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

Document, AVC-356
October 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 92/535

Title: **Proposal for Test Model 2, revision 2**

Status: **Draft**

Source: **Test Model Editing Committee**

CONTENTS

1 INTRODUCTION 8
2 GENERAL CODEC OUTLINE 9
2.1 Arithmetic Precision 9
3 SOURCE FORMATS 10
3.1 Input Formats 10
3.2 Definition of fields and frames 11
3.3 Conversion of CCIR 601 to the Input formats 12
3.3.1 Conversion of CCIR 601 to the 4:2:0 format 12
3.3.2 Conversion of CCIR 601 to SIF 13
3.3.3. Conversion of CCIR 601 to SIF Odd and SIF Even 14
3.3.4 Conversion of CCIR 601 to HHR 15
3.3.5 Conversion of CCIR 601 to SIF Interlaced (SIF-I) 16
3.4 Conversion of the Input Formats to CCIR 601 16
3.4.1 Conversion of the 4:2:0 Format to CCIR 601 16
3.4.2 Conversion of SIF to CCIR 601 17
3.4.3 Conversion of SIF Odd and SIF Even to CCIR 601 18
3.4.4 Conversion of HHR to CCIR 601 19
3.4.5 Conversion of SIF interlaced to CCIR 601 19
4 LAYERED STRUCTURE OF VIDEO DATA 21
4.1 Sequence 21
4.2 Group of pictures 21
4.3 Picture 22
4.3.1 Field-Structure Picture 22
4.3.2. Frame Pictures 22
4.4 Macro block Slice 23
4.4.1 Slave Slice (Frequency scalable extension) 24
4.4.2 Slices in a PictureCompatibility Experiment G.2(i) 24
4.4.3 Slices in a PictureCompatibility Experiment G.2(ii) 24
4.4.4 Slices in a PictureCompatibility Experiment G.3 24
4.4.5 Slice in a PictureCompatibility Experiment G.4 25
4.5 Macroblock 25
4.5.1 Slave_macroblock (Frequency scalable extension) 26
4.6 Block 27
4.6.1 Scaled_block (Frequency scalable extension) 27
4.6.2 Slave_block (Frequency scalable extension) 27
5 MOTION ESTIMATION AND COMPENSATION 28
5.1 Motion Vector Estimation 28
5.1.1 Full Search 29
5.1.2 Half pel search 29
5.2 Motion Compensation 30
5.2.1 Frame Motion Compensation 31
5.2.2 Field Motion Compensation 31
6 MODES AND MODE SELECTION 33
6.1 Picture types 33
6.2 Macroblock types in an intra picture 33
6.3 Macroblock types in a predicted picture 34
6.4 Macroblock types in an interpolated picture 34
6.5 Selection criteria 35
6.5.1 Motion Compensation/No Motion Compensation - Frame/Field 35
6.5.2 Forward/Backward/Interpolative - Field/Frame prediction 35

6.5.3 Compatible prediction 35
6.5.4 Intra/Inter coding 35
6.5.5 Modified Quantizer 36
6.5.6 Field/Frame DCT coding decisions 36
6.5.7 Coded/Not Coded 37
7 TRANSFORMATION AND QUANTIZATION 38
7.1 Quantization of Intra Macroblocks 38
7.1.1 DC Coefficients 38
7.1.2 AC Coefficients 38
7.2 Quantization Non Intra Macroblocks 39
7.3 Dequantization 40
7.3.1 Intra-coded macroblocks 40
7.3.2 Non-Intra-coded macroblocks 40
8 CODING 41
8.1 Macroblock Addressing 41
8.2 Macroblock Type 42
8.2.1 Compatible Prediction Flag 42
8.2.2 Field/Frame Coding Flag 42
8.2.3 Field/Frame Motion Compensation Flag 42
8.3 Motion Vectors 42
8.4 Coded Block Pattern 45
8.4.1 4:2:0 45
8.4.2 4:2:2 45
8.4.3 4:4:4 45
8.5 Intra picture Coefficient Coding 45
8.5.1 DC Prediction 45
8.5.2 AC Coefficients 46
8.6 Non-Intrapicture Coefficient Coding 47
8.6.1 Intra blocks 47
8.6.2 Non intra blocks 47
8.6.3 Frequency scalable blocks 47
8.7 Coding of Transform Coefficients 47
9 VIDEO MULTIPLEX CODER 48
9.1 Method of Describing Bitstream Syntax 48
9.2 Mnemonics 49
9.3 Specification of the Coded Video Bitstream Syntax 50
9.3.1 Start Codes 50
9.3.2 Video Sequence Layer 51
9.3.3 Sequence Header 52
NOTE: This syntax is an interim solution for the representaion of coding standard, picture format and relative picture size. 54
9.3.4 Group of Pictures Layer 56
9.3.5 Picture Layer 57
9.3.6 Slice Layer 59
9.3.7 Macroblock Layer 60
9.3.8 Block Layer 64
10 RATE CONTROL AND QUANTIZATION CONTROL 66
Step 1 - Bit Allocation 66
Complexity estimation 66
Picture Target Setting 67
Step 2 - Rate Control 68
Step 3 - Adaptive Quantization 69

Known Limitations 70
APPENDIX A: DISCRETE COSINE TRANSFORM (DCT) 71
Accuracy Specification 71
APPENDIX B: VARIABLE LENGTH CODE TABLES 72
Introduction 72
B.1 Macroblock Addressing 72
B.2 Macroblock Type and Compatible Macroblock Type 73
B.3 Macroblock Pattern 75
B.4 Motion Vectors 76
B.5 DCT Coefficients 77
APPENDIX C : VIDEO BUFFER VERIFIER 82
C.1 Video Buffering Verifier 82
APPENDIX D: SCALABILITY SYNTAX, ENCODER, AND DECODER DESCRIPTION 84
D.1 INTRODUCTION 84
D.2 LAYERED STRUCTURE OF VIDEO DATA AND MULTIPLEXING OF FREQUENCY SCALES 86
D.3 MOTION ESTIMATION AND COMPENSATION 86
D.4 MODES AND MODE SELECTION 87
D.5 UPWARD DCT COEFFICIENT PREDICTION 87
D.6 TRANSFORMATION AND QUANTIZATION 88
D.6.2 Transformation 88
D.6.2 Upward Coefficient Prediction and Quantization 88
D.6.3 BANDWIDTH CONTROL OF RESOLUTION LAYERS 88
D.7 CODING 89
D.7.1 DCT COEFFICIENT CODING 89
D.8 VIDEO MULTIPLEX CODER 91
D.9 RATE CONTROL AND QUANTIZATION CONTROL 92
Appendix F: CELL LOSS EXPERIMENTS 93
F.1 Cell loss 93
F.1.1 Bitstream specification 93
F.1.2 Calculation of cell loss probabilities 93
F.1.3 Calculation of mean cell loss rate 94
F.1.4 Calculation of mean burst of consecutive cells lost 94
F.1.5 Calculation of cell loss probabilities 95
F.1.6 Simulation of cell loss 95
F.1.7 Random number generation 96
F.2 Parameters 96
F.3 Experment to be supplied by Mike Biggar 97
F.4 Core Experiment on Cell Loss resilience by using frequency scanning 97
F.4.1 Global parameters of the experiment 97
F.4.2 Description of MUVLC 97
F.4.3 Syntax modifications 97
Appendix G: COMPATIBILITY AND SPATIAL SCALABILITY 101
G.1 EXPERIMENTS LIST 101
G.2 EXPERIMENT DETAILS 102
G.2.1 Spatial-temporal weighting 102
G.2.2 Coding 105
Selection method 107
Quantization 107
Coefficient coding 107
Appendix H: LOW DELAY CODING 109
H.1 Simulation guidelines for low delay profile 109

H.1.1 Coding structure. 109
H.1.2 Handling of scene change to maintain low delay. 109
H.1.3 How to handle the first picture using forced intra slice 114
H.1.4 Definition of intra slice/column coding. 114
H.1.5 Rate control 115
H.1.6 Influence of leaky prediction on low delay coding 116
H.2 Experiments 116
H.2.1 Comparison of coding efficiency among predictions at low delay. 117
H.2.2 Comparison of coding efficiency and delay between using intra picture and forced intra slice 118
H.2.3 Comparison of coding efficiency and delay between with and without picture skipping 118
APPENDIX I : FREQUENCY DOMAIN SCALABILITY CORE EXPERIMENTS 119
Core Experiment I.1: Interlace-in-Interlace Extraction 119
I.1.1: Background 119
I.1.2: Description 120
I.1.3: Syntax Changes 120
Core Experiment I.2: Pyramid Encoder Improvements 121
I.2.1: Background 121
I.2.2: Description 122
I.2.3: Syntax Changes 122
Core Experiment I.3: Maximum Quality Encoder 123
Core Experiment I.4: Scalable Side Information 125
I.4.2. Experiment details 125
I.4.3. Syntax extensions 125
I.4.4. Coding 127
I.4.5. Slave Macroblock Type 129
Core Experiment I.5: Scalable VLC coding 129
I.5.1: Experiment 1 129
I.5.2: Experiment I.5.2 131
I.5.2.2: Syntax Changes and VLC's (from MPEG 92/361) 132
Core Experiment I.6: Slice vs. MB Rate Control 132
I.6.1: Background and Description 132
I.6.2: Syntax extensions 134
Core Experiment I.7: Encoder with Drift Correction Layer and Improved Motion Rendition 135
I.7.1 SIMULATION CONDITIONS 135
I.7.2 SYNTAX OF THE PROPOSED EXPERIMENT 136
I.7.3 ENCODER DESCRIPTION 136
I.7.4 DECODER DESCRIPTION 137
I.7.5 EXPERIMENTS 138
Core Experiment I.8: Frequency Scanning 138
I.8.1 Global parameters of the experiment 139
I.8.2 Principle of MUVLC 139
I.8.3 Arrangement of luminance and chrominance coefficients 140
I.8.4 Syntax modifications 141
I.8.5 Supplementary experiments 143
I.8.6 References 144
Core Experiment I.9: Efficient Frequency Scalability Core Experiment 144
I.9.1: Background and Description 144
I.9.2 Syntax Specification 145
Core Experiment I.10: Spatial/Frequency Hybrid Scalability 145
I.10.1: Background and Description 145
I.11 Core experiment on multi loop decoding 145

I.11.1 Experimental conditions 145
I.11.2 Parameter under study 146
I.11.3 Variations of the Experiment 146
Core Experiment I.12: Adaptive Inter-scale Prediction 146
I.12.1. Objective 146
I.12.2. Simulation Conditions 147
Appendix K: MOTION COMPENSATION FOR SIMPLIFIED FAMC 148
K.1. Motion Compensation 148
Appendix L: CORE EXPERIMENTS ON PREDICTION MODES 158
Core Experiment No.1 Simplified FAMC 158
Core Experiment No.2 SVMC (Single Vector Motion Compensation) 158
L.2.1 SVMC motion estimation for frame picture 158
L.2.2 SVMC Program (Except FAMC) for frame picture 161
L.2.3 Syntax change corresponding to SVMC 165
Core Experiment No.3 DUAL-PRIME (Dual') 166
Core Experiment No.4 Global Motion Compensation 173
Core Experiment No.5 Leaky Prediction 1 174
Core Experiment No.6 Leaky Prediction 2 175
Core Experiment No.7 Reverse Order Prediction 175
Core Experiment No.8 Simplification of Test Model 176
Core Experiment No.9 16x8 Sub-Macroblock MC 177
Core Experiment No.10 Special Prediction Modes 178
L.10.1. Definitions 178
L.10.2. Temporal Scaling of the Motion Vector 178
L.10.3. Reference Fields for SVMC3 and DUAL-PRIME 179
L.10.4. Decision for Field-based Prediction 179
L.10.5. Concise Specification of SVMC3 179
L.10.6. Concise Specification of DUAL-PRIME 181
L.10.7. Core Experiment L-10 182
L.11 8x8 Motion Vectors 184
L.12 Intra/Inter Mode Decisions for 8x8 Blocks 186
APPENDIX Q: QUANTISATION EXPERIMENTS 188
Q.1 SCANNING 188
Q.1.1 DEPENDENT SCANNING 188
Q.1.2 INDEPENDENT SCANNING 188
Q.2 EXPERIMENTS ON MATRIX ADAPTATION 189
Q.2.1 MATRIX ADAPTATION WITH MACROBLOCK TYPE 189
Q.2.2 EXPERIMENT OF ADAPTIVE CONTROL OF MATRIX SELECTION AT MB LEVEL 189
Q.2.3 Quantizer Weighting Matrices 190
Q.3 EXPERIMENTS ON QUANTISER RANGE, PRECISION, AND CONTROL 191
Q.3.1 EXPERIMENT EXTENDING RANGE OF TRANSMITTED COEFFICIENT 191
Q.3.2 EXPERIMENT TO INCREASE PRECISION OF INTRA DC COEFFICIENTS 192
Q.3.3 EXPERIMENT TO EXTEND RANGE OF MQUANT 192
Q.3.4 EXPERIMENT ON BLOCK ADAPTIVE QUANTISATION 193
Q.4 Alternate DCT (DCT/NTC) 194
Q.4.1 INTRODUCTION 194
Q.4.2 CORE EXPERIMENT REFERENCE MODEL 194
Q.4.3 DESCRIPTION OF NTC1 METHOD 196
Q.4.4 DESCRIPTION OF NTC2 METHOD 196
Q.5 NON-8 x 8 DCT 200
Q.6 Adaptive VLC's 202
Q.6.1 Frequency scanning 202

Q.6.2 Macroblock Switable VLC (AT&T) 209
Q.6.3 Alternative VLC (Sony) 210
Q.7 Vector Quantisation 214
Q.7.1 Desciption of Method 214
Q.7.2 Assessment Criteria 231

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.

Those parts which have been changed from TM2, revision 1 are marked up by a bar in the margin.

The readers are asked to give comments and corrections to remove ambiguous parts. Please send them to:

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

2 GENERAL CODEC OUTLINE

The generic structure of the test 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 bitstreams as described in Appendix D.
- Experiments for cell loss are given in Appendix F.
- Experiments for compatibility are given in Appendix G.

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.

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, except for CCIR 601 which is well defined in the CCIR documentation.

3.1 Input Formats

The Input Formats 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.

Input Formats derived from CCIR 601 and defined in this chapter are:

- 4:2:2-50 The actual CCIR 601 signal at 50Hz
- 4:2:2-60 The actual CCIR 601 signal at 60Hz
- 4:2:0 CCIR 601 with half chrominance resolution at 50Hz
- 4:2:0 CCIR 601 with half chrominance resolution at 60Hz
- SIF-525 Progressive format, with 1/8 of the 4:2:0-60 resolution
- SIF-625 Progressive format, with 1/8 of the 4:2:0-50 resolution
- CIF 625 Progressive format, as SIF but fixed paramets, being the maximum of SIF-525 and SIF-
- SIF-odd SIF taken from CCIR 601 odd fields
- SIF-even SIF taken from CCIR 601 even fields
- HHR Interlaced format with Half horizontal resolution of 4:2:0
- SIF-I Interlaced format, with 1/8 of the 4:2:0 resolution

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

	4:2:0-625	4:2:2-625	4:2:0-525	4:2:2-525
Number of active lines				
Luminance (Y)	576	576	480	480 / 488 ?
Chrominance (Cb,Cr)	288	576	240	480 / 488 ?
Number of active pixels per line				
Luminance (Y)	704	704	720	720
Chrominance (Cb,Cr)	352	352	360	360
Frame rate (Hz)	25	25	30	30
Frame aspect ratio (hor:ver)	4:3	4:3	4:3	4:3

Table 3.1: Active 4:2:0 and 4:2:2 Formats

For compatibility with MPEG1 or scalability a second set of formats is defined, the MPEG1 SIF. The term SIF is used to indicate this format defined in table 3.2. 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 Hz or 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

For conversions to several formats a number of filters will be defined. At the edges of the picture data is it recommended to repeat the pixel value at edge.

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.

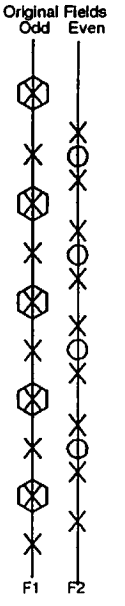


Figure 3.1: 4:2:0 Chrominance sub sampling in the fields

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

NOTE: The horizontal positions of the chrominance sample is not rigth in between 4 luminance pixels.

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.

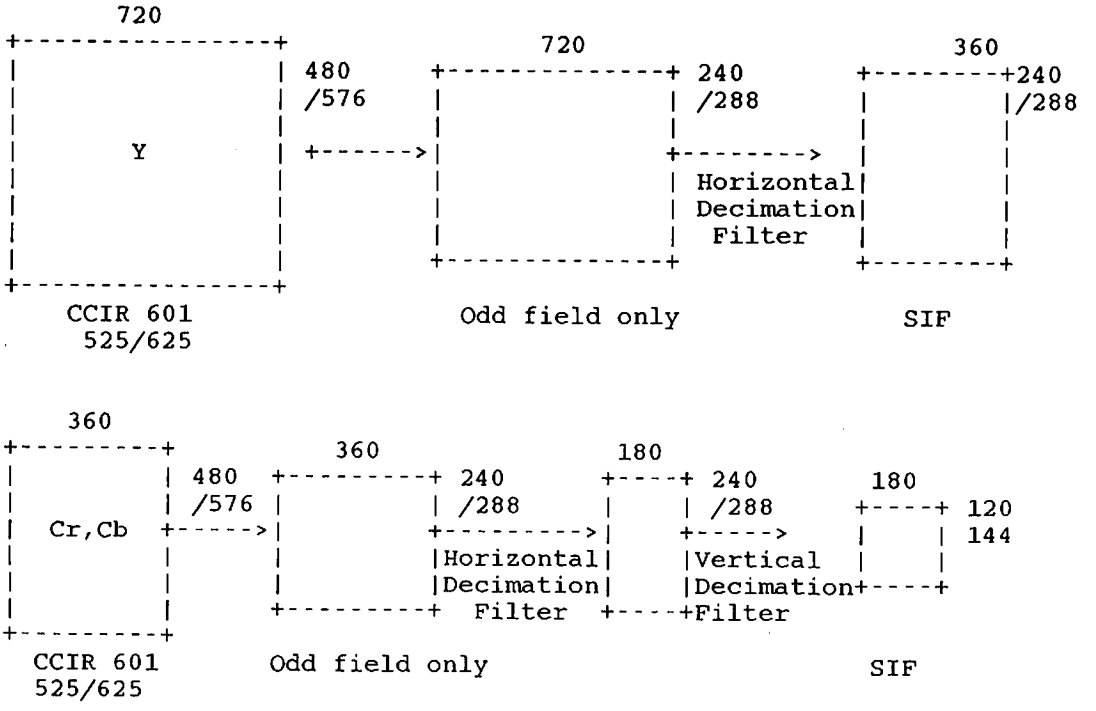


Figure 3.3 Conversion from CCIR 601 into SIF

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

3.3.3. Conversion of CCIR 601 to SIF Odd and SIF Even

The CCIR 601 formats are converted into their corresponding SIF Odd by sampling odd fields and SIF Even by sampling even fields and then applying horizontal decimation files of Table 3.4 in each case.

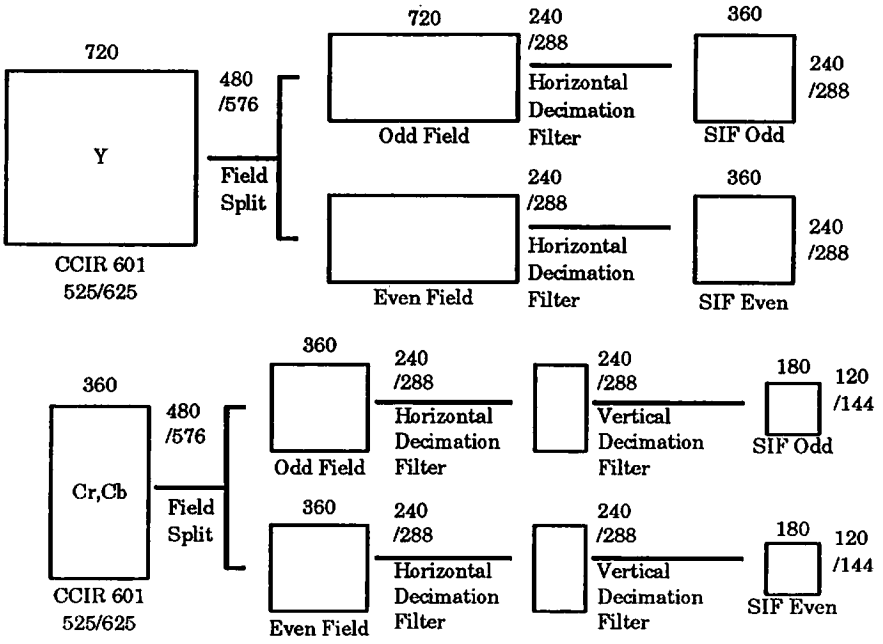


Fig 3.4: Conversion from CCIR 601 into SIF Odd and SIF Even

3.3.4 Conversion of CCIR 601 to HHR

The CCIR 601 formats are converted into their corresponding HHR's by first decimating to SIF Odd and SIF Even as in previous section, and then creating interlaced frames by alternating between lines of SIF Odd's and SIF Even's.

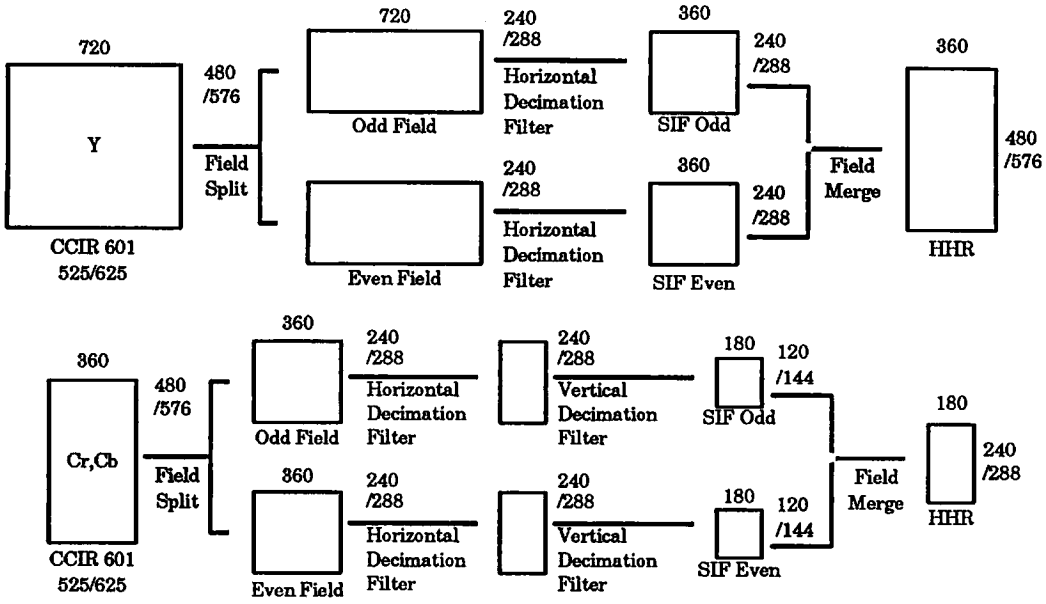


Fig. 3.5 Conversion from CCIR 601 into HHR

3.3.5 Conversion of CCIR 601 to SIF Interlaced (SIF-I)

The CCIR formats are converted into their corresponding SIF interlaced by following the sequence of decimation operations show in Fig. 3.6. The horizontal filter used for decimation is from Table 3.4. The filter used for vertical decimation of odd fields is also from Table 3.4; for decimation of even fields a new filter is specified below.

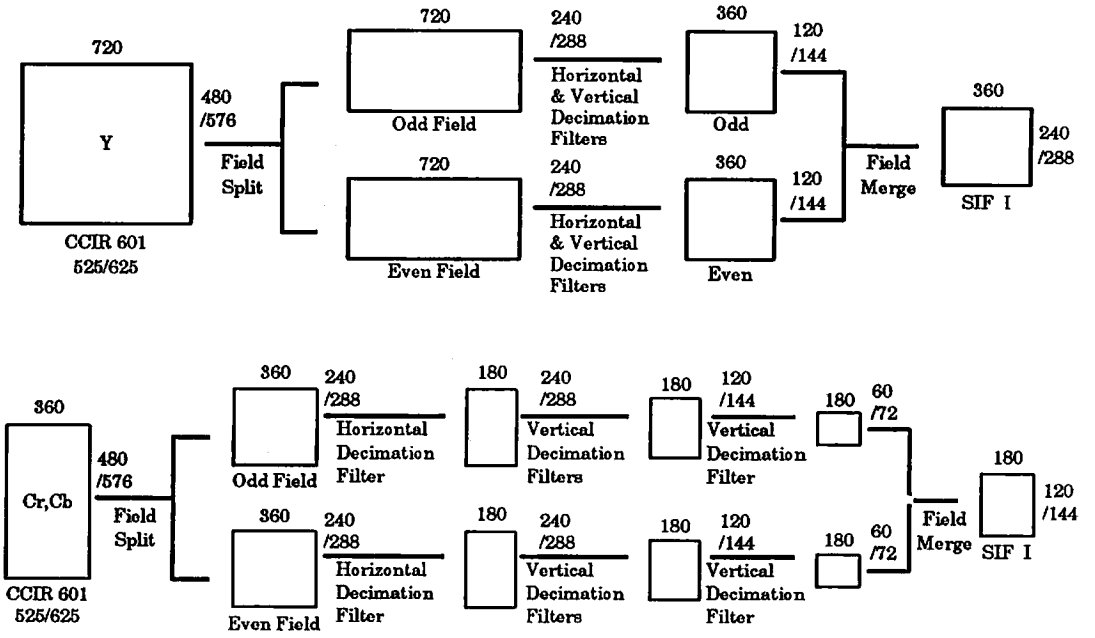


Fig. 3.6: Conversion from CCIR 601 into SIF Interlaced (SIF-I)

Horizontal Filter: Table 3.4

Vertical Filter:

Odd Field	Table 3.4
Even Field	-4, 23, 109, 109, 23, -4

Note: The SIF-I interlaced pictures generated seem devoid of jerkiness but appear blurry. Better choice of decimation filters needs further investigation.

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.

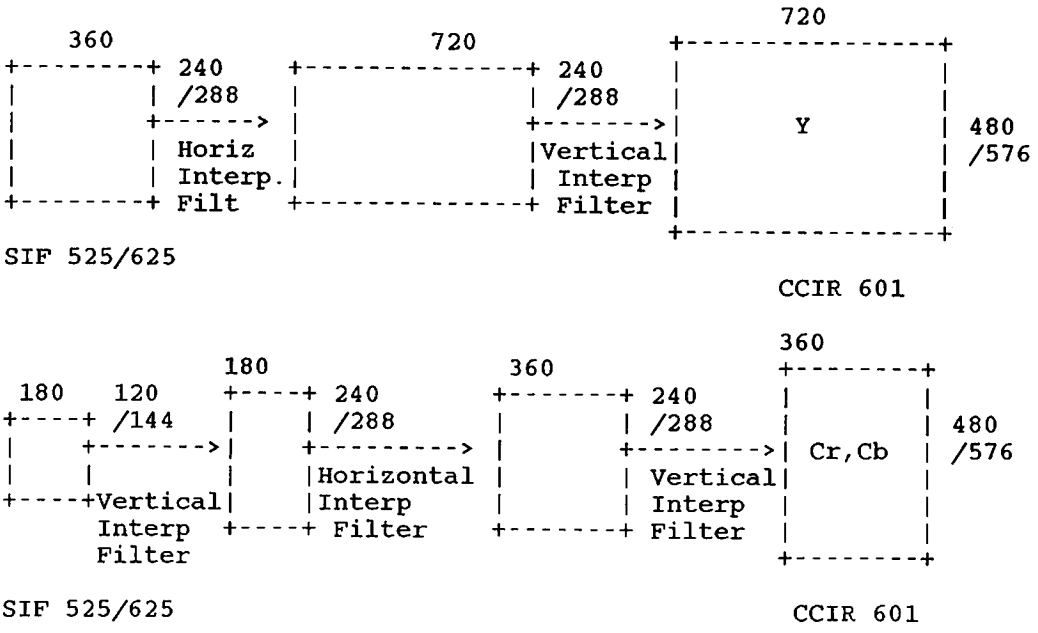


Figure 3.7: Conversion of SIFs to CCIR 601 formats

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.

3.4.3 Conversion of SIF Odd and SIF Even to CCIR 601

SIF Odd and SIF Even are interpolated using interpolation filters of Table 3.5 and interlaced CCIR 601 is created by merging of fields by alternating between lines of upsampled odd and even fields.

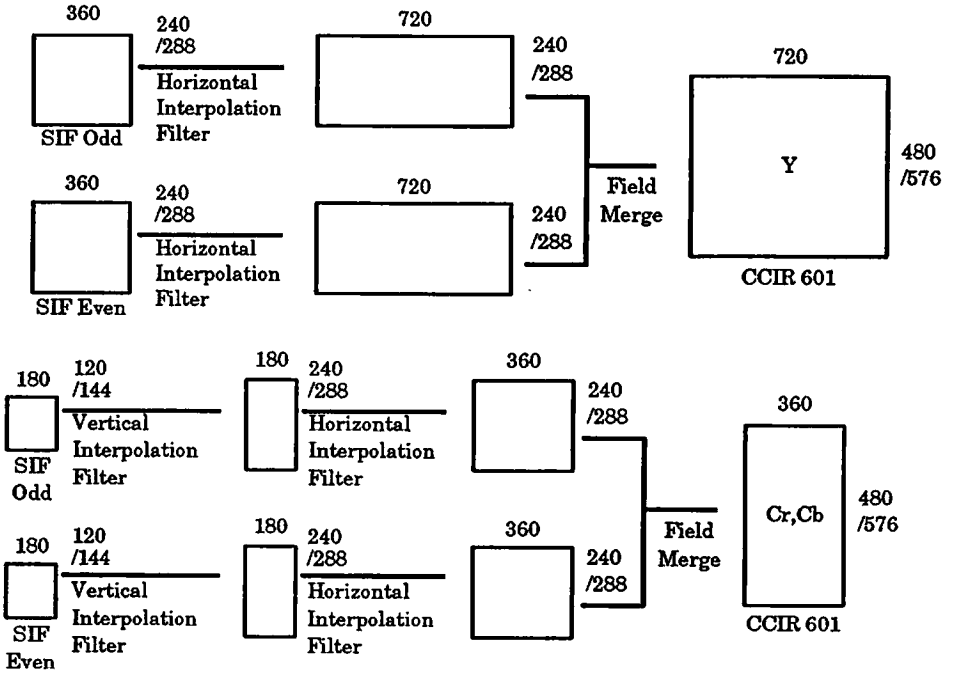


Figure 3.8: Conversion of SIF Odd and Even to CCIR 601

3.4.4 Conversion of HHR to CCIR 601

HHR is split into SIF Odd and SIF Even, each of which are interpolated using interpolation filter of Table 3.5 and interlaced CCIR 601 is created by merging of fields consisting of alternating between lines of upsampled odd and even fields.

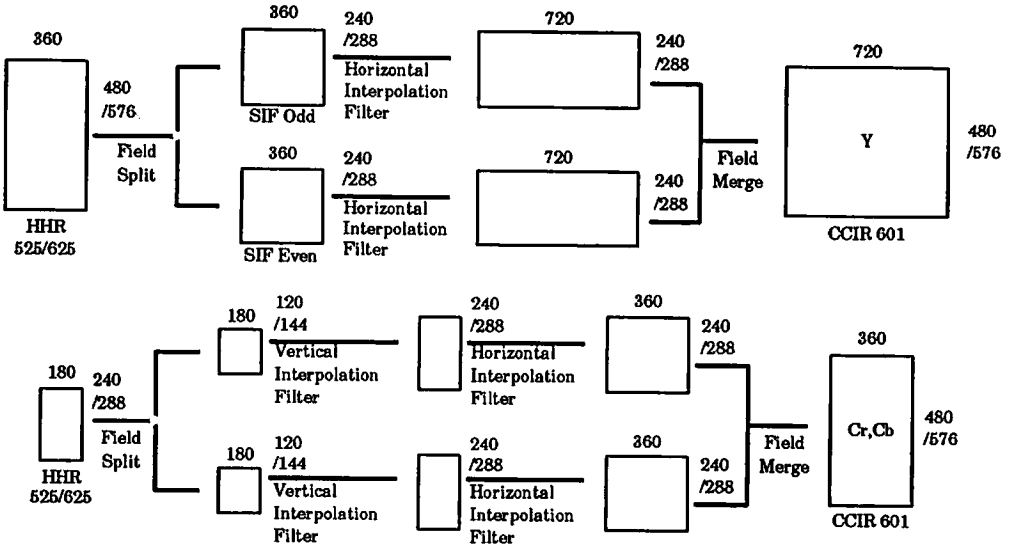


Figure 3.9: Conversion of HHR to CCIR 601

3.4.5 Conversion of SIF interlaced to CCIR 601

SIF interlaced format is interpolated to CCIR 601 by following the sequence of operations shown in Fig. 3.9. The horizontal filter used for interpolation is that of Table 3.5. The filter used for vertical interpolation of odd fields is also that of Table 3.5; for vertical interpolation of even fields a new filter is specified below.

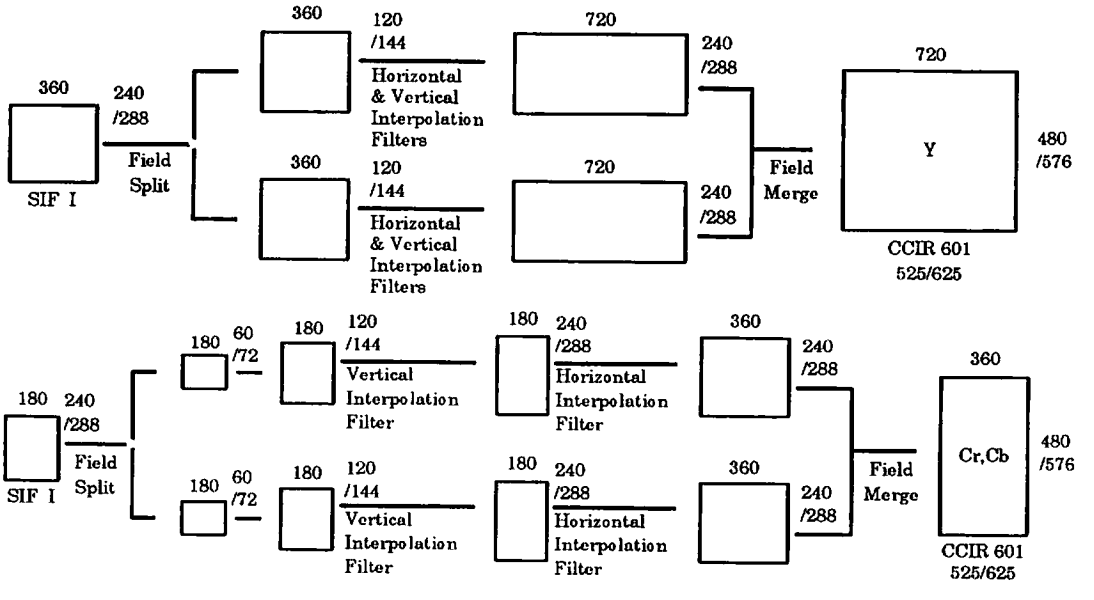


Figure 3.10: Conversion of SIF interlaced (SIF-I) to CCIR 601

Horizontal Filter:	Table 3.5
Vertical Filter:	
Odd field:	Table 3.5
Even field:	-4, 40, 220, 220, 40, -4

Note: The upsampled CCIR 601 pictures seem devoid of jerkiness but appear quite blurry. Better choice of interpolation filters needs further investigation.

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 pictures. The order in which pictures are displayed differs from the order in which the coded versions appear in the bitstream. In the bitstream, the first frame in a Group of Pictures is always an intra picture. 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 bi-directional picture of the consecutive series of bi-directional pictures which immediately precedes the first intra picture.

It should be noted that the first Group of pictures will start with an Intra Picture, and as consequence this Group of pictures will have less Bi-directional pictures then the other Groups of pictures.

A sequence can contain at the same time Field-Pictures and Frame-Pictures. However, in most experiments, the Field/Frame nature of the Pictures is predefined according to the picture-coding-type, and according to the hardware complexity that would be required to decode such sequences.

As guideline, the following levels are proposed for experiments (cf document 334):

Level 5 : (Equivalent to the Frame-Sequence of TM-1)
All pictures are transmitted as Frame-picture
Level 4 :.
I- and P-pictures are transmitted as Frame-picture.
B-Pictures are transmitted as Field-pictures only.
Level 4 (alternate) :.
P-Pictures are transmitted as Frame-picture.
I-Pictures are Field-pictures, field 2 being a P-field.
B-Pictures must be transmitted as Field-picture only.
Level 3 :
Each I-, P- and B-Picture is transmitted in Field-picture only.
B-Field are predicted using only 3 reference fields
Level 2 : (Equivalent to the Field-Sequence of TM-1 with 2 reference fields)
Each I-, P- and B-Picture is transmitted in Field-picture only.
B-Field are predicted using only 2 reference fields
Level 1 : (for low delay experiments)
Each Picture is transmitted in Field-picture.
No B-Picture is allowed.
P-Fields are predicted using 2 reference fields
Level 0 : (cheapest hardware)
Each Picture is transmitted in Field-picture.
No B-Picture is allowed.
P-Fields are predicted using only 1 reference field

Field 1 can be used as prediction for field 2 except in the case of B-Fields.

The number of fields that can be used for prediction is flexible when Field-Picture are used. For forward prediction, the minimum number of reference fields is 1, and the maximum is 2.

Reference fields for forward prediction are the latest two fields NOT part of a B-picture. This means in particular that when a Field-structure P-Picture is decoded, field 2 will use field 1 as one reference. As a side effect, unless M=1, FAMC cannot be used to predict the field 2 of a Field-structure P-Picture.

4.3 Picture

Each Picture can be Frame-Structure or Field-Structure this is applicable to interlaced and progressive material. Frame structure, frame prediction may be used for progressive material. However it does not prohibit to use frame field adaptive or field structure.

4.3.1 Field-Structure Picture

The terms *Field-Picture* and *Frame-Picture* is used instead of *Field-Structure Picture.* and *Frame-Structure Picture.*

A Field-Picture is formed of two fields transmitted in a successive order. The transmission order of the fields is fixed and same as display order (FIELD1 first, FIELD2 second) in case of the simplified syntax.

In the syntax for the core experiments the transmission order of the fields is flexible in case of Field-Structure P- or I-Pictures, but not flexible in case of Field-Structure B-Pictures (field 1 followed by field 2),

In Field-Structure P- and B-Pictures, both fields must be P- or B-. The fields of those Field-Pictures are called *P-Field* and *B-Field.*.

However, when an Intra Field-Picture, It is possible (but not required) to use Predictive-coding-type to transmit the second field.

A Picture Header is transmitted *before each field* of the Field-Picture. Therefore, if a transmission error causes the decoder to desynchronize while decoding field 1 of a Field-Structure B-Picture, a resync can be done at the Picture Header of field 2, and field 2 can be decoded correctly.

In case of Field-Pictures, Field 1 can be used as prediction for field 2 except in the case of B-Fields.

For all P-Field-Pictures the two last coded P-Field-Pictures are used as prediction.

4.3.2. Frame Pictures

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 for the frame-picture coding mode is shown in Figure 4.1. In the figure frames -1 to 13 are part of a Group of Picture.

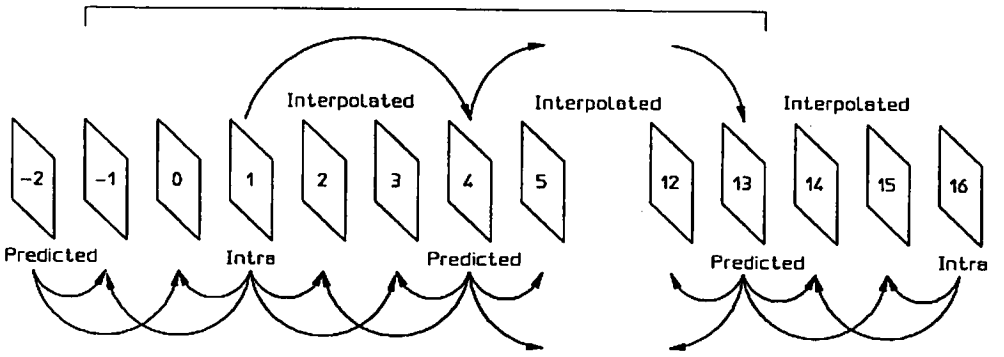


Figure 4.1 Structure of a Group of Pictures in Frame Picture Coding mode in display order

4.4 Macro block Slice

A frame is devided into a number of contiguous macroblock slices, for this TM a fixed structure is used and given in figure 4.2.

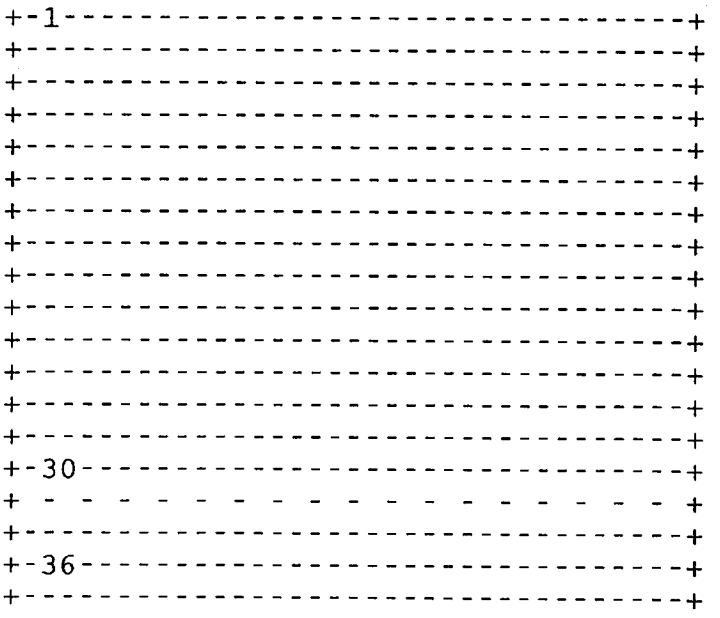


Figure 4.2 Arrangement of Slices in a Picture in Frame Coding mode

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.

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.

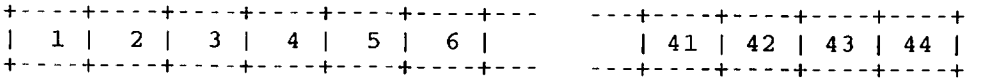


Figure 4.3: Test model Macroblock Slice Structure

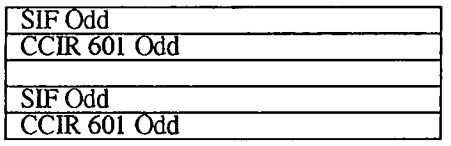
When scalable extensions are used (Annex D), the Slice layer may contain Macroblocks of resolution lower than 16x16.

4.4.1 Slave Slice (Frequency scalable extension)

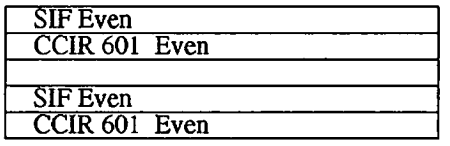
Slave slices are layers of Slave macroblocks which are spatially co-located with the Macroblocks in the Slice layer.

4.4.2 Slices in a PictureCompatibility Experiment G.2(i)

Odd Field Slice Multiplexing

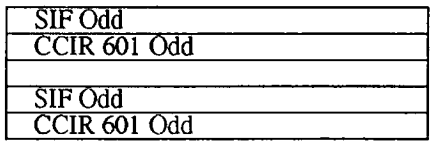


Even Field Slice Multiplexing

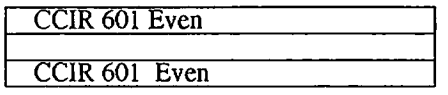


4.4.3 Slices in a PictureCompatibility Experiment G.2(ii)

Odd Field Slice Multiplexing

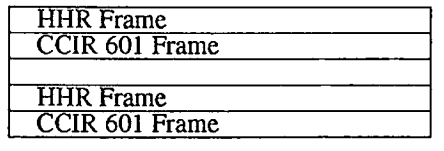


Even Field Slice Multiplexing



4.4.4 Slices in a PictureCompatibility Experiment G.3

Frame Slice Multiplexing



4.4.5 Slice in a PictureCompatibility Experiment G.4

Frame Slice Multiplexing

SIF Odd
SIF Odd
SIF Even
CCIR 601 Frame
CCIR 601 Frame
SIF Odd
SIF Even
CCIR 601 Frame
CCIR 601 Frame
SIF Odd
SIF Even
CCIR 601 Frame
CCIR 601 Frame
SIF Even
CCIR 601 Frame
CCIR 601 Frame

4.5 Macroblock

A 4:2:0 Macroblock consists of 6 blocks. This structure holds 4 Y, 1 Cb and 1 Cr Blocks and is depicted in figure 4.4a.

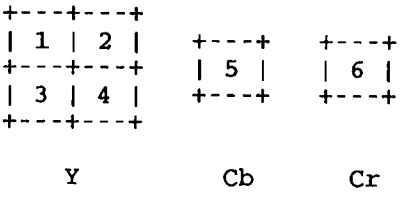


Figure 4.4a: General 4:2:0 Macroblock structure

When the picture format is 4:2:2, a Macroblock consists of 8 blocks.

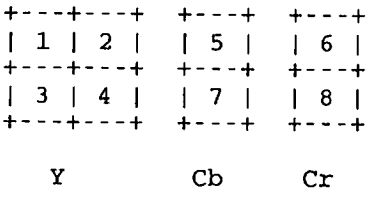


Figure 4.4b: 4:2:2 Macroblock structure

When the picture format is 4:4:4, a Macroblock consists of 12 blocks.

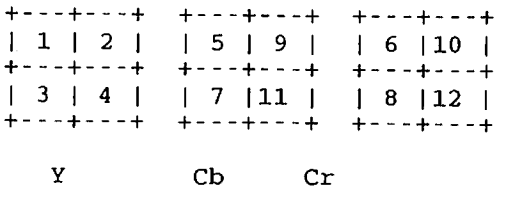


Figure 4.4c: 4:4:4 Macroblock structure

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.

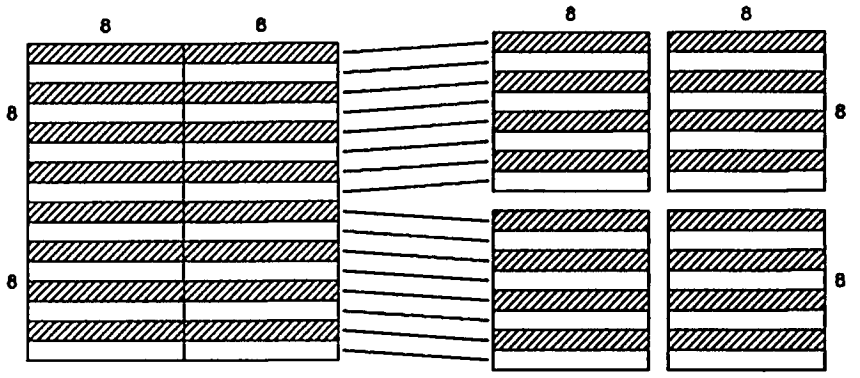


Figure 4.5 : Luminance Macroblock Structure in Frame DCT Coding

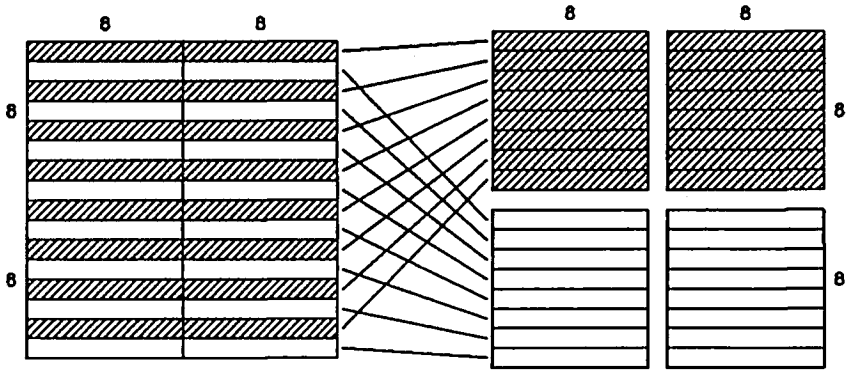


Figure 4.6 : Luminance Macroblock Structure in Field DCT Coding

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

4.5.1 Slave_macroblock (Frequency 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.

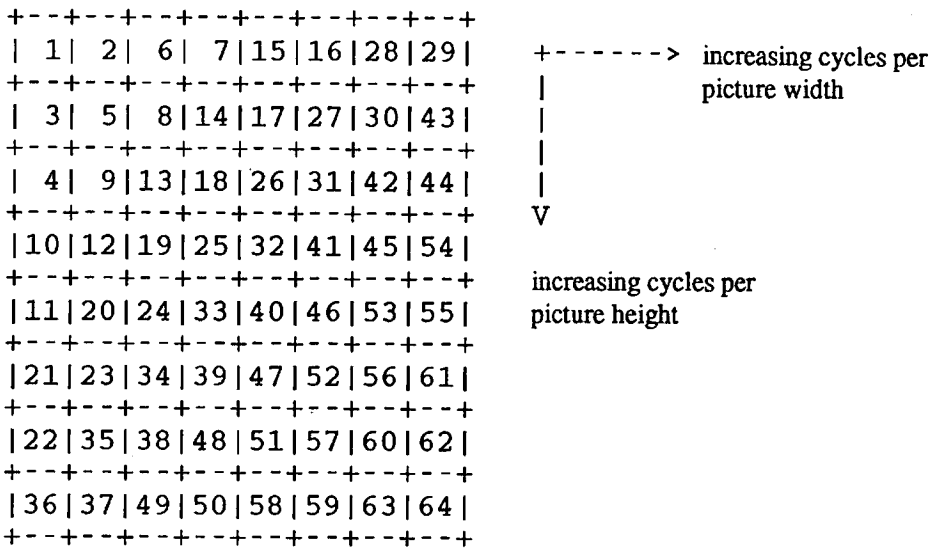


Figure 4.7: Block structure

4.6.1 Scaled_block (Frequency scalable extension)

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

4.6.2 Slave_block (Frequency 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 frequency scalable bitstream.

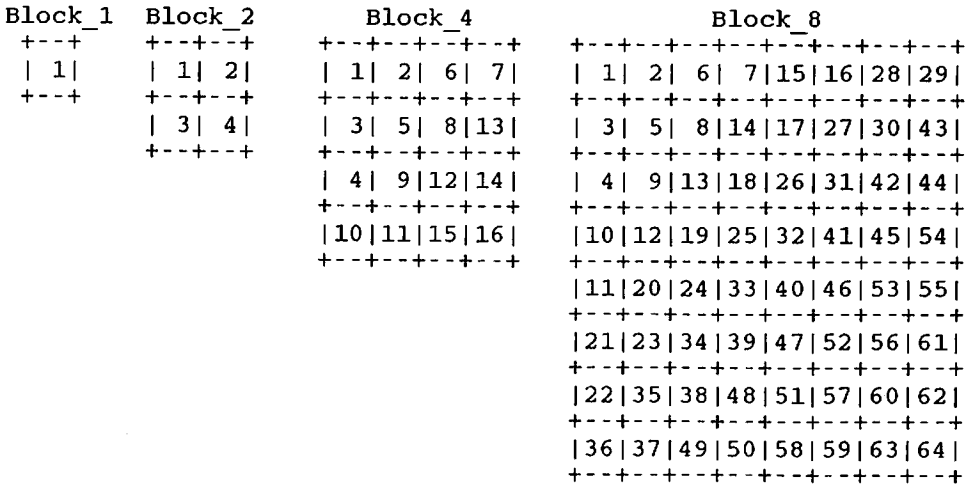


Figure 4.8: Block structures for scalable bitstreams

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	± 7 pels horizontal, ± 3 pels vertical/field
Flower Garden	±15 pels/frame	± 7 pels horizontal, ± 3 pels vertical/field
Calendar	±15 pels/frame	± 7 pels horizontal, ± 3 pels vertical/field
Popple	±15 pels/frame	± 7 pels horizontal, ± 3 pels vertical/field
Football	±31 pels/frame	±15 pels horizontal, ± 7 pels vertical/field
PRL CAR	±63 pels/frame	±31 pels horizontal, ±15 pels vertical/field

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 referenced 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.

NOTE: In field searches, the refence system is the correspondig field. In a field the line distance is 1.

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.

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

where FMV is the forward motion compensated macroblock, thus making reference to a 'previous picture', and BMV is the backward motion compensated macroblock, making reference to a 'future picture'.

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

```
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.

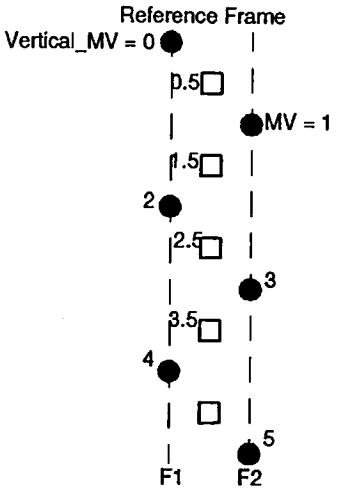
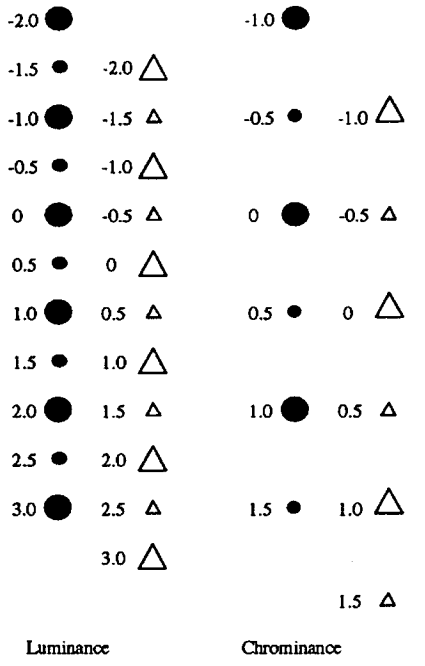


Figure 5.1: Frame Motion Compensation

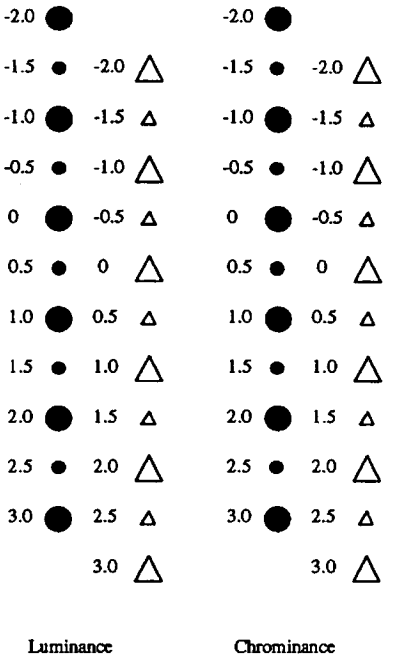
5.2.2 Field Motion Compensation

Field-based MV is expressed in the very same way as frame-based vectors would be if the source (reference) field and the destination field were considered as "frames" (see Figure).

Considering that in each field, lines are numbers 1.0, 2.0, 3.0, ... (1 is the top line of the field), if the pel located at line "n" of the destination field is predicted from line "m" of the reference field, the vertical coordinate of the field vector is "n-m".
Note: "m" and "n" are expressed in units of one vertical half-pel in the field.
When necessary, **motion_vertical_field_select** (one bit) will be transmitted to identify the selected field.



4:2:0 format



4:2:2 format and 4:4:4 format

5.2.2.1. Chrominance Field-based MV

In 4:2:0 sequences :

- The vertical coordinate of the chrominance Field-based MV is derived by dividing by 2 the vertical coordinate of the luminance Field-based MV, as done in MPEG-1.
- The horizontal coordinate of the chrominance MV (Field-based or Frame-based) is derived by dividing by 2 the horizontal coordinate of the luminance MV, as done in MPEG-1.

In 4:2:2 sequences :

- The vertical coordinate of the Field-based MV for chrominance is equal to the vertical coordinate of the luminanceField-based MV.
- The horizontal coordinate of the chrominance MV (Field-based or Frame-based) is derived by dividing by 2 the horizontal coordinate of the luminance MV, as done in MPEG-1.

In 4:4:4 sequences :

- The horizontal (resp. vertical) coordinate of MV for chrominance is equal to the horizontal (resp. vertical) coordinate of the luminance MV.

We use "field_motion _type" in the macroblock layer to specify 16x8 motion compensation block. If "field_motion_type" is "10", 18x8 motion compensation is used instead of SFAMC.

"field_motion_type"

code	prediction type	motion_vector_count	mv_format
10	16x8	2	field

"sub_MB_type" should be added just after "backward_reference_field".
This is 1 bit flag, uimsbf.
sub_MB_type=="0" indicates the condition written in Core Exp. L9 3(1).
sub_MB_type=="1" indicates the condition written in Core Exp. L9 3(2).

6 MODES AND MODE SELECTION

In section 6.1, a coding structure with different picture modes is introduced. Within each picture, macroblocks may be coded in several ways, thus aiming at high coding efficiency. The MB modes for intra, predicted and interpolated pictures 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 picture of a sequence starting with the first picture is coded as intra picture i.e. pictures 1, N+1, etc. (see Fig. 5.1). Following every M-th picture in between (within a Group of Pictures) is a predicted picture coded relative to the previous predicted or intra picture. The interpolated pictures are coded with reference to the two closest previous and next predicted or intra pictures. In this TM, M=3 and N=15 for 29.97 Hz and M=3 and N=12 for 25 Hz.

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

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

Coding modes available for predicted and interpolated pictures 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 bitstream, 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 pictures 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 bitstream, see also chapter 9, the macroblock layer section.

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

NOTE: If two previous fields are allowed as reference fields, the noMC mode does not specify which of the reference fields is to be used for the prediction. Thus, the noMc mode shall refer to the reference field of the same parity as the target field. In the case a (0,0) MV is to be used with the reference field of the opposite parity, the noMC mode cannot be used. The (0,0) MV must be explicitly transmitted (after the appropriate selection bit).

6.4 Macroblock types in an interpolated picture

In interpolated pictures 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 bitstream, see also chapter 9, the macroblock layer section.

Macroblock type selection is done in the following order:
- Interpolative/Forwards/Backwards - Frame/Field prediction
- Intra/Inter
- Compatible prediction

- Modified quantizer
- Field/frame DCT coding
- Coded/not Coded

6.5 Selection criteria

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

6.5.1 Motion Compensation/No Motion Compensation - Frame/Field

For P-pictures, the decision of selecting the Frame Motion Vector or the Field Motion Vector is by SE 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-picture 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.

See appendix G.

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 picture 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.)

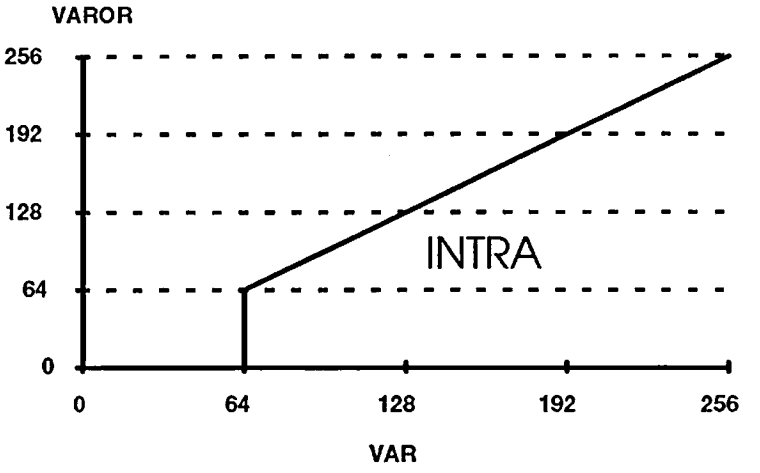


Figure 6.1: Characteristic Intra/Inter

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 decisions

Frame based DCT coding rather than field based DCT coding is used if the following equation holds:

```
if (var_1 <= var_2 + offset)
        Frame based DCT coding
else
        Field based DCT coding
```

offset = 4096 for intra MB
offset = 0 for inter MB
Where var_1 and Var_2 are calculated with the following lines:

```
var_1 = 0;
var_2 = 0;
for (Pix = 0; Pix < 16; Pix++) {
    for (Line = 0; Line < 16; Line += 2) {
        Sum = O(Pix, Line) - O(Pix, Line+1);
        var_1 += (sum * sum);
    }
    for (Line = 0; Line < 16; Line += 4) {
        Sum_1 = O(Pix, Line) - O(Pix, Line+2);
        Sum_2 = O(Pix, Line+1) - O(Pix, Line+3);
        var_2 += (sum_1 * sum_1) + (sum_2 * sum_2)
    }
}
```

where O(Pix, Line) denotes a pel of 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 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 quantised by individual quantisation factors,

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

where q(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 (also called quantiser_scale see section 9). Mquant is calculated for each macroblock by the algorithm defined in Section 10 and is stored in the bitstream 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:

$$QAC(i,j) = [ac\sim(i,j) + sign(ac\sim(i,j))*((p * mquant) \,//\, q)] \,/\, (2*mquant)$$

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

For this TM p=3, and q = 4.

7.2 Quantization Non Intra Macroblocks

Non-intra macroblocks in Predicted and Interpolated pictures 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 pictures 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:

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

$$QAC(i,j) = \begin{cases} ac\sim(i,j) \;/\; (2*mquant) & \text{IF mquant == odd} \\ (ac\sim(i,j)+1) \;/\; (2*mquant) & \text{IF mquant == even AND } ac\sim(i,j)<0 \\ (ac\sim(i,j)-1) \;/\; (2*mquant) & \text{IF mquant == even AND } ac\sim(i,j)>0 \end{cases}$$

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

7.3 Dequantization

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) * wI(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 EVEN 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 bitstream 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 Pictures. 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 * wN(i,j) / 16

if (QAC(i,j) < 0)
        rec(i,j) =  (2 * QAC(i,j) - 1) * mquant * wN(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 bitstream 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 frequency and spatially scalable bitstreams, refer to Annex D, G and I.

8.1 Macroblock Addressing

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

In Intra pictures there are no skipped macroblocks. In predicted pictures 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 pictures, 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 picture. 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.

The macro_block_escape is a fixed bit-string "0000 0001 000" which is used when the difference macroblock_address_increment is greater then 33. It causes the value of macroblock_address_increment to be 33 greater than the value that will be decoded by subsequenct macroblock_escapes and the macroblock_address_increment codewords.

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

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.

8.2 Macroblock Type

Each picture has one of the three modes:

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

For these three picture 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

$$$$ Update
A one or two-bit codeword, **compatible_type,** immediately follows the MBTYPE VLC, if the bitstream is indicated as being compatible in the sequence header.

The definition of this codeword is in the definition of the Macroblock layer.

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 4. 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 pictures 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 pictures a No MC decision corresponds to a reset to zero of the prediction motion vector.)

- In interpolative pictures, 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

In MBs that are field DCT coded, chrominance block structure is as follows :
o When the picture format is 4:2:2 and 4:4:4, the chrominance blocks structure is analogous to that of the luminance.
o When the picture format is 4:2:0, the chrominance blocks is structure is equal to that used for frame coded MBs. In other words, chrominance is always frame coded.

It was agreed that when frame-based prediction is used in non-progressive pictures, the reference field for chrominance prediction may not be the correct one.

There are four prediction motion vectors : PMV1, PMV2, PMV3 and PMV4. They are reset to zero at the start of a slice and at intra-coded MBs.

The prediction MVs (PMV1 to PMV4) *are always expressed in Frame MV coordinates.*
For the prediction of Field-based MVs (mv_type == "field"), the following rules are used:

- **On the decoder side :**
  When a Field-based MV is derived, the vertical coordinate of the PMV is shifted right by 1 bit (with sign extension) before adding the vertical differential.
  Then the Field-based MV is stored in the appropriate PMV(s) after shifting left by 1 bit its vertical coordinate.
- **On the encoder side :**
  When a Field-based MV is encoded, the vertical coordinate of the PMV is shifted right by 1 bit (with sign extension) before it is subtracted from the field MV vertical coordinate.
  Then the Field-based MV is stored in the appropriate PMV(s) after shifting left by 1 bit its vertical coordinate.

1. mv_type == "frame" :

In P-Pictures or P-Fields, PMV1 is used. PMV2, PMV3 and PMV4 are reset to PMV1

In B-Pictures or B-Fields, PMV1 is used for forward motion vector prediction, and PMV3 is used for backward motion vector prediction. PMV2 is reset to PMV1, and PMV4 is reset to PMV3.

2. mv_type == "field" :

In P-Frame-Pictures or P-Field-Pictures:
PMV1 is used for vectors used to predict FIELD1 from FIELD1
PMV2 is used for vectors used to predict FIELD1 from FIELD2
PMV3 is used for vectors used to predict FIELD2 from FIELD1
PMV4 is used for vectors used to predict FIELD2 from FIELD2

In B-Picture, PMV1 and PMV3 are used for forward motion vector prediction from field 1 and 2, and PMV2 and PMV4 are used for backward motion vector prediction from fields 1 and 2.

8.4 Coded Block Pattern

There are three types for the coded block pattern, one for the 4:2:0, one the 4:2:2 coding modes and one for 4:4:4.

8.4.1 4:2:0

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 Coded Block pattern

8.4.2 4:2:2

When the picture format is 4:2:2, the pattern number is coded with an 8 bit FLC.

8.4.3 4:4:4

When the picture format is 4:4:4, use 12 bit FLC.

8.5 Intra picture 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 loss less 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-Intrapicture Coefficient Coding

8.6.1 Intra blocks

Intra blocks in non-intra pictures are coded as in intra pictures. 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 pictures (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.

8.6.3 Frequency scalable blocks

Coding of intra and non intra blocks in a frequency 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 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. See table B.5g

DOCUMENT 408 PROPOSES MAXIMUM 24 BITS WORD FOR RUN/LEVEL ESCAPE AND EXTENDED RANGE

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 Bitstream Syntax

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

The action caused by a decoded data element in a bitstream depends on the value of that data element and on data elements previously decoded. The 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 bitstream. 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 bitstream 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 bitstream.

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 Bitstream 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_positions	00000101 through 000001AF
reserved	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.

9.3.1.1 Slice Start Codes - For Scalability and Compatibility

name	hexadecimal value
picture_start_code	00000100
slice_start_codes (including slice_vertical_positions	00000101 through 000001AF
slice_start_codes (note)	000001B0
sif_even_start_code	000001B1
user_data_start_code	000001B2
sequence_header_code	000001B3
sequence_error_code	000001B4
extension_start_code	000001B5
ccir_slice_start_code	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	

note: In the above table, slice_start_codes identify lowest spatial or frequency scal in the bitstream. Thus in a compatible coding scheme (G.2..G4.) they are used to identify sif_odd_slice or hhr_slice, whereas in frequency scalable schems or in spatial/frequency hybrid schemes they identify the lowest frequency scale present.

[Editors note: proper names need to be defined for sif_even_start_code and ccir_slice_start_code in order to reflect the generic coding approach of MPEG, this table needs to be mixed with the table in 9.3.1]

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_value	**12**	**uimsbf**
vertical_size_value	**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**
sscalable	**1**	**uimsbf**
fscalable	**1**	**uimsbf**
chroma_format	**2**	**uimsbf**
extent_horizontal_size	**1**	**uimsbf**
if (extenet_horizontal_size)		
horizontal_size_extension	**4**	**uimsbf**
extent_vertical_size	**1**	**uimsbf**
if (extenet_vertical_size)		
vertical_size_extension	**4**	**uimsbf**
reserved	**1**	**uimsbf**
if (fscalable) {		
do {		
fscale_code	**8**	**uimsbf**
} while (nextbits != '00000111')		
end_of_fscales_code	**8**	**'00000111'**
}		
if (sscalable) {		
do {		
sscale_code	**8**	**uimsbf**
} while (nextbits != '00001111')		
end_of_sscales_code	**8**	**'00001111'**
}		
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()
  }
}
```

horizontal_size_value - This word forms the 12 least significant bits from horizontal_size.

vertical_size_value - This word forms the 12 least significant bits from vertical_size.

horizontal_size - The horizontal_size is a 16 bit unsigned integer, the 12 least significant bits are defined in horizontal_size_value, the 4 most signifincant bits are defined in horizontal_size_extension. The horizontal_size is the width of the displayable part of each luminance picture in pels. The width of the encoded luminance picture in macroblocks, mb_width, is (horizontal_size+15)/16. The displayable part of the picture is left-aligned in the encoded picture.

vertical_size - The vertical_size is a 16 bit unsigned integer, the 12 least significant bits are defined in vertical_size_value, the 4 most signifincant bits are defined in vertical_size_extension. The vertical_size is the height of the displayable part of each luminance picture in pels. The height of the encoded luminance picture in macroblocks, mb_height, is (vertical_size+15)/16. The displayable part of the picture is top-aligned in the encoded picture.

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

0 No horizontal extension
1 Horizontal extension

Default value: 0

horizontal_size_extension - This word forms the 4 most significant bits from horizontal_size.

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

0 No vertical extension
1 Vertical extension

Default value: 0

vertical_size_extension - This word forms the 4 most significant bits from vertical_size.

NOTE: It should be take into consideration that by extending the horizotal and vertical size to 64 K pels, the requirement on picture size which was new in the Rio meeting, is not yet met. Therefore it is suggested that the requirements group will take this new requirement into it's due consideration.

extended_syntax - This is NOT a syntax element, **extended_syntax** is set to 1 when the extension_start_code is found in the sequence_header, otherwise it is set to 0.

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

0	not fscalable
1	fscalable

Default value: 0

sscalable - This is a one-bit integer defined in the following table:

0	not sscalable
1	sscalable

Default value: 0

sscale_code - This is an 8-bit integer that defines the coding standard and compatibility for each spatial resolution lower layer. The DCT size for all spatial scales is 8. The 8 bit integer is split into 3 fields to indicate coding standard, format and subsampling ratio.

standard name 3 bits		format 2 bits		subsampling ratio 3 bits	
code	standard	code	format	code	ssr
0	H.261	0	progressive	0	
1	MPEG-1	1	odd	1	1/2 h
2	MPEG-2	2	even	2	1/2 v
3	reserved	3	interlaced	3	1/2 h, 1/2 v
4	"			4	reserved
5	"			5	"
6	"			6	"
7	end			7	"

Example: A bitstream with sscale_code 43 (hex) would be interpreted to mean that the lower layer is MPEG-1 coded, ODD field only and the resolution is 1/2 both vertically and horizontally (i.e if this layer CCIR601 then the lower layer is SIF_ODD).

NOTE: This syntax is an interim solution for the representaion of coding standard, picture format and relative picture size.

chroma_format - This is a 2 bit integer defined in the following table:

00	reserved
01	4:2:0
10	4:2:2
11	4:4:4

Default value: 01

fscale_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:

fscale_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_fscales_code**
8	reserved
256	reserved

Example: The bitstream with fscale_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.

sscale_code - This is an 8-bit integer that defines the coding standard and compatibility if any for each spatial resolutioni layer. The DCT size for all spatial scales is 8. The values of this integer are:

sscale code	Feature of Spatial Scale
0	not compatible
1	MPEG-1 compatible sif odd
2	H@261 compatible cif
3	MPEG-1 compatible sif i
4	MPEG-1 compatible hhr
5	sif even coded with MPEG-1 compatible sif odd
6	ccir 601 coded with MPEG-1 compatible sif odd
7	ccir 601 coded with MPEG-1 compatible sif i
8	ccir 601 coded with sif even and MPEG-1 compatible sif odd
9	ccir 601 coded with MPEG-1 compatible hhr
10	reserved
11	reserved
@	@
@	@
@	@
15	end of sscale code
@	@
@	@
@	@
255	reserved

Example: A bitstream with sscale-codes 1,5,8,15 would be interpreted to mean that there are three spatial layers, MPEG-1 compatible SIF Odd, followed by SIF Even coded with respect to SIF Odd, followed by CCIR 601 coded with respect to both SIF Odd and SIF Even.

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 )
}
```

temporal_reference -- The temporal_reference is an unsigned integer associated with each input picture. It is incremented by one, modulo 1024, for each input picture. For the earliest picture (in display order) in each group of pictures, the temporal_reference is reset to zero.

Temporal Reference is the straightforward source picture order because of the allowance of picture drop.

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 == 3) {
    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
    if ( picture_coding_type == 2 || picture_coding_type == 3) {
      forward_vertical_f_code                                 3           uimsbf
    }
    if ( picture_coding_type == 3 ) {
      backward_vertical_f_code                                3           uimsbf
    }
    picture_structure                                         2           uimsbf
    forward_reference_fields                                  2           uimsbf
    backward_reference_fields                                 2           uimsbf
    if ( chroma_format == "01" ) {    /* 4:2:0 */
      chroma_postprocessing_type                              1           uimsbf
    } else {
      reserved                                                1           uimsbf
    }
    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()
    if ( fscalable || sscalable ) {
```

```
        while ( nextbits() == slave_slice_start_code ) {
          slave_slice()
        }
      }
    } while ( nextbits() == slice_start_code )
}
```

temporal_reference -- The temporal_reference is an unsigned integer associated with each input picture. It is incremented by one, modulo 1024, for each input picture. For the earliest picture (in display order) in each group of pictures, the temporal_reference is reset to zero. Its unit is "Frame". (see also Appendix H on Low delay)

forward_vertical_f_code --

backward_vertical_f_code --

picture_structure - This is a 2-bit integer defined in the table below.

11	Frame-Picture
01	Field 1 of a Field-Picture
10	Field 2 of a Field-Picture
00	reserved

Default value: 11

forward_reference_fields - This is a 2-bit integer defined in the table below.

11	Forward prediction from Field 1 and Field 2
01	Forward prediction only from Field 1
10	Forward prediction only from Field 2
00	No forward prediction in this Picture

In Intra Pictures this field is allways set to "00".
Default value: 11, what is the meaning of this field for non-interlaced pictures

backward_reference_fields - This is a 2-bit integer defined in the table below.

11	Backward prediction from Field 1 and Field 2
01	Backward prediction only from Field 1
10	Backward prediction only from Field 2
00	No backward prediction in this Picture

In Intra and Predicted Pictures this field is always set to "00".
Default value: 11, what is the meaning of this field for non-interlaced pictures

chroma_postprocessing_type - This is a 1bit integer that indicate how the chrominance samples of a 4:2:0 Picture must be postprocessed for display. It is defined in the table below.

0	SIF interlaced (for interlaced Pictures)
1	SIF (for progressive Pictures)

Default value: 1

9.3.6 Slice Layer

slice() {		
slice_start_code	**32**	**bslbf**
quantizer_scale	**5**	**uimsbf**
if (fscalable) {		
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 frequency 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**
if (extended_syntax) { /* not MPEG-1 syntax */		
if (sscalable) {		
compatible_type	**1**	**uimsbf**
if (compatible_type) {		
compatible_code_macroblock()	**....**	**.....**
} else {		
macroblock_type	**1-6**	**vlclbf**
}		
} else {		
macroblock_type	**1-6**	**vlclbf**
}		
if (macroblock_motion_forward ||		
macroblock_motion_backward) {		
if (picture_structure == "11") { /* Frame-Picture */		
frame_motion_type	**2**	**uimsbf**
} else {		
field_motion_type	**2**	**uimsbf**
}		
}		
if (picture_structure == "11") /* Frame-Picture */		
&& ((macroblock_intra || macroblock_pattern))		
dct_type	**1**	**uimsbf**
}		
} else { /* MPEG-1 syntax */		
macroblock_type	**1-6**	**vlclbf**
}		
if (macroblock_quant)		
quantizer_scale	**5**	**uimsbf**
if (macroblock_motion_forward)		
forward_motion_vectors()	**...**	**...**
if (macroblock_motion_backward)		
backward_motion_vectors()	**...**	**...**
if (macroblock_pattern)		
coded_block_pattern()	**...**	**...**
for (i=0; i<block_count; i++) {		
if (scalable) {		
scaled_block(i)		
}		
else {		
block(i)		
}		
}		
if (picture_coding_type == 4)		
end_of_macroblock	**1**	**"1"**
}		

compatible_code_macroblock() {		
weight_code	2	uimsbf
compatible_macroblock_type	1-6	vlclbf
}		

compatible_type -- This is a one-bit integer indicating whether the macroblock has prediction from a lower layer enabled. If this is set to "1", the macroblock has prediction from the lower layer.

weight_code -- This is a two-bit code (for the moment) indicating the temporal-spatial weight used for the prediction from the lower layer, defined in the following table:

	Case 1	Case 2		Case 3	
weight_code	w1	w1	w2	w1	w2
00					
01					
10					
11					

compatible_macroblock_type -- For the three picture types I,P and B there are different VLC tables for the compatible_macroblock_types. These are given in tables B.2e for intra pictures, B.2f for predicted pictures and B.2g for bidirectional pictures.

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

field_motion_type - This is a 2-bit code indicating the macroblock motion prediction, defined in the following table:

code	prediction type	**motion_vector_count**	**mv_format**
00	Field-based prediction	1	field
01	Dual-field prediction	2	field
10	Simplified FAMC	1	frame
11	CORE EXPERIMENTS	...	...

frame_motion_type - This is a 2-bit code indicating the macroblock motion prediction, defined in the following table:

code	prediction type	**motion_vector_count**	**mv_format**
00	Field-based prediction	2	field
01	Frame-based prediction	1	frame
10	Simplified FAMC	1	frame
11	CORE EXPERIMENTS	...	...

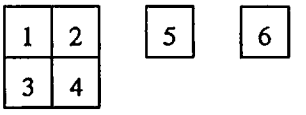
motion_vector_count - This is NOT a syntax element. **motion_vector_count** is derived from **field_motion_type** or **frame_motion_type** as indicated in the tables.

mv_format- This is NOT a syntax element. **mv_format** is derived from **field_motion_type** or **frame_motion_type** as indicated in the tables. **mv_format** indicates if the motion vector is a field-motion vector or a frame-motion vector. **mv_format** is used in the syntax of the motion vectors.and in the process of motion vector prediction.

block_count - This is *not* a syntax element. **block_count** is derived from chroma_format as follows :

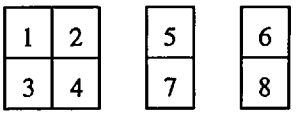
chroma_format	block_count
4:2:0	6
4:2:2	8
4:4:4	12

The 6 blocks in a 4:2:0 MB are numbered in the following order :



In this case, ***cbp*** is a 6-bit integer represented as : <b1><b2><b3><b4><b5><b6>, where <b1> is the most significant bit. <b*n*> is set to 1 when block *n* is coded.

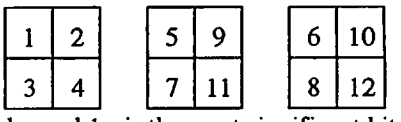
The 8 blocks in a 4:2:2 MB are numbered in the following order :



In this case, ***cbp*** is a 8-bit integer represented as : <b1><b2><b3><b4><b5><b6><b7><b8>, where <b1> is the most significant bit. <b*n*> is set to 1 when block *n* is coded.

NOTE THAT ORDER HAS BEEN CHANGED FOR 4:2:2 FOR EASIER HARDWARE IMPLEMENTATION.

The 12 blocks in a 4:4:4 MB are in the following order :



In this case, ***cbp*** is a 12-bit integer represented as : <b1>...<b12>, where <b1> is the most significant bit. <b*n*> is set to 1 when block *n* is coded.

```
motion_vector () {
   motion_horizontal_code                          1-11        vlclbf
   if ( (horizontal_f != 1) &&  (motion_horizontal_code != 0) )
      motion_horizontal_r                          1-6         uimsbf
   motion_vertical_code                            1-11        vlclbf
   if ( (vertical_f != 1) && (motion_vertical_code != 0) )
      motion_vertical_r                            1-6         uimsbf
}
```

```
forward_field_motion_vector () {
   if (forward_reference_field == "11")
      motion_vertical_field_select                 1           uimsbf
   motion_vector()                                 ...         ...
}
```

```
backward_field_motion_vector () {
   if (backward_reference_field == "11")
      motion_vertical_field_select                 1           uimsbf
   motion_vector()                                 ...         ...
}
```

COMMENT : IN CASE OF DUAL-FIELD PREDICTION AND "DUAL-PRIME" CORE EXPERIMENT,, THE SYNTAX ALLOWS ARBITRARY REFERENCE FIELD SELECTION FOR EACH VECTOR. THIS FLEXIBILITY MAY BE NOT NECESSARY

motion_vertical_field_select - This a 1 bit defined as follows :

0	prediction comes from Reference Field 1
1	prediction comes from Reference Field 2

```
forward_motion_vectors () {
  if ( motion_vector_count == 1 ) {
    if ( mv_format == frame ) {
      motion_vector()                          ...    ...
    } else {
      forward_field_motion_vector()            ...    ...
    }                                          ...    ...
  } else {
    forward_field_motion_vector()              ...    ...
    forward_field_motion_vector()              ...    ...
  }
}
```

```
backward_motion_vectors () {
  if ( motion_vector_count == 1 ) {
    if ( mv_format == frame ) {
      motion_vector()                          ...    ...
    } else {
      backward_field_motion_vector()           ...    ...
    }                                          ...    ...
  } else {
    backward_field_motion_vector()             ...    ...
    backward_field_motion_vector()             ...    ...
  }
}
```

```
coded_block_pattern () {
  if ( chroma_format == "01" )       /* 4:2:0 */
    coded_block_pattern_420                    3-9    vlclbf
  else if ( chroma_format == "10" )  /* 4:2:2 */
    coded_block_pattern_422                    8      uimsbf
  else                               /* 4:4:4 */
    coded_block_pattern_444                    12     uimsbf
}
```

9.3.7.1 Slave Macroblock Layer

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

9.3.8 Block Layer

NOTE : 9-BIT DC PRECISION AND EXTENDED RANGE OF QAC COEFFICIENT WOULD IMPLY CHANGES IN THE SYNTAX OF 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 correlated 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 picture. It is performed before coding the picture.

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 picture 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:

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

where S_i, S_p, S_b are the number of bits generated by encoding this picture 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 picture in the Group of pictures (T_i, T_p, or T_b) is computed as:

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

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

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

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 picture , $R = R - S_{i,p,b}$

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 picture in a GROUP OF PICTURES (an I-picture):

$R = G + R$

$G = \text{bit\_rate} * N / \text{picture\_rate}$

N is the number of pictures in the GROUP OF PICTURES.

At the start of the sequence R = 0.

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

I	B	B	P	B	B	P	B	B	P	B	B
						R-bits					
						$N_p = 2$					
						$N_b = 4$					

Figure 10.1 - GROUP OF PICTURES structure

Step 2 - Rate Control

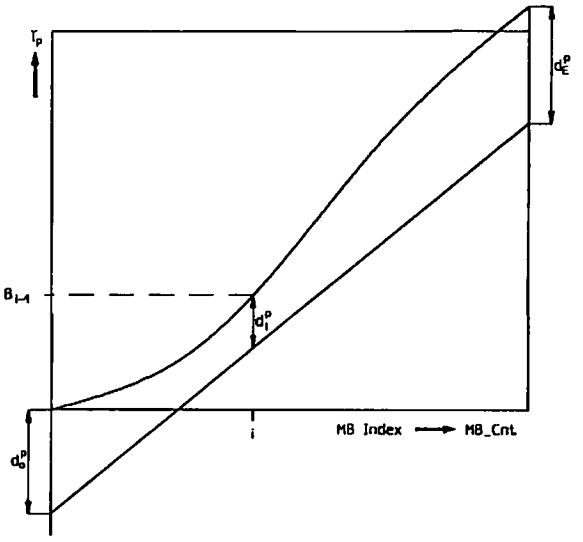


Figure 10.2 : Rate Control for P-pictures

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 picture type.

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

MB_cnt is the number of macroblocks in the picture.

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

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

$$Q_j = \frac{d_j * 31}{r}$$

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 (i.e. original) pixels values:

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

where

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

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

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

Normalise 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 picture to be encoded. On the first picture, 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 picture 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

$$C(u), C(v) = \begin{cases} 1/\mathrm{SQRT}(2) & \text{for } u, v = 0 \\ 1 & \text{otherwise} \end{cases}$$

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 and Compatible 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	macroblock_ compatible
1	0	0	0	0	1	0
01	1	0	0	0	1	0

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	macroblock_ compatible
1	0	1	0	1	0	0
01	0	0	0	1	0	0
001	0	1	0	0	0	0
00011	0	0	0	0	1	0
00010	1	1	0	1	0	0
00001	1	0	0	1	0	0
000001	1	0	0	0	1	0

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	macroblock_ compatible
10	0	1	1	0	0	0
11	0	1	1	1	0	0
010	0	0	1	0	0	0
011	0	0	1	1	0	0
0010	0	1	0	0	0	0
0011	0	1	0	1	0	0
00011	0	0	0	0	1	0
00010	1	1	1	1	0	0
000011	1	1	0	1	0	0
000010	1	0	1	1	0	0
000001	1	0	0	0	1	0

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

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

VLC code	macroblock_ quant	macroblock_ motion_ forward	macroblock motion_ backward	macroblock_ pattern	macroblock_ intra	macroblock_ compatible
1	0	0	0	1	0	1
01	1	0	0	1	0	1
001	0	0	0	0	0	1

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

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

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

VLC code	macroblock_ quant	macroblock_ motion_ forward	macroblock_ motion_ backward	macroblock_ pattern	macroblock_ intra	macroblock_ compatible
10	0	1	0	1	0	1
11	0	0	1	1	0	1
010	0	1	0	0	0	1
011	0	0	1	0	0	1
0010	1	1	0	1	0	1
0011	1	0	1	1	0	1
00011	1	1	0	0	0	1
00010	1	0	1	0	0	1
000011	1	0	0	1	0	1
000010	0	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_next 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 positive '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_next 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 positive, '1' for negative.

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

dct_coeff_first and dct_coeff_nex 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 positive, '1' for negative.

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

dct_coeff_first and dct_coeff_nex 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 positive, '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 0001	-255
1000 0000 0000 0010	-254
...	
1000 0000 0111 1111	-129
1000 0000 1000 0000	-128
1000 0001	-127
1000 0010	-126
...	...
1111 1110	-2
1111 1111	-1
forbidden	0
0000 0001	1
...	...
0111 1111	127
0000 0000 1000 0000	128
0000 0000 1000 0001	129
...	
0000 0000 1111 1111	255

APPENDIX C : VIDEO BUFFER VERIFIER

Constant rate coded video bitstreams 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 of underflow. *If low buffering delay is intended with allowing picture dropping, the buffer occupancy just after decoding a picture shall further conform to Item 5 below.*

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. {For low delay operation, the buffer size B is interpreted as corresponding to the maximum number of bits per picture generated at the encoder when picture dropping takes place}

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. *This item applies to cases that all pictures are coded and transmitted.* 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.

5. *This item applies to cases that some pictures are not coded nor transmitted.*

Encoder may wish to realize low buffering delay with allowing occasional picture droppings. It will regulate its information generation by setting a virtual buffer size Bl smaller than B for stationary pictures. The VBV operates as follows;

All of the data for the non-dropped picture which has been in the buffer longest is instantaneously removed. Then after each interval all data of the non-dropped picture which (at that time) has been in the buffer longest is instantaneously removed. Sequence header, group of picture layer data elements and headers of dropped pictures which immediately precede a picture are removed at the same time as that picture. At some decoding timing where picture dropping takes place in the coder, there will be no sufficient data to remove. In that case, no data removing takes place.

During the stationary state with low buffering delay, the VBV occupancy immediately after each picture is removed shall lie between zero and (Bl - R/P).

To meet this requirement the number of bits dc for the coded picture just before the steady state (including any preceding header and group of picture layer data elements) must satisfy;

no dropped picture	$Bp - Bl + R/P < dc < Bp$
one dropped picture	$Bp - Bl + 2R/P < dc < Bp + R/P$
two dropped pictures	$Bp - Bl + 3R/P < dc < Bp + 2R/P$

n dropped pictures $Bp - Bl + (n+1)R/P < dc < Bp + nR/P$

where:

Bp: VBV buffer occupancy just before removing the data
Bl: VBV buffer corresponding to low delay operation
R : bit rate
P : picture rate

This is a requirement on the video bitstream 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 = bit rate
P = number of pictures per second
t_n is the time when the n'th coded picture is removed from the VBV buffer

$$$$ insert figure here

Figure C.1 VBV Buffer Occupancy

APPENDIX D: SCALABILITY SYNTAX, ENCODER, AND DECODER DESCRIPTION

D.1 INTRODUCTION

The syntax for frequency domain scalable bitstreams has been detailed in Chapter 9. This appendix describes the operation of the Test Model encoder and decoder for frequency domain scalability experiments, in the spirit of a delta with respect to the non-scalable Test Model. Where core experiments depart from these descriptions, the departures are documented in the descriptions of the core experiments themselves.

The frequency domain scalability syntax extensions enable the implementation of hierarchical pyramid and sub-band schemes in the frequency (DCT) domain. Although the syntax is flexible, the Test Model corresponds to a three layer case with the following resolutions:

1. CCIR 601	704x480(576)	(Scale-8)
2. SIF	352x240(288)	(Scale-4)
3. QSIF	176x120(144)	(Scale-2)

Figures D.1 and D.2 are block diagrams of the Test Model encoder and decoder for the frequency domain scalability experiments.

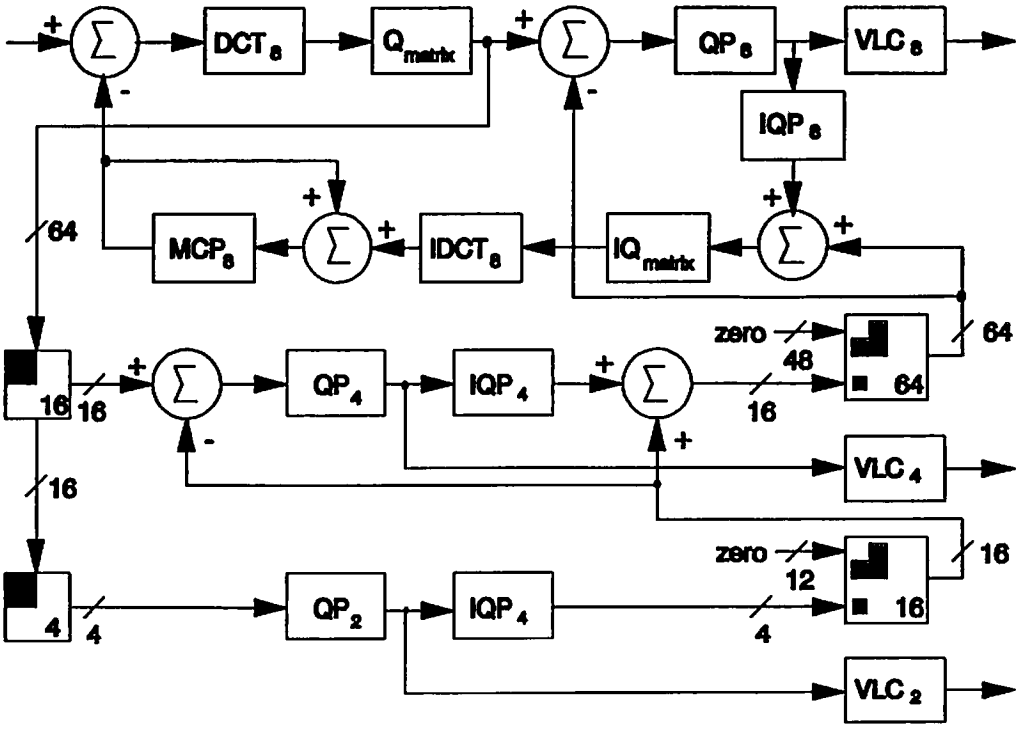


Figure D.1: Frequency domain encoder

(D.1) shift = 1 + log2(8/s)
(D.2) factor = 16/s

(D.3) xs = n >> shift
(D.4) fxs = n - factor * xs
(D.5) ys = m >> shift
(D.6) fys = m - factor * ys

Then, the prediction picture at pixel (x,y) of scale-s Ps(x,y) is given by

Ps(x,y) = ((factor-fys)/factor) * [p1] + (fys/factor) * [p2],

where

p1 = ((factor-fxs)/factor) * Rs(x+xs, y+ys) + (fxs/factor) * Rs(x+xs+1, y+ys),

and

p2 = ((factor-fxs)/factor) * Rs(x+xs, y+ys+1) + (fxs/factor) * Rs(x+xs+1, y+ys+1).

Overall, there is division by factor**2, which can be implemented as a logical shift right by 2* log2(factor) = 2*(4 - log2(scale)) bits.

Chrominance motion-compensation is the same, except that at scale-s, equations (D.1-6) are applied using s/2 instead of s.

D.4 MODES AND MODE SELECTION

The same picture types and coding modes used in the non-scalable Test Model are used in the frequency domain scalable extension. However, the selection criteria are not completely identical. The **motion compensation/no motion compensation, forward/backward/interpolated** and **intra/inter coding** decisions are made by examining the appropriate high resolution signals, in the same manner as for the non-scalable case. However, the determination whether a block is **coded** or **not coded** is made by examining the quantized DCT coefficients of **all** layers. Furthermore, only the frame modes of DCT coding and motion compensation are used, so no frame/field tests are required. The modified quantizer scheme of Test Model 1, Chapter 10 is sufficient as is, since the same rate control strategy is used in the frequency domain scalable extension.

D.5 UPWARD DCT COEFFICIENT PREDICTION

In a pyramidal frequency domain scalable codec, 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 **slice** layer are used to predict the upper-left 2x2 coefficients of the corresponding coefficients in the 4x4 **slave_slice** layer. Similarly, the 4x4 coefficients of the first **slave_slice** layer are used to predict the upper-left 4x4 coefficients of the corresponding coefficients in the 8x8 **slave_slice** 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 of Test Model 1.

D.6 TRANSFORMATION AND QUANTIZATION

D.6.2 Transformation

In order to facilitate upward DCT coefficient prediction, DCT and IDCT formulas with non-standard normalization are used, as defined in Table D.1.

Table D.1: DCT definitions for frequency domain scalability.

Scale	Forward DCT	Inverse DCT
2	4*DCT(2x2)	IDCT(2x2)/4
4	2*DCT(4x4)	IDCT(4x4)/2
8	DCT(8x8)	IDCT(8x8)

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

D.6.2 Upward Coefficient Prediction and Quantization

In the scalable Test Model, the same quantization matrix is used for all resolution scales. For the 2x2 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. Prior to quantization of a particular scale, the quantized and rebuilt coefficients from the next lower scale are used as a prediction of the corresponding coefficients in the current scale. Because of these features and the fact that, in the scalable Test Model extension, the same MQUANT value is applied to the coefficients of all layers, the quantized DCT coefficients in the 2x2 and 4x4 layers could be obtained by simply extracting the appropriate coefficients from the corresponding set of quantized 8x8 coefficients.

(We emphasize that, the generality exists to use at least different MQUANT values in different layers, as is done in some of the Core Experiments. See section D.6.3. It may also be desirable to use different quantization matrices in different scales, in some applications. However, this is not supported in the current Test Model.)

Aside from the above considerations, quantization in the scalable extensions is identical to that in the non-scalable Test Model. In general, to rebuild the DCT coefficients for a target scale, the following steps are needed:

1. dequantization of the DCT coefficients of the target scale and all lower scales using the appropriate quantizer scale factor "MQUANT_s" (MQUANT_2 for scale 2, MQUANT_4 for scale 4, and MQUANT_8 for scale 8), and quantization matrix, and

2. for appropriate coefficients, summation of the corresponding coefficient resulting from the previous step (upward DCT coefficient prediction).

Note that this method differs from that in the scalable extension of Test Model 1.

D.6.3 BANDWIDTH CONTROL OF RESOLUTION LAYERS

The Slave_slice layer specification includes a **quantizer_delta** parameter. For each MB in a **slave_slice** layer, this delta is 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 basic encoder, **quantizer_delta** is always zero.

D.7 CODING

D.7.1 DCT COEFFICIENT CODING

Coding of DCT coefficients is done using a combination of two-dimensional run/amplitude VLC tables and FLC escape tables. The FLC coding is exactly as in Test Model 1. However, there is one VLC table for each scale. The VLC tables are specified below.

Table I: Scale 2 run/amplitude variable length codes

Run	Amplitude	length	Codeword
EOB		1	1
0	1	2	01s
0	2	4	0011s
1	1	4	0010s
0	3	5	00011s
2	1	6	000101s
0	4	6	000100s
1	2	7	0000111s
0	5	7	0000110s
0	6	7	0000101s
1	3	8	00001001s
3	1	8	00001000s
0	7	8	00000111s
ESCAPE		8	00000110
0	8	8	00000101s
2	2	8	00000100s
0	9	9	000000111s
1	4	9	000000110s
0	10	9	000000101s
0	11	9	000000100s
1	5	10	0000000111s
0	12	10	0000000110s
0	13	10	0000000101s
1	6	10	0000000100s
2	3	10	0000000011s
0	14	11	00000000101s
0	15	11	00000000100s
1	7	11	00000000011s
0	16	11	00000000010s
0	17	12	000000000011s
1	8	12	000000000010s
0	18	12	000000000001s

Table II: Scale 4 run/amplitude variable length codes

Run	Amp	Len	Codeword	Run	Amp	Len	Codeword
0	1	2	11s	1	5	10	0000001110s

EOB		2	10		14	1	10	0000001101s
1	1	3	011s		13	1	10	0000001100s
0	2	4	0101s		0	11	10	0000001011s
2	1	4	0100s		3	3	10	0000001010s
3	1	5	00111s		6	2	10	0000001001s
0	3	5	00110s		0	12	10	0000001000s
4	1	5	00101s		1	6	10	0000000111s
1	2	6	001001s		15	1	11	00000001101s
5	1	6	001000s		2	4	11	00000001100s
0	4	6	000111s		0	13	11	00000001011s
6	1	7	0001101s		0	14	11	00000001010s
0	5	7	0001100s		4	3	11	00000001001s
2	2	7	0001011s		1	7	11	00000001000s
9	1	7	0001010s		5	3	11	00000000111s
7	1	7	0001001s		0	15	11	00000000110s
1	3	7	0001000s		9	2	12	000000001011s
8	1	7	0000111s		3	4	12	000000001010s
0	6	7	0000110s		0	16	12	000000001001s
3	2	8	00001011s		1	8	12	000000001000s
0	7	8	00001010s		0	17	12	000000000111s
10	1	8	00001001s		7	2	12	000000000110s
0	8	8	00001000s		8	2	12	000000000101s
1	4	9	000001111s		2	5	13	0000000001001s
4	2	9	000001110s		0	18	13	0000000001000s
11	1	9	000001101s		1	9	13	0000000000111s
5	2	9	000001100s		0	19	13	0000000000110s
0	9	9	000001011s		4	4	13	0000000000101s
2	3	9	000001010s		1	10	13	0000000000100s
ESCAPE		9	000001001		5	4	13	0000000000011s
12	1	9	000001000s		0	20	13	0000000000010s
0	10	10	0000001111s		0	21	13	0000000000001s

Table III: Scale 8 run/amplitude variable length code table

Run	Amp	LEN	CODEWORD	Run	Amp	LEN	CODEWORD
EOB		2	11	31	1	10	0000010001s
0	1	3	101s	29	1	11	00000100001s
1	1	3	100s	2	3	11	00000100000s
2	1	4	0111s	19	2	11	00000011111s
10	1	5	01101s	10	5	11	00000011110s
3	1	5	01100s	8	3	11	00000011101s
4	1	5	01011s	13	4	11	00000011100s
9	1	5	01010s	0	7	11	00000011011s
5	1	5	01001s	3	3	11	00000011010s
0	2	5	01000s	1	4	11	00000011001s
8	1	6	001111s	28	1	11	00000011000s
6	1	6	001110s	21	2	11	00000010111s
7	1	6	001101s	14	5	11	00000010110s
11	1	6	001100s	20	3	11	00000010101s
14	1	6	001011s	9	4	11	00000010100s

12	1	6	001010s	0	8	11	00000010011s
20	1	6	001001s	12	3	11	00000010010s
13	1	7	0010001s	13	5	12	000000100011s
10	2	7	0010000s	15	2	12	000000100010s
0	3	7	0001111s	48	1	12	000000100001s
15	1	7	0001110s	37	1	12	000000100000s
35	1	7	0001101s	4	3	12	000000011111s
19	1	7	0001100s	10	6	12	000000011110s
21	1	7	0001011s	14	6	12	000000011101s
1	2	8	00010101s	56	1	12	000000011100s
ESCAPE		8	00010100	5	3	12	000000011011s
16	1	8	00010011s	1	5	12	000000011010s
0	4	8	00010010s	0	9	12	000000011001s
9	2	8	00010001s	11	3	12	000000011000s
10	3	9	000100001s	2	4	12	000000010111s
2	2	9	000100000s	13	6	12	000000010110s
14	2	9	000011111s	3	4	12	000000010101s
22	1	9	000011110s	35	3	12	000000010100s
8	2	9	000011101s	40	1	12	000000010011s
13	2	9	000011100s	16	2	12	000000010010s
3	2	9	000011011s	12	4	12	000000010001s
17	1	9	000011010s	9	5	12	000000010000s
23	1	9	000011001s	14	7	12	000000001111s
18	1	9	000011000s	24	2	12	000000001110s
0	5	9	000010111s	8	4	12	000000001101s
24	1	9	000010110s	6	3	12	000000001100s
20	2	9	000010101s	10	7	13	0000000010111s
25	1	9	000010100s	7	3	13	0000000010110s
34	1	9	000010011s	38	1	13	0000000010101s
1	3	10	0000100101s	0	10	13	0000000010100s
14	3	10	0000100100s	39	1	13	0000000010011s
4	2	10	0000100011s	50	1	13	0000000010010s
36	1	10	0000100010s	17	2	13	0000000010001s
26	1	10	0000100001s	20	4	13	0000000010000s
11	2	10	0000100000s	13	7	13	0000000001111s
10	4	10	0000011111s	0	11	13	0000000001110s
35	2	10	0000011110s	41	1	13	0000000001101s
12	2	10	0000011101s	14	8	13	0000000001100s
5	2	10	0000011100s	46	1	13	0000000001011s
33	1	10	0000011011s	49	1	13	0000000001010s
13	3	10	0000011010s	1	6	13	0000000001001s
9	3	10	0000011001s	12	5	13	0000000001000s
27	1	10	0000011000s	25	2	13	0000000000111s
0	6	10	0000010111s	10	8	13	0000000000110s
32	1	10	0000010110s	4	4	13	0000000000101s
6	2	10	0000010101s	51	1	13	0000000000100s
7	2	10	0000010100s	2	5	13	0000000000011s
14	4	10	0000010011s	13	8	13	0000000000010s
30	1	10	0000010010s	19	3	13	0000000000001s

D.8 VIDEO MULTIPLEX CODER

See Test Model 2, Chapter 9.

D.9 RATE CONTROL AND QUANTIZATION CONTROL

Core Experiments that require methods other than those described in Test Model 1 contain descriptions.

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 bitstream 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.

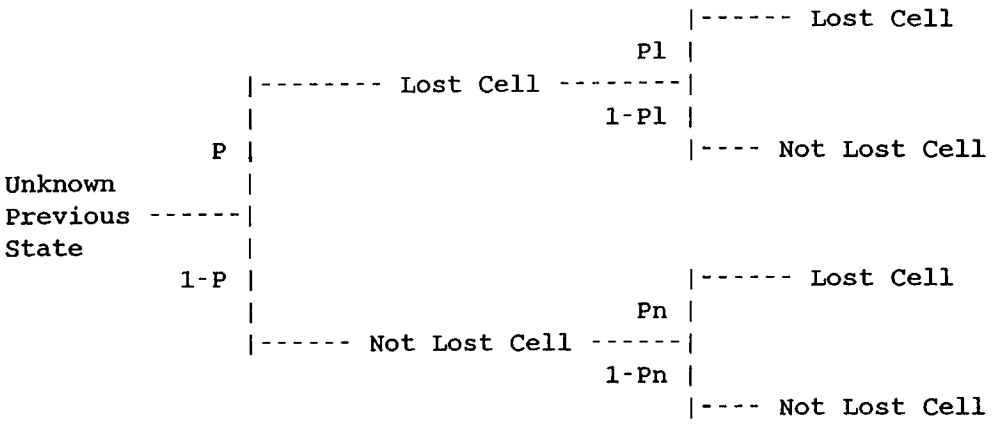
For a lost cell the cell is discarded.

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.



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

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

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:

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

Summing this series leads to the result:

$$B = 1 / (1 - P_l) \quad (2)$$

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.

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

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, cell loss rates such as 1 in 100.

For all experiments the following table should be completed.

		High priority bit rate	Low priority bit rate
1-layer			

2-layer	base		
	enhance		

F.3 Experment to be supplied by Mike Biggar

F.4 Core Experiment on Cell Loss resilience by using frequency scanning

The purpose of this experiment is to compare the impacts of cell losses on both the block scanning (Basic Mode) and the frequency scanning with MUVLC techniques.

The output bitstream of the frequency scanning mode will be split up into 2 layers (with 2 different CLR's). The layer 1 will contain the (n-1) MSB's of the 4X4 LF coefficients; the layer 2 will contain the LSB of the 4X4 LF coefficients and the full amplitude of the remaining 48 coefficients.

No concealment technique will be applied at the decoder side.

F.4.1 Global parameters of the experiment

The core experiment will be carried out by using the following parameters:

chroma_format: 4:2:0
picture_structure: frame picture
bit_rate: 4 Mbit/s
group of pictures structure: N=12 (15), M=3
prediction field/frame adaptive
transform field/frame adaptive
number of frames 50 (60)

The simulation of Cell Loss will be performed according to Appendix F - Section F.1 of TM2.

The parameters for Cell Loss will be :

-Total bit rate : 4 Mbit/s
-High priority bit rate : 1.6 Mbit/s (40% of total).

F.4.2 Description of MUVLC

The MUVLC is described in Appendix Q - Section Q.8.2. and Appendix I - Section I.8.3.

A C-program of the MUVLC algorithm will be distributed to interested parties by e-mail (send requests to B. Hammer (Siemens), ha@bvax4.zfe.siemens.de, or J. De Lameillieure (HHI), grunaaie@w204zrz.zrz.tu-berlin.de).

F.4.3 Syntax modifications

For purpose of the frequency scanning experiments the slice layer, the macro block layer and the block layer must be changed as indicated below:

F.4.3.1 Slice Layer

```
slice() {
  slice_start_code
  quantizer_scale
        ::::
    no change
        ::::
  extra_bit_slice
  slice_control()
  slice_data()
}
```

F.4.3.2 Slice Control Layer

```
slice_control() {
  for (mb=0; mb<44; mb ++) {
    while(nextbits() == '0000 0001 111')
      macroblock_stuffing
        ::::
        same syntax as in macroblock() of TM2
        ::::
    if (macroblock_pattern)
      coded_block_pattern()
  }
}
```

F.4.3.3 Slice Data Layer

```
slice_data() {
   for (coef_i=1;coef_i<65; coef_i ++) {
     buf_size = 0
     for (mb = 0; mb < 44; mb++) {
       for(block_i=1; block_i<5; block_i++) {
          if (pattern_code[mb][block_i]) {
           c_buf[buf_size] = dct_coef[mb][block_i][coef_i]
            buf_size++
          }
       }
     }
     for (mb = 0; mb < 44; mb++) {
       if (pattern_code[mb][5]) {
         c_buf[buf_size] = dct_coef[mb][5][coef_i]
          buf_size++
       }
     }
     for (mb = 0; mb < 44; mb++) {
       if (pattern_code[mb][6]) {
         c_buf[buf_size] = dct_coef[mb][6][coef_i]
          buf_size++
       }
     }
    if (coef_i<17)
      muvlc0(c_buf,buf_size)
    else
```

```
        muvlc1(c_buf,buf_size)
    }
  while(nextbits()!='0000 0000 0000 0000 0000 0001')
    next_start_code()
}


muvlc0(c_buf,buf_size) {
pc_layer1 = pc_code(c_buf,buf_size)                                    3
  for(class=pc_layer1; class=0; class--) {
    pl_layer1 = pl_code(c_buf, buf_size)                               3
      run = 0
      for (buf_i = 0; buf_i < buf_size; buf_i ++) {
        if(!c_coded[buf_i]) {
          if (((c_buf[buf_i] >> class) & 0x1) == 1) {
            rl = rl_code(run,pl_layer1)
             ncb_layer1 = ncb_code_MSB(c_buf[buf_i],pc_layer1)         1-7
             ncb_layer2 = ncb_code_LSB(c_buf[buf_i],pc_layer1)         1
              c_coded[buf_i] = 1
              run = 0
           }
           else
              run++
         }
     }
       eoc = eoc_code(run,pl_layer1)
   }
}

where   ncb_code_LSB is the LSB of the coefficient;
        ncb_code_MSB are the coefficient bits without the LSB.

muvlc1(c_buf,buf_size) {
pc_layer2 = pc_code(c_buf,buf_size)                                    3
  for(class=pc_layer2; class=0; class--) {
    pl_layer2 = pl_code(c_buf, buf_size)                               3
      run = 0
      for (buf_i = 0; buf_i < buf_size; buf_i ++) {
        if(!c_coded[buf_i]) {
          if (((c_buf[buf_i] >> class) & 0x1) == 1) {
            rl = rl_code(run,pl_layer2)
             ncb_layer2 = ncb_code(c_buf[buf_i],pc_layer2)             1-8
              c_coded[buf_i] = 1
              run = 0
           }
           else
              run++
         }
     }
       eoc = eoc_code(run,pl_layer2)
   }
}
```

rl_code(run,pl) {			
while (run >= 1<<pl) {			
max_run		1	"0"
run -= 1<<pl			
}			
prefix	1	"1"	
run		1-8	
}			

eoc_code(run,pl) {			
while (run > 0) {			
max_run		1	"0"
run -= 1<<pl			
}			

Appendix G: COMPATIBILITY AND SPATIAL SCALABILITY

Scalability is achieved by spatial reduction in the pel and temporal domain. Compatibility is a specific implementation of the spatial scalabilty. The following section will describe several core experiments. An ad-hoc group will be formed, which will discuss results and will work on improvements.

G.1 EXPERIMENTS LIST

Expt. No.	Description	Organizations	Doc. No.	Picture structure
1(a)	Spatio-temporal weighted compatible field coding	PTT, Col. U., AT&T	506	Field, Frame
1(b)	Spatio-temporal weighted compatible frame coding	BT, Bellcore, AT&T, Sarnoff	506	Frame
1(c)	H.261 based spatio-temporal weighted compatible frame coding	CNET	506	Frame
2(a)	Spatio-temporal adaptive interlace-interlace conversion	AT&T, Bellcore, Col. U, PTT, BT	509, new prp	Frame
2(b)	Interlace-interlace compatible coding	AT&T, Bellcore, Col. U., PTT, BT	506, 509, new prp	Field, Frame

GUIDELINES

Bitrates

Expt. No.	Number of Layers	Bitrate Layer 1 Mbit/s	Bitrate Layer 2 Mbit/s
1(a)	2	1.15	2.85
1(b)	2	1.15	2.85
1(c)	2	1.15	2.85
2(a)	2	-	-
2(b)	2	1.50	2.50

CODING PARAMETERS AND STANDARD

M, N

Expt No.	M,N for 30 Hz	M,N for 25 Hz
1(a), 1(b),2(b)	M=3, N=15	M=3, N=12
1(c)	M=1, N=∞	M=1, N=∞

MC Prediction Modes

Frame Picture	Field Picture
Frame MC	Field MC
Field MC	Field submacro MC

G.2 EXPERIMENT DETAILS

G.2.1 Spatial-temporal weighting

Expt 1(a) Spatio-temporal weighted compatible field coding

This experiment compares two techniques, one that uses full "spatio-temporal weighted compatible field coding" with that which uses only "spatial compatible field coding". The lower layer consists of SIF odd fields which are coded using MPEG-1. The upper layer is either CCIR-601 fields or frames depending on whether field- picture or frame- picture is employed. The upper layer is coded with MPEG-2 coding and uses optional compatible prediction from the lower layer. The explanation of spatio-temporal weighted compatible field coding, as well as the syntax is provided in Doc. MPEG 92/506.

The SIF layer decoded picture is upsampled for compatible prediction of odd- fields. The even- fields are initially obtained by line shifting of odd- fields (vertical averaging). In the "spatio-temporal weighted compatible field coding technique", the even fields just computed are weighted averaged with a temporal mc prediction. The prediction signal is derived from the CCIR 601 decoded (frame or field) by using the motion vectors included for macroblocks that may use spatio-temporal weighted mode. A 2- bit weight code that represents spatial weight used in spatio-temporal prediction is specified on a macroblock basis. This weight code corresponds to the least MSE and is selected after having compared MSE produced by using all four weight codes to generate prediction.

It is important to recognize that for odd fields in CCIR 601 layer, full spatial prediction is available, i.e., in frame- pictures, for example, an 8x8 block from SIF- layer is upsampled horizontally to obtain a 16x8 block and used for compatible prediction without any modification. On the other hand, for even- fields in CCIR-601 layer, a line shifted version of odd field obtained in the previous step froms thespatial prediction and is weighted by "w" and summed with temporal prediction block of 16x8 macroblock by mc prediction and weighted by "1-w".

Any syntax changes , macroblock ype tables or change of weights from those specified in Doc. MPEG 92/506 will be communicated with in the ad hoc group.

In Fig. 1(a), spatio-temporal weighted prediction is explained further. As shown, spatio-temporal weighting is applied to 16x8 even field submacroblock for frame- pictures. In case of field pictures weighting is applied to 16x16 even field submacroblock.
All coding parameters and mc modes to be used are shown in tables.

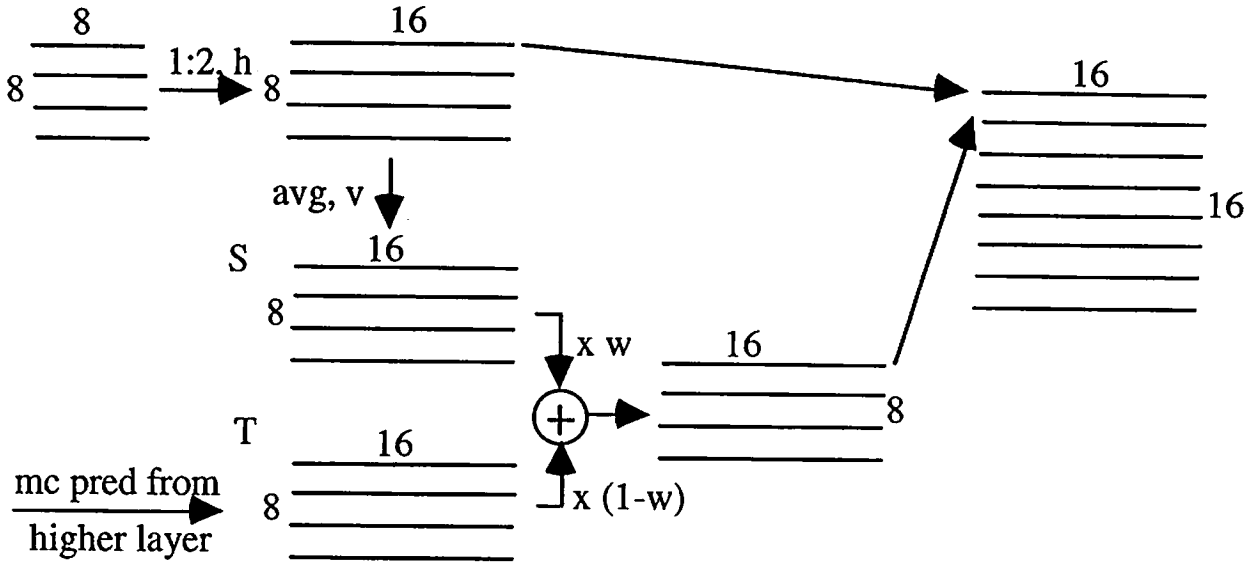


Fig. 1(a)

weight table:

w
1
1/2
1/4
0

Expt 1(b) Spatio-temporal weighted compatible frame coding

This experiment compares two techniques , one that uses full "spatio-temporal weighted compatible frame coding" with a reference technique, "spatial compatible frame coding". The lower layer is SIF odd fields which is CCIR-601 frames coded frame- picture coding using MPEG-2 is applied for this layer along with optional compatible prediction from the lower layer. Some explanation of spatio-temporal weighted compatible frame coding, as well as syntax is provided in Doc. MPEG 92/506. Some changes from the description in that document will be described here.

The SIF layer decoded picture is upsampled to provide compatible prediction for odd field of frame-picture. The even- fields are initially obtained by line shifting of odd- fields (vertical averaging). In the full "spatio-temporal weighted compatible frame coding" technique, a frame- spatial prediction macroblock obtained by merging 16x8 blocks of upsampled odd and even fields is weighted averaged with a 16x16 macroblock obtained by temporal mc prediction. The prediction signal is derived from the CCIR-601 decoded frames by using the motion vectors included for macroblocks that may use spatio-temporal weighted mode. A table for 2- bit weight code is provided in Fig. 1(b). This code actually a pair of spatial weights (w1,w2) where w1 is applied to lines of field1 and w2 to lines of field2. The temporal weights are easily computed as (1-w1, 1-w2) and are applied to lines of field1 and field2. Fig. 1(b) also

shows the all necessary operations needed to derive a 16x16 macroblock for CCIR-601 layer by weighted spatio-temporal combination. The values of weights provided in the table are a preliminary guess, experimenters using different weights to determine the best set of weights must document statistics of both, this set and alternate set. For information purposes alternate better weights should also be communicated with in the ad hoc group.

It is important to recognize the difference in this experiment as compared to experiment 1(a). Here, spatio-temporal weighting is performed on 16x16 macroblocks, where as in experiment 1(a), for frame-pictures, spatio-temporal weighting is performed on 16x8 even field macroblock. This has some impact, not only related to the block size, but also to which predictions get filtered.

All coding parameters and motion compensation modes to be used are provided in the tables.

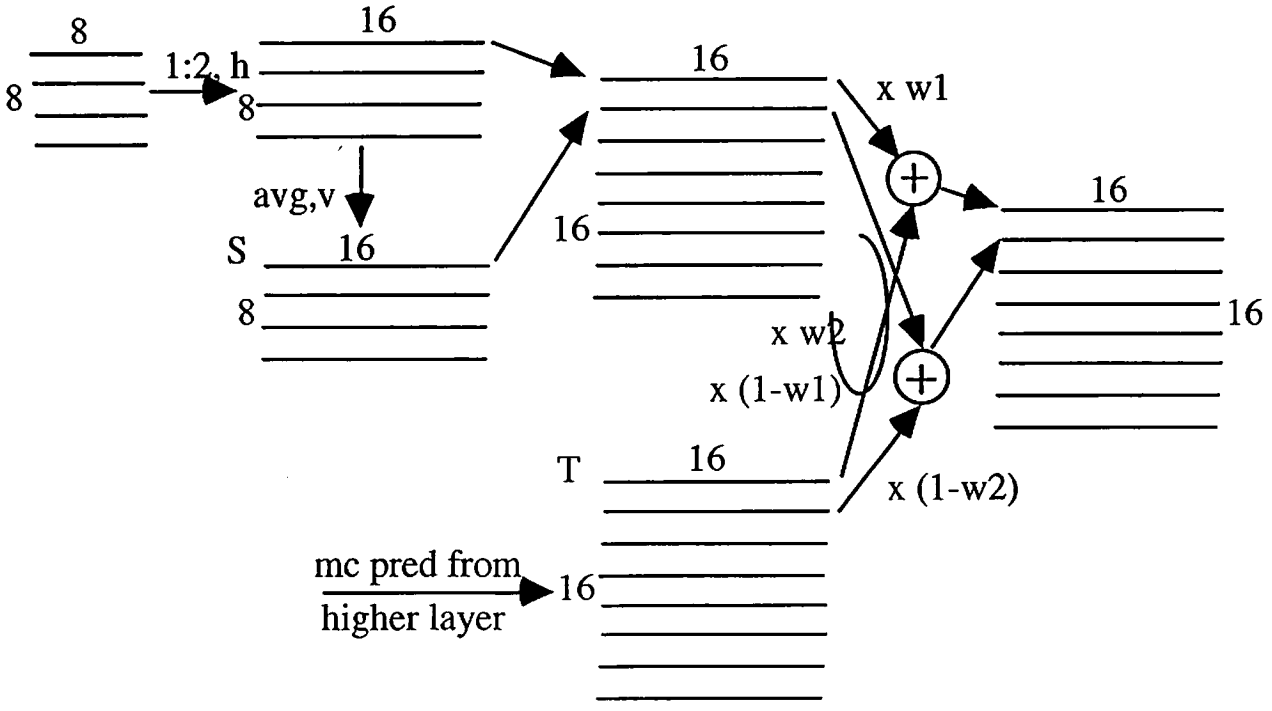


Fig. 1(b)

weights:

(w1,w2)
(1,0)
(3/4,1/4)
(3/4,1/2)
(1/2,1/2)

Expt 1(c) frame coding H.261 based Spatio-temporal weighted compatible

This experiment is similar to Expt 1(b)., except that specifications for H.261 are used for coding the SIF odd field (lower) layer. Two techniques are compared, one in which the higher layer uses full, "spatio-temporal compatible coding", with the reference technique of "spatial compatible frame coding".

Expt 2(a) Spatio-temporal adaptive interlace-interlace conversion

This experiment compares two techniques, one that uses "spatio-temporal adaptive interlace-interlace conversion " and the other, a reference "spatial interlace-interlace conversion of TM2". The higher layer is CCIR-601 frame pictures; a lower layer SIF_I is derived from it. Both down sampling of CCIR-601 to SIF_I as well as upsampling of SIF_I to CCIR-601 is investigated.

An example of "spatio-temporal adaptive interlace-interlace conversion" is specified in Doc. MPEG 92/509. This technique does not use motion compensation for deinterlacing. Any suggested modifications to these specifications should be fully documented and documented and communicated with in the ad hoc group. alternately, other methods of "spatio-temporal adaptive interlace-interlace" conversion are also allowed in this experiment as long as a comparison with "spatial interlace-interlace conversion of TM2" is made and new technique is fully documented and communicated with in the ad hoc group as soon as possible.

It is realized that the downsampling process is not a matter of standardization, however, both downsampling and upsampling documentation is needed as a certain upsampling method which is standardized may work well with specific downsampling method.

In comparing results of this experiment, a pair consisting of down sampled images shown at SIF_I resolution, one corresponding to "spatio-temporal adaptive interlace-interlace" technique and the other corresponding to the reference, "spatial interlace-interlace of TM2" is mandatory. In addition to down sampled images, down and upsampled images are also necessary, again, a comparison as in downsampling case is made, this time at upsampled CCIR-601 resolution. It is recommended that 3 test scenes "Flowergarden", "Mobil Calendar" and "Football" be used for experiment. It is also suggested that SNR value of SIF_I and upsampled CCIR-601 image with respect to original CCIR-601 image with respect to original CCIR-601 should be reported for comparison purposes.

Expt 2(b) Interlace-interlace compatible coding

In this experiment, compatible coding is performed using 2 layers, SIF_I and CCIR-601. Both SIF_I and CCIR-601 are inherently frame- pictures. SIF_I is derived from CCIR-601 using"spatio-temporal adaptive interlace-interlace conversion" (Doc. MPEG 92/509 or a new proposal). SIF_I is coded using MPEG-2 and upsampled both horizontally and vertically as compatible predictions for the higher layer. For each macroblock in the higher layer, a 16x16 temporal prediction is obtained from upsampling an 8x8 block in the lower layer. Spatio-temporal weighted averaging is performed on these two 16x16 prediction to produce the final prediction macroblock. A 2- bit weight code is selected from the table shown in Fig. 1(b) based on the minimum MSE. Therefore, the field1 spatial prediction is weighted by w1, and the field2 spatial prediction is weighted by w2, and the temporal predictions for field1 and 2 are weighted by (1-w1) and (1-w2) respectively.

G.2.2 Coding

The use of coded block pattern is allowed for all compatible coded macroblocks.

When compatible prediction is selected for an Intra macroblock, the DC value is not differentially coded and all DC predictors are reset.

Non-intra Weighting matrix is employed for quantizing all compatible predicted macroblocks.

The macroblock types for compatible macroblocks are as follows:

NOTE: For compatible macroblocks, in the block layer it is no longer necessary to transmit the DC coefficient with the differential DC code if a compatible prediction is made for this block.

Motion vector predictors (PMV)
(see 8.3)

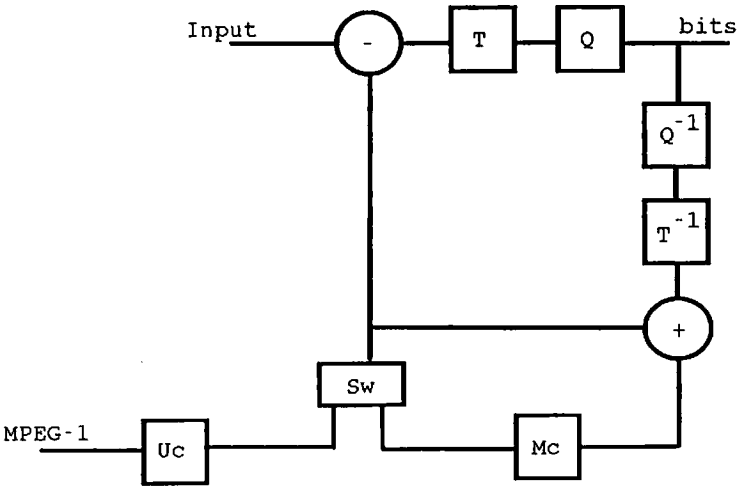
Macroblock Addressing and Skipped Macroblocks
In Intra pictures there are no skipped macroblocks. For compatibile macroblock_type a skipped macroblock indicates that the macroblock is using base layer prediction only, no coeficients are coded.

In predicted pictures 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. For compatible macroblock_type a skipped macroblock is not used.

In interpolated pictures, a macroblock is skipped if it has the same MTYPE or compatible_macroblock_type as the prior macroblock, its motion vectors are the same as the corresponding motion vectors in the prior macroblock, the compatible weight_codes are the same, all its quantized DCT coefficients are zero, and it is not the first or last macroblock in the slice.

Compatible prediction method

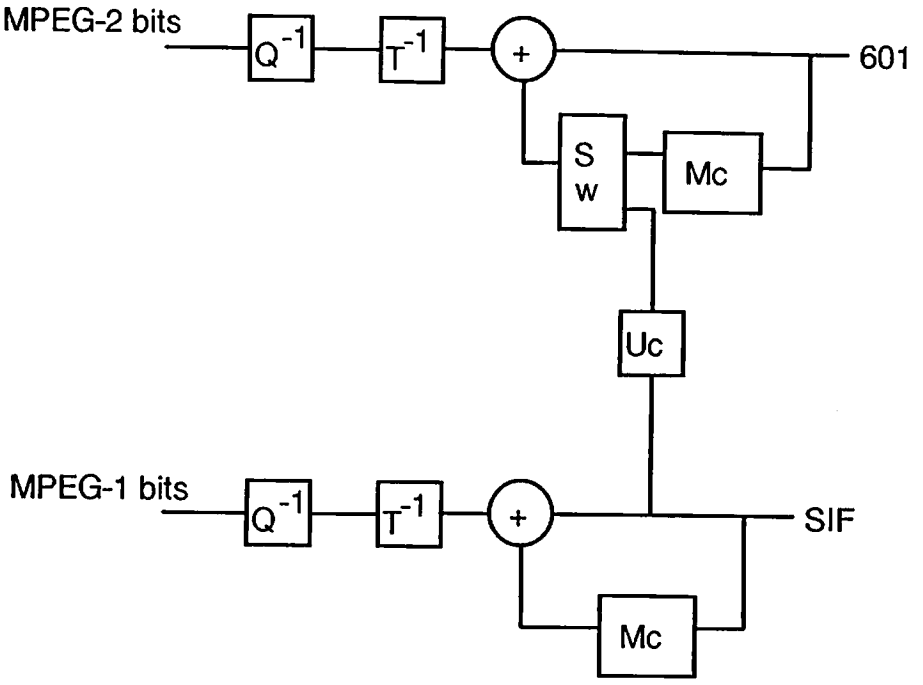
On a macroblock basis a prediction from the locally decoded picture from the base layer may be chosen. This prediction is generated from a weighted average of the corresponding upsampled coded 8*8 block of the lower layer(to a 16*8 block) and the normal prediction of the upper layer(See figure G.1 and G.2). The 8*8 block is upsampled to a 16*8 block by a [1/2, 1, 1/2] filter (a similar operation is perform on the chrominance 4*4 subblock to give a 8*4 prediction block).



T Transform
Q Quantizer
MC Motion compensation

Sw Weighted Switch Uc Up conversion

Figure G.1: Compatible encoder structure



T Transform Q Quantizer MC Motion compensation
Sw Weighted Switch Uc Up conversion

Figure G.2: Compatible decoder structure

Selection method

Selection is MSE based. The mode which gives the minimum is chosen.

Quantization

Quantization is in line with chapter 7. There is one exception, which is intra quantization in only used on intra macroblock type and no compatible prediction (compatible_type = 0).

Coefficient coding

In all picture types, macroblocks in which a compatible prediction has been made (i.e. when compatible_type is not 0) the quantized coefficients are sequentially transmitted according to the zig-zag sequence given in figure 4.5.

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

Appendix H: LOW DELAY CODING

H.1 Simulation guidelines for low delay profile.

H.1.1 Coding structure.

In the low delay profile it is assumed that the total coding/decoding delay shall be kept below 150 ms. However, it is realized that coding/decoding delay considerably below that limit could be desirable for many applications (e.g. two way communication over satellite links).

In the following the delay will be measured in "field periods" - fp. For 50 and 60 Hz pictures the 150 ms correspond to:

- 50 Hz -> 150 ms = 7.5 fp.
- 60 Hz -> 150 ms = 9.0 fp.

The coding/decoding delay is considered to consist mainly of two parts:

- Buffer delay. For low delay mode, forced intra slice/column is recommended for updating. With these updating method, it is assumed that the buffer delay may be 2fp.
- Delay resulting from frame/field reordering. This will be called basic delay.

The sum of these two contribution should be below 150 ms - or as low as possible. The possible coding modes fulfilling this criteria are listed in the table below.

Predictor performance:
Coding performance is important also for low delay coding. Performance depends to a large extent on the quality of the predictor. The preferred predictors for the different modes are:

- Field coding and M>1: The predictors in TM1 performs well.
- Frame coding and M=1: FAMC - or similar - are the preferred modes.
- Field coding and M=1: A prediction mode similar to FAMC have shown very promising performance and would be preferable. The syntax is the same as for field coding with one vector pr. MB as defined in TM1.

The table below list:

- Possible coding modes that fulfills the delay criteria.
- Preferred prediction modes (for M=1).
- Different delays.
- Performance ranking estimated by computer simulations.

Coding mode	M	Buffer delay	Basic delay	Total delay	Performance ranking (low="good")
Frame FAMC - or similar	1	2 fp	2 fp	4 fp	2
Field FAMC - or similar	1	2 fp	0 fp	2 fp	3
Field FAMC -or similar	2	2 fp	4 fp	6 fp	1

H.1.2 Handling of scene change to maintain low delay.

A major contribution to the buffer delay is the complete INTRA frames/fields coding. To get around this delay it is necessary for the low delay profile to have the possibility of picture skipping. The encoder may decide to skip frames after a "large" picture due to scene cut.

For the prediction of pictures followed skipped one, the re-constructed image of the skipped picture is forced to set to the re-constructed image of the "large" picture.

VBV consideration.

VBV specification for skipped picture can be found in Annex. C.

Decoding Operation
The following decoder is assumed

1) The decoder and the encoder have the same video clock frequency as well as the same picture rate, and are operated synchronously [current VBV specification]. Some means are provided externally to achieve this synchronism, e.g. by using sampling clock information contained in the picture header, AAL, time stamp.

2) It has a receiving buffer of size B, which is given in the vbv_buffer_size field in the sequence header [current VBV specification].

3) It receives coded data at a constant rate and write in the buffer [current VBV specification].

4) The buffer is initially empty [current VBV specification].

5) Decoding starts after filling the buffer for the time specified by the vbv_delay field in the video stream [current VBV specification].

6) At decoding timing with the same interval as that of encoder, all of the data for the non-dropped picture which has been in the buffer longest is instantaneously removed, instantaneously decoded and starts to be displayed [current VBV specification]..

7) If there is no complete data for a coded picture, there takes place no decoding operation and the most recent decoded field is displayed repeatedly [TM2 specification].

Coding control

Model for encoding

Two coding control examples, M=1 frame coding and M=2 field coding, are given to show that there exist methods to meet the above mentioned decoder requirements.

This encoder operates as follows;

1) It has a transmission buffer of size B. This means that the maximum number of bits per picture should be B.

2) The buffer is empty when encoding starts.

3) Just after a complete picture is input to the encoder, it instantaneously encodes the picture and instantaneously sends coded data to the buffer.

4) The buffer content is read out at a constant rate.

5) A buffer occupancy value Bl (less than B) is set corresponding to the required buffering delay time (Bl/R where R is transmission rate). For stationary signals, the coding operation is controlled so that the buffer occupancy stays between zero and B_l at any time.

6) If a scene change or forced updating takes place, the buffer occupancy exceeds Bl, and necessary number of pictures are dropped.

The encoder controls the bit generation for a picture by referring to the buffer occupancy just before the encoding, and the estimated buffer occupancy just before encoding the next picture. If there is no picture dropping, the low delay mode encoder controls so that no overflow nor underflow of the buffer with size Bl should take place.

Example for M=1 frame coding

Figure 1 shows an example for M=1 frame coding with Bl corresponding to 1.5 frame time delay. Until frame F, low delay operation is maintained and G is the first frame of a new scene. Three frames H,I,J are dropped and low delay operation is resumed from frame K. In order to do so, the number of bits for frame G should be regulated to fall in the zone between 0 and Bl-R/P (R: transmission rate, P: picture rate), filling dummy bits if necessary, just before starting to encode the next frame. Figure 2 shows such regulation.

Example for M=2 field coding

Figure 3 shows an example for M=2 field coding with Bl corresponding to 2 field time delay. In this case, the display order is different from the encoding order. Hence there may be two strategies; to drop fields in the encoding order or in the display order. Figure 3 illustrates the two methods, which will give different subjective impressions.

When fields are dropped in the encoding order, such as p6,b3,b4,p9, repetition of the most previously decoded field happens at maximum three places. Encoding of b7 and b8 should use only p5 and p10, or p5 only in the worst case where p10 is also dropped, thus the reproduced pictures will degrade. When fields are dropped in the display order, such as b3,b4,p6,b7,b8, repetition of the previously coded field happens at maximum two places. In this case, encoding of p9,p10 is carried out by anticipating that b7,b8 are to be dropped.

In either case, low delay operation is resumed from field p13.

Since subjective impression is involved, the two methods should be experimented. If there is no significant difference in impression, field dropping in the encoding order is more straightforward.

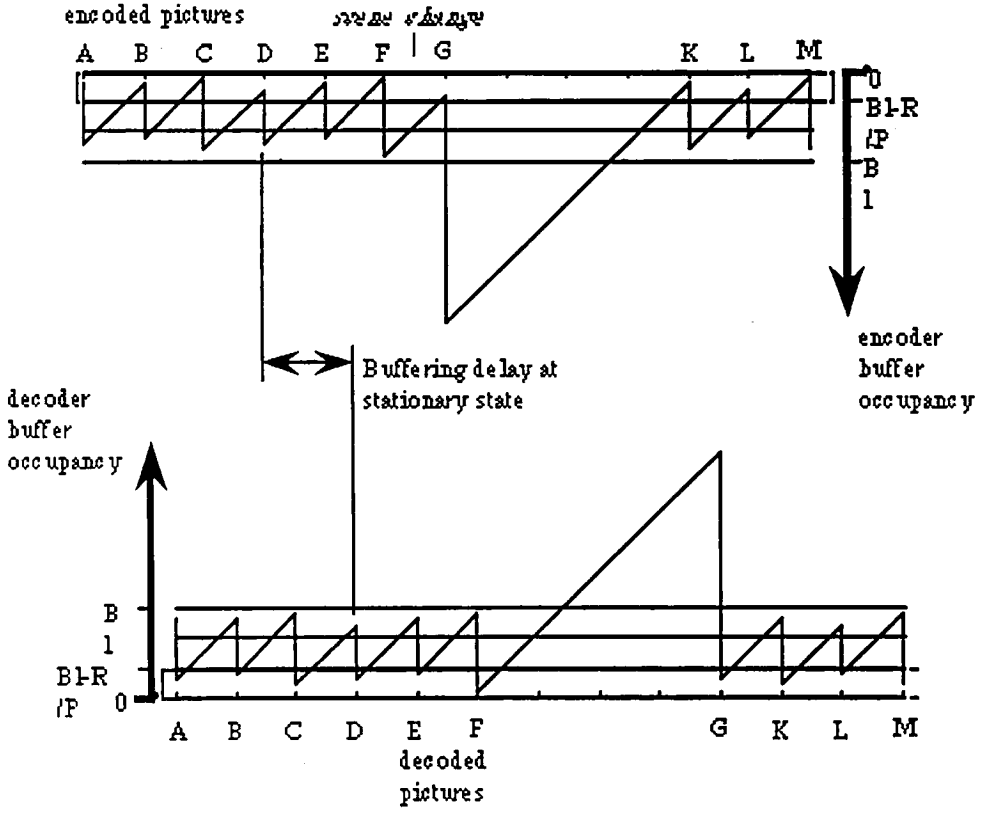
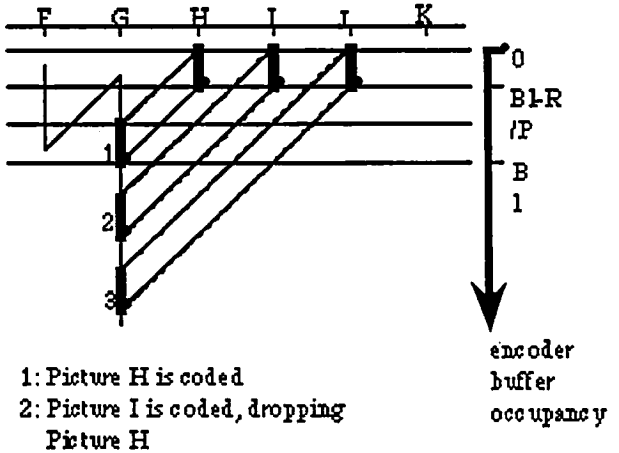


Figure 1 Decoder and encoder buffers for M=1 frame coding



1: Picture H is coded
2: Picture I is coded, dropping Picture H
3: Picture J is coded, dropping Pictures H and I

Figure 2 Information generation of Picture G

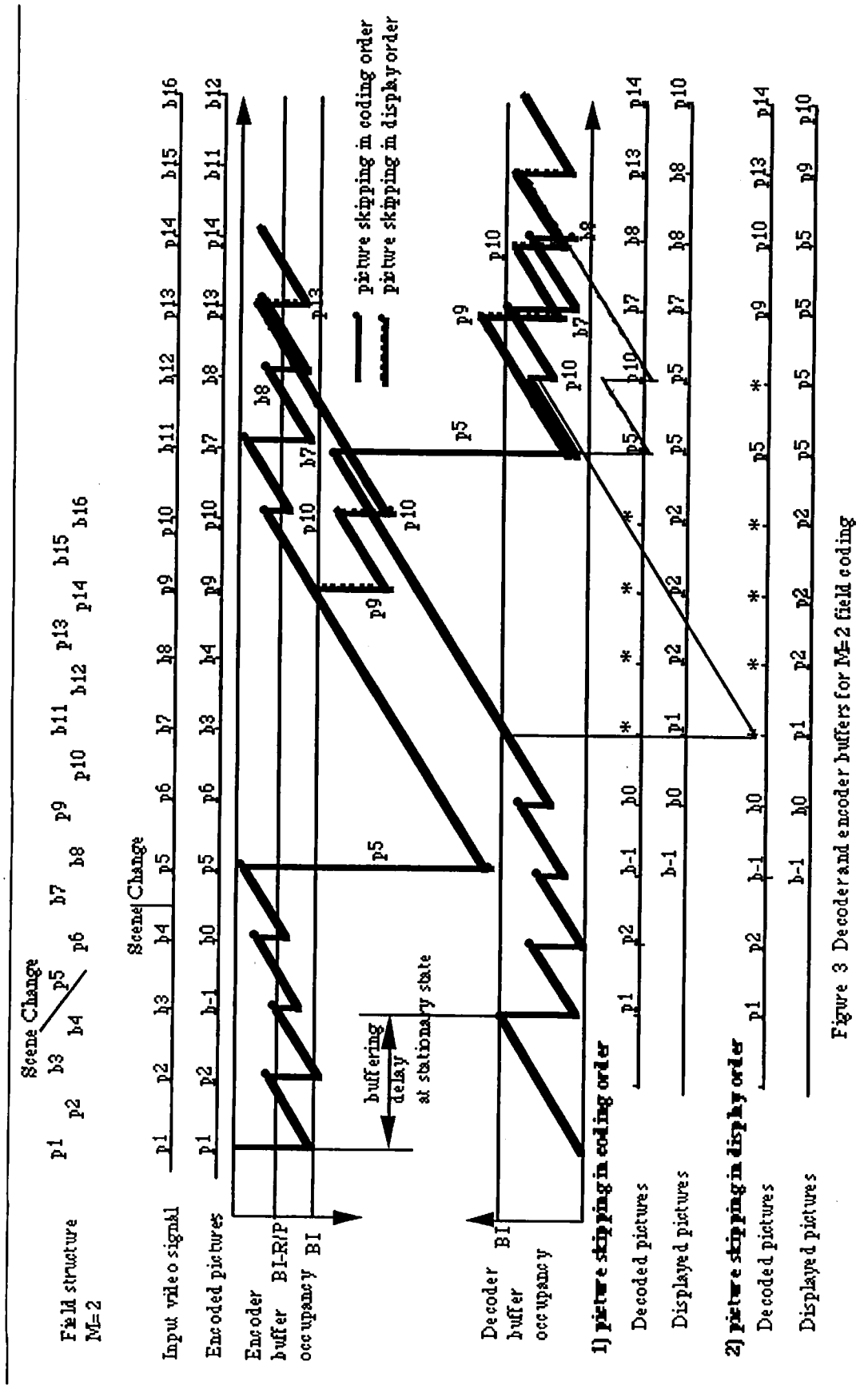


Figure 3 Decoder and encoder buffers for M=2 field coding

Hander for skipped picture
In the skipped picture, only their picture headers is sent with indication of skipped picture in **dropped_picture**, which is one bit integer defined in picture header.

Numbering of Temporal Reference

Temporal Reference is the straightforward source picture TR method as described in bit stream syntax.

Source picture	A B C D E F G H I J K L M N
Coded picture	A B C D E F G * * * K L M N
Source picture TR	1 2 3 4 5 6 7 8 9 10 11 12 13 14

H.1.3 How to handle the first picture using forced intra slice

In the low delay profile it is of interest to study the "steady state" performance of buffer delay. It is therefore desirable to pretend that we are in the steady state situation from the beginning of the sequence. The proposed way to obtain this is the following:

- Code the first picture INTRA with QP=16 (for 4 Mb/s).
- After the first picture the buffer is set to a value corresponding to 1/60th of a second (66667 bits for 4 Mb/s).
- The number of bits for the first picture - used for buffer regulation - is set to the average number of bits for the sequence, and this picture is treated as P-picture in term of rate control.
- In evaluating the delay only buffer filling after the first picture is considered.

Note) In case of using intra picture, the first picture of the test sequence is coded as NORMAL Intra Picture, and its rate control follows NORMAL rate control described in TM.

H.1.4 Definition of intra slice/column coding.

To reduce buffer delay the updating is made with forced INTRA slices rather than forced INTRA pictures. The procedure for doing the forced INTRA slice coding is shown in the figure below. All the modes given in the table above are covered. The updating for the different modes are arranged so that the time for total update is the same for all modes.

Instead of intra slice updating, intra column updating may be used. In this case one column of MBs is treated as an "update slice". The updating procedure for columns can be the same as for slices indicated in the figure below.

o Frame mode, M=1: Two slices pr. frame.
o Field mode, M=1: One slice every other field.
o Field mode, M=2: One slice every field for P-fields.

To guarantee that errors do not propagate there is restriction on predictions in region 1. Motion vectors in region 1 may not refer to areas in region 2 (refer to the figure).

With this method the time for total refresh is 500 ms for 60 Hz sequences and 720 ms for 50 Hz sequences. It should be noted that the maximum entry times for channel hopping are twice as large as the mentioned values.

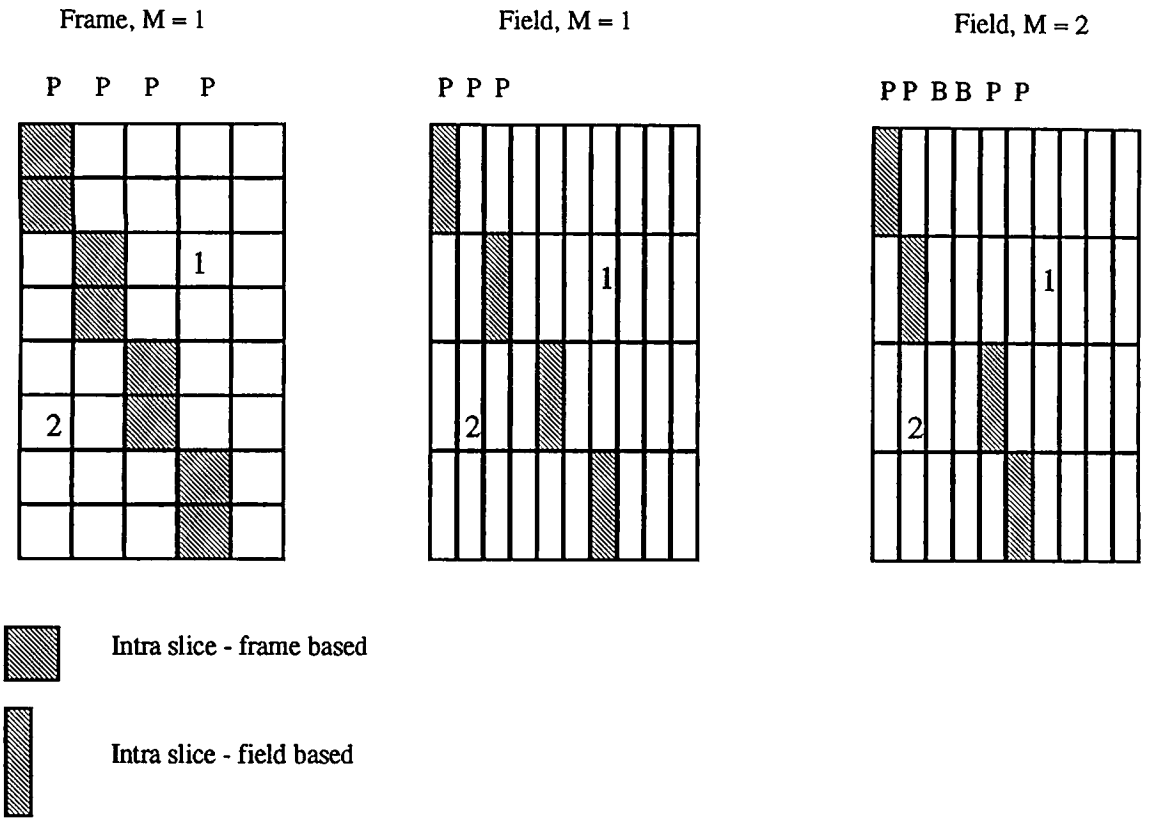


Figure H.1 Low Delay Coding stategy

H.1.5 Rate control.

INTRA SLICES

For intra slice updating, a modified buffer regulation is introduced.

TM1 rate control assumes that the generated bit amount Bj increases constantly. Therefore buffer occupancy d_j^p is calculated as follows.

$$d_j^p = d_0^p + B_{j-1} - \tilde{B}_{j-1} \qquad (1)$$

$$\tilde{B}_j = T_p * \frac{j}{MB\_cnt} \qquad (2)$$

Here $\tilde{B}_j$ is a prediction value of Bj. It increases linearly as shown in figure 2. When using intra slices, $\tilde{B}_j$ is modified as in figure 3.

Target bit amount is also modified to $T_{p_r} + T_{p_i}$. The target bit amounts are calculated as :

$$T_{p_p} = \frac{Number\_of\_P\_slices * Tp}{Number\_of\_P\_slices * Tp + Number\_of\_I\_slices * Ti} * T_p \quad (3)$$

$$T_{p_i} = \frac{Number\_of\_I\_slices * Ti}{Number\_of\_P\_slices * Tp + Number\_of\_I\_slices * Ti} * T_I \quad (4)$$

The rate control may then be operated as in equation (1).

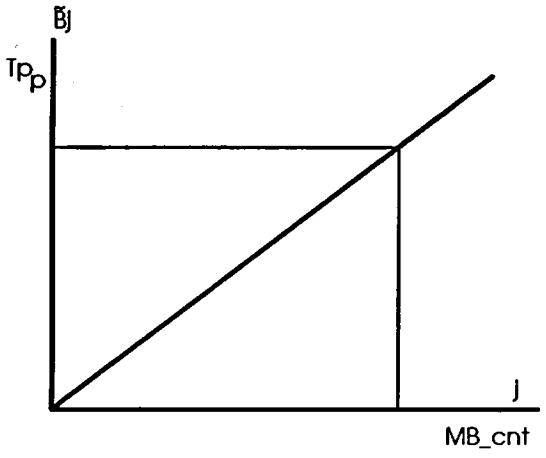


Figure H.2

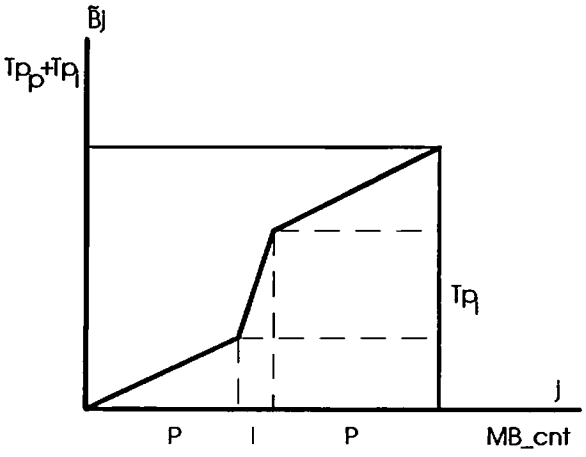


Figure H.3

SKIPPED PICTURES

For picture skipping the following buffer regulation is used:

- For the first picture after a scene change QP is fixed to the same value as for the first picture.
- Rate control is activated again from the second picture after the scene change.
- The number of bits generated in the skipped picture is forced to set to zero.
- The Step 1 of rate control in TM is active even in the skipped pictures.
- The Step2 and Step3 of rate control are active only in the coded pictures.

H.1.6 Influence of leaky prediction on low delay coding.

If leaky prediction is needed for other purposes, it could also be useful for the low delay profile. Forced INTRA slices could be replaced by leaky prediction. Possible advantages could be reduced buffer delay and no visible INTRA updating.

H.2 Experiments

The three coding structures were picked up for low delay at Angra meeting.

- Frame structure M=1
- Field structure M=1
- Field structure M=2

The core experiments for low delay should be carried out on the above structures. The rate control with forced intra picture and with picture skipping can be found in Appendix H "Guide line of simulation with low delay profile". In other cases, rate control is as in TM1.

Experiment 1 Comparison of coding efficiency among predictions at low delay.
(Revised to Experiment 4)

Experiment 2 Comparison of coding efficiency and delay between using intra picture and forced intra slice

Forced intra slice can reduce the buffering delay because of smoothing the number of bits per frame. To check the impact of intra slice , the coding efficiency and delay will be compared between using intra picture and forced intra slice as the same table of experiment 1.

Experiment 3 Comparison of coding efficiency and delay between with and without picture skipping

At scene change, picture skipping is useful technique to keep the delay low. The coding efficiency and delay with/without picture skipping at scene change will be compared on the same table of experiment 1. To get scene change, two sequences are combined (e.g. Flower Garden / Mobile Calendar).

Experiment 4 Comparison of coding efficiency among predictions at low delay.
(Core experiment L10 in prediction ad-hoc group)

Every coding structure has several prediction modes. In order to seek the suitable prediction for low delay, the performance of predictions will be compared within the following table.

prediction structure	Fr/Fi/Dual'	Fr/Fi/SVMC3	Fr/Fi/SVMC3-1/2-pel	Fr/Fi/dual'/SVMC3	Fr/Fi/dual'/(SVMC3-dual)
Fr M=1					
Fi M=1					
Fi M=2					

Experiments should be performed so that proper comparisons cab be made concerning;

- Prediction
- Coding structure.

Experiment No.5 (picture skipped order in field structure with M=2)

Picture skipped order in case of field structure M=2

Now we have two way of skipped order for field structure M=2 as

- Encoding order
- Display order.

These two ways should be compared in term of subjective impression. The detail can be found in the revised "Simulation guide line for low delay profile."

The core experiments for low delay should be carried out on the above structures. The rate control with forced intra picture and with picture skipping can be found in Appendix H "Guide line of simulation with low delay profile". In other cases rate control in as in TM1.

H.2.1 Comparison of coding efficiency among predictions at low delay.

Every coding structure has several prediction modes. In order to seek the suitable prediction for low delay, the performance of predictions will be compared within the following table.

prediction structure	Fr/Fi	Fr/Fi/FAMC	SVMC	Fr/Fi/Dual'/Dual
Fr M=1				
Fi M=1				
Fi M=2				

Experiments should be performed so that proper comparisons cab be made concerning;

- Prediction
- Coding structure.

H.2.2 Comparison of coding efficiency and delay between using intra picture and forced intra slice

Forced intra slice can reduce the buffering delay because of smoothing the number of bits per picture. To check the impact of intra slice , the coding efficiency and delay will be compared between using intra picture and forced intra slice as the same table of experiment 1.

H.2.3 Comparison of coding efficiency and delay between with and without picture skipping

At scene change, picture skipping is useful technique to keep the delay low. The coding efficiency and delay with/without picture skipping at scene change will be compared on the same table of experiment 1. To get scene change, two sequences are combined (e.g. Flower Garden / Mobile Calendar).

APPENDIX I : FREQUENCY DOMAIN SCALABILITY CORE EXPERIMENTS

Core Experiment I.1: Interlace-in-Interlace Extraction

I.1.1: Background

It has become apparent that the case of a two-layer scalable video hierarchy with both layers interlaced is very important. It is further noted that the quality of each layer is critical in some applications. It is therefore important to investigate encoder solutions that address this issue.

To produce a lower resolution layer in frequency scalable video, a subset of the DCT coefficients of the 8x8 blocks are extracted. In the case of non-interlaced video, the upper left 4x4 coefficients are chosen. When this technique is applied to interlaced source material, the temporal resolution is reduced by a factor of two. This is unacceptable for high quality applications.

A first solution to this problem was to adaptively use frame and field coding and motion compensation modes. However, field-based extraction of the lower layer produces samples on non-equally spaced lines, none of which are spatially co-located with those produced by frame-based extraction as shown in Figure I.1.

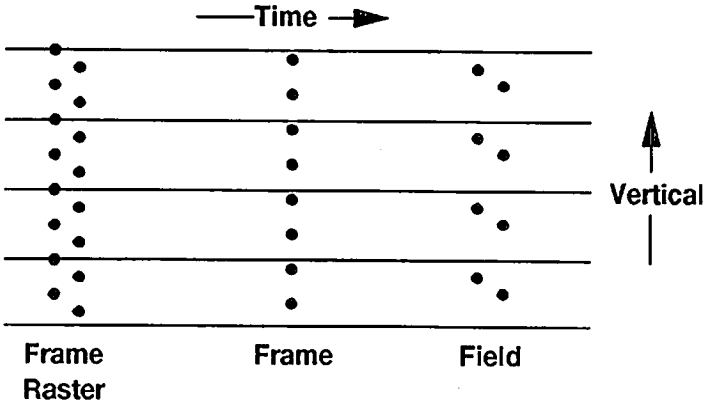


Figure I.1:Raster mis-match between frame and field based 4x4 extraction.

This effect results in very poor motion-compensated prediction when the current macroblock and the prediction macroblock in the reference picture are coded in different modes. Moreover even at high bit rates, a high quality lower layer signal is not available.

In an attempt to avoid the complexity of correcting the raster position of the field-based lower layer pixels, some alternative solutions, involving the use of non-lowpass subsets of the DCT coefficients in scale-4, have been proposed and will be evaluated during this core experiment.

It is well-understood that the interlaced-in-interlaced mode of operation could well be used with complex encoder structures designed to satisfy quality requirements of specific applications. However, in order to isolate the effect of the options under test from other aspects of frequency domain scalable systems, we propose to conduct this core experiment using a simple encoder structure which produces a single layer bit-stream.

Core experiment I.1 addresses the low layer encoder, which produces SIF resolution pictures. This decoder, given in Figure I.2, is identical in structure to the upper layer decoder with the exception of the quarter-pixel resolution for the motion-compensation as described in Appendix D. The decoder will

decode the entire bit-stream, extracting one of the subsets of the scale-8 DCT coefficients given in Figure I.3. Pictures produced with each of these subsets will be compared to determine which subset gives the subjectively best tradeoff of motion rendition, vertical resolution, and aliasing.

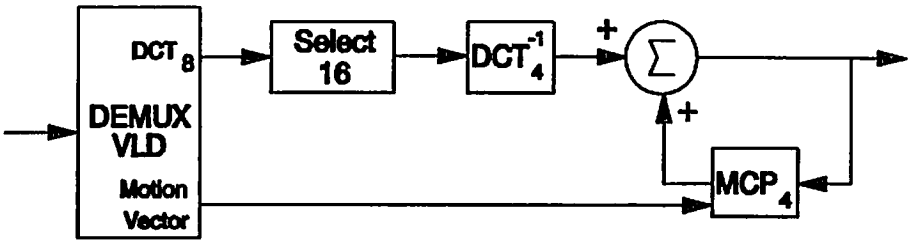


Figure I.2: Scale-4 decoder for core experiment I.1

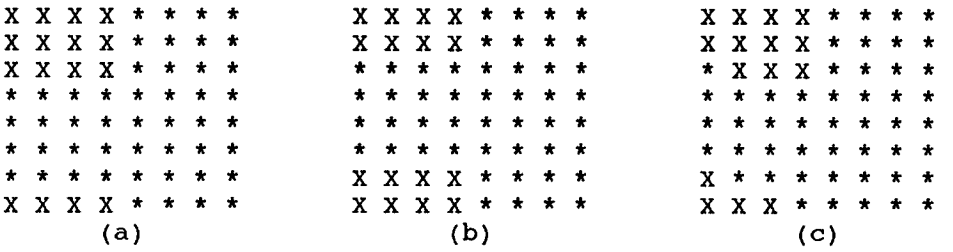


Figure I.3. Three possible DCT coefficient subsets.

Note: This core experiment uses a single-layer bit-stream. Bit-stream scalability requires additional base/slave layer multiplexing, which in turn requires the scanning order to be defined appropriately. The best scanning path depends on the selected set of DCT coefficients, and is still under investigation.

I.1.2: Description

The encoder and upper layer decoder are identical to their Test Model 1 non-scalable equivalents under the following conditions:

- Chrominance sub-sampling 4:2:0
- Bit rate 4.0 Mb/s
- N=15 (12), M=3
- Frame coding and motion compensation modes only
- Other aspects as in TM1 and/or as in TM2 Appendix D

The lower layer decoder has the following features which differ from the scalable Test Model 1 scale-4 decoder:

- scale-4 DCT coefficients extracted from single layer bit-stream
- 1/4 pixel motion vector resolution in scale 4

Test sequences: Bicycle, Cheer Leaders

I.1.3: Syntax Changes

None required for this core experiment.

Core Experiment I.2: Pyramid Encoder Improvements

This core experiment addresses the issue of retaining the high quality high resolution layer of frequency domain scalable encoders, while improving the quality of the lower layer(s). The fundamental nature of the experiment should make the results useful across a range of application profiles.

I.2.1: Background

A single-loop two-layer encoder using frame coding and motion-compensation modes only can be made efficient at high resolution, but suffers from drift problems at lower resolutions. A significant improvement in lower resolution quality is realized when the full precision of the motion vector information as it exists in the bit-stream is used to perform the lower resolution motion compensation as described in Appendix D. However, it is felt that more improvements can be made to the basic framework of a multi-layer encoder to further improve the quality.

One investigation along this line has been the use of motion-compensation loops in more than the highest resolution layer. When the multiple loops are uncoupled except for the upward coefficient prediction, as shown in MPEG 91/212 Figures 5a and 6, the drift problem is solved, but at the expense of reducing the upward prediction efficiency. The result is a tilt in the balance of quality away from the high resolution layer. Overall, the coding noise is increased. In an attempt to control this quality balance more carefully, coupled motion compensation prediction loops have been studied, in which the coupling takes the follow form. At each layer, the prediction error signal is formed by a weighted average of the appropriate DCT coefficients from more than one scale, as in MPEG 92/288 Figure 1. These weights allow one to realize encoders which vary smoothly from the uncoupled case, which represents the best drift correction we currently know of, to the simple case in which the low resolution coefficients are extracted from the high resolution layer. It was hoped that some in-between setting of the weights would produce a better quality balance. However, the basic problems of low layer drift and upward prediction efficiency are not simultaneously resolved.

A solution to the drift problem was presented in MPEG 92/288 on the basis of a subband scheme using multiple prediction loops in encoder and decoder. This scheme will be addressed in the next Core Experiment trying to identify the maximum quality scheme with less hardware complexity constraints.

In an attempt to find simple solutions to the fundamental problems, we propose a Core Experiment in which the value of the lower layer coefficients as a predictor for those in the higher layer is studied. In particular, we envision two classes of variations. The first would use this upward DCT coefficient prediction in all MBs of certain picture types but not in those of others. The second class would allow adaptive use of the upward prediction on a macroblock basis. The purpose of the core experiment is to compare the picture quality resulting from the various approaches.

The first variation to be tested is one in which the upward prediction is unconditionally disabled for P- and B-pictures, to determine whether global prediciton of the prediction error (which is precisely what the upward prediction is doing in P- and B-pictures) is effective. This encoder is known as Version A. The second variation (Versions B and C) will employ adaptive upward coefficient prediction. This adaptivity will be possible at the MB level.

In the conditional upward prediction encoder Version B, upward prediction is done if the mean square prediction error is less than the energy of the predicted signal. As a benchmark, an encoder that uses the mode requiring fewer bits will also be implemented (Version C). The results in terms of SNR and picture quality produced by these two encoders will be compared with that of the basic encoder (with upward prediction), as well as with Version A.

In either of these experiments, improvements in the upward prediction will result in quality improvement at both resolutions, since the quantization factors used in each scale are identical. It is also of interest to consider the effect of conditional upward coefficient prediction in cases where the lower layer is bit-rate-constrained through more coarse quantization, aa this is likely to worsen the prediction efficiency. It should be noted that even in the worst case in which upward prediction never helps, the frequency pyramid has advantages over simulcast in not requiring redundant coding of side information and in efficiently coding I-picture data. Although no core experiment along these lines is proposed at this time, independent investigation is encouraged. Furthermore, we feel that important improvements in frequency domain scalable encoder performance can be made in areas other than conditional upward prediction and motion vector resolution, and urge our MPEG colleagues to consider this problem.

These encoders all fall within the framework shown in Figure I.4.

figure to be inserted

Figure I.4: Two-layer scalable encoder with conditional upward prediction.

The use of upward prediction is summarized in Table I.1.

Table I.1. Use of upward prediction in different picture types

Encoder Version	Upward Prediction I	P	B
A	y	n	n
B	y	c	c
C	y	c	c

y: upward prediction used
n: upward prediction not used
c: conditional

I.2.2: Description

The base encoder for this experiment has the following specifications:

- two resolution layers (scale 8 and scale 4), pyramidal structure
- two motion-compensation loops with down-sampling filter equivalant to an 8x8 forward DCT performed on each block followed by a 4x4 IDCT on the upper left 4x4 subblock.
- unconditional upward DCT coefficient prediction
- equal MQUANT in all layers
- other aspects as in TM1 and/or TM2 Appendix D

Test sequence: Mobile and Calendar

I.2.3: Syntax Changes

To indicate the use of Version A in the Test Model, 3 single-bit codes, **up_I**, **up_P**, and **up_B**, are inserted in the sequence layer after **end_of_fscales_code**. The interpretation of these bits are as follows:

up_I = 0 -> upward prediction used in I-pictures
up_I = 1 -> upward prediction not used in I-pictures

up_P = 0 -> upward prediction used in P-pictures
up_P = 1 -> upward prediction not used in P-pictures

up_B = 0 -> upward prediction used in B-pictures
up_B = 1 -> upward prediction not used in B-pictures

For Versions B and C, we use the **up_IPB** flags at the sequence layer, but with the following interpretation:

up_I = 0 -> adaptive upward prediction used in I-pictures
up_I = 1 -> upward prediction always used in I-pictures

up_P = 0 -> adaptive upward prediction used in P-pictures
up_P = 1 -> upward prediction always used in P-pictures

up_B = 0 -> adaptive upward prediction used in B-pictures
up_B = 1 -> upward prediction always used in B-pictures

In order to indicate whether upward prediction is to be used in a given MB, we introduce a MB layer codeword **up_pred**, used in a fashion similar to the currently existing MB syntax for **compatible_type**. The interpretation of the codeword is simply:

up_pred = 1 -> perform upward prediction in the current MB
up_pred = 0 -> no upward prediction in the current MB

Syntax modification at the Macroblock layer (directly after the **compatible_type** test) is as follows:

```
    if (picture_type == 1)
        if (up_I == 1)
            up_pred                         1       uimsbf
    else if (picture_type == 2)
        if (up_P == 1)
            up_pred                         1       uimsbf
    else if (picture_type == 3)
         if (up_B == 3)
            up_pred                         1       uimsbf
```

To emphasize, this codeword exists in the bit-stream only for Versions B and C of Core Experiment I.2, and only for picturetypes using adaptive upward prediction.

Core Experiment I.3: Maximum Quality Encoder

The purpose is to find a two-layer coding scheme with best possible subjective quality, which should also be balanced between the layers. The performance of subband and pyramid solutions to frequency scalability will be compared.

Theoretically a critically sampled (subband) scheme for layered coding should yield a superior coding performance over a pyramid scheme. It will then be suited to applications that are not satisfied with the quality from the pyramid approaches developed in the scalability group as well as in the compatibility group.

However this involves optimization of the underlying algorithm, i.e. taking away as many constraints as possible. Specifically no rate control will be used on the individual layers. Moreover one of the schemes proposes to use a multiple loop decoder for the top layer(s).

The following algorithms are proposed for this experiment:

a) the best pyramid scheme that can be identified from Core Experiment I.2
b) the subband scheme in Fig. 1 of MPEG 92/288 that was developed after the Haifa meeting.
c) the new subband scheme proposed in Fig. 3 of MPEG 92/288

Common parameters:

- Chrominance subsampling: 4:2:0
- Bit rate: 4 MBit/s
- N=12 (15), M=3
- two resolution layers (scale 8 and scale 4)
- Frame prediction and coding only
- 1/4 pixel motion vector resolution in scale 4
- equal QP in all layers
- other aspects as in TM1

The quality reference, for subjective comparisons as well as SNR calculations, is the SIF version of the original signal.

The most appropriate scheme for a) and its parameters are subject to the results of core experiment 2. Initially the encoder in MPEG 92/288 Fig 1. might be used with weighting factors of 0.5:0.5 for the two prediction error signals. The downsampling filter is the same as in Core Experiment I.2.

The weighting factors for b) shall be selected (iteratively) such that the SNR of the upper layer matches the one of a). The SNR and - more important - the subjective quality of the lower layer will be compared with a).

Proposal c) has no variables, so that a straight forward comparison of SNRs and picture quality with the other two candidates can be done.

The two subband schemes need a change in DCT scanning order to allow for bitstream scalability. The following scan order is proposed:

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

To signal the use of this scanning order for the subband schemes, a flag **subband** is introduced in the sequence header right after the **end_of_fscale_code** flag:

subband = 1 -> alternate scan order

Note: The maximum quality issue is relatively independent from the-interlace-in interlace problem as long as the results of Core Experiment I.1 are applicable here. Therefore the above scanning can be used

which is suitable for non-interlaced material. Mobile & Calendar (as a mildly interlaced sequence) is selected as test sequence for this experiment.

Core Experiment I.4: Scalable Side Information

I.4.1. Application

Any application which uses frequency scalability techniques and requires one or more layers with a sufficiently low bandwidth that all overhead information can not be transmitted within or below that layer. For example, multichannel transmission with a coding layer below 1 mbit/s.

I.4.2. Experiment details

I.4.2.1 Multichannel scalability, each channel with a fixed bandwidth.

Resolution scales: 1/16, 1/4, 1 (QSIF, SIF and 601)
Bitrates: 0.75, 1.5 and 4.0.
All layers must be at full temporal resolution.

I.4.2.2 Multichannel scalability for entertainment (each channel has fixed bandwidth)

Resolution scales: 1/4, 1, 1 (SIF, 601 and 601)
Bitrates: 1.5, 3.0, 4.0.
All layers must be at full temporal resolution.

I.4.3. Syntax extensions

The primary extension to the syntax is to support sending side information (macroblock type, address increment, coded block pattern and motion vector information) in the higher layers of the encoding. We introduce a new type of slave slice, called an incremental_slice, which contains incremental_macroblocks. Also we change the method of interpreting motion vector information. Rather than send full size motion vectors in the lowest layer and scaling them in the decoder for use at each layer , the motion vectors are sent at the scale appropriate to the dct_size of the slice within which they are transmitted. For use in higher quality slices they are scaled up. The higher layer slice may optionally contain refinement information for the motion vector.

I.4.3.1 Sequence header:

Add a one bit code **incremental_side_information** which is set to one to indicate that the slave_slice code now signals a incremental_slice and the use of incremental motion vectors.

```
incremental_slice()
{
      slave_slice_code                                          32    bslbf
      slave_slice_uantizer_ratio                                9     uimsbf
      dct_size                                                  8     uimsbf
      do {
       incremental_macroblock(dct_size)
      } while (nextbits() != 000 0000 0000 0000 0000 0000)
      next_start_code
}

incremental_macroblock(dct_size)
{
      while ( nextbits() == 0000 0001 111)
         macroblock_stuffing                                          11    vlclbf
      while ( nextbits() == 0000 0001 000)
```

```
            macroblock_escape                                          11    vlclbf
         macroblock_address_increment                                  1-11  vlclbf
         if (new_macroblock)
            macroblock_type                                            1-6   vlclbf
         else
            slave_macroblock_type                                      1-3   vlclbf
         if (macroblock_quant)
            quantizer_scale                                            5     uimsbf
         if (new_macroblock)
            if (compatible)
               compatible_type                                         2     uimsbf
            if (interlaced) {
               if (picture_structure == 0) {
                  if (macroblock_intra || macroblock_pattern)
                     interlaced_macroblock_type                        1     uimsbf
                  if (macroblock_motion_forward ||
macroblock_motion_backward)
                     interlace_motion_type                             2     uimsbf
               } else {
                  if (macroblock_motion_forward XOR
macroblock_motion_backward)
                     field_interlace_type                              1     uimsbf
               }
            }
         }
         if (motion_refinement) {
            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 (chroma_format == 0) {
        if (macroblock_pattern)
            coded_block_pattern                                 3-9    vlclbf
        for (i = 0; i < 6; i++)
            if (new_macroblock)
                scaled_block(i)
            else
                slave_block(i, dct_size)
    } else {
        if (macroblock_pattern)
            coded_block_pattern                                 8      uimsbf
        for (i = 0; i < 8; i++)
            if (new_macroblock)
                scaled_block(i)
            else
                slave_block(i, dct_size)
        }
    }
    if (picture_coding_type == 4)
        end_of_macroblock                                       1      1
}
```

new_macroblock is true if no previous information has been sent about the macroblock at the current macroblock address.

I.4.4. Coding

I.4.4.1 Slave Macroblock Addressing

Relative slave macroblock address are coded as described in section 8.1.

I.4.4.2 Slave Macroblock Type

Each macroblock has one of the three modes:

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

For these block types different VLC tables for the Slave Macroblock types are used. See table a for Intra, table b for predictive-coded blocks and table c for bidirectionally predictive-coded blocks. I blocks are all blocks within an I picture, intra and no motion compensation coded blocks within a P picture and intra coded blocks within a B picture. P blocks are predicted blocks within a P picture and forward or backward (but not both) predicted blocks within a B picture. B blocks are those blocks within a B picture which are bi-directionally predicted. The block type is determined by the first macroblock_type transmitted for the macroblock.
Methods for mode decisions are described in section 6. MTYPE is based on the highest resolution.

I.4.4.3 Macroblock_quant

In each slave macroblock the quantizer (mquant) used is the product of quantizer_scale and slave_slice_quantizer_ratio.

mquant = quantizer_scale * slave_slice_quantizer_ratio // 16

When macroblock_quant is 1 a new quantizer_scale is transmitted. When macroblock_quant is 0 the quantizer_scale from the same macroblock at the underlying layer is inherited. When new_macroblock is true (1) and macroblock_quant is false (0), the incremental_macroblock inherits its quantizer_scale from the left, rather than from below. If no macroblock to the left has a quantizer_scale, then quantizer_scale is taken from the base layer slice header.

I.4.4.4 Motion Vectors

Motion vectors are determined as described in Annex D. The scaled motion vector is transmitted in the base layer. When a block is not transmitted then the motion vector is zero. Coding of this scaled motion vector is as described in section 8. When **Δ_mv_present** is set, slave motion vectors are transmitted. Slave motion vectors are coded as an increment to the appropriately scaled up motion vector of the closest layer below. for which a motion vector exists. For example, if a motion vector was originally transmitted in a layer with a dct_size of 2 and is now scaled up for use in a layer with dct_size of 4, it is multiplied by two along each axis and the slave motion vector is added. This new vector forms the basis for the scaled up vector of the next layer. The incremental motion vector is coded using the same variable length code table B.4. The first value transmitted for a particular motion vector, base_vector, is calculated as described in section 8.3 except the appropriate f_code is reduced by 1 for each scale reduction factor of 2. All refinement motion vectors are calculated with an f_code of 1.

I.4.4.5 Slave Coded Block Pattern

When the coded block pattern is not sent as part of an incremental_macroblock, it is assumed to be zero -- no DCT coefficients are decoded. The coded_block_pattern is not inherited between layers unlike in normal frequeny scalability. Coded block pattern are determined by the quantized coefficients of the current layer.

I.4.4.6 Coefficient Coding

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

I.4.4.7 Multi-layer Rate Control

A seperate mquant value is determined for each incremental macroblock in the bitstream. The quantizer scale is only transmitted when needed as described above. The method for determining the mquant values is described in Section I.6, steps 1, 2a, and 3a.

I.4.4.8 Code / No Code decisions

Incremental macroblocks are not coded when there is no new information to be transmitted. When there are DCT or motion vector refinements the incremental macroblock must be coded.

I.4.4.9 Motion Estimation

A multiple level motion estimation routine is necessary for optimum refinement of the motion vectors at each layer. A top down hierarchical search is used. A full search is done at the highest resolution from original to original with a half pel refinement performed between original and reconstruction. The original picture is decimated using a forward DCT kernel at high resolution, followed by a smaller

inverse DCT of the low frequency components. The motion search in the lower scale is done from the decimated original to the lower scale reconstruction *by a refinement search around the scaled motion vector form the layer above. A search window of +/-1 pixel is used followed by a half pixel search.

I.4.5. Slave Macroblock Type

Table a. Variable length codes for slave_macroblock_type for blocks that are intra-coded (I-pictures, P-pictures, B-pictures).

VLC code	macroblock_quant	delta_mv_present	macroblock_pattern
0	0	0	1
11	1	0	1
10	0	0	0

Table b. Variable length codes for slave_macroblock_type for blocks that are predictive-coded (P-pictures, B-pictures).

VLC code	macroblock_quant	delta_mv_present	macroblock_pattern
0	0	1	1
11	0	1	0
10	1	1	1

Table c. Variable length codes for slave_macroblock_type for blocks that are bidirectionally predictive-coded (B-pictures).

VLC code	macroblock_quant	Δ_mv_present	macroblock_pattern
00	0	0	1
01	0	1	0
11	0	1	1
100	1	0	1
101	1	1	1

Core Experiment I.5: Scalable VLC coding

Two suggestions have been presented aimed at improving the efficiency of VLC coding in scalable systems (MPEG 92/361 & MPEG 92/356). MPEG 92/356 suggests a way to reduce the number of EOB's that need to be sent by extendng the last run in each layer into the next one, and using that "run over" to implicity specify an EOB for the layer. MPEG 92/361 suggests that different Run/Length VLC table be used at each layer. Below, the 2 experiments are described. It is suggested that both experiments be combined.

I.5.1: Experiment 1

I.5.1.1: Description

The basic idea is to send implied EOB's at lower layers in order to reduce the total number of EOB's transmitted, thereby improving the efficiency of the algorithm. There are two variations, one for subband and one for pyramid scalable algorithms.

Subband Case (as suggested in MPEG 92/356)

Layer	Coefficients always decoded	Coefficients carried over to next layer
fscale_2	1-5	4
fscale_4	6-14	11
fscale_8	15-64	

At fscale_2, the first 5 coeffients in the standard zigzag scan are coded. Coefficient 4 is transmitted to the decoder, but not used to decode the small picture, and simply stored for use at the next level. If coefficient 5 is a zero, the encoder can choose to terminate the block at this layer with an EOB, or it can code it as a zero in a run/length pair that runs over into the information for the next layer - implicitly specifying the EOB. Any extra coefficients are thereby also transmitted to the decoder, but not used to produce the lowest layer picture - they are stored for use at the next levels.

At fscale_4, decoding of VLC's continues at the position beyond the last coefficient transmitted from the previous level, and then proceeds as it did for the previous level.
Fscale_8 continues similiarly, and terminates with an EOB or a coefficient in position 64.

Pyramid Case

Layer	Coefficients always decoded	Coefficients carried over to next layer
fscale_2	1-5	4
fscale_4	6-25	11
fscale_8	26-64	

At fscale_2, coefficients for the first 5 coefficients are sent. If these 5 coefficients end in one or more zeros, the encoder can choose to send it as an EOB, or code the next VLC, resulting in an implied EOB as above in the sub-band case.

At fscale_4, differential values are coded for each coefficient that was transmitted at fscale_2, and quantized values are transmitted for the remaining values in the block up to coefficient 14.

At fscale_8, values transmitted for coefficients that were transmitted at the previous layer are differential, while values for the others are the quantized coefficients.Syntax Changes from MPEG 92/356.

I.5.1.2: Syntax Changes (from MPEG 92/356)

Sequence Layer:

```
do {
            :
      fscale_num_coef
      fscale_max_run
            :
} while(next_bits != '10000000')
```

Scaled Block Layer:

```
fscale_coef_cnt = coef_block(fscale_num_coef, fscale_max_run)
fscale_extra_prev = fscale_coef_cnt - fscale_num_coef
```

Slave Block Layers:

```
fscale_rem_coef = fscale_num_coef - fscale_extra_prev
if(fscale_rem_coef < 0) fscale_num_coef = 0
fscale_coef_cnt=coef_block(fscale_rem_coef, fscale_max_run)
fscale_etra_prev = fscale_coef_cnt - fscale_rem_coef

int coef_block(fscale_num_coef, fscale_max_run)
{
      fscale_coef_cnt = 0
      run = 0
      while( fscale_coef_cnt < fscale_num_coef)
            next_dct_coef(dct_size)
            fscale_coef_cnt++
            if(dct_coef_zero())run++
            else run = 0
      }
      if (run != 0){
            do{
                  next_dct_coef(dct_size)
                  fscale_coef_cnt++
                  if(dct_coef_zero()){
                        run++
                        if(run== fscale_max_run + 1){
                              end_of_block
                              break
                        }
                  }
            } while(dct_coef_zero())
      }
      return(fscale_coef)
}
```

I.5.2: Experiment I.5.2

I.5.2.1: Background and Description

This experiment is to compare the result of using the standard VLC run/length table to using different tables at each layer. Also, a different coefficient scan pattern derived from Subband coding can be used in both pyramid coding and sub-band coding with the new set of Huffman tables.

Alternate Scan:

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

I.5.2.2: Syntax Changes and VLC's (from MPEG 92/361)

```
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)
                next_dct_coef(dct_size)                  2-16  vlclbf
            end_of_block (dct_size)                      2-16  vlclbf
        }
    }
}
```

Slave block Layer:

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

Core Experiment I.6: Slice vs. MB Rate Control

I.6.1: Background and Description

This core experiment will compare the effect of performing rate control at a slice or MB granularity. The outcome of this experiment is to determine whether higher resolution layer rate control at slice granularity performs as well as at macroblock granularity.

In both approaches, rate control is carried out at macroblock granularity for the low resolution layer. Also, for both approaches a multiplication factor is sent in the slice header for the higher resolution layers. In the slice granularity approach, for the higher resolution layers, each macroblock Mquant is the product of the multiplication factor and Mquant from the low resolution layer. For the macroblock granularity approach, the encoder can signal the use of :

- a new Mquant
- the slice multiplication factor
- the previous macroblock's Mquant

for each MB.

Both approaches are based on the TM1 rate control algorithm. Only the equations that differ from TM1 are given below. The picture target setting equations are common to the two approaches:

Step 1 - Bit allocation

Complexity estimation

$$X_{ipb}(l,m,h) = S_{ipb}(l,m,h) * Q_{ipb}(l,m,h)$$

where:

l refers to the lower resolution layer
m refers to the medium resolution layer
h refers to the higher resolution layer

Initial conditions:

$$X_i(l,m,h) = 160 * br\%(l,m,h) * bit\_rate/115$$
$$X_p(l,m,h) = 60 * br\%(l,m,h) * bit\_rate/115$$
$$X_b(l,m,h) = 42 * br\%(l,m,h) * bit\_rate/115$$

Picture target setting

$$T_i(l,m,h) = \max \{ \textbf{Error!}, br\%(l,m,h)*bit\_rate / (8*picture\_rate)\}$$

$$T_p(l,m,h) = \max \{ \textbf{Error!}), br\%(l,m,h)*bit\_rate / (8*picture\_rate)\}$$

$$T_b(l,m,h) = \max \{ \textbf{Error!}, br\%(l,m,h)*bit\_rate / (8*picture\_rate)\}$$

After coding a picture, $R(l,m,h) = R(l,m,h) - S_{ipb}(l,m,h)$. Before encoding the first picture in a GOP:

$$R(l,m,h) = br\%(l,m,h)*G + R(l,m,h).$$

Step 2 - Rate Control

The two rate control aproaches differ at this step.

a) MB granularity approach

$$d_j^{ipb}(l,m,h) = d_{j0}^{ipb}(l,m,h) + B_{j-1}(l,m,h) - \textbf{Error!}$$

$$Q_j(l,m,h) = \textbf{Error!}$$

Initial conditions:

$$d_0^i(l,m,h) = \textbf{Error!}$$
$$d_0^p(l,m,h) = K_p * d_0^i(l,m,h)$$
$$d_0^b(l,m,h) = K_b * d_0^i(l,m,h)$$

b) Slice granularity approach

$$d_s^{ipb}(l,m,h) = d_{s0}^{ipb}(l,m,h) + B_{s-1}(l,m,h) \text{ - Error!}$$

$$Q_s(l,m,h) = \text{Error!}$$

where Slc_cnt is the number of slices in a picture and s is the current slice index.

Initial conditions:

$$d_0^i(l,m,h) = \text{Error!}$$
$$d_0^p(l,m,h) = K_p * d_0^i(l,m,h)$$
$$d_0^b(l,m,h) = K_b * d_0^i(l,m,h)$$

Step 3 - Adaptive quantization
This step also differs for the two approaches.

a) MB granularity approach

$$mquant_j(l,m,h) = Q_j(l,m,h) * N\_act_j$$

where N_act_j is as defined in the TM1

b) Slice granularity approach

For the lower resolution layer:

$$mquant_j(l) = Q_j(l) * N\_act_j;$$

For the higher resolution layers:

$$M\_ratio_s(m,h) = \text{Error!}$$

A 9 bit pattern (slave_slice_quantizer_ratio)

16 8 4 2 1 0.5 0.25 0.125 0.0625

is used to represent M_ratio_s, e.g a slave_slice_quantizer_ratio of 0.75 would be represented as 000001100. At the decoder the MB mquant is

$$mquant_j(m,h) = M\_ratio_s(m,h)*mquant_j(l).$$

I.6.2: Syntax extensions

a) MB granularity approach

A different mquant value may be required for each layer within the same macro-block. The syntax was extended to support transmission of the extra quantizer information. The new slave macroblock definition is:

```
slave_macroblock(dct_size) {
      slave_macroblock_quant                    1     uimsbf
      if (slave_macroblock_quant)
            slave_macroblock_quantizer_scale    5     uimsbf
      if (slave_macroblock_quant != 1)
            slave_macroblock_quantizer_source   1     uimsbf
```

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

When **slave_macroblock_quant** is 1, the slave macroblock contains a new 5 bit quantizer for use in decoding the slave DCT coefficients. Otherwise a one bit field, **slave_macroblock_quantizer_source** is transmitted to indicate whether the slave_blocks should be dequantized using the same **mquant** as the previous slave macroblock (**slave_macroblock_quantizer_source** = 1), or using the product of the lower resolution layer macroblock's quantizer and the slice quantizer multiplier - $\mathbf{M\_ratio}_S$ (**slave_macroblock_quantizer_source** = 0)

b) Slice granularity approach

Slave slice layer:

```
slave_slice() {
        slave_slice_start_code                  32      bslbf
        slave_slice_quantizer_ratio              9      bslbf
        dct_size 8          uimsbf
        for(s=0; s<slice_size; s++) {
                slave_macroblock(dct_size)
        }
}
```

Core Experiment I.7: Encoder with Drift Correction Layer and Improved Motion Rendition

The purpose of this experiment is to investigate the effect of field/frame adaptive MC and DCT on a low resolution layer of a frequency scalable bitstream. Another purpose is to investigate improvements to the motion rendition of this layer.

The encoders we investigate generate a bitstream with three f-scalable layers: two 4x4 layers, and one 8x8 layer.

The data format is 4:2:2 to avoid the mismatch of the DCT mode. In the 4:2:0 format, the chrominance blocks are coded with frame DCT even if the luminance blocks are coded with field DCTs.

The frame/field adaptive MC and DCT are as described in TM1. The decision algorithm is the same as TM1 adapted to the simulation conditions below. Mode and other attributes of the slave layers are inherited from the master layer, as with the basic codec.

I.7.1 SIMULATION CONDITIONS

- Frame MC mode + Frame DCT mode
- Field MC mode + Frame DCT mode
- Frame MC mode + Field DCT mode
- Field MC mode + Field DCT mode
- Field/Frame adaptive mode, where DCT mode follows MC mode.

The actual experiments will use conditions 1 or 5 only.

I.7.2 SYNTAX OF THE PROPOSED EXPERIMENT

Most of the basic syntax is preserved in this experiment. Some minor modifications are explained throughout this text. One such modification is an extension of the basic multiplexing, to allow for the existence of two layers at scale 4, only one of which (the base layer) is used in the reconstruction of the high resolution (scale 8) pictures. The other scale 4 layer (slave 2 below) is used as an SNR enhancement for the scale 4 base layer only.

4x4 base bitstream	master
8x8 bitstream	slave 1
4x4 difference bitstream	slave 2

To indicate this, the "fscalable" flag in the sequence header is set, followed by the following **fscales**:

bitstream	**scale code**
4x4	2
8x8	3
4x4 difference	2
end_of_scale	7

In decoding at the 8x8 resolution, the last 4x4 layer should be ignored.

I.7.3 ENCODER DESCRIPTION

The block diagram of the drift correction encoder is specified in Figure I.6. This encoder generates three hierarchical layers. Two of the hierarchical layers are generated in the same manner as the basic encoder, i.e. by extracting the 4x4 DCT coefficients from the full resolution 8x8 blocks. These two layers are the only ones required to reconstruct the full resolution video. The third layer is a difference layer generated by subtracting the output of a second encoder (operating at the 4x4 resolution) from the extracted 4x4 coefficient layer. Therefore this layer contains data that can be used to correct the drift error that would be generated by decoding with the extracted 4x4 layer alone.

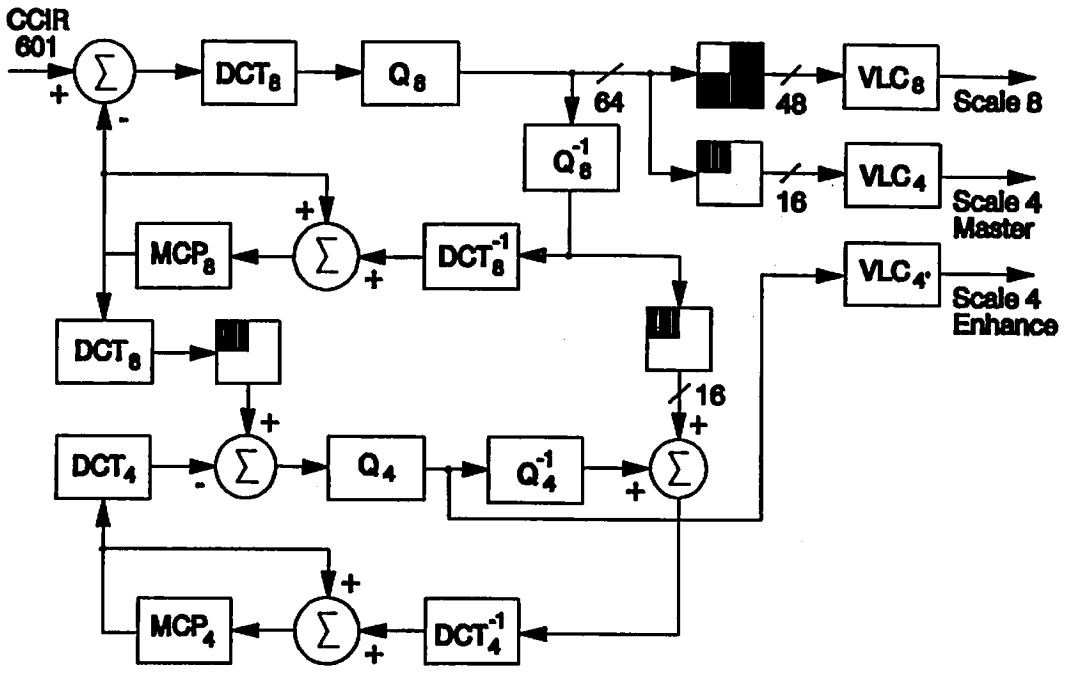


Figure I.5: Drift correcting encoder

I.7.3.1 Rate Control of the difference bitstream

Basically this drift correction method is able to be applied to every picture type: I, P, and B. However, the I-picture does not need to be compensated, because it does not have a "drift error". B-pictures also may not need to be compensated, because drift error in these pictures do not propagate. Therefore compensation may only be needed for P-pictures. If this is the case, we propose to keep the 4x4 difference slave_slices in the bitstream, but only with some flag to indicate that it is empty.

The rate control of P-picture is based on TM1. The modification is only the initial values:

Xi = bitrate/ 115
Xp = 260*bitrate/ 115
Xb = bitrate/ 115

There are two sets of target bitrates for this experiment:

1)	difference bitstream:	0.5 Mbit/s
	4x4 + 8x8 bitstream:	3.5 Mbit/s
2)	difference bitstream:	1.0 Mbit/s
	4x4 + 8x8 bitstream:	3.0 Mbit/s

I.7.4 DECODER DESCRIPTION

The corresponding decoder is shown in Figure I.7. The decoder for the 8x8 resolution is the same as a basic encoder with two layers. It therefore has no drift error. To decode 4x4 resolution video without drift error, the two 4x4 layers should be dequantized and added before applying the inverse DCT.

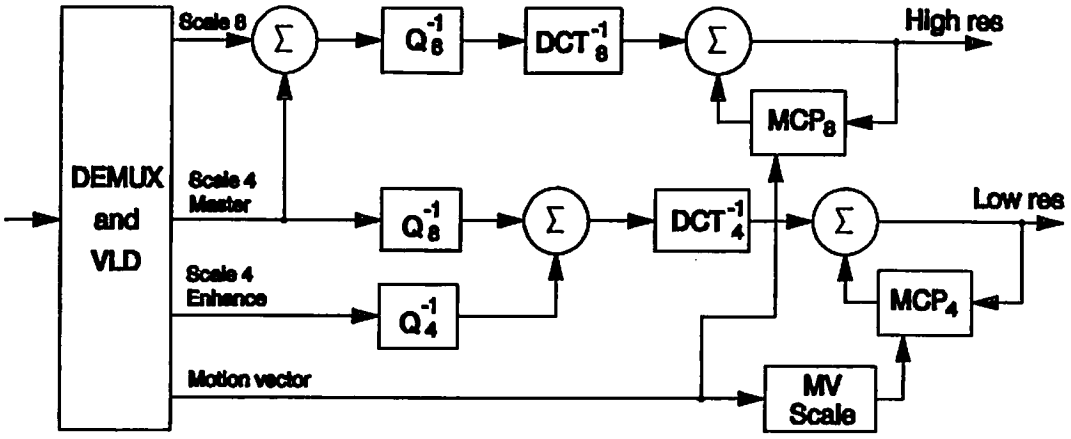


Figure I.6: Drift correcting decoder

I.7.5 EXPERIMENTS

The results from the following experiments should be compared.

I.7.5.1 ADAPTIVE DCT/MC CODING

In this experiment Simulation codition 5 is used.

I.7.5.2 SCAN PATTERNS FOR INTERLACE-IN-INTERLACE DECODING

For this experiment Simulation condition 1 is used. In addition, instead of the upper left 4x4 coefficients, the upper left and the lower left 4x2 coefficients are extracted and supplied to the 4x4 IDCT. The zig-zag scan needs to be modified for this experiment:

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

Core Experiment I.8: Frequency Scanning

Frequency scanning combined with an entropy coding technique, called MUVLC (Modified Universal Variable Length Coding) has been proposed to increase the coding efficiency of the test model and to improve the performance of scalable systems both in terms of coding efficiency and functionality.[4,6]. The purpose of this experiment is to compare the coding efficiency of the proposed method when applied in the framework of a single loop scalable encoder with the performance of a corresponding nonscalable coder using block scanning.

In the following the principle of the algorithm and its adaptation to the TM2 syntax are described. A C-program of the MUVLC algorithm will be distributed to interested parties by e-mail.
(send your request to: Bernard Hammer (Siemens), e-mail: ha@bvax4.zfe.siemens.de)

I.8.1 Global parameters of the experiment

The core experiment should be carried out using the syntax of a single loop scalable encoder (as described in 9.3.3 ff and in appendix D) with the following parameters:

chroma_format:	4:2:0
picture_structure:	frame picture
bit_rate:	4 Mbit/s
group of pictures structure:	N=12 (15), M=3
resolution layers:	3 (scale_8, scale_4, scale_2)
rate control	equal QP in all layers, mquant = non adaptive

I.8.2 Principle of MUVLC

In contrast to entropy coding using block scanning, as described in 8.5 - 8.7, the proposed frequency scanning method applies run length coding to coefficients of different blocks, but which have the same scanning number respectively representing the same frequency band.

The code for a slice is generated in six basic steps:

1) arrange the coefficients of all coded blocks (indicated by the CBP), which belong to a macroblock slice, according their scanning number in 64 stripes;

2) for encoding a stripe select from the table, given below, the magnitude class in which the coefficient with highest amplitude falls and transmit the 3 bit class prefix code PC;

magnitude range	class C	prefix class code (PC)
0 - 1	0	000
2 - 3	1	001
4 - 7	2	010
8 - 15	3	011
16 - 31	4	100
32 - 63	5	101
64 - 127	6	110
128 - 255	7	111

3) transmit the 3 bit line prefix code PL,which determines the Adaptive Truncated Runlength (ATRL) code [2] used in step 4).for this class.

4) encode each non zero coefficient of class C using a code of the form

(RL NCB)

where RL gives the position of the coefficient in the class relativ to its previous nonzero coefficient. NCB gives the exact value of the coefficient by transmitting the C least significant bits of its magnitude plus one sign bit. Each class code string is terminated by an End of Class (EOC) code word.

5) repeat step 3) to 4) for all remaining magnitude classes . Coefficients which have been already encoded in the previous classes are ignored for calculating the runlength code RL.

6) repeat step 2) to 5) for all stripes of the slice

Thus the code for a slice gets the following structure:

```
PC PL (RL NCB) (RL NCB) (RL NCB) .....EOC        stripe 1, class C
   PL (RL NCB) (RL NCB) (RL NCB) .....EOC        stripe 1, class C-1
           ------------
   PL (RL NCB) (RL NCB) (RL NCB) .....EOC        stripe 1, class 0
PC PL (RL NCB) (RL NCB) (RL NCB) .....EOC        stripe 2, class C
           ------------
   PL (RL NCB) (RL NCB) (RL NCB) .....EOC        stripe 64 , class 0
```

I.8.2.1 Principle of ATRL

Table I.8.1 shows the basic code structure of an ATRL-code with PL=3. A "0"in the source pattern denotes a coefficient of a lower class while "1" indicates the occurence of a coefficient within the current magnitude class.
The maximal run length, which can be encoded by ATRL with a single codeword is truncated to a length of M = 2**PL-1.To cope with runs having more than M preceding "0"s the first codeword is used for extension by runs of M+1 "0"s.
The extension code word is coded by a single bit set to "0", while run lengths codewords consist of a prefix set to "1" followed by PL bits which give the number of preceding "0"s.

To indicate the End of Class (EOC) a position outside the stripe is addressed by sending one or more extension codewords

source pattern	Run length (RL)	RL- Code
00000000	8	0
1	0	1000
01	1	1001
001	2	1010
0001	3	1011
00001	4	1100
000001	5	1101
0000001	6	1110
00000001	7	1111

Table I.6.1: Truncated run-length code for PL=3 and M=8.

To cope with the different statistics of each magnitude class PL may vary between 0 and 7 in order to minimize the code length.
The code length is calculated by the formula below:

```
for (i=1; i<=k; i++)
  code_length += run_length[i] / (M+1)

code_length += k * (1+PL) + zero_run_length / (M+1) +1
```

with run_length [i] = run length of the nonzero coefficient i in a class
k = number of non zero coefficients in a class
zero_run_length = number of zero coefficients following coefficient k

I.8.3 Arrangement of luminance and chrominance coefficients

I.8.3.1 Interleaving of luminance and chrominance coefficients within a stripe

Y-coefficients (max. 176)	Cb-coefficients (max. 44)	Cr-coefficients (max. 44)

I.8.3.2 Scanning order of blocks within a macroblock

Blocks are scanned according the numbering as being described for block_count in 9.3.7

I.8.3.3 Scanning order of coefficients within a block

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

scale_2 layer: 1 - 4
scale_4 layer: 5 - 16
scale_8 layer: 17 - 64

I.8.4 Syntax modifications

For purpose of the frequency scanning experiments the slice layer, the macro block layer and the block layer must be changed as indicated below:

I.8.4.1 Slice Layer

```
slice() {
  slice_start_code
  quantizer_scale
        ::::
    no change
        ::::
  extra_bit_slice
  slice_control()
  slice_data()
}
```

I.8.4.2 Slice Control Layer

```
slice_control() {
  for (mb=0; mb<44; mb ++) {
    while(nextbits() == '0000 0001 111')
      macroblock_stuffing
        ::::
        same syntax as in macroblock() of TM2
        ::::
    if (macroblock_pattern)
      coded_block_pattern()
  }
}
```

I.8.4.3 Slice Data Layer

```
slice_data() {
 for (layer=0; layer<3; layer++) {
  if (layer!=0)
   slave_slice_start_code                                    32
  for (coef_i=start[layer];coef_i<end[layer]; coef_i ++) {
    buf_size = 0
    for (mb = 0; mb < 44; mb++) {
      for(block_i=1; block_i<5; block_i++) {
          if (pattern_code[mb][block_i]) {
           c_buf[buf_size] = dct_coef[mb][block_i][coef_i]
            buf_size++
          }
      }
    }
    for (mb = 0; mb < 44; mb++) {
      if (pattern_code[mb][5]) {
        c_buf[buf_size] = dct_coef[mb][5][coef_i]
         buf_size++
      }
    }
    for (mb = 0; mb < 44; mb++) {
      if (pattern_code[mb][6]) {
        c_buf[buf_size] = dct_coef[mb][6][coef_i]
         buf_size++
      }
    }
   muvlc(c_buf,buf_size)
  }
 }
 while(nextbits()!='0000 0000 0000 0000 0000 00001)
  next_start_code()
}
```

```
with     start [0] =  1;          end [0] =  4
         start [1] =  5;          end [1] = 16
         start [2] = 17;          end [2] = 64
```

```
muvlc(c_buf,buf_size) {
pc = pc_code(c_buf,buf_size)                                    3
  for(class=pc; class=0; class--) {
   pl = pl_code(c_buf, buf_size)                              3
    run = 0
    for (buf_i = 0; buf_i < buf_size; buf_i ++) {
      if(!c_coded[buf_i]) {
        if (((c_buf[buf_i] >> class) & 0x1) == 1) {
          rl = rl_code(run,pl)
          ncb = ncb_code(c_buf[buf_i],pc)                       1-8
           c_coded[buf_i] = 1
           run = 0
        }
        else
           run++
      }
    }
     eoc = eoc_code(run,pl)
  }
}
```

```
rl_code(run,pl) {
  while (run >= 1<<pl) {
   max_run                                                  1        "0"
    run -= 1<<pl
  }
  prefix                                                    1        "1"
  run                                                               1-8
}
```

```
eoc_code(run,pl) {
  while (run >= 0) {
   max_run                                                  1        "0"
    run -= 1<<pl
  }
```

I.8.5 Supplementary experiments

- Change of processing order of luminance and chrominance coefficients within a stripe

  - the coding efficiency may be increased by processing luminance and chrominance coefficients with similar statistical behaviour within a stripe, i.e. Y- and Cb- and Cr-coefficients corresponding to the same sampling frequency

  - to improve the access to image data it is desirable to have luminance and chrominance information of an image region close together in the bit stream. Instead of processing luminance and chroninance coefficients in separate blocks as proposed in the core experiment, interleaving of the components may be possible without significant loss of coding efficiency.

- Removal of Coded Block Pattern (CBP) in the Slice Control Layer

The use of CBP as proposed in the core experiments implies, that the length of each stripe will differ.which may adds some complexity to the hardware. Having in mind that CBP gives no significant gain in coding efficiency at bitrates > 1 Mbit/s it seems obvious to skip CBP and use the run length coding ability of MUVLC instead.

I.8.6 References

[1] B. Macq: "An Universal Entropy Coder for Transform or Hybrid Coding", Picture Coding Symposium, session 12, March 26-28 1990, Cambridge, Mass., USA.

[2] H. Tanaka and A. Leon-Garcia: "Efficient Run-Length Encoding", IEEE Transactions on Information Theory, vol. IT-28, no.6, pp. 880-890, November 1982.

[3] CCIR Study Group - Document CMTT-2/31, March 1990 - Belgium RTT

[4] G. Schamel et al "Experiments with UVLC-coding" doc MPEG92/289

[5] Th. Selinger, "Implementation Study of a MUVLD", doc MPEG 92/388

[6] A. Knoll "Experiments with UVLC-coding" doc MPEG 92/391

Core Experiment I.9: Efficient Frequency Scalability Core Experiment

I.9.1: Background and Description

Reference: MPEG 92/356

Goals:

1. Study impact on picture quality of encofrcing DCT coefficients of lower frequency scales to be subset of coefficients along the zigzag scan of 8x8 DCT coefficient block.

2. Measure saving in overhead bits (in sub-band approach) of sending EOB code conditionally, assuming lower layer frequency scale docoders with limited run/length capability.

Experiment:

This experiment consists of studying the impact on picture quality of using for lower frequency scales, coefficients that fall along the zigzag scan of the 8x8 DCT coefficient block, compared to choosing coefficients for lower frequency scales corresponding to a square block. A two layer approach with the following parameters is used:

Frequency Scale	# of coefficients belonging to sub-band	# of coefficients selected in zigzag scan
fscale 4	16	14
fscale 8	48	50

The second part of this experiment measures the savings in EOB overhead bits because of including implicit EOB (marked by the first coefficient of the next higher frequency scale). A three layer sub-band splitting of 8x8 DCT coefficients is employed and EOBs are sent only when absolutely necessary, depending on the decoding capabilities of the corresponding decoder. The parameters used are:

Frequency Scale	# of coefficients belonging to sub-band	# of coefficients selected in zigzag scan	Maximum run allowed in VLC
fscale 2	4	3	3
fscale 4	12	11	15
fscale 8	48	50	63

I.9.2 Syntax Specification

See Section I.5.1.2 for this syntax.

Core Experiment I.10: Spatial/Frequency Hybrid Scalability

I.10.1: Background and Description

Reference: MPEG 92/355, MPEG 92/356

Goal:

Investigate the effectiveness of spatial/frequency hybrid to derive well-balanced multi-resolution scales.

Experiment:

This is a revised specification of experiment I.3.1 of Appendix I in TM1. In this experiment, one spatial layer and two frequency layers are generated. The SIF odd fields are encoded using MPEG-1, but the resulting bit-stream is partitioned in "fscale4" and "fscale8" sub-streams representing frequency scales derived from SIF resolution. In generating "fscale4" and fscale8", use of efficient frequency scaling (Doc MPEG 92/356) is made to keep complexity of generating frequency scales low and to minimize the overhead. The total bit-rate for "fscale4" and "fscale8" is 1.15 Mbit/s.

The higher layer is CCIR-601 field-structure encoded using MPEG-2 at 2.85 Mbit/s. The motion compensation prediction options in field structure coding, such as adaptive field/dual field are utilized. For CCIR-601 Odd fields, an optional prediction is generated by interpolating the reconstructed SIF picture.

The block diagram and syntax are described in Doc MPEG 92/355, syntax for more efficient frequency scalability is given in Doc MPEG 92/356.

I.11 Core experiment on multi loop decoding

The purpose of the experiment is to identify the impact of different methods of coupling the prediction signals in a pyramid two loop frequency scalable coder. These methods are compared against a scheme without this coupling and the "minumum drift decoder" (described in document MPEG 92/466) working on the bit stream of a single prediction loop encoder.

I.11.1 Experimental conditions

* Test Model 2 with frame MC / frame DCT only
* 4 Mbit/s, M=3, N=12(15), 4:2:0
* two prediction loops & two bit stream layers
* ME on upper layer; 1/4 pel MC on lower layer
* same quantizer on both layers
* DCT coefficient prediction
* syntax as in App. D of TM2
* lower layer input signal: DCT decimated
* all interscale links with 4*4 DCT pattern extraction
* Test sequences: Mobile, Bus, Table Tennis (2 seconds each)

I.11.2 Parameter under study

Under study is the method of merging the 4*4 and 8*8 prediction signals in the 8*8 coding loop. The following versions are suggested (MPEG documents in brackets give more details, e.g. block diagrams of coders):

(a) Conditional replacement of low-freq. 4*4 coefficients (92/488)
(b) Unconditional replacement of low-freq. 4*4 coefficients (92/288)
(c) Weighted addition (ratio 1:1) of both prediction signals (92/501)
(d) No connection between prediction loops (92/486)

and, as a benchmark with minimum impact on the upper layer quality:

(e) "Minimum drift decoder" for the lower layer, working on a single prediction loop encoder, with a two layer bit stream generated by employing the scanning pattern proposed in Fig. I.x.1
(Leading zeros in the scan for the 8*8 layer shall not be transmitted, i.e. coefficient 17 is coded as {Run=0,Amplitude}!)

Versions (a) to (c) require the use of a multi-loop decoder.

I.11.3 Variations of the Experiment

As a variation to versions (a) to (c) the output of the 4*4 VLC can be switched off. To maintain the two bit stream layers, the 8*8 coefficient stream can be distributed the same way as in (e). The 4*4 reconstructed signal will then be retrieved from this second layer. This results in a "DCT-subband" structure of the scheme.

Furthermore all sequences can be processed employing the 4*3+4*1 interlace extraction pattern for (a) DCT decimation; (b) coupling of prediction signals and (c) prediction of DCT coefficients.

Experiments with MB adaptive switching of these pattern as well as with the use of adaptive frame/field MC in both layers are encouraged.

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

Figure I.11.1 Scanning pattern for (c) and experiment variation (same as in I.3)

Core Experiment I.12: Adaptive Inter-scale Prediction

I.12.1. Objective

The objective of this core experiment is to assess the effectiveness of using inter-scale prediction adaptively on a macroblock by macroblock basis, in a frequency domain layered coding system. It was shown in MPEG 92/486 and 92/501 (Core Experiment I.2) that adaptive inter-scale prediction is not

particularly effective when the quantization factors of the two layers are identical in all macroblocks. In such cases, corresponding coefficients in different scales are highly correlfated, even in motion-compensated difference blocks, because the differences result only from non-identical motion-compensation. However, in cases where lower layers are rate-controlled by quantization which differs from that of the higher layer, the increase in quantization noise levels of the lower scale should reduce the correlation between corresponding coefficients in different scales. This experiment is an extension of Core Experiment I.2 (MPEG 92/NO245 Test Model 2), under identical conditions, except for the use of rate control in the lower layer.

I.12.2. Simulation Conditions

Sequence format:	4:2:0
Picture structure:	Frame
Coding and MC modes:	Frame only
Number of layers:	2 (scales 4 and 8)
Scale-8 bit rate:	4.0 Mbit/sec
Scale-4 bit rate:	1.5, 2.0 Mbit/sec
GOP size:	15 frames/GOP
M:	3

Rate control for scale-4 will be performed as in Core Experiment I.6 (slice granularity).

Appendix K: MOTION COMPENSATION FOR SIMPLIFIED FAMC

K.1. Motion Compensation

The basic idea of FAMC is that a trajectory is defined by one transmitted motion vector. To calculate the prediction value, the two pixels are used in each horizontal and vertical direction with corresponding weighting coefficients, and these two pixels should be located in the each side of the trajectory. As the results, FAMC prediction is the interpolation from four pixels in reference frame. The address calculations of these 4 points are defined in the followings, and it is illustrated in Fig. K.1. The horizontal (X) axis interpolation is rounded in half pixel precision because of hardware simplification, and the vertical interpolation is arithmetic precision interpolation.

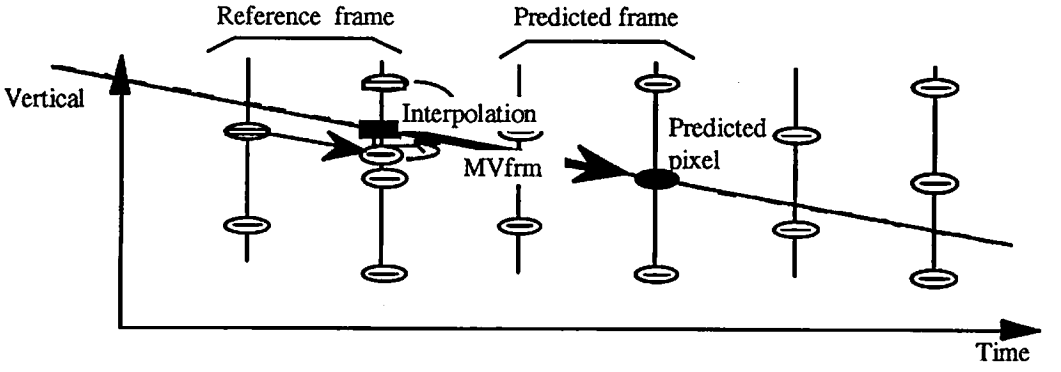


Figure K.0 Principle of FAMC (Vertical direction)

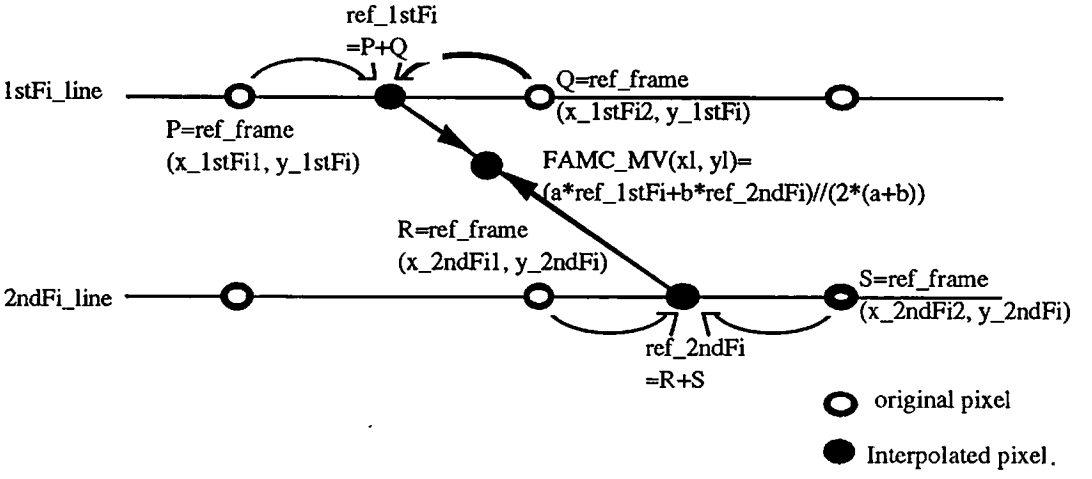


Figure K.1: FAMC prediction for 1stFi line

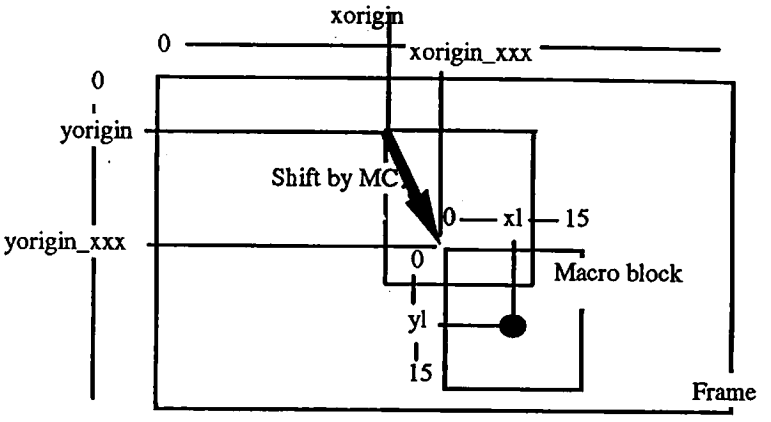


Figure K.2: The relation of variables

Get_Simplified_FAMC_MB_for_Forward (Frame_Distance, Origin, FAMC_MV, FAMC_MB){
LeftPel = 0, RightPel = 703, Top1stFiLine = 0, Bottom1stFiLine = 478,
Top2ndFiLine = 1, Bottom2ndFiLine = 479

*M = 16 // (2*Frame_Distance))* **/* M = N * 16; N = 1/(2*Frame_Distance) */**
/* = 8 When Frame_Distance=1 */
/* = 4 When Frame_Distance=2 */
/* = 3 When Frame_Distance=3 */

/* For first Field */

*xorigin_1stFi1 = xorigin + (2 * FAMC_MVx)/2*
*xorigin_1stFi2 = xorigin + (2 * FAMC_MVx)//2*
*xorigin_2ndFi1 = xorigin + (2*FAMC_MVx - (M*FAMC_MVx)//8)/2*
*xorigin_2ndFi2 = xorigin + (2*FAMC_MVx - (M*FAMC_MVx)//8)//2*
yorigin_1stFi = yorigin + Adjacent_1stFi_Line_for_1st_Field_for_Forward (Frame_Distance, FAMC_MVy)
yorigin_2ndFi = yorigin + Adjacent_2ndFi_Line_for_1st_Field_for_Forward(Frame_Distance, FAMC_MVy)

for (xl=0; xl<16; ++xl) {
x_1stFi1 = xl + xorigin_1stFi1 **/* Addressing X of pixel P */**
x_1stFi2 = xl + xorigin_1stFi2 **/* Addressing X of pixel Q */**
x_2ndFi1 = xl + xorigin_2ndFi1 **/* Addressing X of pixel R */**
x_2ndFi2 = xl + xorigin_2ndFi2 **/* Addressing X of pixel S */**

/* In case that the required pixel is out of frame */
if (x_1stFi1 < LeftPel) x_1stFi1 = LeftPel
if (x_1stFi1 > RightPel) x_1stFi1 = RightPel
if (x_2ndFi1 < LeftPel) x_2ndFi1 = LeftPel
if (x_2ndFi1 > RightPel) x_2ndFi1 = RightPel

for (yl=0; yl<16; yl=yl+2) {
y_1stFi = yl + yorigin_1stFi **/* Addressing Y of P & Q pixels */**
y_2ndFi = yl + yorigin_2ndFi **/* Addressing Y of R & S pixels */**

/* In case that the required pixel is out of frame */
if (y_1stFi < Top1stFiLine) y_1stFi = Top1stFiLine
if (y_1stFi > Bottom1stFiLine) y_1stFi = Bottom1stFiLine

```
    if (y_2ndFi < Top2ndFiLine) y_2ndFi = Top2ndFiLine
    if (y_2ndFi > Bottom2ndFiLine) y_2ndFi = Bottom2ndFiLine

                                        /* interpolation    */
      ref_1stFi = ref_frame(x_1stFi1,y_1stFi) + ref_frame(x_1stFi2,y_1stFi)
      ref_2ndFi = ref_frame(x_2ndFi1,y_2ndFi) + ref_frame(x_2ndFi2,y_2ndFi)
      FAMC_MB(xl,yl) = (a*ref_1stFi + b*ref_2ndFi)//16
    }
  }
```

/* For Second Field */

```
xorigin_2ndFi1 = xorigin +(2 * FAMC_MVx)/2
xorigin_2ndFi2 = xorigin +(2 * FAMC_MVx)//2
xorigin_1stFi1 = xorigin +(2 * FAMC_MVx + (M*FAMC_MVx)//8)/2
xorigin_1stFi2 = xorigin +(2 * FAMC_MVx + (M*FAMC_MVx)//8)//2
yorigin_2ndFi  = yorigin + Adjacent_2ndFi_Line_for_2nd_Field_for_Forward (Frame_Distance,
FAMC_MVy)
yorigin_1stFi  = yorigin + Adjacent_1stFi_Line_for_2nd_Field_for_Forward (Frame_Distance,
FAMC_MVy)

for (xl=0; xl<16; ++xl) {
  x_2ndFi1 = xl + xorigin_2ndFi1        /* Addressing X of pixel R    */
  x_2ndFi2 = xl + xorigin_2ndFi2        /* Addressing X of pixel S    */
  x_1stFi1 = xl + xorigin_1stFi1        /* Addressing X of pixel P    */
  x_1stFi2 = xl + xorigin_1stFi2        /* Addressing X of pixel Q    */

                                        /* In case that the required pixel is out of frame  */
  if (x_2ndFi1 < LeftPel)  x_2ndFi1 = LeftPel
  if (x_2ndFi1 > RightPel) x_2ndFi1 = RightPel
  if (x_1stFi1 < LeftPel)  x_1stFi1 = LeftPel
  if (x_1stFi1 > RightPel) x_1stFi1 = RightPel

  for (yl=1; yl<16; yl=yl+2) {
    y_2ndFi = yl + yorigin_2ndFi        /* Addressing Y of R & S pixels */
    y_1stFi = yl + yorigin_1stFi        /* Addressing Y of P & Q pixels */

                                        /* In case that the required pixel is out of frame  */
    if (y_1stFi < Top1stFiLine) y_1stFi = Top1stFiLine
    if (y_1stFi > Bottom1stFiLine) y_1stFi = Bottom1stFiLine
    if (y_2ndFi < Top2ndFiLine) y_2ndFi = Top2ndFiLine
    if (y_2ndFi > Bottom2ndFiLine) y_2ndFi = Bottom2ndFiLine

                                        /* interpolation    */
      ref_2ndFi = ref_frame(x_2ndFi1,y_2ndFi) + ref_frame(x_2ndFi2,y_2ndFi)
      ref_1stFi = ref_frame(x_1stFi1,y_1stFi) + ref_frame(x_1stFi2,y_1stFi)
      FAMC_MB(xl,yl) = (a*ref_2ndFi + b*ref_1stFi)//16
    }
  }
}
```

Backward Motion Compensation is a little different from forward one because the relative position of both field of reference and predicted frame is inversed. That is, in forward prediction, 2nd field of reference is near to predicted frame, but in backward prediction, 1st field of reference is near to predicted one in time domain. Different points from forward are indicated by UNDER LINE.

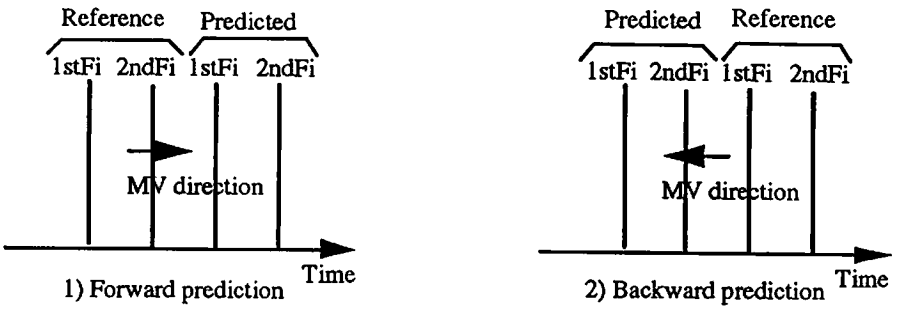


Figure K.3 Difference of forward and backward prediction in FAMC

```
Get_Simplified_FAMC_MB_for_Backward (Frame_Distance, Origin, FAMC_MV, FAMC_MB){
LeftPel = 0, RightPel = 703, Top1stFiLine = 0, Bottom1stFiLine = 478,
Top2ndFiLine = 1, Bottom2ndFiLine = 479
M = 16 // (2*Frame_Distance))          /* M = N * 16; N = 1/(2*Frame_Distance) */
                                       /*   = 8   When Frame_Distance=1 */
                                       /*   = 4   When Frame_Distance=2 */
                                          /*   = 3   When Frame_Distance=3 */
```

/* For first Field */

```
xorigin_1stFi1 = xorigin + (2 * FAMC_MVx)/2
xorigin_1stFi2 = xorigin + (2 * FAMC_MVx)//2
xorigin_2ndFi1 = xorigin + (2*FAMC_MVx + (M*FAMC_MVx)//8)/2
xorigin_2ndFi2 = xorigin + (2*FAMC_MVx + (M*FAMC_MVx)//8)//2
yorigin_1stFi  = yorigin + Adjacent_1stFi_Line_for_1st_Field_for_Backward (Frame_Distance,
FAMC_MVy)
yorigin_2ndFi  = yorigin + Adjacent_2ndFi_Line_for_1st_Field_for_Backward (Frame_Distance,
FAMC_MVy)

for (xl=0; xl<16; ++xl) {
  x_1stFi1 = xl + xorigin_1stFi1       /* Addressing X of pixel P    */
  x_1stFi2 = xl + xorigin_1stFi2       /* Addressing X of pixel Q    */
  x_2ndFi1 = xl + xorigin_2ndFi1       /* Addressing X of pixel R    */
  x_2ndFi2 = xl + xorigin_2ndFi2       /* Addressing X of pixel S    */

                                       /* In case that the required pixel is out of frame  */
  if (x_1stFi1 < LeftPel)  x_1stFi1 = LeftPel
  if (x_1stFi1 > RightPel) x_1stFi1 = RightPel
  if (x_2ndFi1 < LeftPel)  x_2ndFi1 = LeftPel
  if (x_2ndFi1 > RightPel) x_2ndFi1 = RightPel

  for (yl=0; yl<16; yl=yl+2) {
    y_1stFi = yl + yorigin_1stFi       /* Addressing Y of P & Q pixels */
    y_2ndFi = yl + yorigin_2ndFi       /* Addressing Y of R & S pixels */

                                       /* In case that the required pixel is out of frame  */
    if (y_1stFi < Top1stFiLine) y_1stFi = Top1stFiLine
    if (y_1stFi > Bottom1stFiLine) y_1stFi = Bottom1stFiLine
    if (y_2ndFi < Top2ndFiLine) y_2ndFi = Top2ndFiLine
```

```
    if (y_2ndFi > Bottom2ndFiLine) y_2ndFi = Bottom2ndFiLine

                                        /* interpolation   */
      ref_1stFi = ref_frame(x_1stFi1,y_1stFi) + ref_frame(x_1stFi2,y_1stFi)
      ref_2ndFi = ref_frame(x_2ndFi1,y_2ndFi) + ref_frame(x_2ndFi2,y_2ndFi)
      FAMC_MB(xl,yl) = (a*ref_1stFi + b*ref_2ndFi)//16
    }
  }
/* For Second Field */
xorigin_2ndFi1 = xorigin +(2 * FAMC_MVx)/2
xorigin_2ndFi2 = xorigin +(2 * FAMC_MVx)//2
xorigin_1stFi1 = xorigin +(2 * FAMC_MVx - (M*FAMC_MVx)//8)/2
xorigin_1stFi2 = xorigin +(2 * FAMC_MVx - (M*FAMC_MVx)//8)//2
yorigin_2ndFi  = yorigin + Adjacent_2ndFi_Line_for_2nd_Field_for_Backward (Frame_Distance,
FAMC_MVy)
yorigin_1stFi  = yorigin + Adjacent_1stFi_Line_for_2nd_Field_for_Backward (Frame_Distance,
FAMC_MVy)

for (xl=0; xl<16; ++xl) {
  x_2ndFi1 = xl + xorigin_2ndFi1        /* Addressing X of pixel R    */
  x_2ndFi2 = xl + xorigin_2ndFi2        /* Addressing X of pixel S    */
  x_1stFi1 = xl + xorigin_1stFi1        /* Addressing X of pixel P    */
  x_1stFi2 = xl + xorigin_1stFi2        /* Addressing X of pixel Q    */

                                        /* In case that the required pixel is out of frame   */
  if (x_2ndFi1 < LeftPel)  x_2ndFi1 = LeftPel
  if (x_2ndFi1 > RightPel) x_2ndFi1 = RightPel
  if (x_1stFi1 < LeftPel)  x_1stFi1 = LeftPel
  if (x_1stFi1 > RightPel) x_1stFi1 = RightPel

  for (yl=1; yl<16; yl=yl+2) {
    y_2ndFi = yl + yorigin_2ndFi        /* Addressing Y of R & S pixels */
    y_1stFi = yl + yorigin_1stFi        /* Addressing Y of P & Q pixels */

                                        /* In case that the required pixel is out of frame   */
    if (y_1stFi < Top1stFiLine) y_1stFi = Top1stFiLine
    if (y_1stFi > Bottom1stFiLine) y_1stFi = Bottom1stFiLine
    if (y_2ndFi < Top2ndFiLine) y_2ndFi = Top2ndFiLine
    if (y_2ndFi > Bottom2ndFiLine) y_2ndFi = Bottom2ndFiLine

                                        /* interpolation   */
      ref_2ndFi = ref_frame(x_2ndFi1,y_2ndFi) + ref_frame(x_2ndFi2,y_2ndFi)
      ref_1stFi = ref_frame(x_1stFi1,y_1stFi) + ref_frame(x_1stFi2,y_1stFi)
      FAMC_MB(xl,yl) = (a*ref_2ndFi + b*ref_1stFi)//16
    }
  }
}
```

(*Adjacent_XXX_Line_for_XXX_Field_for_XXX*) functions are shown in Table 1.1 to 1.4, and the vertical interpolation coefficients (a,b) are shown in Table 2.1 to 2.2.

Prediction for Chrominance is calculated in the same manner of luminance with replacing the FAMC_MV to the vector which is derived by halving component value of the corresponding MB vector, using the formula from DC 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;
```

An example of FAMC for luminance and chrominance are depicted in Fig. 1. ????

2. Motion Vector Estimation for FAMC

For ME simplification, 2 Step MV search algorithm is used.

1) Step 1 ; Integer pel, 2 line accuracy
In step 1, frame-base ME is performed with 2(V) x 1(H) accuracy.

```
Min_AE = MAXINT
for (j=(-YRange); j<(YRange+1); j+=2) {
  for (i=(-XRange); i<(XRange+1); ++i) {
    Get_Prediction_MB_by_Frame_Prediction (i, j, prediction_mb)
    AE_mb = AE_macroblock (current_mb, prediction_mb)
    if (AE_mb < Min_AE) {
      Min_AE = AE_mb
      FAMC_MV = (i,j)
    }
  }
}
```

Note: YRange used here should be an even number.

2) Step 2 ; Half pel accuracy
In step 2, simplified FAMC based-ME is performed on the twenty neighboring positions which are evaluated the following order;

```
 1  2  3
 4  5  6
 7  8  9
10  0 11
12 13 14
15 16 17
18 19 20
```

where 0 represents the evaluated position in step 1. Min_AE as a result of in Step 1 is used as an initial value in Step 2.

```
for (j=-3; j<4; ++j) {
  for (i=-1; i<2; ++i) {
    Get_Simplified_FAMC_MB_for_xxxx (Frame_Distance, Origin,
          (FAMC_MVx_2int+0.5*i, FAMC_MVy_2int+0.5*j), FAMC_MB)
    AE_famc = AE_macroblock (current_mb, FAMC_MB)
    if (AE_famc < Min_AE) {
      Min_AE = AE_famc
      FAMC_MV = (FAMC_MVx_2int+0.5*i, FAMC_MVy_2int+0.5*j)
    }
```

}
}

where *(FAMC_MV_2int)* represents the motion vector which is detected in step 1 motion estimation stage.

\

Table 1.1
Adj._1stFi_L._for_1st_F._for_Forward
Adj._2ndFi_L._for_2nd_F._for_Backward

FAMC_ MVy	Frame_Distance 1	2	3
0.0	0	0	0
0.5	0	0	0
1.0	0	0	0
1.5	0 + 0	0 + 2	0 + 2
2.0	2	2	2
2.5	4	2	2
3.0	4	2	2
3.5	4	2	2
4.0		4	4
4.5		6	4
5.0		6	4
5.5	4+	6	4
6.0		6	6
6.5		6	8
7.0		8	8
7.5		8	8
8.0			8
8.5			10
9.0			10
9.5	8+	8+	10
10.0			10
10.5			10
11.0			12
11.5			12
12.0			
12.5			
13.0			
13.5	12+		12+
14.0			
14.5			
15.0			
15.5			
16.0			
16.5	REPEATED		
17.0			

Table 1.2
Adj._2ndFi_L._for_1st_F._for_Forward
Adj._1stFi_L._for_2nd_F._for_Backward

FAMC_ MVy	Frame_Distance 1	2	3
0.0	1	1	1
0.5	1	1	1
1.0	1	1	1
1.5	0 + 1	0 + 1	0 + 1
2.0	1	1	1
2.5	1	3	3
3.0	1	3	3
3.5	1	3	3
4.0		3	3
4.5		3	5
5.0		3	5
5.5	2+	3	5
6.0		5	5
6.5		5	5
7.0		5	5
7.5		5	5
8.0			7
8.5			7
9.0			7
9.5	4+	6+	7
10.0			9
10.5			9
11.0			9
11.5			9
12.0			
12.5			
13.0			
13.5	6+		10+
14.0			
14.5			
15.0			
15.5			
16.0			
16.5	REPEATED		
17.0			

Table 1.3

Adj._1stFi_L._for_2nd_F._for_Forward
Adj._2ndFi_L._for_2nd_F._for_Forward
Adj._2ndFi_L._for_1st_F._for_Backward
Adj._1stFi_L._for_1st_F._for_Backward

FAMC_ MVy	Frame_Distance 1	2	3
0.0	1	1	1
0.5	1	1	1
1.0	1	1	1
1.5	0 + 1	0 + 1	0 + 1
2.0	3	3	3
2.5	5	3	3
3.0	5	3	3
3.5	5	3	3
4.0		5	5
4.5		7	5
5.0		7	5
5.5	6+	7	5
6.0		7	7
6.5		9	9
7.0		9	9
7.5		9	9
8.0			9
8.5			11
9.0			11
9.5	12+	10+	11
10.0			11
10.5			13
11.0			13
11.5			13
12.0			
12.5			
13.0			
13.5	18+		14+
14.0			
14.5			
15.0			
15.5			
16.0			
16.5	REPEATED		
17.0			

Table 1.4

FAMC_ MVy	Frame_Distance 1	2	3
0.0	0	0	0
0.5	0	0	0
1.0	2	2	2
1.5	0 + 2	0 + 2	0 + 2
2.0	2	2	2
2.5	2	4	2
3.0	2	4	4
3.5	4	4	4
4.0		4	4
4.5		4	6
5.0		4	6
5.5	4+	4	6
6.0		6	6
6.5		6	6
7.0		6	6
7.5		8	6
8.0		0	8
8.5		0	8
9.0		2	8
9.5	8+	8 + 2	10
10.0		2	10
10.5		4	10
11.0		4	10
11.5		4	12
12.0		4	
12.5		4	
13.0		4	
13.5	12+	4	12+
14.0		6	
14.5		6	
15.0		6	
15.5		8	
16.0			
16.5	REPEATED		
17.0			

*In case of FAMC_MVy < 0, the sign of all figures in Table 1.1 - 1.4 should be changed to minus.

Table 2.1 (a,b)
(yl == 1stFi) in forward prediction
(yl == 2ndFi) in backward prediction

FAMC_ MVy	Frame_Distance 1 a	2 a	3 a
-1.0	3	2	1
-0.5	5	4	4
0.0	8	8	8
0.5	5	4	4
1.0	3	2	1
1.5	1	2	3
2.0	8	8	8
2.5	1	6	5
3.0	3	3	3
3.5	5	2	0
4.0		8	8
4.5	REPEATED	2	6
5.0		3	4
5.5		6	2
6.0		8	8
6.5		2	2
7.0		2	4
7.5		4	6
8.0			8
8.5	REPEATED		0
9.0			3
9.5			5
10.0			8
10.5			3
11.0			1
11.5			4
12.0			
12.5			
13.0			
13.5			
14.0			
14.5			
15.0			
15.5			
16.0			
16.5	REPEATED		
17.0			

Table 2.2 (a,b)
(yl == 2ndFi) in forward prediction
(yl == 1stFi) in backward prediction

FAMC_ MVy	Frame_Distance 1 a	2 a	3 a
-1.0	3	2	1
-0.5	3	3	4
0.0	8	8	8
0.5	3	3	4
1.0	3	2	1
1.5	6	5	5
2.0	8	8	8
2.5	6	1	1
3.0	3	3	3
3.5	3	6	5
4.0		8	8
4.5	REPEATED	6	1
5.0		3	4
5.5		1	6
6.0		8	8
6.5		5	6
7.0		2	4
7.5		3	1
8.0			8
8.5	REPEATED		5
9.0			3
9.5			1
10.0			8
10.5			5
11.0			1
11.5			4
12.0			
12.5			
13.0			
13.5			
14.0			
14.5			
15.0			
15.5			
16.0			
16.5	REPEATED		
17.0			

Note: a+b = 8

*In case of FAMC_MVy < 0, all coefficients (a,b) are cyclically repeated in Table 2.1 and 2.2.

Appendix L: CORE EXPERIMENTS ON PREDICTION MODES

Nine core experiments on prediction modes are listed in this appendix:

1. Simplified FAMC (Matsushita, Philips)
2. SVMC (KDD, TCE)
3. DUAL-PRIME (Dual') (Toshiba, GCT, Mitsubishi)
4. Global Motion Compensation (Matsushita, KDD)
5. Leaky Prediction 1 (JVC, AT&T)
6. Leaky Prediction 2 (CCITT, CCET)
7. Reverse Order Prediction (Columbia Univ., AT&T, Bellcore)
8. Simplification of Test Model (SONY, GCT, Mitsubishi)
9. 6x8 Macroblock (JVC, Columbia Univ., AT&T, Bellcore)

Core Experiment No.1 Simplified FAMC

The specification of simplified FAMC can be found in ANNEX B of MPEG 92/249.

1) The coefficient for interpolation is truncated to 3 bits, namely the multiplication of 1/8.

2) Simplified FAMC is not allowed in averaged MBs in B-pictures.

Simplified FAMC is applied to field structure same as frame structure. Attension should be paid in "field FAMC" with M=1, because the relative parity of the two reference fields are inversed for the first and second field, in the forward prediction direction.

Core Experiment No.2 SVMC (Single Vector Motion Compensation)

SVMC motion estimation is described in **L.2.1.** The sample program is shown in **L.2.2.** The corresponding syntax for SVMC is described in **L.2.3.** The reference and performance of SVMC can be found in MPEG 92/246. Since SVMC already includes frame type prediction as well as field type prediction, such modes as frame and field prediction can be replaced or represented by SVMC which uses only one MV per MB. Although SVMC has four modes in it, three modes except simplified FAMC use the same fractional pel picture, and only the selection of prediction points differs as shown in L.2.1. As for simplified FAMC, interpolation mode in B-picture is excluded and fast search method is used, where first stage of this search method is common for all the four modes.

SVMC is also applicable to field picture by considering only one field(instead of frame) to predict for each time (Figure L2). Although the most part is the same, the complete description of SVMC for field picture will be specified at a later time.

NOTE: This has to be done at the Tarrytown meeting

L.2.1 SVMC motion estimation for frame picture

SVMC motion estimation is performed in two steps.

Step 1 Frame vector search
In this case, frame block search using integer pel accuracy is performed. For vertical direction, block matching is carried out for every other line(..-4,-2,0,+2,+4,..) which enables motion estimation of same parity field. By this line hopping, the amount of calculation is half of that of normal frame block matching in integer pel accuracy.

Step 2 SVMC search for 21 points for each mode

By using the motion vector obtained in the above as an offset, SVMC search is performed for limited search area. The area consists of +/-0.5 for horizontal direction and +/-1.5 for vertical direction which makes total of 21[=3(H) x 7(V)} search points for each mode.
There are four prediction modes in SVMC for frame picture.
a) Simplified FAMC
b) Same parity field MC
c) Near field MC
d) Modified dual field MC
The first one, simplified FAMC(Figure L1-(1)) is used where ME and MC are simplified according to MPEG92/249 and an interpolation mode in B-picture is excluded in order to reduce the memory band width (same condition as Core Experiment No.1 Simplified FAMC).

The second one, same parity field MC is performed by searching motion vector using the same parity field data as shown in Figure L1-(2). In this case, each point in each field is predicted using a point in the same parity field with 1/2 pel accuracy to the horizontal and 1/4 pel to the vertical direction.

The third one, near field MC is performed by searching motion vector using the nearest field in the time domain to the input picture as shown in Figure L1-(3). In this case, each point in both fields is predicted using a point in the nearest field in terms of time axis with 1/2 pel accuracy to the horizontal and 1/4 pel to the vertical direction.

The fourth one, modified dual field MC is performed by using the average data of two fields as shown in Figure L1-(4). In this case, each point in field is predicted using two points in two fields with 1/2 pel accuracy to the horizontal and 1/4 pel to the vertical direction. In this fourth mode, an interpolation mode in B-picture is excluded.

In each mode a) - d), 21 points are searched using above prediction data and best position is searched in terms of MAD. Then mode decision is made by finding the smallest MSE among four modes. When MSE is the same, the priority is given to b),c),a),d) order in the above modes.

Unlike FAMC, modes b) and c) do not require complicated weighting calculation by multiplier using two points, since these two schemes only access fixed points no matter how frame distance value and motion vector may change on condition that pixel data of 1/2 pel accuracy in the horizontal and 1/4 pel in the vertical direction is obtained. Although mode d) requires access of two pixels for simple averaging, there is no need of multiplier.
As for chrominance, half value of motion vector is used in the same way as TM1.

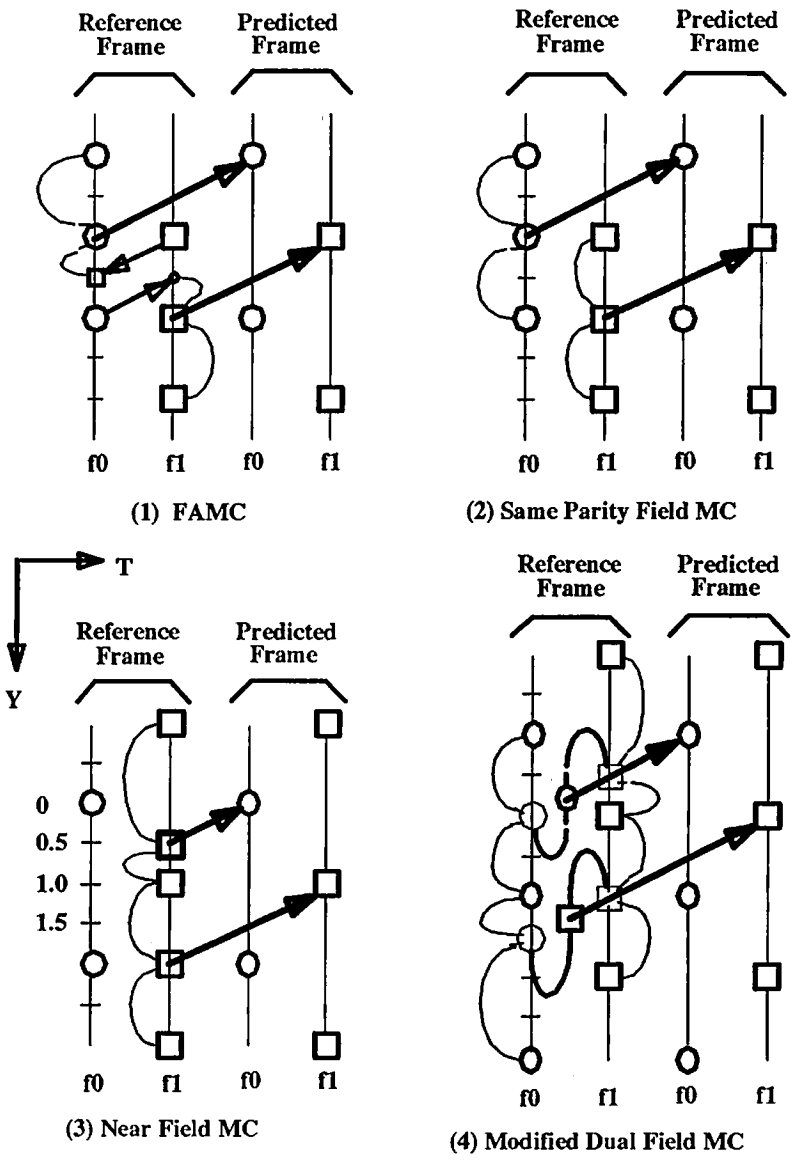


Figure L1 SVMC(Single Vector Motion Compensation) for frame picture

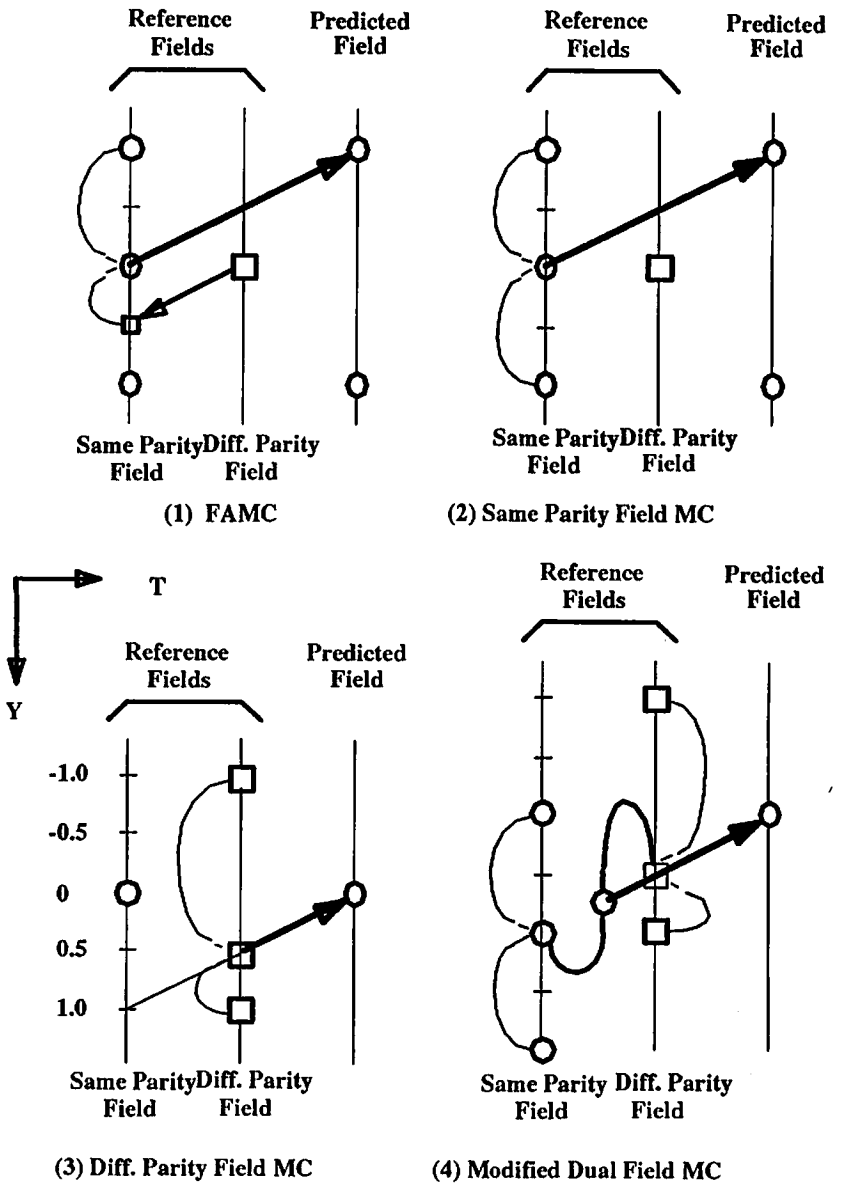


Figure L2 SVMC(Single Vector Motion Compensation) for field picture

L.2.2 SVMC Program (Except FAMC) for frame picture

```
Get_SVMC_MB (mvx,mvy,XSRT,YSRT)
/*      mvx ... -1.5,-1.0,-0.5,0,0.5,1.0,1.5 ...
        mvy ... -1.5,-1.0,-0.5,0,0.5,1.0,1.5 ...
        XSRT ... MB horizontal position
        YSRT ... MB vertical position      */
{
/*      Same_Field_MB ... Same Parity Field Prediction MB
        Diff_Field_MB ... Different Parity Field Prediction MB
        Near_Field_MB ... Near Field Prediction MB
        Dual_Field_MB ... Field Interpolation MB
        EtoE ... Field Distance from Reference Field Even to Predicted Field Even
        EtoO ... Field Distance from Reference Field Odd to Predicted Field Even
```

```
        OtoO ... Field Distance from Reference Field Odd to Predicted Field Odd
        OtoE ... Field Distance from Reference Field Even to Predicted Field Odd*/

        if (Forward Prediction) {
                EtoE = Frame_Distance*2 ;
                EtoO = Frame_Distance*2-1 ;
                OtoO = Frame_Distance*2 ;
                OtoE = Frame_Distance*2+1 ;
        }
        else {
                EtoE = Frame_Distance*2 ;
                EtoO = Frame_Distance*2+1 ;
                OtoO = Frame_Distance*2 ;
                OtoE = Frame_Distance*2-1 ;
        }

/* Positioning (Predicted Field = Even) */
        /* Halfpel horizontal position to Even Field */
        xofs_even1 = (mvx*2)/2 ;
        xofs_even2 = (mvx*2)//2 ;

        /* Quarter pel vertical position to Even Field */
        yofs_even1 = ((mvy*2)/4)*2 ;
        if (mvy < 0) yofs_even2 = (((mvy*2)-3)/4)*2;
        else         yofs_even2 = (((mvy*2)+3)/4)*2;

        /* Halfpel horizontal position to Odd Field */
        xofs_odd1 = ((mvx*2*EtoO)//EtoE)/2 ;
        xofs_odd2 = ((mvx*2*EtoO)//EtoE)//2 ;

        /* Quarter pel vertical position to Odd Field */
        if (mvy < 0) {
                yofs_odd1 = (((mvy*2*EtoO)//EtoE - 2)/4)*2 + 1 ;
                yofs_odd2 = (((mvy*2*EtoO)//EtoE - 5)/4)*2 + 1 ;
        }
        else {
                yofs_odd1 = (((mvy*2*EtoO)//EtoE + 2)/4)*2 - 1 ;
                yofs_odd2 = (((mvy*2*EtoO)//EtoE + 5)/4)*2 - 1 ;
        }

        /* Weighting Parameter for Same Parity Field Prediction */
        if (yofs_even1 == yofs_even2) {
                same_wt1 = 4 ;
                same_wt2 = 0 ;
        }
        else {
                same_wt1 = absolute (yofs_even2*2-(mvy*2)) ;
                same_wt2 = absolute (yofs_even1*2-(mvy*2)) ;
        }

        /* Weighting Parameter for Difference Parity Field Prediction */
        if (yofs_odd1 == yofs_odd2) {
                diff_wt1 = 4 ;
                diff_wt2 = 0 ;
```

```
        }
        else {
                diff_wt1 = absolute (yofs_odd2*2 - (mvy*2*EtoO)//EtoE) ;
                diff_wt2 = absolute (yofs_odd1*2 - (mvy*2*EtoO)//EtoE) ;
        }

/* Prediction main (Predicted Field = Even) */
        for (y = 0;y < 16;y += 2)
                for (x = 0;x < 16;x ++) {
                        /* Even Field to Even Field Prediction */
                        x_even1 = XSRT + x + xofs_even1 ;
                        x_even2 = XSRT + x + xofs_even2 ;
                        y_even1 = YSRT + y + yofs_even1 ;
                        y_even2 = YSRT + y + yofs_even2 ;
                        ref_evenx1 = ref_frame[y_even1][x_even1] + ref_frame[y_even1][x_even2];
                        ref_evenx2 = ref_frame[y_even2][x_even1] + ref_frame[y_even2][x_even2] ;
                        Same_Field_MB[y][x] = (same_wt1*ref_evenx1 + same_wt2*ref_evenx2)//8 ;

                        /* Odd Field to Even Field Prediction */
                        x_odd1 = XSRT + x + xofs_odd1 ;
                        x_odd2 = XSRT + x + xofs_odd2 ;
                        y_odd1 = YSRT + y + yofs_odd1 ;
                        y_odd2 = YSRT + y + yofs_odd2 ;
                        ref_oddx1 = ref_frame[y_odd1][x_odd1] + ref_frame[y_odd1][x_odd2] ;
                        ref_oddx2 = ref_frame[y_odd2][x_odd1] + ref_frame[y_odd2][x_odd2] ;
                        Diff_Field_MB[y][x] = (diff_wt1*ref_oddx1 + diff_wt2*ref_oddx2)//8 ;

                        if (Forward Prediction)   Near_Field_MB[y][x] = Diff_Field_MB[y][x] ;
                        else                      Near_Field_MB[y][x] = Same_Field_MB[y][x] ;
                }

/* Positioning (Predicted Field = Odd) */
        /* Halfpel horizontal position to Odd Field */
        xofs_odd1 = (mvx*2)/2 ;
        xofs_odd2 = (mvx*2)//2 ;

        /* Quarter pel vertical position to Odd Field */
        yofs_odd1 = ((mvy*2)/4)*2 ;
        if (mvy  < 0)  yofs_odd2 = (((mvy*2)-3)/4)*2;
        else       yofs_odd2 = (((mvy*2)+3)/4)*2;

        /* Halfpel horizontal position to Even Field */
        xofs_even1 = ((mvx*2*OtoE)//OtoO)/2 ;
        xofs_even2 = ((mvx*2*OtoE)//OtoO)//2 ;

        /* Quarter pel vertical position to Even Field */
        if (mvy < 0) {
                yofs_even1 = (((mvy*2*OtoE)//OtoO - 2)/4)*2 + 1 ;
                yofs_even2 = (((mvy*2*OtoE)//OtoO - 5)/4)*2 + 1 ;
        }
        else {
                yofs_even1 = (((mvy*2*OtoE)//OtoO + 2)/4)*2 - 1 ;
                yofs_even2 = (((mvy*2*OtoE)//OtoO + 5)/4)*2 - 1 ;
        }
```

```
        /* Weighting Parameter for Same Parity Field Prediction */
        if (yofs_odd1 == yofs_odd2) {
                same_wt1 = 4 ;
                same_wt2 = 0 ;
        }
        else {
                same_wt1 = absolute (yofs_odd2*2-(mvy*2)) ;
                same_wt2 = absolute (yofs_odd1*2-(mvy*2)) ;
        }

        /* Weighting Parameter for Difference Parity Field Prediction */
        if (yofs_even1 == yofs_even2) {
                diff_wt1 = 4 ;
                diff_wt2 = 0 ;
        }
        else {
                diff_wt1 = absolute (yofs_even2*2 - (mvy*2*OtoE)//OtoO) ;
                diff_wt2 = absolute (yofs_even1*2 - (mvy*2*OtoE)//OtoO) ;
        }

/* Prediction main (Predicted Field = Odd)*/
        for (y = 1;y < 16;y += 2)
                for (x = 0;x < 16;x ++) {
                        /* Odd Field to Odd Field Prediction */
                        x_odd1 = XSRT + x + xofs_odd1 ;
                        x_odd2 = XSRT + x + xofs_odd2 ;
                        y_odd1 = YSRT + y + yofs_odd1 ;
                        y_odd2 = YSRT + y + yofs_odd2 ;
                        ref_oddx1 = ref_frame[y_odd1][x_odd1] + ref_frame[y_odd1][x_odd2] ;
                        ref_oddx2 = ref_frame[y_odd2][x_odd1] + ref_frame[y_odd2][x_odd2] ;
                        Same_Field_MB[y][x] = (same_wt1*ref_oddx1 + same_wt2*ref_oddx2)//8 ;

                        /* Even Field to Odd Field Prediction */
                        x_even1 = XSRT + x + xofs_even1 ;
                        x_even2 = XSRT + x + xofs_even2 ;
                        y_even1 = YSRT + y + yofs_even1 ;
                        y_even2 = YSRT + y + yofs_even2 ;
                        ref_evenx1 = ref_frame[y_even1][x_even1] + ref_frame[y_even1][x_even2];
                        ref_evenx2 = ref_frame[y_even2][x_even1] + ref_frame[y_even2][x_even2];
                        Diff_Field_MB[y][x] = (diff_wt1*ref_evenx1 + diff_wt2*ref_evenx2)//8 ;

                        if (Forward Prediction)   Near_Field_MB[y][x] = Same_Field_MB[y][x] ;
                        else                      Near_Field_MB[y][x] = Diff_Field_MB[y][x] ;
                }

/* Dual Field Prediction */
        for (y = 0;y < 16;y ++)
                for (x = 0;x < 16;x ++)
                        Dual_Field_MB[y][x] = (Same_Field_MB[y][x]
                                                + Diff_Field_MB[y][x])//2 ;
}
```

L.2.3 Syntax change corresponding to SVMC

Macroblock Layer

```
macroblock() {
            .
            .
            .
    if (interlaced) {  /* not MPEG-1 syntax */
      if ( macroblock_motion_forward ||
                    macroblock_motion_backward ) {
        if ( picture_structure == "11" ) { /* Frame-Picture */
          frame_motion_type                          2          uimsbf
          if ( frame_motion_type == "11" ) {
            if ( macroblock_motion_forward &&
                        macroblock_motion_backward )
              SVMC_type1                             2          uimsbf
            else
              SVMC_type2                             2          uimsbf
          }
        } else {
          field_motion_type                          2          uimsbf
          if ( field_motion_type == "11" ) {
            if ( macroblock_motion_forward &&
                        macroblock_motion_backward )
              SVMC_type3                             2          uimsbf
            else
              SVMC_type4                             2          uimsbf
          }
        }
      }
      if (( picture_structure == "11" ) /* Frame-Picture */
          && ( macroblock_intra || macroblock_pattern ))
        dct_type                                     1          uimsbf
    }
            .
            .
            .
  }
```

SVMC_type1 - this is two-bit integer indicating the macroblock motion prediction when bidirectional mode for frame-picture, defined in the following table:

binary value	Motion Prediction
00	Forward = Same, Backward = Same Field Prediction
01	Forward = Same, Backward = Near Field Prediction
10	Forward = Near, Backward = Same Field Prediction
11	Forward = Near, Backward = Near Field Prediction

SVMC_type2 - this is two-bit integer indicating the macroblock motion prediction when unidirectional mode for frame-picture, defined in the following table:

binary value	Motion Prediction
00	Same Parity Field Prediction
01	Near Field Prediction
10	Simplified FAMC
11	Modified Dual Field Prediction

SVMC_type3 - this is two-bit integer indicating the macroblock motion prediction when bidirectional mode for field-picture, defined in the following table:

binary value	Motion Prediction
00	Forward = Same, Backward = Same Field Prediction
01	Forward = Same, Backward = Diff. Field Prediction
10	Forward = Diff., Backward = Same Field Prediction
11	Forward = Diff., Backward = Diff. Field Prediction

SVMC_type4 - this is two-bit integer indicating the macroblock motion prediction when unidirectional mode for field-picture, defined in the following table:

binary value	Motion Prediction
00	Same Parity Field Prediction
01	Diff. Parity Field Prediction
10	Simplified FAMC
11	Modified Dual Field Prediction

field_motion_type- this is a 2-bit code indicating the macroblock motion prediction, defined in the following table:

code	prediction type	**motion vector count**	**mv format**
11	CORE EXPERIMENTS(SVMC)	1	frame

frame_motion_type- this is a 2-bit code indicating the macroblock motion prediction, defined in the following table:

code	prediction type	**motion vector count**	**mv format**
11	CORE EXPERIMENTS(SVMC)	1	frame

If you have any questions or request on MV, please contact to nakajima@spg.elb.kddlabs.co.jp or Y.Nakajima, KDD R&D Labs., tel +81(492)66-7891, fax +81(492)66-7510

Core Experiment No.3 DUAL-PRIME (Dual')

DUAL-PRIME prediction is an improved version of dual field prediction defined in TM1, reducing overhead for transmitting motion vectors. It needs only one field motion vector and one additional very small differential vector per macroblock. Its operation is considered to be both improvement of vertical resolution and adaptive spatiotemporal loop filtering. No pel by pel basis multiplication is needed and its decoder memory bandwidth is one-half that for original FAMC.

All the information for DUAL-PRIME is described in MPEG92/259, according to the syntax and the field motion vector transmission method of TM1. This document is the revised version of the syntax, decoding

process and motion vector estimation method for DUAL-PRIME, according to the syntax and the field motion vector transmission method of TM2 with extension to the field-picture.

1. Syntax Extension for Dual' in TM2

```
forward_motion_vectors() {
    if (motion_vector_count == 1) {
        if (mv_format == frame) {
            forward_motion_vector()                ...      ...
        } else {
            forward_field_motion_vector()          ...      ...
            if (frame_motion_type == '11'
             || field_motion_type == '11')
              dmv()                                ...      ...
        }
    } else {
        forward_field_motion_vector_1()            ...      ...
        forward_field_motion_vector_2()            ...      ...
    }
}
```

```
backward_motion_vectors() {
    if (motion_vector_count == 1) {
        if (mv_format == frame) {
            backward_motion_vector()               ...      ...
        } else {
            backward_field_motion_vector()         ...      ...
            if (frame_motion_type == '11'
             || field_motion_type == '11')
              dmv()                                ...      ...
        }
    } else {
        backward_field_motion_vector_1()           ...      ...
        backward_field_motion_vector_2()           ...      ...
    }
}
```

```
dmv() {
    dmv_horizontal                                 1-2      vlclbf
    dmv_vertical                                   1-2      vlclbf
}
```

```
forward_field_motion_vector() {
    if (forward_reference_fields == '11'
     && !(frame_motion_type == '11' ||
          field_motion_type == '11'))
        motion_vertical_field_select               1        uimsbf
    }
```

```
|     motion_vector()                              ...      ...     |
| }                                                                  |
---------------------------------------------------------------------


---------------------------------------------------------------------
| backward_field_motion_vector() {                                   |
I     if (backward_reference_fields == '11'                          I
I      && !(frame_motion_type == '11' ||                             I
I           field_motion_type == '11'))                              I
|         motion_vertical_field_select       1       uimsbf        |
|     }                                                              |
|     motion_vector()                              ...      ...     |
| }                                                                  |
---------------------------------------------------------------------
```

field_motion_type -- This is a 2-bit code indicating the macroblock motion prediction, defined in the following table:

code	prediction type	motion_vector_count	mv_format
00	Field-based prediction	1	field
01	Dual-field prediction	2	field
10	Simplified FAMC	1	frame
11	Core Experiments (Dual')	1	field

frame_motion_type -- This is a 2-bit code indicating the macroblock motion prediction, defined in the following table:

code	prediction type	motion_vector_count	mv_format
00	Field-based prediction	2	field
01	Frame-based prediction	1	frame
10	Simplified FAMC	1	frame
11	Core Experiments (Dual')	1	field

VLC table for dmv_horizontal/vertical

dmv_horizontal/vertical	value
0	0
10	1
11	-1

2. Decoding process of motion vectors

In dual' field based forward prediction, the following procedure is used to reconstruct the motion vectors. In backward prediction, it is also valid except for replacing the term 'for' to 'back'. If field_motion_type is dual' ('11') or frame_motion_type is dual' ('11'), foward_field_motion_vector() and dmv() are transmitted.

Let recon_{right/down}_for means the horizontal/vertical component of MV_for, which is reconstructed from foward_field_motion_vector(). Let dmv_{horizontal/vertical}_for means the horizontal/vertical component of DMV_for, which is reconstructed from dmv().

Let recon_{right/down}_for_OtoO, recon_{right/down}_for_EtoO, recon_{right/down}_for_EtoE and recon_{right/down}_for_OtoE mean the horizontal/vertical component of MV_for_OtoO, MV_for_EtoO, MV_for_EtoE and MV_for_OtoE.

Let dist_OtoO, dist_EtoO, dist_EtoE and dist_OtoE be the field distance between the {odd(#1)/even(#2)} reference field and the {odd(#1)/even(#2)} field to be decoded. Note that dist_OtoO and dist_EtoE is equal to the value 2*Frame_distance in FAMC description.

```
/* Substitute the decoded motion vector for the motion
   vector of same parity */
recon_right_for_OtoO = recon_right_for_EtoE = recon_right_for;
recon_down_for_OtoO = recon_down_for_EtoE = recon_down_for;

/* Derive the cross point of MV_for_OtoO in the even reference field,
    truncate it to the nearest one-half pixel unit point toward zero,
    add DMV_for to get MV_for_EtoO
    and express it in the opposite parity field coordinate */
recon_right_for_EtoO = ((recon_right_for_OtoO * dist_EtoO) / dist_OtoO)
                     + dmv_horizontal_forward;
recon_down_for_EtoO = ((recon_down_for_OtoO * dist_EtoO) / dist_OtoO)
                    + dmv_vertical_forward - 1;

/* Derive the cross point of MV_for_EtoE in the odd reference field,
    truncate it to the nearest one-half pixel unit point toward zero,
    add DMV_for to get MV_for_OtoE
    and express it in the opposite parity field coordinate */
recon_right_for_OtoE = ((recon_right_for_EtoE * dist_OtoE) / dist_EtoE)
                     + dmv_horizontal_forward;
recon_down_for_OtoE = ((recon_down_for_EtoE * dist_OtoE) / dist_EtoE)
                    + dmv_vertical_forward + 1;
```

For a Frame-Picture (picture_structure == '11'), all of the procedure mentioned above is executed to reconstruct the motion vectors. For a Field-Picture (picture_structure == '10' or picture_structure == '01'), part of the procedure mentioned above is executed to reconstruct the motion vectors. That is, if the parity of the field to be decoded is odd (picture_structure == '10'), the procedure related with *_OtoO and *_EtoO should be executed, and if the parity of the field to be decoded is even (picture_structure == '01'), the procedure related with *_EtoE and *OtoE should be executed.

3. Motion vector estimation process

At the first step, org_MV_for_{O/E}to{O/E}, which is the field motion vector from the odd/even field to be referenced, to the odd/even field to be coded, is derived, according to the field motion vector estimation process of this TM with half pel search using original pictures. The second step is as follows:

```
min_AE = MAXINT;
foreach i (OtoO EtoO EtoE OtoE) {
     /* $i may be OtoO, EtoO, EtoE or OtoE (4 candidates) */
     if ($i == 'OtoO' || $i == 'EtoE') {
          /* org_MV_$i is the motion vector for the field of same parity */
          recon_right_for = recon_right_for_OtoO
                          = recon_right_for_EtoE = org_recon_right_for_$i;
          recon_down_for = recon_down_for_OtoO
```

```
                    = recon_down_for_EtoE = org_recon_down_for_$i;
}
else {
     /* org_MV_$i is the motion vector for the field
        of opposite parity */
     recon_right_for = recon_right_for_0to0 = recon_right_for_EtoE
                     = (org_recon_right_for_$i * dist_0to0) / dist_$i;

     /* If ($i == 'Eto0'), the vertical component of the motion vector
        should be incremented by one to be expressed in the same
        parity field coordinate.
        If ($i == '0toE'), the vertical component of the motion vector
        should be decremented by one to be expressed in the same
        parity field coordinate. */
     recon_down_for = recon_down_for_0to0 = recon_down_for_EtoE
                    = ((org_recon_down_for_$i + ($i == 'Eto0') * 1
                    + ($i == '0toE') * (-1)) * dist_0to0) / dist_$i;
}

/* Cut the local decoded reference fields of the same parity
   according to the motion vectors. */
cut_ref(recon_right_for_0to0, recon_down_for_0to0, 0dd_reference,
        lines_of_0dd_field_of_MB1, MBadrs);
cut_ref(recon_right_for_EtoE, recon_down_for_EtoE, Even_reference,
        lines_of_Even_field_of_MB1, MBadrs);

for (dmv_vertical_forward = -1;dmv_vertical_forward<=1;
                              dmv_vertical_forward++) {
     for (dmv_horizontal_forward = -1;dmv_horizontal_forward<=1;
                                   dmv_horizontal_forward++) {
          /* DMV's are prepared (9 candidates) */

          /* Derive MV_for_Eto0 from MV_for_0to0.
             The vertical component should be decremented by one,
             according to the TM2 motion vector definition. */
          recon_right_for_Eto0 = ((recon_right_for_0to0 * dist_Eto0)
                               / dist_0to0) + dmv_horizontal_forward;
          recon_down_for_Eto0 = ((recon_down_for_0to0 * dist_Eto0)
                              / dist_0to0)  + dmv_vertical_forward - 1;

          /* Derive MV_for_0toE from MV_for_EtoE.
             The vertical component should be incremented by one,
             according to the TM2 motion vector definition. */
          recon_right_for_0toE = ((recon_right_for_EtoE * dist_0toE)
                               / dist_EtoE) + dmv_horizontal_forward;
          recon_down_for_0toE = ((recon_down_for_EtoE * dist_0toE)
                              / dist_EtoE) + dmv_vertical_forward + 1;

          /* Cut the local decoded reference fields of the opposite
             parity according to the motion vectors. */
          cut_ref(recon_right_for_Eto0, recon_down_for_Eto0,
                  Even_reference, lines_of_0dd_field_of_MB2, MBadrs);
          cut_ref(recon_right_for_0toE, recon_down_for_0toE,
                  0dd_reference, lines_of_Even_field_of_MB2, MBadrs);

          /* Interpole the fields of both parity */
          interpole(lines_of_0dd_field_of_MB1,
                    lines_of_0dd_field_of_MB2,
                    lines_of_0dd_field_of_prediction_MB);
          interpole(lines_of_Even_field_of_MB1,
                    lines_of_Even_field_of_MB2,
                    lines_of_Even_field_of_prediction_MB);

          /* Calculate the absolute error */
          AE_0dd = AE_Macroblock(lines_of_0dd_field_of_prediction_MB,
                                 lines_of_0dd_field_of_current_MB);
          AE_Even = AE_Macroblock(lines_of_Even_field_of_prediction_MB,
```

```
                                        lines_of_Even_field_of_current_MB);
                AE = AE_Odd + AE_Even;
                if (AE < min_AE) {
                     min_AE = AE;
                     MV_for = (recon_right_for, recon_down_for);
                     DMV_for = (dmv_horizontal_forward, dmv_vertical_forward);
                }
           }
      }
}
```

cut_ref(recon_right, recon_down, Reference, Buf, MBadrs): The reference signals pointed by recon_{right/down}_for and MBadrs in field buffer "Reference" are copied to macroblock buffer "Buf".
interpole(Buf1, Buf2, Buf): Average signals of buffer "Buf1" and "Buf2" are stored in buffer "Buf".

For a Frame-Picture, all of the procedure mentioned above is executed to reconstruct the motion vectors (number of candidates are 4*9 = 36). Note that MB1, MB2, prediction_MB and current_MB are frame macroblocks, and odd/even lines in them are in the odd/even field.

For a Field-Picture, part of the procedure mentioned above is executed to reconstruct the motion vectors. If the parity of the field to be coded is odd, the procedure related with *_OtoO and *_EtoO should be executed (number of candidates are 2*9 = 18). Note that MB1, MB2, prediction_MB and current_MB are field macroblocks, and all lines in them are in the odd field. If the parity of the field to be coded is even, the procedure related with *_EtoE and *OtoE should be executed (number of candidates are 2*9 = 18). Note that MB1, MB2, prediction_MB and current_MB are field macroblocks, and all lines in them are in the even field.

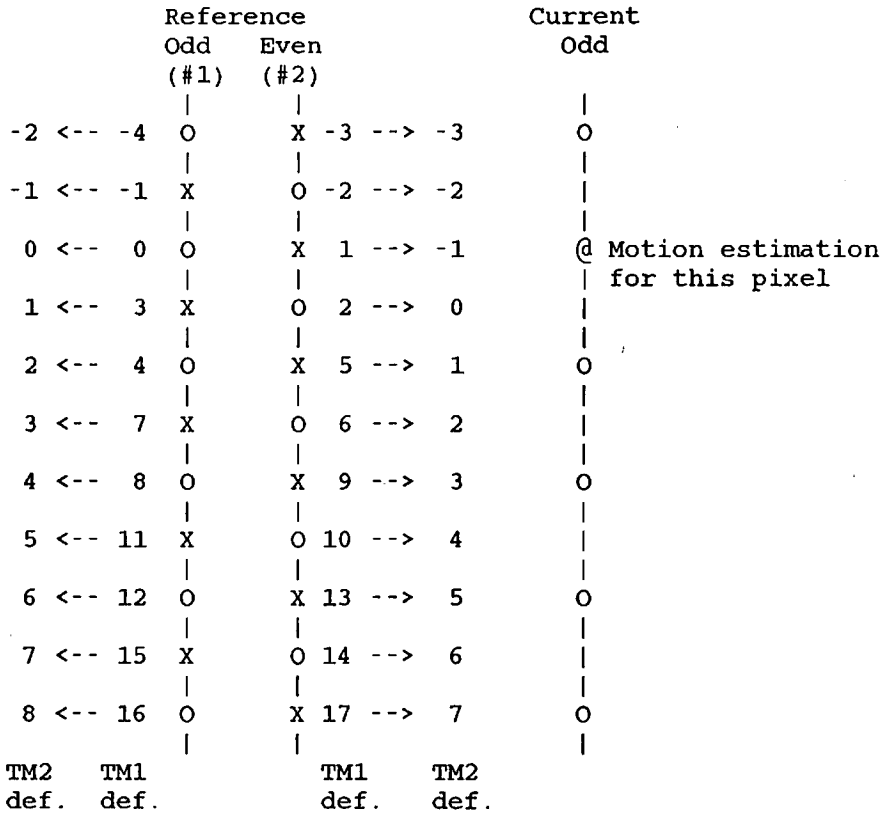
Note:
1) In motion vector estimation process, if the motion vector, obtained by the extension of the vector and summation of dmv, points out of the picture, the candidate is not used.
2) In both motion vector estimation process and decoding process of motion vectors, if the value of the half pixel point located in the edge of the picture must be used, the value of the integer pixel point next to it is used.

If you have some questions, please contact to: odaka@eel.rdc.toshiba.co.jp
or T.Odaka, Toshiba R&D center, fax +81(44)549-2276

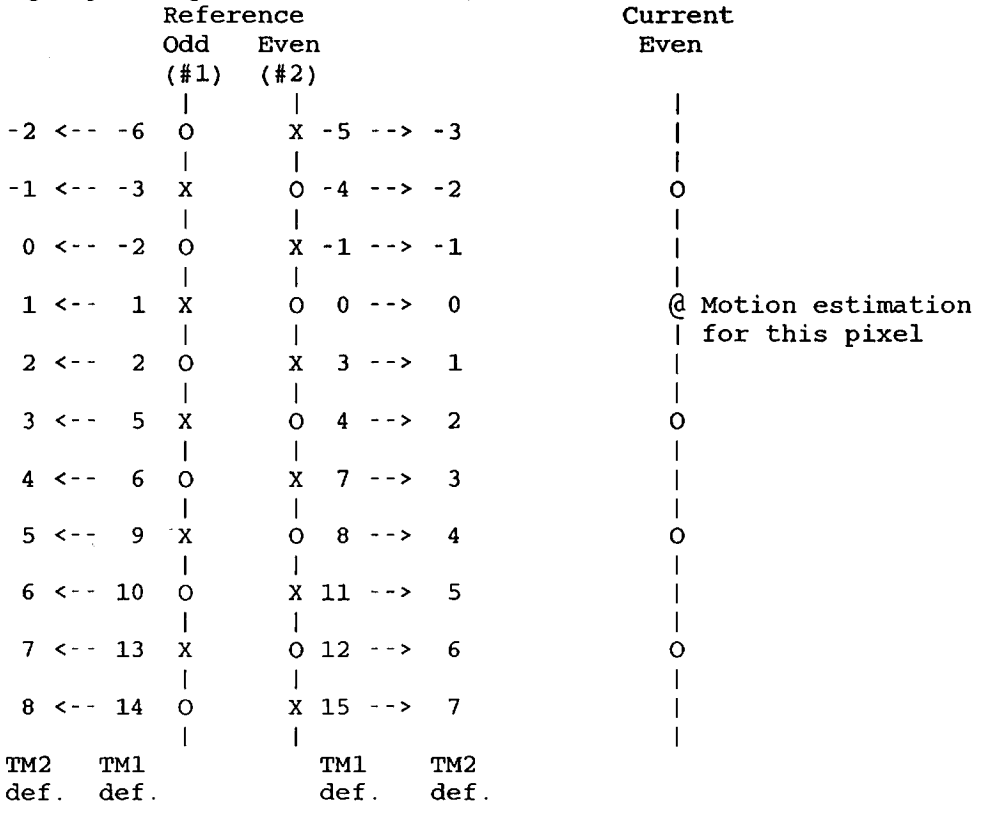
4. Supplimentary explanation for Figure A-1 in MPEG92/259.

The notation '(TM1 def.) --> (TM2 def.)' means the replacement of motion vector expressed in TM1 definition with motion vector expressed in this TM definition in half-pel unit.

[Lower left figure of Figure A-1 in MPEG92/259]



[Lower right figure of Figure A-1 in MPEG92/259]



Core Experiment No.4 Global Motion Compensation

Global motion compensation is applied to I-pictures coded in frame mode. It is a kind of pre-processing that shifts the relative position between the two fields in an I-picture based on a pair of global MV. After reconstruction, the relative position between the two fields has to be shifted back to the original position for display (at decoder side) or as the reference picture (at encoder side).

1. Global Motion Estimation

1) Global MV is determined from the luminance pixels only.
2) Fields 1 and 2 of an I-picture are splitted. Field 2 is fixed while field 1 becomes the reference field.
3) The motions between fields 1 and 2 of an I-picture is estimated in full pel precision horizontally and half pel precision vertically.Thus, the reference field is composed of field 1 plus its vertical interpolation.
4) A pair of motion vector is determined for each macroblock by full search block matching method under the minimum absolute error criterion. The block size in each field is 16(H)x8(V). The difference is always between fields 2 and 1, or between field 2 and the interpolated lines of the reference field.
5) The search area is set to half of the search area for inter-frame motion vectors as the global motion between two fields cannot be larger than one half of the motions between two frames. Thus, for an inter-frame motion search range of +- 15 pels, the inter-field motion search range is +- 7 pels.
6) With the MV's obtained above, a pair of global MV is determined by taking the average of the components which occupy more than a certain percentage (majority) of all the MV's. This is calculated separately for the x and y components.
7) For the y component, GlobalMV_y is modified as follows to avoid breaking down the interlaced structure of the original picture.:

```
if (GlobalMV_y == negative odd number)
        GlobalMV_y = GlobalMV_y + 1;
if (GlobalMV_y == positive odd number)
        GlobalMV_y = GlobalMV_y - 1;
if (GlobalMV_y == even number)
        GlobalMV_y = GlobalMV_y;
```

2 Global Motion Compensation

1) Global motion compensation is applied to the luminance of I-pictures only.
2) Based on the pair of global MV obtained, field 1 of an I-picture is shifted in full pel precision both horizontally and vertically.
3) If meaningful pixels cover the whole 720 pels of the lines and the horizontal global motion compensation is performed before the cropping. If meaningful pixels cover exactly the 704 pel area, every other lines at one side of the borders will be left with no meaningful pixels after the compensation. One can fill up the vacancies by repeating the nearest pixel in each line.
4) Every other lines at one side of the horizontal borders will be left with no meaningful lines after the vertical global motion compensation.The values of the nearest line can be repeated to fill up the vacancies.
5) The global-motion-compensated I-picture is then encoded as usual.
6) The pair of global MV is included in the bitstream as describe in the next section.
7) At the decoder side, the global MV is used to shift the field back to its original position after decoding and reconstruction. The vertical and horizontal borders are dealt with one of the methods described in 3) and 4), respectively.

3. Modification of the Bitstream Syntax

The global motion vector (GlobalMV_x, GlobalMV_y) is appended to the Picture Layer. These parameters are integers, range from -7 through +7 or -15 through +15. They are offset by 15 to produce unsigned integers. In the modified syntax, the global_motion_flag is a one bit integer. It is set to 1 when there is global motion, and to 0 if (GlobalMV_x, GlobalMV_y) = (0, 0).

if (nextbits() == extension_start_code) {		
extension_start_code	**32**	**bslbf**
picture_structure	**2**	**uimsbf**
forward_reference_fields	**2**	**uimsbf**
backward_reference_fields	**2**	**uimsbf**
global_motion_flag	**1**	**uimsbf**
if (global_motion_flag==1){		
GlobalMV_x	**5**	**uimsbf**
GlobalMV_y	**5**	**uimsbf**
}		
.....		
}		

If you have some questions, please contact to :
toshi@drl.mei.co.jp
or Toshiya Takahashi, Matsushita,
FAX +81-6-949-2218.

Core Experiment No.5 Leaky Prediction 1

Purpose: To verify the coding efficiency , improvement of visual quality and accessibility
Definition ;
Leaky Prediction 1 (Leaked MC filter)
1. Rational leak prediction for high spatial frequency (stable loop filter) is used.
2. Apply P-prediction and B-prediction with out FAMC
3. Modify the coefficients of half-pel filter in MC

type 1 (non-DC leak)
b' = (a +14b + c) / 16
e = (- a +9b +9c - d) / 16

a x b x e + c x d x
pel allocation (vertically or horizontally)

type 2 (DC leak)
b' = (a +12b + c) / 16
i = (- a +8b +8c - d) / 16

a , b c , d = reference pels
b' , e = predictive pels

Core Experiments
type 1 (non-DC leak) / type 2 (DC-leak)
Change Horizontal only / Horizontal and Vertical
Syntax
In the sequence layer, one bit flag may be defined to indicate as follow.
"0" : normal prediction
"1" : leaked prediction

Reference
MPEG 92 / 261
If any question , contact ;

Kenji Sugiyama Digital Technologies Research Dept. Central R&D Center JVC
58-7, Shinmei-cho , Yokosuka, Kanagawa 239, JAPAN
e-mail : k-sgym@krhm.jvc-victor.co.jp Tel : +81-468-36-9275 Fax : +81-468-36-8540

Core Experiment No.6 Leaky Prediction 2

The syntax used for transmitting the leak_factor is described precisely in MPEG 92/340.

A 3-bit codeword is sent in the header of each P-picture. This codeword ranges from "001" to "111" ("000" is prohibited). This codeword is mapped to an integer n by natural binary mapping (uimsbf) except for "111" where n is infinite.

The leak_factor LF is obtained as 1 - 1 / 2 ** n. It is equal to 1 for infinite n (no leaky prediction).

The leaky prediction is obtained as LF * (pred_pixel - 128) + 128
where pred_pixel is the prediction pixel obtained by any prediction method.

As suggested by the CCITT Experts Group, the multiplication in the above equation is implemented in ***floating point*** followed by ***truncation towards 128 *** in 8 bits.

An algorithm for computing the leak_factor LF is given in MPEG 92/340.
This method computes the correlation between the the current picture and the (non-leaky) motion-compensated prediction from the previous I/P picture. This correlation is quantized to the nearest allowed leak_factor.

EXPERIMENTS:

1. Compare the performance of M=3, N=12 with fixed LF=0.875 or variable LF , M=3, and no I-pictures (after the first).

2. Simulate channel errors or cell losses and demonstrate resiliency.

3..Check the capability of channel tune-in for M=1 and M=3.

In the above experiments, the quantizer matrices are the default matrices in the TM.
If the matrices are downloaded in the picture layer, this should be stated

Core Experiment No.7 Reverse Order Prediction

(Option to use reverse field order in field structure coding)

In field-structure syntax, each field is preceded by a picture header. The order of field processing may be reversed using the two bits of picture-structure:

11 frame picture
10 field 1 picture
01 field 2 picture
00 reserved

Since "B" frame s are not affected, the improvement of reversing field order is more evident in low-delay mode (M=1).

This experiment compares (a) mormal field order coding and (b) reverse field order coding. CCIR-601 fields are coded using MPEG2 field structure coding at 4 Mbit/s, using adaptive single/dual-field predictions with M=1, N=15 (30Hz) or N=12 (25Hz). In (b), the I and P-picrures are reversed in field processing using the above table, resulting in improved performance in sequences with uncovered background.

(a) Io -> Po -> Po ->
\ / \ / \ NOTE ; UPWARD AND DOWNWARD ARROWS
Pe -> Pe -> Pe -> SHOULD BE WRITTEN !!!!

(b) Po -> Po -> Po ->
\ \ \ NOTE ; UPWARD&&BACKWARD AND
Ie -> Pe -> Pe -> DOWNWARD&&LOWSLOPE ARROWS
SHOULD BE WRITTEN !!!!

---------->
display time

Core Experiment No.8 Simplification of Test Model

Introduction

Test Model 1 (TM1) : Frame/Field Adaptive Coding has two problems. One is algorithm complexity, because TM1 has adaptations of Frame MC/Field MC and Frame DCT/Field DCT. The other is the amount of memories in hardware implementation. TM1 coding method needs more memories, e.g. compared with Frame Base Field Structure Coding.To solve these problems, this document proposes the simplification of Test Model 1 : Frame/Field Adaptive Coding.

Proposed Method

Proposed Method 1
Test Model 1 : Frame/Field Adaptive Coding has 4 modes. They are as follows :

1. Frame MC - Frame DCT
2. Frame MC - Field DCT
3. Field MC - Frame DCT
4. Field MC - Field DCT

In mode 2 and 3, macroblock reordering is needed between Motion Compensation and DCT, and this increases hardware complexity of Frame/Field Adaptive Coding Method. To solve this problem, the simplified method that inhibits mode 2 and 3, i.e. restricts mode 1 and 4 only is investigated. This simplified method decides DCT mode by means of MC mode in non-intra MB. For example, if MC mode is decided to Frame Mode by comparing prediction errors, DCT mode is decided Frame Mode automatically. This simplified method saves not only Macroblock reordering after Motion Compensation but also the process of DCT mode decision in the encoder. Moreover TM1 has two bits(flags) to inform MC and DCT modes, but this simplified method needs only one bit(flag) to inform MC/DCT mode. In intra MB the DCT mode is decided by the same method as TM1.

Another Reason for Proposed Method 1

The combination of Field MC and Frame DCT mode shows poor performance at core experiments of Scalability. Therefore the combination of Field MC and Frame DCT mode is not recommended by the requirement of Scalability.

Proposed Method 2

The second proposed method fixes Picture Structure of B-Pictures to Frame Base Field Structure. By this method, frame memory for the B-Pictures becomes needless because B-Pictures can be displayed as soon as it is decoded.

The problem of this method is that the matching between the time the decoder needs to decode 1 slice and the time the decoder needs to display 1 slice becomes critical. However, This problem is also occurred in the TM1 if TM1 has only 1 field memory for B-Pictures.

Proposed Method 3

This method is a combination of Proposed Method 1 and Proposed Method 2. In P-Pictures, Proposed Method 1 is used. In B-Pictures, Proposed Method 2 is used. This method doesn't need Macroblock reordering, nor does it need frame memory for the B-Pictures.

Simulation Conditions

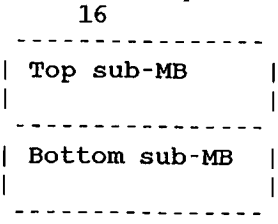
Simulation conditions are as follows, and other conditions are the same as Test Model 1.

GOP & Prediction : M=3, N=12
Sequence : Flower Garden, Mobile & Calendar, CheerLeaders, Bicycle, Football, Ftball, TableTennis
Bitrate : 4Mbps
Rate Control : Test Model 2

Core Experiment No.9 16x8 Sub-Macroblock MC

1. Purpose: To evaluate the coding efficiency and improvement of visual quality

2. Definition of sub-macroblock
   In case of field coding, 16x16 macroblock may not be correct to apply the same MC operation, then 16x16 macroblock may be coded as two 16x8 sub-macrobloks with different MC operation.



3. Description of sub-MB motion compensation
(1) Different motion vector(s) for each 16x8 sub-MB while same MB-type

- decision-criteria-1: whether one 16x16 or two 16x8

```
If(MV_of_top_sub-MB == MV_of_bottom_sub_MB)
  do  (16x16 MC)
  else
     do  (two 16x8 MC)
```

- decision-criteria-2: whether one 16x16 or two 16x8

```
If((16x16 MC_MSE) < (16x8 MC_MSE_sum x1.125))
  do  (16x16 MC)
  else
     do  (two 16x8 MC)
```

where "16x8_MC_MSE_sum" refers to sum of 16x8_MC_MSE of "Top" and "Bottom" sub MB's.

(2) Different MC operation for each 16x8 sub-MB while MVs limited by 2 per this MB

- MVs is limited by two then in case of two 16x8 sub-MB, it can be MC operated as follows.
  - The meaning of "forward-MC-type" will be changed to "forward-MC-type for top-sub-MB" and "backward-MC-type for bottom-sub-MC"
  - The meaning of "backward-MC-type" will be changed to "backward-MC-type for top-sub-MB" and "forward-MC-type for bottom-sub-MC"
  - The meaning of "interpolated-MC-type" will be changed to "both forward-MC-type" with different MV for each sub-MB.
- The decision criteria is the MSE criteria.

4. Syntax change

- In the picture layer, one bit flag will be added to indicate whether 3.(1) or 3.(2) sub-MB scheme will be applied when indicated by "sub-MB-flag in MB layer.
- In the MB layer one bit flag of "sub-MB-flag" will be added just after MB-type field to know whether normal MB or two sub-MBs.

If any question, contact;

Kenji Sugiyama / JVC

or

Atul Puri / AT&T

Core Experiment No.10 Special Prediction Modes

L.10.1. Definitions

SVMC3 is SVMC without FAMC. The latest document is used (TM-2 Erratum), with the modifications and corrections described in this document.

L.10.2. Temporal Scaling of the Motion Vector

Scaling of the motion vector is done in the same manner for all the special prediction modes (FAMC, SVMC3 and DUAL-PRIME).
The transmitted motion vector (x, y) corresponds to a prediction from same-parity field.
The horizontal coordinate is in 1/2-pel units. The vertical coordinate is in 1/2-pel units or 1/4-pel units (depending on the mode and of the experiment performed).
If the same parity reference frame is at a distance of 2*k fields from the predicted field, the coordinates (x', y') of the "scaled-motion-vector" used for accessing the different-parity field is computed as follows :

x' = (x * m * K) // 32 (x and x' are integers)
y' = ((y * m * K) // 32) + e (y and y' are integers)
K = 16 // k (k is integer)
m = field-distance between the predicted field and the different-parity-field (m is integer and can be negative).
The "e" is an adjustment necessary to reflect the vertical shift between the lines of field 1 and field 2. To give an example, line 1 of field 2 is in fact located 1/2 line under line 1 of field 1.
If vertical unit is 1/4-pel, "e" is defined as follows :
e = -2 if the reference field corresponding to the scaled vector is field 2
e = +2 if the reference field corresponding to the scaled vector is field 1
If vertical unit is 1/2-pel, "e" is defined as follows :
e = -1 if the reference field corresponding to the scaled vector is field 2
e = +1 if the reference field corresponding to the scaled vector is field 1

L.10.3. Reference Fields for SVMC3 and DUAL-PRIME

The reference fields used for SVMC3 and DUAL-PRIME are not always contiguous in time. Those modes can now be used in all cases of Field-structure P-Pictures.
When SVMC3 or DUAL-PRIME is used in the second P-Field of a P-Picture, the first P-Field is used as a reference (different-parity) field.
SVMC3 and DUAL-PRIME can be used with reversed order prediction of P-Fields (in this case, m = -1).

L.10.4. Decision for Field-based Prediction

In order to take advantage of the various special prediction modes, the decision rule must be modified for Field-based prediction.
It has been noted by various members that quality is improved by choosing Field-based prediction less often, to the benefit of another special prediction mode, particularly in B-Pictures.
For example, even in cases where Field-based prediction has an MSE slightly better than any of the other prediction modes, it may cost a significant overhead to transmit two field-vectors (four in B-Frames).
Until further improvement, we propose to use the following decision rule in core experiments involving one of the special prediction modes :

- Field-based chosen
  if MSE_field + 8 < MSE_best_of_other_modes in B-pictures
  if MSE_field < MSE_best_of_other_modes in P-pictures

where MSE = Mean Square Error PER PEL of predicted MB

L.10.5. Concise Specification of SVMC3

The transmitted motion vector is scaled with the specified rule to obtain motion vectors origination from each of the reference field, and pointing to the predicted field. When the reference field and the predicted field are of same parity, the motion vector is used directly (no scaling is necessary).

L.10.5.1 Forward Prediction

L.10.5.1.1 Forward Prediction of the pels of Field 1 (16Hx8V)

The coordinates (x'1 , y'1) of the scaled motion vector are computed as specified, with m = m1.
A 16Hx8V prediction block is obtained from reference field 1 with the vector originating from this field Vertical interpolation is 1/4-pel linear interpolation. Horizontal interpolation is 1/2-pel as usual. Like in the "usual" case, horizontal and vertical interpolation are done in a single step, involving only one division (in this case by 8). An example is given in Figure 1 and Figure 2.

A 16Hx8V prediction block is obtained from reference field 2 with the vector originating from this field. Vertical interpolation is 1/4-pel linear interpolation. Horizontal interpolation is 1/2-pel as usual.
The selection of the prediction is done according to the SVMC3 type:
- **Near-field**: The prediction block used is the one corresponding to the reference field closest to the predicted field (in time axis).
- **Same-parity**: The prediction block used is the one corresponding to the reference field of same parity as the predicted field.
- **Dual**: The prediction block used is obtained by averaging the two prediction blocks from field 1 and field 2. The averaging is done like in "Interpolation-mode" in B-Pictures.

L.10.5.1.2 Prediction of the pels of Field 2 (16Hx8V)

The coordinates (x'2 , y'2) of the scaled motion vector are computed as specified, with m = m2.
For the rest, the prediction is done like in 5.1.1.

L.10.5.2 Backward Prediction

The forward rule is simply transposed.

L.10.5.3 Averaged Prediction in B-Pictures

L.10.5.3.1 Averaged Prediction of the pels of Field 1 (16Hx8V)

The predictor blocks for forward prediction are computed as in sections 5.1.1 and 5.1.2.
The predictor blocks for backward prediction are computed as in 5.2.
The selection of the prediction is done according to the SVMC3 type:
- **Near-near**: The prediction block used is obtained by averaging the prediction blocks from the closest forward and backward reference fields (in time axis).
- **Same-near**: The prediction block used is obtained by averaging the prediction block from the same parity forward reference field and the prediction block from the closest backward reference field (in time axis).
- **Near-same**: The prediction block used is obtained by averaging the prediction block from the same parity backward reference field and the prediction block from the closest forward reference field (in time axis).
- **Same-same**: The prediction block used is obtained by averaging the prediction block from the same parity forward reference field and the prediction block from the same parity backward reference field.

Note that the four SVMC3 averaged modes and the SVMC3 dual mode are extremely similar. Only the choice of the two reference fields differs. The averaging is always done like in "Interpolation-mode" in B-Pictures.
The other SVMC3 modes are equivalent to field-based prediction with 1/4 vertical accuracy.

L.10.5.4. Chrominance

The motion vector used for chrominance is obtained from the luminance SVMC3 motion vector with precisely the same rule as in the case of field-based prediction (for 4:2:0 : divide each coordinate by 2 as described section 5.2.2.1. of TM-2). The rules of prediction are same as for luminance.

L.10.5.5. Motion estimation of SVMC3

Search is done by a local refinement around several candidate motion vectors resulting of a first search. The candidate motion vector can be the result of a full-pel accuracy search. The local search covers 5Vx5H = 25 motion vectors and is done on reconstructed. For each of those, all the candidate SVMC3

prediction blocks must be evaluated. For local search, the vertical step is 1/4-pel, and the horizontal step is 1/2-pel.
In the case of Frame-Pictures, the candidate motion vectors used as starting point of local search are:
- The Frame motion (result of Frame-based search).
- The Field motion vector (result of Field-based search) from the closest reference field to the predicted field of same parity. In this case, the vertical coordinate must be multiplied by two to have 1/4-pel vertical field accuracy.
If forward: from field2 to field2
If backward: from field1 to field1
- Optionally, the other field motion vectors (scaled appropriately) could be used as candidate motion vectors.
In the case of Field-Pictures, the candidate motion vectors used as starting point of local search are the field motion vectors (result of Field-based search), scaled to the field-distance corresponding to same-parity. In this case, the vertical coordinate of the field vectors must be multiplied by two to have a candidate motion vector with 1/4-pel vertical field accuracy.

L.10.6. Concise Specification of DUAL-PRIME

In DUAL-PRIME prediction, single motion vector like that of SVMC3 with 1/2 pixel precision and one very small differential motion vector called DMV is transmitted per macroblock. To obtain motion vectors originating from each of the reference field, and pointing to the predicted field of the different parity, the transmitted motion vector is scaled and DMV is added as follows:

$x' = ((x * m * K) // 32) + dmv\_horizontal$ (x and x' are integers)

$y' = ((y * m * K) // 32) + dmv\_vertical + e$ (y and y' are integers)

The variables (x, y), (x', y'), K, m, and e have been already defined above, and vertical unit is 1/2-pel. dmv_horizontal and dmv_vertical are horizontal and vertical components of DMV with 1/2 pixel precision. These values are restricted within the range from -1 to +1. Note that the same DMV is used for the two scaled motion vectors in the frame picture as illustrated in Figure 3.
When the reference field and the predicted field are of same parity, the motion vector is used directly (no scaling is necessary, and the addition of DMV is not necessary.).

L.10.6.1 Forward Prediction

L.10.6.1.1 Forward Prediction of the pels of Field 1 (16Hx8V):

The coordinates (x'1 , y'1) of the scaled motion vector are computed as specified, with m = m1.
A 16Hx8V prediction block is obtained from reference field 1 with the vector originating from this field. Both horizontal and vertical interpolation is 1/2-pel linear interpolation as usual in the field motion vector. Like in the "usual" case, horizontal and vertical interpolation is done in a single step.
A 16Hx8V prediction block is obtained from reference field 2 with the vector originating from this field. Both vertical and horizontal interpolation is 1/2-pel linear interpolation as usual.
The prediction block used is obtained by averaging the two prediction blocks from field 1 and field 2. The averaging is done like in "Interpolation-mode" in B-Pictures.

L.10.6.1.2 Prediction of the pels of Field 2 (16Hx8V):

The coordinates (x'2 , y'2) of the scaled motion vector are computed as specified, with m = m2.
For the rest, the prediction is done like in 6.1.1.

L.10.6.2 Backward Prediction

The forward rule is simply transposed.

L.10.6.3 Prediction mode in B-Pictures

The averaging mode is inhibited in DUAL-PRIME. Only the forward/backward prediction is used in B-pictures.

L.10.6.4. Chrominance

From DUAL-PRIME motion vector, four field motion vectors for luminance from the reference field 1/field 2 to the predicted field 1/field 2 can be obtained. Corresponding four chrominance vectors are obtained with precisely the same rule as in the case of field-base prediction (for 4:2:0 : divide each coordinate by 2 as described section 5.2.2.1. of TM-2). The rules of prediction are same as for luminance.

L.10.6.5. Motion estimation of DUAL-PRIME

The motion estimation of DUAL-PRIME is carried out by the following two steps.
The first step is to obtain four candidate motion vectors as follows. First, four field motion vectors with half-pel accuracy from the reference field 1/field 2 to the predicted field 1/field 2 are searched by the normal field motion vector search method defined in TM2, except that original pictures are used in half-pel refinement. Then, these vectors are appropriately scaled, if the parity of the predicted field is opposite to that of the reference field.
The second step is to evaluate prediction errors using possible combinations of four candidate motion vectors obtained by the first step, and 3Vx3H = 9 candidates for DMV using local decoded pictures, and to select the best combination of the motion vector and DMV.

L.10.7. Core Experiment L-10

L-10.1. Frame + Field + DUAL-PRIME
L-10.2. Frame + Field + SVMC3
L-10.3. Frame + Field + SVMC3-1/2-pel

Same as L-10.2, except that all motion vectors involved are only 1/2-pel vertical accuracy. The motion vector transmitted is field-type, as in DUAL-PRIME. However the rule for selecting the PMV's is same as for SVMC3, i.e., the same as for Frame-prediction. Scaling of the PMV vertical coordinate is done like for field vectors.
L-10.4 Frame + Field + DUAL-PRIME + SVMC3
L-10.5 Frame + Field + DUAL-PRIME + (SVMC3 - dual)
The dual mode of SVMC3 is not used, since DUAL-PRIME may replace it advantageously. However, no significant hardware simplification is expected by implementation of l-10.5 vs. L-10.4. Among the modes in the above core experiment, mode selection is decided by MSE.

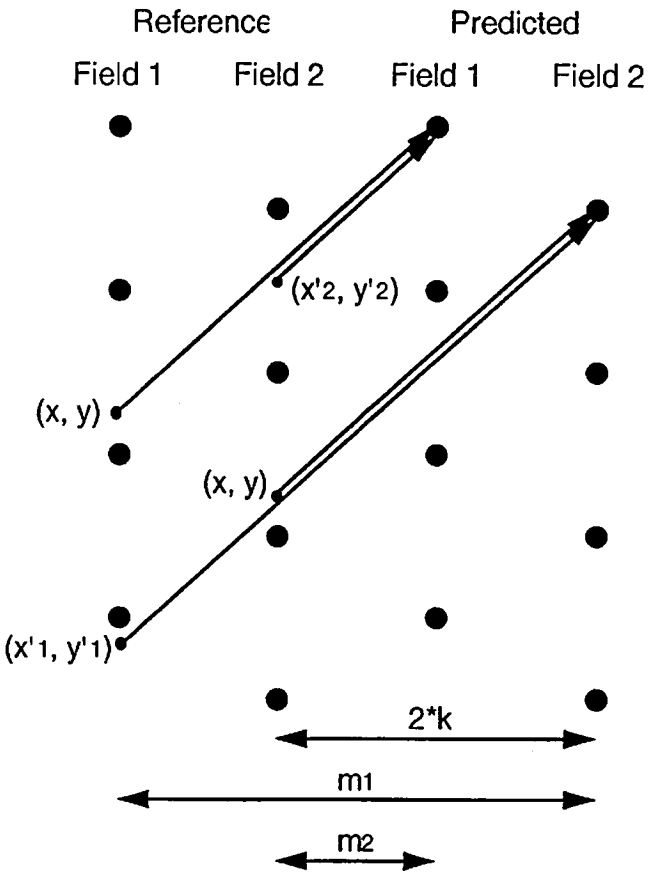


Figure L.10.1 SVMC

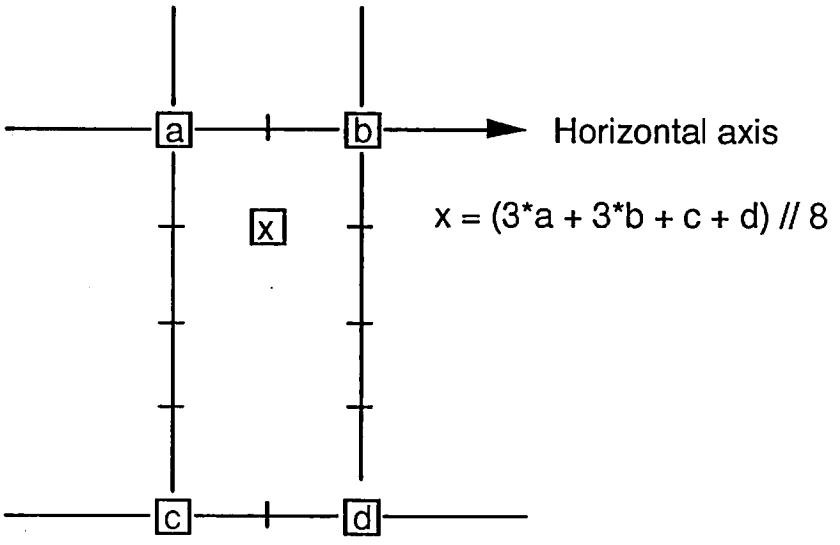


Figure L.10.2 Interpolation at 1/2 H, 1/4 V

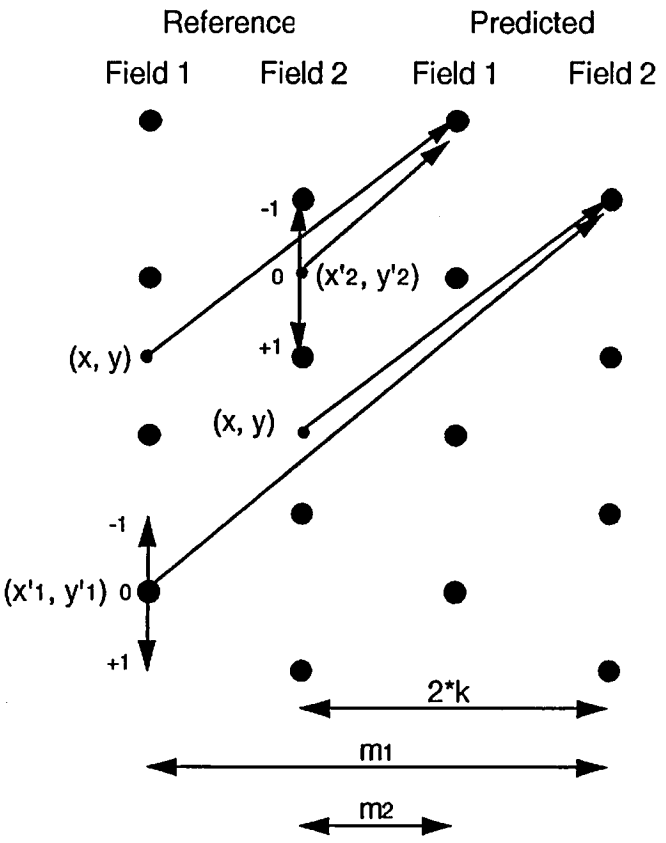


Figure L10.3 DUAL-PRIME

L.11 8x8 Motion Vectors

Test Model 2 is modified to use 8x8 motion vectors. The method will be evaluated by simulating an optimal method of selecting the motion vector block size.1. Macroblock motion vectors are determined for each macroblock which uses forward and/or backwards prediction.2. 8x8 motion vectors are determined for each 8x8 block which uses forward and/or backwards prediction.3. Each macroblock is compressed in the normal mode which does not use 8x8 motion vectors. The number of bits required to encode the 4 luminance blocks are summed. The number of bits required to specify 'macroblock_type' is added to this result. The new total bit count is referred to as N_16x16.4. The same macroblocks are compressed using adaptive 8x8 motion vector encoding (P-frames and B-frames). This mode will use the same field/frame mode as in step 3. For this experiment, the forward interpolation and/or backward interpolation mode will also be the same as in step 3. Each of the 4 luminance blocks are compressed using first the macroblock motion vector and then the 8x8 motion vector and the bit counts are observed in each case. In the second case, the overhead required to specify the 8x8 motion vector(s) is added to the 8x8 bit count. This overhead is the combined length of the horizontal and vertical components of the forward and/or backwards motion vector codewords. Only the difference between the macroblock motion vector and the 8x8 motion vector is coded. If field mode is used then there will be two macroblock motion vectors corresponding to blocks of 8x16 pixels. In this case the overhead for selecting 8x8 motion vector mode is determined by counting the number of bits required to code the difference between the 8x8 motion vector and the corresponding 8x16 motion vector. In this way, the mode requiring the least number of bits is determined for each of the 4 luminance blocks. The resulting total number of bits is then calculated by summing the bit counts corresponding to the more efficient mode for each of the 4 luminance blocks. The number of bits required to specify 'macroblock_type' is then added to this result. The new total bit count is referred to as N_8x8. The 8x8 motion vector codeword table is the same as the one used for encoding

macroblock motion vectors.5. Adaptive 8x8 mode is selected only if N_8x8 is less than N_16x16. If 8x8 mode is selected, then the individual block decisions determined in step 4 are retained.Simple changes to the syntax of the decoder macroblock and block layers are proposed.The following lines are added to the macroblock layer:------ if(macroblock_block_motion) coded_block_motion 3-5 vlclbf------The following lines are added to the block layer:------ if(macroblock_block_motion) { for(i=0; i<4; ++i) if(coded_block_motion & (1 << i)) { if(macroblock_motion_forward) motion_vector() if(macroblock_motion_backward) motion_vector() } }------The variable 'macroblock_block_motion' is determined by macroblock_type. The macroblock_type tables are adjusted to accomodate macroblock_block_motion and as a result, the maximum length of the macroblock_type codeword increases from 7 bits to 11 bits. Tables B.2a, B.2b, and B.2c are changed to the following:------Table B.2a: Variable length codes for macroblock_type in intra coded pictures (I-pictures).VLC

macroblock_quant	macroblock_motion_forward	macroblock_motion_backward	macroblock_pattern	macroblock_intra	macroblock_block_motion1	0
0	0	0	1	001	1	
0	0	0	1	0001	0	
0	0	0	0	0		

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

macroblock_quant	macroblock_motion_forward	macroblock_motion_backward	macroblock_pattern	macroblock_intra	macroblock_block_motion10
0	1	0	1	0	011
0	1	0	1	0	1010
0	0	0	1	0	0011
0	1	0	0	0	00010
0	1	0	0	0	10011
0	0	0	0	1	000011
1	1	0	1	0	000010
1	1	0	1	0	1000011
1	0	0	1	0	0000010
1	0	0	0	1	0000001
0	0	0	0	0	0---------

---Table B.2c: Variable length codes for macroblock_type in bidirectionallypredictive-coded pictures (B-pictures).VLC

macroblock_quant	macroblock_motion_forward	macroblock_motion_backward	macroblock_pattern	macroblock_intra	macroblock_block_motion10
0	11	0	1	1	0
0	1010	0	1	1	1
0	0011	0	1	1	1
0	10010	0	0	1	0
0	00011	0	0	1	0
0	100011	0	0	1	1
0	000010	0	0	1	1
0	10000110	1	0	0	
0	00000100	1	0	0	
0	10000010	0	1	0	1
0	00000011	0	1	0	1
0	100000010	0	0	0	0
1	000000011	1	1	1	1
0	0000000010	1	1	1	1
0	1000000011	1	1	0	1
0	00000000010	1	1	0	1
0	10000000011	1	0	1	1

0 000000000010 1 0 1 1
0 100000000011 1 0 0 0
1 000000000001 0 0 0 0
0 0------

The variable-length codeword for 'coded_block_motion' is sent if macroblock_block_motion is true. It is used to identify the luminance blocks using 8x8 motion vectors. The variable length codes for coded_block_motion are shown below.------Variable length codes for coded_block_motionVLC code coded_block_motion100 1101 2110 4111 80000 150100 30101 50110 100111 1200100 700101 1100110 1300111 1400010 600011 9----------

L.12 Intra/Inter Mode Decisions for 8x8 Blocks

Test Model 2 is modified to allow adaptive intra/inter decisions for each 8x8 block of P-frames and B-frames. The method will evaluated by simulating an optimal mode selection method.1. Each macroblock is first compressed using intra mode for each block. The number of bits required to encode each 8x8 block (luminance and chrominance) is summed. The number of bits required to specify 'macroblock_type' (macroblock_intra = 1) is added to this result and the new total bit count is referred to as N_INTRA.2. The macroblock is again compressed, this time using inter mode for each block. The bit counts for each block is summed with the length of the motion vector codewords and the length of the 'macroblock_type' codeword. The resulting total bit count is referred to as N_INTER.3. The macroblock is again compressed, this time using adaptive intra/inter mode. In this case, each 8x8 luminance and chrominance block is compressed in both intra and inter modes and the mode producing the least number of bits is selected. The total bit count for the macroblock is obtained by summing the bit counts corresponding to the more efficient mode for each of the 8x8 luminance and chrominance blocks. This result is then summed with the length of the motion vector codewords, the length of the 'intra_block_pattern' codeword (described below) and the length of the 'macroblock_type' codeword. The resulting total bit count is referred to as N_ADAP.4. The mode is determined by selecting the minimum of N_INTRA, N_INTER, and N_ADAP. If N_ADAP is the minimum, then the individual intra/inter decisions determined in step 3 are retained.Simple changes to the syntax of the decoder macroblock and block layers are proposed.The following lines are added to the macroblock layer:------ if(!macroblock_intra) intra_block_pattern 1-10 vlclbf------The block layer now becomes:------block(i) { if(pattern_code[i]) { if(macroblock_intra || (intra_block_pattern & (1 << i))) { 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" } }}------The luminance and chrominance blocks which are intra coded are identified by intra_block_pattern. The variable length codes for intra_block_pattern are shown below.------Variable length codes for intra_block_pattern (4:2:0)VLC code intra_block_pattern0 01111 6011101 411100 811011 1611010 32110011 12110010 48110001 20110000 40101111 28101110 44101101 52101100 56101011 1101010 61101001 2101000 621001111 241001110 361001101 310010111 510010110 910010101 1710010100 3310010011 610010010 1010010001 1810010000 34100011111 7100011110 11100011101 19100011100 35100011011 13100011010 49100011001 21100011000 41100010111 14100010110 50100010101 22100010100 42100010011 15100010010 51100010001 23100010000 43100001111 25100001110 37100001101 26100001100 38100001011 29100001010 45100001001

53100001000	57100000111	30100000110	46100000101	54100000100
581000000111	311000000110	471000000101	551000000100	591000000011
2710000000010	39------			

APPENDIX Q: QUANTISATION EXPERIMENTS

Source: Ad-hoc group on quantisation Angra dos Reis ,July 1992

Q.1 SCANNING

Q.1.1 DEPENDENT SCANNING

This experiment is to determine the effectiveness of adapting the scan pattern according to the DCT mode decision (Frame/Field) with the following scans:

Scan Patterns: Frame DCT Zigzag as in Test Model
Field DCT Scan a,b,c or d

with: scan a : As given in CCIR Rec. 723
scan b: As given for field blocks in MPEG 92/261
scan c: As described in MPEG 92/144
scan d: An optimum scan to be determined and communicated within the ad hoc group.

Test Conditions: Bit rate 4 and 9 Mbit/s
Prediction modes must be clearly stated as these may effect the efficiency of adaptive scanning

Criteria for assessment:

i) SNR averaged over sequence and given for I,P and B frames
ii) Bits/frame for the first I,P and B frames in sequence
iii) Bits/frame using MQUANT equal to 5,10 and 15 for the first 3 I, P and B frames and the average over the sequence for I, P and B frames.

Q.1.2 INDEPENDENT SCANNING

This involves adaptation of the scanning pattern independant of the mode selected for the DCT.

Scan 1: Zigzag
Scan 2: Vertical (as in MPEG 92/241)

Decision Criteria: Either "a posteriori" or "a priori" . An a posteriori selection choses the scan pattern which gives the lowest number of bits after the VLC. An a priori decision is made before variable-length coding. A suitable a priori criterion will be circulated within the ad-hoc group.

Syntax for independent scanning: Insert the following after interlaced_macroblock_type

```
if (macroblock_intra || macroblock_pattern) {
    vertical_zigzag_scanning                    1 uimsbf
}
```

Test Conditions: 4 and 9 Mbit/s
Prediction modes must be clearly stated

Criteria for assessment:

1. SNR averaged over sequence (figures given separately for I, P and B frames)
2. Bits/frame for the first I, P and B frames of the sequence.
3. Bits/frame using MQUANT equal to 5 and 10 and 15 for the first I and P and B frame and also the average over the sequence separately for I, P and B frames.

Additional statistics: Percentage usage of each scan in separately for I, P and B frames.

Q.2 EXPERIMENTS ON MATRIX ADAPTATION

Q.2.1 MATRIX ADAPTATION WITH MACROBLOCK TYPE

It is proposed to download 4 matrices at the picture layer. The syntax for this as follows:

Before "picture_extension-data" insert the following:

```
load_quantiser_matrices                 1   uimsbf
if (load_quantiser_matrices) {
    quantiser_matrix_one                8*64 uimsbf
    quantiser_matrix_two                8*64 uimsbf
    quantiser_matrix_three              8*64 uimsbf
    quantiser_matrix_four               8*64 uimsbf
    matrix_assignment_table              16  uimsbf
}
```

where the matrix assignment table, which assigns one of the four matrices to the different macroblock types, is constructed as follows:

Macroblock type			Matrix Selection (bits)
Intra	Frame	Y	2
Intra	Frame	C	2
Intra	Field	Y	2
Intra	Field	C	2
Non_intra	Frame	Y	2
Non_intra	Frame	C	2
Non_intra	Field	Y	2
Non_intra	Field	C	2

SPECIFICATION OF MATRICES: To be optimised and distributed within ad-hoc group for both 4:2:0 and 4:2:2 cases

Initial comparison: For macroblock types Intra/*/* and Non_intra/*/* use matrices as defined in TM together with those defined in MPEG 92/261 for the 4:2:0 case.

SPECIFICATION OF ASSIGNMENT TABLE: To be optimised and distributed within ad-hoc group

ASSESSMENT CRITERIA: Picture quality !

Q.2.2 EXPERIMENT OF ADAPTIVE CONTROL OF MATRIX SELECTION AT MB LEVEL

Four matrices are transmitted at the picture layer as described in previous experiment. Transmission of the assignment table is not necessary.

At the MB layer, after "coded_block_pattern()", insert:

mb_quantiser_matrix_select 2 uimsbf

This variable identifies which of the four matrices is to be used for coding all the blocks in this MB.

CRITERIA FOR MATRIX DEFINITION AND SELECTION: To be determined and distributed in quantisation ad-hoc group.

An initial experiment would be to adapt the weighting matrix according to the criticality of the macroblock; i.e. a macroblock can have one of four possible values of criticality which selects one of the 4 matrices. The meaning of criticality is specified in the following table:

Criticality Weighting Matrix Adjustment

0 Lower precision of all but lower frequency terms
1 No change
2 Increase precision of frequency terms representing interlace
3 Increase precision of low frequency terms

ASSESSMENT CRITERIA: i) Picture Quality

Q.2.3 Quantizer Weighting Matrices

At the sequence layer four matrices wdelta0[64] , wdelta1[64], wdelta2[64] and wdelta3[64] containing 8-bit signed integers are transmitted, these are relative weighting matrices which when added to Intra or NonIntra matrices yield actual weighting matrices used for quantization.

Chrominance Weighting

At the sequence layer, appropriate wdelta matrix to be used for determining the chrominance matrix (by summing it to corresponding Intra or NonIntra matrix) is identified. Thus this matrix is global for all chrominance weighting in the sequence.

Local Weighting Adaptation

The weighting matrix used for a macroblock depends on the "criticality" of the macroblock. A macroblock can have one of the four possible values of criticality and each criticality value corresponds to a specific wdelta matrix. The following table clarifies the meaning of macroblock criticality.

Criticality	Weighting Matrix Adjustment
0	Lower precision of all but low frequency terms
1	no change in any term
2	increase precision of frequency terms representing interlace
3	increase precision of low frequency terms

A fixed 2-bit code per macroblock (this can be made VLC later) identifies the criticality.

The exact wdelta [] matrices will be specified with in the ad hoc group.

Syntax

```
sequence_header() {

        load_delta_matrices                     1
        if (load_delta_matrices) {
                wdelta0[64]                     8*64
                wdelta1[64]                     8*64
                wdelta2[64]                     8*64
                wdelta3[64]                     8*64
        }
        chroma_matrix_enable            1
        wdelta_select                           2

}

macroblock() {

        wdelta_select                           2

}
```

Q.3 EXPERIMENTS ON QUANTISER RANGE, PRECISION, AND CONTROL

Q.3.1 EXPERIMENT EXTENDING RANGE OF TRANSMITTED COEFFICIENT

This experiment is to test the effectiveness and necessity of increasing the maximum range of the transmitted coefficients from +/- 255 to +/- 2047

One bit is added at the Picture Layer, to indicate that a single 12-bit level *tcoef* escape is used, rather that the mechanism of *escape/double-escape*, as in MPEG-1.

tcoef escape format	
0	Compatible MPEG-1
1	12-bit Escape

When *12-bit Escape* is selected, the syntax of the escape changes. The *escape code* "000001" is followed by the *run* (6-bit VLC), followed by the *level* (12-bit VLC).
The table for the escaped level becomes :

FLC	level
100000000001	-2047
100000000010	-2046

...	...
111111111111	-1
forbidden	
000000000001	+1
...	...
011111111111	+2047

The mechanism of double-escape (28 bits) is not used is this case, making the maximum length of an encoded run-level equal to 24 bits.
As a side effect, this proposal may simplify the implementation of encoders and decoders, as it is no more necessary to check whether MQUANT need to be changed to avoid coefficient clamping at 255, and the maximum number of bits per transmitted coefficient is reduced from 28 to 24 bits

ASSESSMENT CRITERION: Picture quality on pictures which generate large coefficients with low MQuant values (e.g. still pictures)

Q.3.2 EXPERIMENT TO INCREASE PRECISION OF INTRA DC COEFFICIENTS

One bit is added at the Picture Layer, to indicate that Intra DC coefficients are transmitted with a 9-bit precision rather than 8-bit.

intra_dc_format	
0	Compatible MPEG-1 (8-bit Precision)
1	9-bit Precision

When *9-bit Precision* is selected, the intra DC coefficients are quantized (resp. dequantized) with no-dead-zone and a step-size equal to 4, rather than 8, and transmitted with 9-bit precision rather than 8-bits.
When *9-bit Precision* is selected, the value 256 is used, rather than 128, to initialize or reset the intra DC predictors.
The tables B5a and B5b used to decoding *bit Precision* intra DC coefficients are extended in a natural way :

Luminance	Size
100	0
00	1
01	2
101	3
110	4
1110	5
11110	6
111110	7
1111110	8
11111110	9

Chrominance	Size
00	0
01	1
10	2
110	3
1110	4
11110	5
111110	6
1111110	7
11111110	8
111111110	9

ASSESSMENT CRITERION: Picture quality on appropriate pictures

Q.3.3 EXPERIMENT TO EXTEND RANGE OF MQUANT

A) INREASING MQUANT TO 6-BIT

The MQUANT range of 1 to 31 is insufficient for very high resolution video and also the precision of MQUANT is not sufficient at low values for high quality coding.

In addition to default table of values for MQUANT from 1 to 31, an mquant_extend table is proposed. This table covers a range of 1 to 63 and is nonlinear in nature, providing higher precision for smaller values of MQUANT and coarser precision for finer values. An example of the MQUANT values allowed for this mquant_extend table are the following:

```
1     1.5   2     2.5   3   ......   14    14.5  15    15.5

16    17    18    ....................  29    30    31

33    35    37    ....................  59    61    63
```

The mquant_extend table is downloaded at the sequence layer with the following syntax in the Sequence Header:

```
sequence_header(){
        .
        .
        .
    mquant_extend              1
    if(mquant_extend)
        mquant_extend[64]       8*64     uimsbf
            .
            .
            .

}
```

If necessary, this table could be loaded at the picture layer

It is recognised that many ways of enabling a larger range of MQUANT are possible and optimisations in spacing of MQUANT values is dependent on bit rate and picture complexity. It is therefore important that any experiments are performed on material containing high resolution details as well as at higher bit rates (above 10 Mbit/s)

B) 5-BIT MQUANT WITH ADAPTIVE ASSIGNMENT OF VALUE

In this experiment the syntax and objectives are similar to the last experiment except that the number of levels assigned to MQUANT remains at 32 and an mquant_assign table (8*32 bits) is downloaded at the picture layer.

Q.3.4 EXPERIMENT ON BLOCK ADAPTIVE QUANTISATION

To improve picture quality locally in a scene, refinement of MQUANT may be necessary on a block basis. A 1-bit code decides whether such a refinement is necessary; if so, 2-bits are sent for every coded luminance block within that macrblock. The mechanism for refinement is via use of BQSCALE, which can have one of four possible values such as {0.5, 0.75, 1.0, 1.5}. The MQUANT value for the macroblock is multiplied by the chosen BQSCALE and used as the block quantiser value. For macroblocks not requiring refinement, no BQSCALE bits are necessary.

The choice of BQSCALE for a block depends on the following:

- The variance of a block
- The variances of its three causal neighbours. These are used to detect the presence of an edge. If such an edge is detected, BQSCALE of 0.5 or 0.75 is selected.

It is noted that the decision criteria, concerning whether or not a macroblock needs to be refined with respect to MQUANT, depends on the bit rate available and picture quality desirable.

The syntax is:

Macroblock Layer

```
macroblock(){
        .
        .
    if(macroblock_quant){
        quantiser_scale                    5 (/6)
}
quantizer_scale_refine                     1
        .
        .
}
```

Block Layer

```
block(i){

    if(pattern_code[i]){

        if(i<4){
            if(quantiser_scale_refine){
              bqscale                      2
            }
        }
    }
}
```

Q.4 Alternate DCT (DCT/NTC)

Q.4.1 INTRODUCTION

In this proposal of core experiment, algorithm enhancement to the current TM to improve coding efficiency of edge information which based on the concept of adaptive DCT/Non-DCT is to be examined. The basic coding process additions to the current TM are Edge Block Detection Process, Prediction Process, and Quantization Process. They are described in detail in the following sections.

Q.4.2 CORE EXPERIMENT REFERENCE MODEL

Reference Model : TM2 Frame structure, with Frame/Field Adaptive Modes only
Data Rate : 4 Mbits/s
Test Sequences : Mobile & Calendar, or similar
Algorithms : NTC1 (Section 3), and NTC2 (Section4)

Bitstream Syntax :

adaptive_coded_type -- This is a 1-bit flag to indicate whether or not the Macroblock is coded adaptively by the DCT/DPCM coder. If this is set to "0", all blocks in the Macroblock are DCT coded.

adaptive_coded_block_pattern -- If the adaptive_coded_type code is "1", the 6-bits adaptive_coded_block_pattern is followed with each bit indicating the block position of the NTC (method 1 or 2) coded blocks in the Macroblock. (eg. first received bit is "1", the first block in the Macroblock is non-DCT coded, and so on.)

Macroblock Layer :

```
    ....

if ((core_experiment==NTC1) || (core_experiment==NTC2)) {
    adaptive_coded_type                                  1
    if ( adaptive_coded_type == 1 )
        adaptive_coded_block_pattern                     6
            }

if ( macroblock_quant )
    quantizer_scale                                      5        uimsbf

    ...
```

Block Layer :

data_scan_path_flag--- This is a 2bits FLC to indicate which data scan path is selected . Table are shown in the document MPEG92/275.

differential_flag--- This is a 1bit to indicate differential on or off. If it is 1, the differential is on, if it is 0, the differential is off.

block_quantizer_scale --- An unsigned integer in the range 1 to 31(5bits) used to scale the reconstruction level of the retreived NTC quantized levels..

In NTC1 coding, the base value is treated the same as DCT dc coefficients. So the DCT dc coefficients and NTC base are coded by the Z-differential technique same as TM0.

```
block (i) {
    if ( pattern_code[i] ) {

        if ((core_experiment==NTC1)&&(adaptive_coded_block_pattern[i]==1)) {
            data_scan_path_type                          2
            differential_flag                            1
            block_quantizer_scale                        5
            }

        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' )
            coded_coeff_next                                 3-28    vlclbf
        end_of_block                                         2
        }
    }
}
```

Q.4.3 DESCRIPTION OF NTC1 METHOD

1) METHOD OVERVIEW

NTC1 is a predictive coding method in the block layer. Examples of NTC1 are shown in Figure 1 and 2 in the document MPEG92/275. The following 4 techniques are applied in NTC.

1.1) Prediction of block representative value

In NTC1 the block representative value is called "base". NTC1 module predicts the base value and encodes the differential values between the base and each pixel in the block.

1.2) Adaptive Quantization

The Quantization is applied to the differential values between the base and each pixel in the block. The values are quantized with a flat weighting and with a deadzone equal to the stepsize. So the quantizer is the same as for non intra macroblocks in the TM0.

1.3) Adaptive Data Scan

For each block the NTC1 module selects one scan path which gives the fewest number of run/level events. Then for each block the NTC1 module selects either "differential on" or "differential off" depending on the number of VLC events. If the differential is on, 1 dimensional differential coding is applied for the scanned data..

2) Algorithm for NTC1

The algorithm is described in the document MPEG92/275. Please refer to the document.

Q.4.4 DESCRIPTION OF NTC2 METHOD

1) METHOD OVERVIEW

A simplified block diagram of the TM0 is described in figure 1 with extensions to Adaptive DCT/DPCM method as shaded boxes. The Macroblock Type Decision Module in figure 1 performs the decision of Motion Prediction Mode (Intra, Forward, Backward, Frame, Field, etc), the Motion Compensate Process, and the Frame/Field DCT Decision.

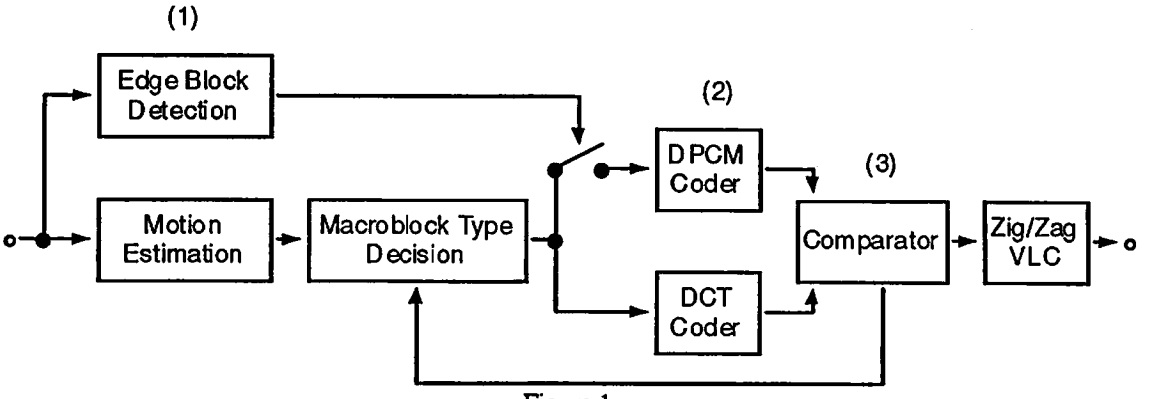


Figure 1

The Edge Block Detection (1) process is performed on original input blocks. Decision on Edge/Non-Edge Block is done based on Block signal energy and the neighboring Block signal energy. If the Block is detected as Edge Block, it will also be coded with the DPCM Coder (2). The reconstructed blocks (after coding and decoding) from the DCT Coder and the DPCM Coder are then compared based on Mean-Square Error (MSE) with reference to the original input Block at the Comparator (3) module, and the coder with best result (lower MSE) will be selected, and output coded Block to the Zig/Zag Scanner and VLC. This module may be omitted for the core experiment to simplify implementation, and selection is then based on Edge Detection result only.

2) EDGE DETECTION

In the Edge Detection process, the amount of activity in each input Luminance 8x8 block is calculated and compared to a pre-defined threshold value *T1*. The sum of the square deviation of the block pixel intensity values from the mean block intensity value is used as representation of the block activity. If the calculated activity is greater than *T1*, then the activities of the surrounding blocks are compared to another threshold value *T2*. If any one of the activities of the surrounding block is lower than *T2*, the input block is considered as a boundary block between an object in the video sequence and a smooth background or region. In such a case, the input block is classified as an Edge Block.

For the Chrominance block, if any one of the Luminance block in the same Macroblock is detected as an Edge Block, and the calculated activity of the chrominance block is greater then a third threshold value *T3*, the chrominance block is considered as an Edge Block. The threshold values *T1*, *T2* and *T3* are set at *5300*, *1000*, and *8500* respectively for the simulation (they may be optimized for special sequences - encoder issue).

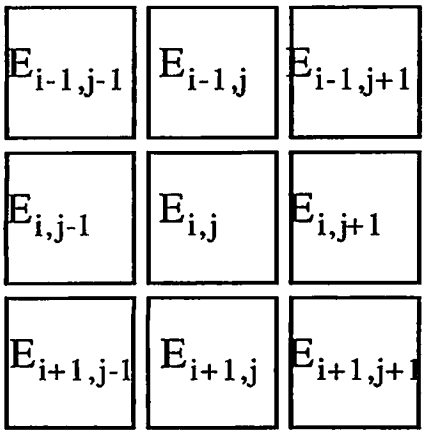


Figure 2

Due to the fact that all surrounding blocks will be examined, an input block will not be considered for edge condition if it is at the picture boundary (first row, last row, first column,, last column). In case of a Field DCT Macroblock in an adaptive Frame/Field system, luminance block 1 & 3 (similarly block 2 & 4) of the macroblock must satisfy the edge block condition together in order to be considered as edge blocks since the edge conditions are checked on frame based block only; hence, corresponding to the Frame/Field DCT, Frame/Field DPCM is applied to the edge blocks.

3) DPCM CODER

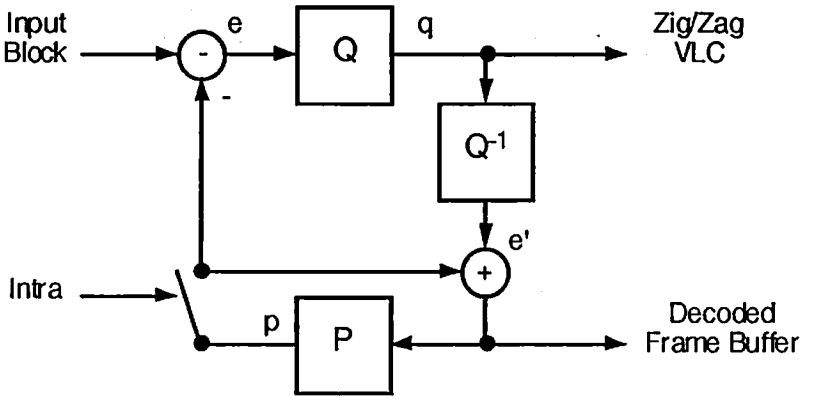


Figure 3

The DPCM Coder is illustrated in figure 3. A pel is scanned left-to-right and top-to-bottom from the input edge block to be coded. For an Intra Coded Block, the input block is coded as following :

1) The first pel (top-left corner of the block) is not coded in the DPCM coder. It is sent to the VLC for transmission as "DC" of the block.
2) The quantization parameter b is calculated from the reference quantization parameter Q (with no adaptive quantization applied for macroblock containing at least one edge block) of the macroblock (j) as :
   $b = Q_j \times 0.1875 + 0.625$
3) For each of all other pels, a predicted value p is made based on the average of passed coded top and left pels (if available), and the prediction error e is quantized as :
   $q = (int)$(**Error!**) $\times$ *Sign (e)*

The Dequantization is :
$e' = (int)((q^2+|q|) \times b^2) \times Sign(q)$

4) All quantized coefficients are output to the TM0 Zig/Zag and VLC for transmission as "AC" of the block.

For a non-intra Coded Block, the input block is coded as following :

1) The quantization parameter b is calculated from the reference quantization parameter Q (with no adaptive quantization applied for macroblock containing at least one edge block) of the macroblock (j) as :
$b = Q_j \times 0.1875 + 1.125$

2) Each pel e in the block is quantized as :
$q = (int)$(Error! $- 0.5) \times Sign(e)$
The Dequantization is :
$e' = (int)((q^2+|q|+$Error!$) \times b^2) \times Sign(q)$
where $k =$ Error!)

3) All quantized coefficients are output to the TM0 Zig/Zag and VLC for transmission as "AC" of the block.

An example "C" implementation of the DPCM Coder is as follow :

```
DpcmBlock (block, qBlock, Q, mbType)

    int     block[8][8];
    int     qBlock[8][8];
    int     Q;
    int     mbType;[

    int     x, y;
    int     pel, p, e, dec;
    int     q, k;
    double  b;

    if (mbType==Intra) b = (double)Q * 0.1875 + 0.625;
    else b = (double)Q * 0.1875 + 1.125;
    if (b<1.7) b = 1.7;

    for (y=0; y<8; y++) {
        for (x=0; x<8; x++) {

            pel = block[y][x];

            if (mbType==Intra) {
                if ((x==0)&&(y==0)) {    // Transmit not Quantized
                    q = block[0][0];
                    dec = pel;}
                else {
                    if (x==0) p = block[y-1][0];
                    else if (y==0) p = block[0][x-1];
                    else p = (block[y][x-1] + block[y-1][x]) / 2;

                    e = pel - p;
                    q = (int)(sqrt((double)(abs(e)))/b) * Sign(e);

                    dec = p + (int)((q*q+abs(q))*b*b) * Sign(q);
                    ]
                }
            else {
                e = pel;
                q = (int)(sqrt(4.0*abs(e)+b*b)/(2.0*b) - 0.5) * Sign(e);
```

```
            if (q==0) k = 0; else k = 1;
            dec = (int)((q*q+abs(q)+(abs(q)+k)/2)*b*b) * Sign(q);
            }
        qBlock[y][x] = q;
        block[y][x] = dec;    //    Local Decoded Value
        }
    }
for (y=0; y<8; y++)
    for (x=0; x<8; x++) block[y][x] = ClampInteger (&block[y][x], 0, 255);
}
```

For real-time implementation consideration, the Quantization and dequantization process can be implemented by table-look-up method.

4) COMPARATOR

The local decoded block from the DCT Coder and the DPCM Coder can be compared to the original (uncoded) block in this module based on the Mean-square-Error measure MSE (other measuring criteria is possible). Final selection of output quantized coefficients of the block from one of the two coders is made based on lowest MSE. This module only maximizes signal/noise ratio of the decoded pictures. Again, this is done only for SNR optimization (not necessary picture quality improvement).

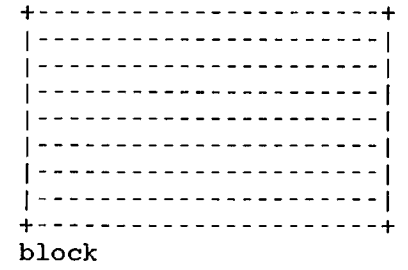
5) REFERENCE

All figures in section 4 (figure 1,2,..) can be refered to in document MPEG 92/368. For further questions, please feel free to contact :

Lucas Hui
AV/Information Research Center
Asia Matsushita Electric (S) Ptd Ltd
Email : lucas@avirc.ams.mei.co.jp
Tel : +65 381-5460
Fax : +65 285-7237

Q.5 NON-8 x 8 DCT

Purpose : To belify the coding efficiency and improvement of visual quality
Definition ;
8 x 1 DCT
1. Horizontally one dimensional 8 DCT that described in MPEG1 CD is used.
2. Block size is same as normal 8 x 8 DCT , then 8 sets of horizontal DCT coeff. exist in each block independently.



3. 8 x 1 DCT is applied to Non-Intra MB only
4. Each coefficient is multiplied by 2 instead of 4 that is in case of 8 x 8 DCT.
5. VLC(AC coeff.) is same as 8 x 8 DCT (TM)
6. Q-Matrix and Scanning for 8 x1 DCT block are basically one dimensional as below;

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

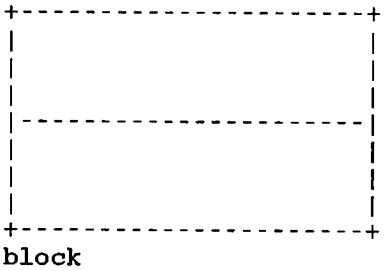
Scan order for 8 x 1 DCT

22	23	24	26	27	28	30	31
22	23	24	26	27	28	30	31
22	23	24	26	27	28	30	31
22	23	24	26	27	28	30	31
22	23	24	26	27	28	30	31
22	23	24	26	27	28	30	31
22	23	24	26	27	28	30	31
22	23	24	26	27	28	30	31

Q-Mat. for 8 x 1 DCT

8 x 4 DCT

1. Use 8 x 4 DCT intead of 8 x 8 DCT.
2. Block size is same as normal 8 x 8 DCT , then 2 sets of DCT coeff. exist in each block independently.



block

3. 8 x 4 DCT is applied to only non-intra MB
4. Each coefficient is multiplied by suare root 8 instead of 4 that is in case of 8 x 8 DCT.
5. VLC(AC coeff.) is same as 8 x 8 DCT (TM)
6. Q-Matrix and Scanning for 8 x4 DCT block are as below;

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

Scan order for 8 x 4 DCT

18	19	20	21	22	23	24	25
19	20	21	22	23	24	25	26
20	21	22	23	24	25	26	27
21	22	23	24	25	26	27	28
18	19	20	21	22	23	24	25
19	20	21	22	23	24	25	26
20	21	22	23	24	25	26	27
21	22	23	24	25	26	27	28

Q-Mat. for 8 x 4 DCT

Adaptation of 8 x 8 / 8 x 1 or 8 x 8 / 8 x 4 DCT

Switching unit : Macro Block

Decision 1 (post)

1. Calculate amount of data in each MB
2. Select smaller one

Decision 2 (pri)

pri-decision method will be added by JVC

Core experiments

1. 8 x 1 DCT / 8 x 4 DCT
2. All non-intra are 8 x 1 (8 x 4) / adaptive
3. Field-structure / Frame-structure
4. Decision 1./ Decision 2

Syntax

```
if (macroblock_pattern)
        coded_block_pattern      3 -- 9     vlclbf
```

non_dct will be signalled in the sequence_layer

in macroblock layer will be changed

```
if (macroblock_pattern) {
        coded_block_pattern      3 -- 9     vlclbf
if (non_dct)
        non_8x8dct_flag  1       uimsbf
```

Reference
MPEG 92 / 093, MPEG 92 / 261

If any question, contact ;

Kenji Sugiyama Digital Technologies Research Dept. Central R&D Center JVC
58-7, Shinmei-cho , Yokosuka, Kanagawa 239, JAPAN
e-mail : k-sgym@krhm.jvc-victor.co.jp Tel : +81-468-36-9275 Fax : +81-468-36-8540

Q.6 Adaptive VLC's

In this Section, three alternative VLC proposals are discussed. These are:

Section Q6.1	MUVLC and frequency scanning proposal
Section Q6.2	Macroblock switchable VLC
Section Q6.3	Alternative VLC

In order to be able to compare the effectiveness of the different proposals, it is suggested that results are quoted for the following two cases:

a) Fixed quantiser (MQUANT = 10), giving number of bits.
b) Fixed bitrate of 4 Mbit/s, comparing S/N and subjective quality.

Other cases are also of interest, such as, other values of MQUANT, other bit rates and the performance for all-intra coding. Suggested parameters are:

	Standard I/P/B frames	All Intra
Bitrate	4/9 Mbit/s	12/20/25 Mbit/s
Fixed MQUANT	5/10 for I and P frames 1.4 times for B frames	3/5/8

(Note: For the experiment on vector quantisation in Section 7 additional constraints are introduced into the comparison criteria.)

Q.6.1 Frequency scanning

Frequency scanning combined with an entropy coding technique, called MUVLC (Modified Universal Variable Length Coding) has been proposed to increase the coding efficiency of the test model and to improve the performance of scalable systems both in terms of coding efficiency and functionality [4-8].

The purpose of this experiment is to evaluate the coding efficiency of the proposed method compared to the present VLC used in TM2.

In the following the principle of the algorithm and its adaptation to the TM2 syntax are described. A C-program of the MUVLC algorithm will be distributed to interested parties by e-mail (send requests to B. Hammer (Siemens), ha@bvax4.zfe.siemens.de, or J. De Lameillieure (HHI), grunaaie@w204zrz.zrz.tu-berlin.de).

Q.6.1.1 Global parameters of the experiment

The experiment relies on the basic TM2 mode. The following specific parameters are required :

chroma format :	4:2:0
picture structure :	frame picture
bit rate :	4 Mbit/s
group of pictures structure :	N=12 (15), M=3
prediction :	frame/field adaptive
DCT :	frame/field adaptive
sequences :	Flower, Mobile, Football, Table Tennis, Bicycle
number of frames :	50 (60) frames, the first two seconds of a sequence

Q.6.1.2 Principle of MUVLC

In contrast to entropy coding using block scanning, as described in the sections 8.5 - 8.7, the proposed frequency scanning method applies run length coding to coefficients of different blocks which have the same scanning number respectively represent the same frequency band.

The code for a slice is generated in six basic steps :

1) arrange the coefficients of all coded blocks (indicated by the CBP), which belong to a macroblock slice, according their scanning number in 64 stripes. This is illustrated in the following figure :

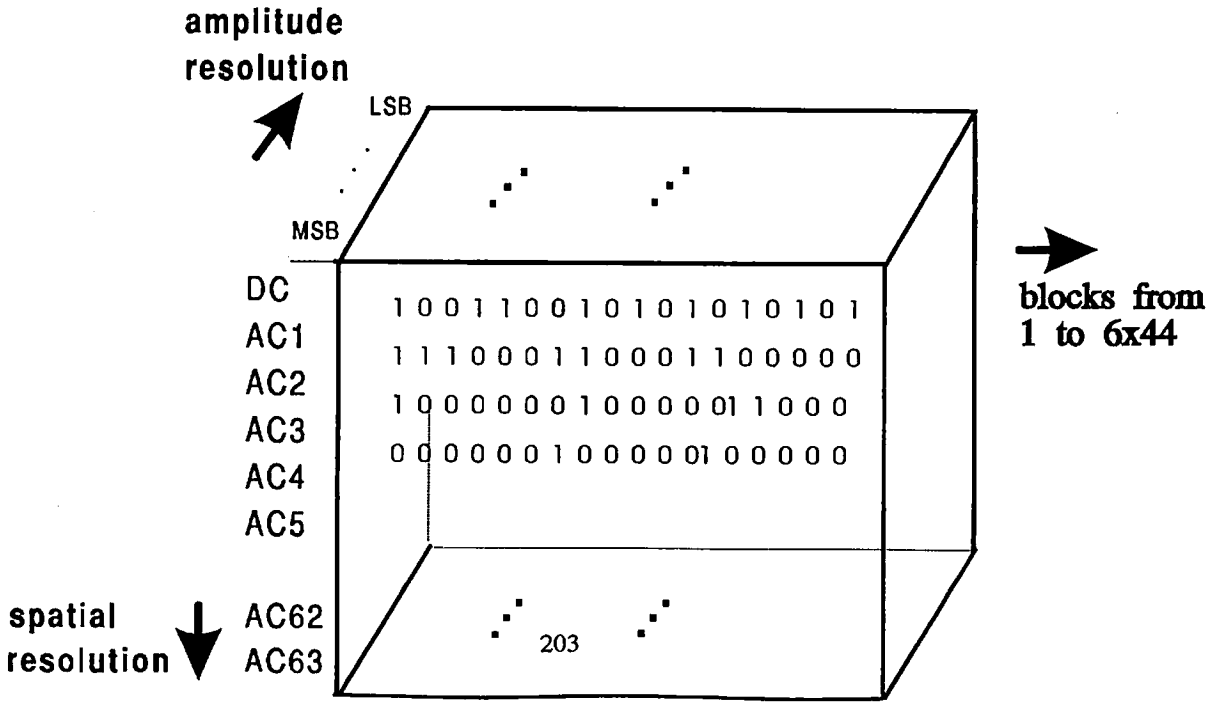


Figure Q.6.1: MUVLC scanning

The horizontal planes of the cube in the figure above are the stripes, used in the MUVLC coding.

2) for encoding a stripe, select from the table given below the mangitude class of the coefficient with highest amplitude, and transmit the 3 bit class prefix code PC ;

magnitude range	class C	prefix class code (PC)
0 .. 1	0	000
2 .. 3	1	001
4 .. 7	2	010
8 .. 15	3	011
16 .. 31	4	100
32 .. 63	5	101
64 .. 127	6	110
128 .. 255	7	111

3) transmit the 3 bit line prefix code PL, which determines the Adaptive Truncated Run Length (ATRL) code [2] used in step 4) for this class ;

4) encode each non zero coefficient of class C using a code of the form

```
(RL NCB)
```

where RL gives the position of the coefficient in the class relative to its previous nonzero coefficient. NCB gives the exact value of the coefficient by transmitting the C least significant bits of its magnitude plus one sign bit. Each class code string is terminated by an End of Class (EOC) code word ;

5) repeat step 3) to 4) for all remaining magnitude classes. Coefficients which have been already encoded in the previous classes are ignored for calculating the run-length code RL.

6) repeat step 2) to 5) for all stripes of the slice.

Thus, the code for a slice gets the following structure :

```
PC PL (RL NCB) (RL NCB) (RL NCB) ... EOC    stripe 1, class C
   PL (RL NCB) (RL NCB) (RL NCB) ... EOC    stripe 1, class C-1

              ...

   PL (RL NCB) (RL NCB) (RL NCB) ... EOC    stripe 1, class 0
PC PL (RL NCB) (RL NCB) (RL NCB) ... EOC    stripe 2, class C
   PL (RL NCB) (RL NCB) (RL NCB) ... EOC    stripe 2, class C-1

              ...

              ...

   PL (RL NCB) (RL NCB) (RL NCB) ... EOC    stripe 64, class 0
```

Q.6.1.2.1 Principle of ATRL

The table below shows the basic code structure of an ATRL-code with line prefix code length PL=3. A "0" in the source pattern denotes a coefficient of a lower class while "1" indicates the occurence of a coefficient within the current magnitude class.

TABLE : Truncated run-length code for PL=3 and M=8

Source pattern	Run length (RL)	RL-code
00000000	8	0
1	0	1000
01	1	1001
001	2	1010
0001	3	1011
00001	4	1100
000001	5	1101
0000001	6	1110
00000001	7	1111

The maximal zero run length, which can be encoded by ATRL with a single codeword is truncated to a length of M=(2**PL)-1. To cope with runs having more than M preceeding "0"s, the first codeword is used for extension by runs of M+1 "0"s.

The extension code word is coded by a single bit set to "0", while run length codewords consist of a prefix set to "1" followed by PL bits which give the number of preceeding "0"s.

To indicate the End of Class (EOC), a position at the border or outside the stripe is addressed by sending extension codewords "0" from the RL-codebook.

As the receiver knows the length of a stripe, no EOC should be sent when the end of a class is reached after coding the last coefficient of the considered class. Sending no EOC means an EOC with length 0.

This is also the case when after the end of a class is reached after transmitting one or more "0"s. Then also, no further run lengts codewords should be transmitted.

The optimal line prefix code length PL, i.e., the one that delivers the shortest code length, is selected for each magnitude class. The class code length that has to be minimized with respect to PL is calculated by the program code below :

```
/*  run_length[i]    = RL-code of the run length preceding the non
                       zero coefficient i in a class
    k                = number of non zero coefficients in a class
    zero_run_length  = number of zero coefficients following
                       coefficient k
*/

code_length = 0 ;

/* how many times has the RL-codeword "0" been transmitted */

for (i=1; i<=k; i++)
  code_length += run_length[i] / (M+1) ;

/* bit rate for transmission of RL-codeword "1xx...x" (with x=0 or
```

```
x=1) for each non zero coefficient in a class */

code_length += k*(1+PL) ;

/* bit rate for EOC */

code_length += (zero_run_length==0) ? 0 :
                                ((zero_run_length-1) / (M+1)) + 1 ;
```

It should be repeated that the "/"-operation is the integer division, as defined in section 2.1 on arithmetic precision.

Q.6.1.3 Arrangement of luminance and chrominance coefficients

Q.6.1.3.1 Arrangement of luminance and chrominance coefficients within a stripe

Y-coefficients (max. 176)	Cb-coefficients (max. 44)	Cr-coefficients (max. 44)

Q.6.1.3.2 Scanning order of blocks within a macroblock

Blocks are scanned according the numbering as described for block_count in 9.3.7

Q.6.1.3.3 Scanning order of coefficients within a block

Zig-zag scanning as usual. The scanning of DCT coefficient has no impact on coding efficiency.

Q.6.1.4 Syntax modifications

For purpose of the frequency scanning experiments, the slice layer, the macro block layer and the block layer must be changed as indicated below :

Q.6.1.4.1 Slice layer

```
slice() {
  slice_start_code
  quantizer_scale

     ...

    (no change)

     ...

  extra_bit_slice
  slice_control()
  slice_data()
}
```

Q.6.1.4.2 Slice control layer

```
slice_control() {
  for (mb)0; mb<44; mb++) {
    while (nextbits()=='0000 0001 111')
      macroblock_stuffing

        ...

      (same syntax as in macroblock() of TM2)

        ...

    if (macroblock_pattern)
      coded_block_pattern()
  }
}
```

Q.6.1.4.3 Slice data layer

```
slice_data() {
  slave_slice_start_code                                          32
  for (coef_i=1; coef_i<65; coef_i++) {
    buf_size=0
    for (mb=0; mb<44; mb++)    /* Y */
      for (block_i=1; block_i<=4; block_i++)
        if (pattern_code[mb][block_i]) {
          c_buf[buf_size] = dct_coef[mb][block_i][coef_i]
          buf_size++
        }
    for (mb=0; mb<44; mb++)    /* U */
      if (pattern_code[mb][5]) {
        c_buf[buf_size] = dct_coef[mb][5][coef_i]
        buf_size++
      }
    for (mb=0; mb<44; mb++)    /* V */
      if (pattern_code[mb][6]) {
        c_buf[buf_size] = dct_coef[mb][6][coef_i]
        buf_size++
      }
    muvlc(c_buf, buf_size)
  }
  while (nextbits()!='0000 0000 0000 0000 0000 0001')
    next_start_code()
}
```

```
muvlc(c_buf, buf_size) {
  pc = pc_code(c_buf, buf_size)                                    3
  for (class=pc; class>=0; class--) {
    pl = pl_code(c_buf, buf_size)                                  3
    run = 0
    for (buf_i=0; buf_i<buf_size; buf_i++)
      if (!c_coded[buf_i]) {
        if (((c_buf[buf_i]>>class) & 0x1) == 1) {
          rl = rl_code (run,pl)
          ncb = ncb_code(c_buf[buf_i],pc)                          1 - 8
          c_coded[buf_i] = 1
          run = 0
        }
      else
        run++
    eoc = eoc_code(run,pl)
  }
}
```

```
rl_code(run,pl) {
  while (run>=(1<<pl)) {
    max_run                                                  1      "0"
    run -= 1<<pl
  }
  prefix                                                             1
      "1"
  run                                                        1 - 8
}
```

```
eoc_code(run,pl) {
  while (run > 0) {
    max_run                                                  1      "0"
    run -= 1<<pl
  }
}
```

Q.6.1.5 Supplementary experiments

** Removal of Coded Block Pattern (CBP) and macroblock_address_increment (MAI) in the Slice Control Layer*

The use of CBP and MAI (i.e., macroblock skiping) as proposed in the core experiments implies, that the length of each stripe can differ. Having in mind that CBP might give no significant gain for intra pictures at low bit rates, and for all kinds of pictures at higher bit rate, it seems obvious to skip CBP, and use the run length coding ability of MUVLC instead.

Q.6.1.6 References

[1] B. Macq, "A universal entropy coder for transform or hybrid coding," Picture Coding Symposium, session 11, Mar. 26-28, 1990, Cambridge, Mass., USA.

[2] H. Tanaka and A. Leon-Garcie, "Efficient run-length encoding," IEEE Trqnsactions on Information Theory, vol. IT-28, no. 6, pp. 880-890, Nov. 1982.

[3] CCIR Study Group - Document MTT-2/31, Mar. 1990, RTT Belgium.

[4] G. Schamel et al., "Experiments with UVLC coding," doc. MPEG 92/289.

[5] Th. Selinger, "Implementation study of a MUVLD," doc. MPEG 92/388.

[6] A. Knoll, "Experiments with UVLC coding," doc. MPEG 92/391.

[7] O. Poncin, B. Maison, "Core experiment I.8 results," doc. MPEG 92/503.

[8] G. Schamel and H. Li, "Some results on frequency scanning using MUVLC", doc. MPEG 92/504.

Q.6.2 Macroblock Switable VLC (AT&T)

In this experiment, the VLC for all coefficients except the Intra-DC coefficient is selected at the macroblock level. The Intra-DC coefficients are encoded as in TM2. For initial experiments the selection is between two VLC tables. The VLC tables are too long to be included in this document but will be supplied on request by:

Barry Haskell
AT&T Bell Laboratories
Room BO 4C-538
101 Crawford Corners Road
Holmdel, NJ
Fax: +1 908 949 3697
Email: bgh@vax135.att.com

Syntax:

Picture Layer

```
picture()
{
...
  vlc_select_enable                    1          ulmsbf
...
}
```

Macroblck Layer

```
macroblock()
{
...
 if (vlc_select_enable) {
   ac_table_select                     1          uimsbf
  }
}
```

Decision Criterion: A posteriori

Assessment Criteria: As described in introduction to Section Q6.

Q.6.3 Alternative VLC (Sony)

In this core-experiment, every Intra MB is coded with a second VLC table designed for Intra coding, and every Inter MB is coded with MPEG1 compatible VLC table as defined in TM2. No syntax modification is required.

The second VLC table designed for Intra coding is follows.

VLC	Run	Level
"00"	0	1
"010"	0	2
"011"	1	1
"1000"	EOB	
"1001"	0	3
"10100"	0	4
"10101"	0	5
"10110"	1	2
"10111"	2	1
"11000"	3	1
"110010"	0	6
"110011"	0	7
"110100"	4	1
"110101"	5	1
"1101100"	0	8
"1101101"	0	9
"1101110"	1	3
"1101111"	2	2
"1110000"	6	1
"1110001"	7	1
"1110010"	8	1
"1110011"	9	1
"1110100"	10	1
"11101010"	ESCAPE	
"11101011"	0	10
"11101100"	0	11
"11101101"	0	12
"11101110"	0	13
"11101111"	1	4
"11110000"	3	2
"11110001"	11	1
"11110010"	12	1
"11110011"	13	1
"111101000"	0	14
"111101001"	0	15
"111101010"	0	16
"111101011"	0	17
"111101100"	1	5
"111101101"	2	3

"111101110"	4	2
"111101111"	14	1
"1111100000"	0	18
"1111100001"	0	19
"1111100010"	0	20
"1111100011"	0	21
"1111100100"	0	22
"1111100101"	1	6
"1111100110"	1	7
"1111100111"	3	3
"1111101000"	5	2
"1111101001"	6	2
"1111101010"	7	2
"1111101011"	15	1
"11111011000"	0	23
"11111011001"	0	24
"11111011010"	0	25
"11111011011"	0	26
"11111011100"	0	27
"11111011101"	0	28
"11111011110"	1	8
"11111011111"	2	4
"11111100000"	4	3
"11111100001"	8	2
"11111100010"	9	2
"11111100011"	10	2
"11111100100"	16	1
"11111100101"	17	1
"11111100110"	24	1
"11111100111"	25	1
"111111010000"	0	29
"111111010001"	0	30
"111111010010"	0	31
"111111010011"	0	32
"111111010100"	0	33
"111111010101"	1	9
"111111010110"	1	10
"111111010111"	1	11
"111111011000"	2	5
"111111011001"	3	4
"111111011010"	5	3
"111111011011"	11	2
"111111011100"	12	2
"111111011101"	13	2
"111111011110"	18	1
"111111011111"	19	1
"111111100000"	20	1
"111111100001"	21	1
"111111100010"	22	1
"111111100011"	23	1
"111111100100"	26	1
"111111100101"	29	1
"111111100110"	30	1
"111111100111"	32	1

"1111111010000"	0	34
"1111111010001"	0	35
"1111111010010"	0	36
"1111111010011"	0	37
"1111111010100"	0	38
"1111111010101"	0	39
"1111111010110"	0	40
"1111111010111"	1	12
"1111111011000"	1	13
"1111111011001"	1	14
"1111111011010"	2	6
"1111111011011"	3	5
"1111111011100"	4	4
"1111111011101"	6	3
"1111111011110"	7	3
"1111111011111"	8	3
"1111111100000"	9	3
"1111111100001"	10	3
"1111111100010"	14	2
"1111111100011"	27	1
"1111111100100"	28	1
"1111111100101"	31	1
"1111111100110"	33	1
"11111111001110"	0	41
"11111111001111"	0	42
"11111111010000"	0	43
"11111111010001"	0	44
"11111111010010"	0	45
"11111111010011"	0	46
"11111111010100"	1	15
"11111111010101"	1	16
"11111111010110"	1	17
"11111111010111"	2	7
"11111111011000"	2	8
"11111111011001"	3	6
"11111111011010"	4	5
"11111111011011"	5	4
"11111111011100"	6	4
"11111111011101"	7	4
"11111111011110"	8	4
"11111111011111"	11	3
"11111111100000"	12	3
"11111111100001"	13	3
"11111111100010"	34	1
"111111111000110"	0	47
"111111111000111"	0	48
"111111111001000"	0	49
"111111111001001"	0	50
"111111111001010"	0	51
"111111111001011"	1	18
"111111111001100"	1	19
"111111111001101"	1	20
"111111111001110"	1	21
"111111111001111"	1	22

"1111111110100000"	2	9
"1111111110100001"	3	7
"1111111110100010"	4	6
"1111111110100011"	5	5
"1111111110100100"	9	4
"1111111110100101"	10	4
"1111111110101101"	13	4
"1111111110101110"	14	3
"1111111110110000"	35	1
"1111111110110001"	36	1
"1111111110110100"	40	1
"1111111110110101"	41	1
"1111111110111000"	0	52
"1111111110111001"	0	53
"1111111110111010"	0	54
"1111111110111011"	0	55
"1111111110111100"	0	56
"1111111110111101"	1	23
"1111111110111110"	1	24
"1111111110111111"	1	25
"1111111111000000"	1	26
"1111111111000001"	1	27
"1111111111000010"	1	28
"1111111111000011"	1	29
"1111111111000100"	2	10
"1111111111000101"	2	11
"1111111111000110"	3	8
"1111111111000111"	4	7
"1111111111001000"	5	6
"1111111111001001"	6	5
"1111111111001010"	7	5
"1111111111001011"	8	5
"1111111111001100"	9	5
"1111111111001101"	11	4
"1111111111001110"	12	4
"1111111111001111"	12	5
"1111111111010000"	13	5
"1111111111010001"	14	4
"1111111111010010"	15	2
"1111111111010011"	16	2
"1111111111010100"	24	2
"1111111111010101"	25	2
"1111111111010110"	37	1
"1111111111010111"	38	1
"1111111111011000"	39	1
"1111111111011001"	42	1
"1111111111011010"	43	1

For any further information on experiment Q6.3 contact:

YAGASAKI yoichi Junet: yagasaki@av.crl.sony.co.jp
(Phone: +81 (03) 3448-5605) (Fax: +81 (03) 3448-5611)

Q.7 Vector Quantisation

Q.7.1 Desciption of Method

1 Definitions
A selection vector is defined as a 64 bit number that indicates which of the 64 quantized coefficients are transmitted to the receiver. A codebook is defined as a set of selection vectors. A selection vector index is defined as an address of a selection vector within the codebook.

2 Introduction
It is proposed that the encoding of the positions and amplitudes of the non-zero DCT coefficients can be efficiently achieved by vector quantization of the coefficient patterns together with variable length coding (VLC) the amplitudes of the coefficients.

In this scheme, either one or four selection vectors are transmitted to represent the selection patterns of the four luma blocks within a macroblock, while one selection vector is transmitted for each of the chroma channels (cb and cr).
Separate variable length code tables are used to encode the amplitudes of the selected DC coefficients and the AC coefficients.

3 Vector Quantization Process
Figure 1 gives an example of the vector quantization process. First a selection vector is chosen that best fits the quantized coefficients by choosing the one with the least error and the lowest encoded bit rate (i.e. lowest "cost", see section 4).

Next, the selection vector is applied to the data. Finally, the selection vector index and the vector quantized coefficients are variable length coded and transmitted to the receiver.

Figure 1: Example of Vector Quantization.
(** see MPEG 92/525, distributed at Tarrytown)

The receiver decodes the selection vector index and applies it to the address the vector codebook to obtain the 64 bit selection vector. The selection vector is then used to place the decoded coefficients in an 8x8 block.

4 Choosing selection vectors.
The selection vectors are picked by applying the selection vectors in the reference codebook to the data to be vector quantized and calculating a cost for each. The selection vector with the minimum cost is chosen.

The cost of a selection vector is defined as follows: (EQ 1)
(** see MPEG 92/525, distributed at Tarrytown)

where the droperror of a selection vector is defined as the sum of the absolute quantized coefficients that are not selected, coefficientBitRate is defined as the number of bits needed to transmit the quantized coefficients that are selected and selectionBitRate is the number of bits to send the selection vector index itself.

Generally values of around 16 for have been found to give good results.

5 Luma Vector Quantization.
A luma macroblock may be encoded using either one luma macroblock selection vector or four luma block selection vectors. A luma macroblock selection vector is a selection vector that is applied to each of the four blocks in a luma macroblock. A luma block selection vector is a selection vector that is applied to only one block of the four luma blocks.

Figure 2: Applying luma macroblock selection vector.
(** see MPEG 92/525, distributed at Tarrytown)

Figure 3: Applying luma block selection vectors.
(** see MPEG 92/525, distributed at Tarrytown)

The cost function given above is used to determine whether one or four selection vectors should be sent for every macroblock by minimizing the cost over a macroblock.

Hence, the cost of sending four selection vectors is defined as: (EQ 2)
(** see MPEG 92/525, distributed at Tarrytown)

while the cost of sending only one selection vector is: (EQ 3)
(** see MPEG 92/525, distributed at Tarrytown)

Generally, sending four selection vectors improves the matching of the vector patterns to the data while sending one vector reduces the selection overhead.

6 Chroma Vector Quantization
There is one selection vector in each macroblock for each of the two chroma channels and adaptive division into chroma block selection vectors is not permitted. In this case, the cost function is calculated over the one (4:2:0) or two (4:2:2) blocks that form the chroma macroblock. Again, intra macroblocks are restricted to the codebook entries that always select the DC coefficient.

7 Coding of selection vector indices.
Four variable length codes are used to encode the selection vector indexes as summarized in the table below:

Table 1: Coding of selection vector indexes.

1 Inter luma index
2 Intra luma index
3 Inter chroma index
4 Intra chroma index
(see appendices)

8 Coding of vector quantized coefficients.
Intra DC coefficients are differentially encoded, as in TM-2.
A 1-d VLC table is provided to code the intra-AC and inter-DC-AC coefficients that are indicated by the selection vector:

Table 2: Coding of coefficients.
(see appendices)

9 Syntax:

Macroblock Layer:
Instead of the coded_block_pattern, insert the following:

```
if (macroblock_intra || macroblock_coded) {
 luma_selection_escape                          1        uimsbf
 if (luma_selection_escape)
   for (i=0; i < 4; i++)
     luma_selection_vector                      1-19     vlclbf
 else
   luma_selection_vector                        1-19     vlclbf

 Cb_selection_vector                                     1-19     vlclbf
 Cr_selection_vector                                     1-19     vlclbf
}
```

Block Layer:

```
block(i)
{
 if (macroblock_intra) {

     ..... same as in TM-2 .....

 }
 else {
   if (codebook[selection_vector][0])
      dc_coeff_nonintra                        1-16   vlclbf
 }

 for (k=1; k < 64; k++) {
   if (codebook[selection_vector][k])
      ac_coeff[k]                                       1-16     vlclbf

}
```

Further, all the selected coefficients are transmitted in raster-scan, (left to right and top to bottom), NOT according to the zigzag scan of TM-2. The selection vectors also follow the raster scan.

10 Selection Codebooks and Variable Length Tables

The selection vectors and the associated VLCs are appended below, and they are also available (by e-mail) from Barry Haskell bgh@vax135.att.com.

(the 1-d table for the VLC coefficients is located at the very end)

INTER-LUMA SELECTION VECTORS and VLC TABLE:

```
  Vector                                   VLC
0 0000000000000000000000000000000000000000000000000000000000000000 0
1 1111110011111100111111001111110011111100111111001111110011111100 10100
```

2 1100000011000000110000001100000011000000110000001100000011000000 111101
3 1111000011110000111100001111000011110000111100001111000011110000 110110
4 0000110000001100000011000000110000001100000011000000110000001100 101110
5 1000000000000000000011100000111000001110000000000000000000000000 100100
6 1111100011111000111110001111100011111000111110001111100011111000 100001
7 1000000000000000000000000000111000001110000011100000000000000000 1111000
8 0000000000000000000000000000000000000000000000000000000000010000000 1110011
9 1000000000000000000000000000011000000110000000000000000000000000 1100101
10 1111111011111100111110001111000011100000110000001000000000000000 1100011
11 1000000000000000000000000000000000000000000000000000000000000000 1100010
12 0000000000000000000000000000000000000000100000000000000000000000 1011011
13 0000000000000000000000000000000000000000000000010000000000000000 1010101
14 1000000000000000000000000000001111000011110000111100001111000000 1001111
15 0000000000000000000000000000000100000000000000000000000000000000 1001101
16 0000000000000000000000010000000000000000000000000000000000000000 1001010
17 1000000000001000000111000111111101111111000111000000100000000000 1000110
18 1000000000000000000000000000000000000011110000111100001111000000 1000001
19 1000000010000000100000001000000010000000100000001000000010000000 11111101
20 1000000000000000000000000000000000000011110000111100001111000011 11110011
21 0000000010000000000000000000000000000000000000000000000000000000 11110010
22 0000000000000000100000000000000000000000000000000000000000000000 11101110
23 1000000000000000000000000011000000110000000000000000000000000000 11101100
24 1000000010000000100000001000000010000000100000001000000000000000 11101011
25 1000000000000000100000001000000010000000100000001000000010000000 11101001
26 1100000011000000000000000000000000000000000000011000000110000000 11101000
27 1000000000000000000000000000000010000000100000001000000010000000 11100100
28 0011000000110000001100000011000000110000001100000011000000110000 11100000
29 1110000011100000111000000000000000000011100000111000001110000000 11011110
30 1000000000000000000000000000000000000000000000001111000011110000 11010001
31 1000000010000000100000001000000010000000100000001000000010000000 11001101
32 1000000000000000000000000000000000000111000001110000011100000000 11000011
33 1011011000110110000000000000000000000000000000000000000000000000 11000001
34 1000000000000000000000000000000000000000000000001111000011110000 10111111
35 1000000000001110000011100000111000000000000000000000000000000000 10110100
36 1000000011100000111000001110000011100000111000001110000000000000 10110010
37 1000111100001111000011110000111100000000000000000000000000000000 10110001
38 0000001100000011000000110000001100000011000000110000001100000011 10110000
39 1100000000000000000000000000000000000000000000000000000000000000 10101111
40 1000000000000000011110000111100001111000011110000000000000000000 10101110
41 1000000000000000000000000000000000000000000000000000000011110000 10011101
42 1000000010000000000000001000000000000000100000000000000010000000 10011001
43 1000000000000000000000000000000000000001000000010000000100000000 10011000
44 1000000000000000000000000000000000000011000000110000000000000000 10001110
45 1000111000001110000011100000000000000000000000000000000000000000 10001000
46 1000000000000000000000001000000010000000100000001000000010000000 11111111
47 1000000000000000000000000000000011000000110000001100000011000000 11111101
48 1110000001100000000000000000000000000000000000000000000000000000 11111000
49 1111000011100000110000001000000010000000100000001000000010000000 11110110
50 1001110000011100000111000000000000000000000000000000000000000000 11110100
51 1110000011000000100000001000000010000000100000001000000010000000 11110010
52 1000000000000000000000000000000000000000000000001000000010000000 11110000
53 1111111011111100100000001000000000000000000000000000000000000000 11010100
54 1000000000000000000000000000000000000000000000011000000011000 11000111
55 1100000010000000010000001000000001000000100000000100000010000000 11000101

56 11110000011100000111000000000000000000000000000000000000000000000000 111000100
57 10000000000000000000000011100000111000001110000011100000000000000000 111000010
58 10110000001100000000000000000000000000000000000000000000000000000000 110111111
59 10111000001110000011100000000000000000000000000000000000000000000000 110111010
60 10000000000000000000000000000011000000110000001100000000000000000000 110111001
61 11000000110000000000000000000000000000000000000000000000000000000000 110101111
62 11100000110000001000000010000000000000000000000000000000000000000000 110101101
63 11111000110000001000000010000000000000000000000000000000000000000000 110101011
64 10000000000000000000000000011100000111000001110000000000000000000000 110101001
65 10000000000000000110011001100110011001100110011000000000000000000000 110100111
66 10000000000000000000000000000000011100000111000001110000000000000000 110100110
67 10000000000000000111000001110000011100000000000000000000000000000000 110100001
68 11000000110000001100000011000000100000001000000100000000100000000000 110011110
69 10011000000110000000000000000000000000000000000000000000000000000000 110011101
70 10000000000000001110000011100000111000001110000011100000000000000000 110011001
71 10000110000001100000000000000000000000000000000000000000000000000000 110011000
72 10000000000000001110000011100000111000000000000000000000000000000000 110010011
73 11111100111110001000000010000000000000000000000000000000000000000000 110010001
74 10001111000000000000000000000000000000000000000000000000000000000000 110010000
75 11000000110000001100000011000000010000001000000001000000100000000000 110000000
76 10000000000000000000000000000000000000000000000000000000000000001100110 101111101
77 11111100111110001111000011100000110000001000000000000000000000000000 101111011
78 10000000000000000000000000000000000000011000000110000000000000000000 101111010
79 10000000000000000000000000000000000000000000000000000011000000011100 101101011
80 11110000110000001000000010000000000000000000000000000000000000000000 101100111
81 10001111000011110000000000000000000000000000000000000000000000000000 101011011
82 10000000111000001110000011100000111000001110000000000000000000000000 101011010
83 10000000000000000000110000001100000000000000000000000000000000000000 101010011
84 10000000110000001100000000000000000000000000000000000000000000000000 100111001
85 10000000000000000000000000000000000000000010000000100000000000000000 100101111
86 10000000000000000000000011100000111000001110000000000000000000000000 100101101
87 10000000100000001000000010000000100000000000000000000000000000000000 100101100
88 10000000000000001100000011000000110000001100000011000000000000000000 100011110
89 10000000000000000000000000000000011100000111000001110000000000000000 100010110
90 10000000100000001000000010000000000000000000000000000000000000000000 100010101
91 10000000000000000000000000000000110000001100000000000000000000000000 100010100
92 10000000000000001000000010000000100000001000000000000000000000000000 100010010
93 10000000000000000000000011000000110000001100000011000000000000000000 100000011
94 10000000000000001100110011001100000000001100110011001100000000000000 100000010
95 10000000000000000000000000000000011000000110000000000000000000000000 100000001
96 10000000000000000000000010000000100000001000000010000000000000000000 111111101
97 10000000000000000000000000000000110000001100000000000000000000000000 111111100
98 10000000110000001100000011000000110000001100000000000000000000000000 111111001
99 10000000000000000000000000000000000000011000000110000000000000000000 111111001
100 10000000000000000000000000000000000000000000000000000111100001111 111110111
101 10000000100000000000000000000000000000000000000000000000000000000000 111110101
102 10000000000000000000000000000000000000000000000000011000000110000 111110101
103 10000000000000000000000000011000000110000000000000000000000000000000 111110011
104 11111100110000001000000010000000000000000000000000000000000000000000 111110001
105 10000000000000000000000011000000110000000000000000000000000000000000 111110001
106 10000000100000001000000000000000000000000000000000000000000000000000 111011111
107 10000000000000000000000000000011100000111000001110000000000000000000 111011101
108 10000000000000000000000000000010000000000000010000000000000000000000 111011100
109 10000000000000000000000010000000100000001000000000000000000000000000 111011011

110 1111111011111100111100001000000000000000000000000000000000000000000000 1110110101
111 1000000011100000111000001110000011100000000000000000000000000000000000 1110110100
112 1110000011100000111000000000000000000000000000000000000000000000000000 1110101011
113 1000000000000000111000001110000011100000111000000000000000000000000000 1110010110
114 1000000000000000000011000000110000000000000000000000000000000000000000 1110010100
115 1000110000001100000000000000000000000000000000000000000000000000000000 1110001100
116 1111100011110000100000001000000000000000000000000000000000000000000000 1110000110
117 1000000000111000001110000011100000000000000000000000000000000000000000 1101111100
118 1000000000000110000001100000000000000000000000000000000000000000000000 1101110110
119 1111111011000000100000001000000000000000000000000000000000000000000000 1101110001
120 1000111100001111000011110000000000000000000000000000000000000000000000 1101011101
121 1000000000000000000000000000000010000000100000000000000000000000000000 1101011100
122 1000000000000000110000001100000011000000110000000000000000000000000000 1101011001
123 1000000000000000000000000000000000000000111000001110000000000000000000 1101010101
124 1000000000000000000000011000000110000001100000000000000000000000000000 1101010100
125 1010000000000000000000000000000000000000000000000000000000000000000000 1101010001
126 1000000000111000001110000011100000000000000000000000000000000000000000 1101001011
127 1000000000000000000000000000000000000010000000000000000000000000000000 1101001010
128 1000000000000000000000000000000011100000111000001110000000000000000000 1101001001
129 1000000011000000110000001100000011000000000000000000000000000000000000 1101001000
130 1000000000000000110000001100000000000000000000000000000000000000000000 1101000000
131 1100000010000000100000001000000010000000100000001000000010000000100000 1100111111
132 1000000000000000000000000011000000110000000000000000000000000000000000 1100111000
133 1110000011100000111000001100000011000000100000001000000000000000000000 1100100101
134 1000000000000000001100000011000000000000000000000000000000000000000000 1100001011
135 1000000000000000001100000011000000000000001100000011000000000000000000 1100001010
136 1000000000000000001110000011100000111000000000000000000000000000000000 1100001001
137 1000000011000000110000001100000000000000000000000000000000000000000000 1100001000
138 1111110011100000111000001110000010000000100000000000000000000000000000 1100000010
139 1000000011000000110000000000000000000000000000000000000000000000000000 1011111001
140 1000000000000000000000001000000010000000000000000000000000000000000000 1011111000
141 1000000001110000011100000111000000000000000000000000000000000000000000 1011110010
142 1000000000000000000000000111000001110000011100000000000000000000000000 1011110001
143 1111111011100000111000001110000011100000110000001000000000000000000000 1011110000
144 1000000000000000000000001110000011100000111000000000000000000000000000 1011010100
145 1000000000000000100000001000000000000000000000000000000000000000000000 1011001100
146 1111111011000000100000001000000010000000100000001000000000000000000000 1010110011
147 1111111011100000111000001110000011100000100000001000000000000000000000 1010110000
148 1000000000000000111000001110000011100000000000000000000000000000000000 1010100011
149 1001100000011000000000000000000000000000000000000000000011000000110001010100010
150 1110011001100110000000000000000000000000000000000000000000000000000000 1010100000
151 1000000000000000100000000000000010000000000000000000000000000000000000 1001110000
152 1000000000000000000000000000000000000000000000011000000110000000000 1001011101
153 1110000011000000110000001100000010000000000000000000000000000000000000 1000111111
154 1110000011000000110000001000000000000000000000000000000000000000000000 1000111110
155 1000000000000000000000001100000011000000000000000000000000000000000000 1000101110
156 1001000000000000000000000000000000000000000000000000000000000000000000 1000100111
157 1000000000000000000000000000000011000000110000000000000000000000000000 1000000000
158 1000000000000000000000000000000000000000000011000000110000000000 11111110011
159 1000000011100000111000001110000000000000000000000000000000000000000000 11111110001
160 1000000000000000000000000000001000000000000000000000000000000000000000 11111001101
161 1000000011100000111000000000000000000000000000000000000000000000000000 11111001100
162 1000000000000000000000000000000011000000110000000000000000000000000000 11101101101
163 1000100000000000000000000000000000000000000000000000000000000000000000 11101010100

164 1000000000000000100000000000000000000000000000000000000000000000000000 11100101110
165 1000000000110000001100000000000000000000000000000000000000000000000000 11100101010
166 1000000000000000011000000110000000000000000000000000000000000000000000 11100011011
167 1111110011100000111000001110000011000000100000000000000000000000000000 11100001111
168 1000000001000000000000000000000000000000000000000000000000000000000000 11100001110
169 1000000000000000100000000000000000000000000000010000000000000000000000 11011111011
170 1110000011000000100000001000000010000000000000000000000000000000000000 11011111010
171 1000000010000000100000001000000010000000100000000000000000000000000000 11011101111
172 1111110011000000100000001000000010000000100000000000000000000000000000 11011100001
173 1000010000000000000000000000000000000000000000000000000000000000000000 11011100000
174 1000000000000000111000001110000000000000000000000000000000000000000000 11010110001
175 1110000011000000100000001000000010000000100000000000000000000000000000 11010100001
176 1110000011100000110000001100000010000000000000000000000000000000000000 11010000011
177 1111100011100000111000001000000010000000000000000000000000000000000000 11010000010
178 1000000000000000000000000000000000000000011000000110000000000000000000 11001111101
179 1000000000011000000110000000000000000000000000000000000000000000000000 11001111100
180 1000000000000000000000011100000111000000000000000000000000000000000000 11001110010
181 1111100011100000111000001100000010000000000000000000000000000000000000 11001001000
182 1000000000000000000000000000000111000001110000000000000000000000000000 10111100110
183 1000000000000000000000001000000000000000000000000000000000000000000000 10110101011
184 1000000000000000000110000001100000000000000000000000000000000000000000 10110011010
185 1000000000000000000000000000000000000000000000001000000000000000000000 10101100101
186 1111111011111100111100001100000000000000000000000000000000000000000000 10101100010
187 1000000000000000100000000000000000000000000000000000000000000000000000 10101001011
188 1110000011100000110000001100000011000000100000001000000000000000000000 10101001010
189 1000000000000000000000000000000000000001000000000000000000000000000000 10101001000
190 1000000000000000000000001000000000000000000000000000000000000000000000 10101000011
191 1110000011000000110000001100000011000000100000001000000000000000000000 10011100010
192 1000000000000000000000000000000000000000000000000000000000000000001111 10001001101
193 1111110011111100011000000100000000000000000000000000000000000000000000 10001001100
194 1000000000000000000000000000000010000000000000001000000000000000000000 10000000011
195 1111110011111100011100000100000000000000000000000000000000000000000000 111111100101
196 1000000000000000100000001000000010000000100000001000000000000000000000 111111100100
197 1000000000000000000000000000000000000000110000001100000000000000000000 111111100001
198 1100000000000000110000000000000011000000000000001100000000000000000000 111111100000
199 1000000000000000000000000000000000000001000000000000000000000000000000 111110111011
200 1000000000000000110000001100000011000000000000000000000000000000000000 111110111001
201 1000000000001100000011000000000000000000000000000000000000000000000000 111110111000
202 1000000000000000000000000000000000000001100110011001100000000000000000 111011111010
203 1100000000000000100000000000000000000000000000000000000000000000000000 111011111001
204 1000000000000000000000000000000010000000000000000000000000000000000000 111011111000
205 1111111011111100110000001000000000000000000000000000000000000000000000 111011011001
206 1000000000011000000110000000000000000000000110000001100000000000000000 111011011000
207 1100000000000000110000000000000011000000000000001000000000000000000000 111010101010
208 1111100011000000100000001000000010000000000000000000000000000000000000 111001011111
209 1000000001000000000000000000000000000000000000000000000000000000000000 111001011110
210 1000001000000000000000000000000000000000000000000000000000000000000000 111001010110
211 1000000000000100000000000000000000000000000000000000000000000000000000 111000110100
212 1000000000000000000000000000000000000000000000000000000000011000011 110111011101
213 1000000000000000000000000000000000000000000000000000000110011000000000 110101100001
214 1100000000000000000000000000000010000000000000001000000000000000000000 110101000001
215 1111110011111100011000001100000000000000000000000000000000000000000000 110101000000
216 1000000000000000010000000000000010000000000000000000000000000000000000 110011100111
217 1000000000000000000000000000000010000000100000001000000000000000000000 110010010011

```
218 1000000000000000000000000000000001000000000000000000000000000000 110010010010
219 1111100011110000111000001100000010000000000000000000000000000000 110000001111
220 1111100011110000111000001100000000000000000000000000000000000000 110000001110
221 1000000000000000010000000000000000000000000000000000000000000000 110000001100
222 1000000000000000000000001000000000000000000000000000000000000000 101111001111
223 1100000000000000100000000000000000000000000001000000000000000000 101101010101
224 1000000000000000000000000000001000000000000000000000000000000000 101101010100
225 1100000000000000000000000000000000000000000001000000000000000000 101100110111
226 1110000011100000110000001000000000000000000000000000000000000000 101011001000
227 1100001111000011000000000000000000000000000000000000000000000000 101011000111
228 1000000000100000000000000000000000000000000000000000000000000000 101011000110
229 1111111011100000111000001100000010000000100000001000000000000000 101010010010
230 1000000000000000011000110110001101100011011000110000000000000000 101010000101
231 1111111011100000111000001110000010000000100000001000000000000000 101010000100
232 1000000000000010000000000000000000000000000000000000000000000000 100111000111
233 1000000000000000000000000000000000000000000001000000000000000000 100111000110
234 1000000000000000000000000000000000000000001000000000000000000000 100101110010
235 1000000000000000000000001100011011000110000000000000000000000000 100101110001
236 1111100011110000111000001000000000000000000000000000000000000000 100101110000
237 1100000000000000100000000000001000000000000001000000000000000000 100010111111
238 1000000000000000000000000000000000000000000000000000001100000011 100010111110
239 1000000000000000000001100000011000000000000000000000000000000000 100000000101
240 1110000011000000100000001000000010000000100000001000000000000000 1111101110101
241 1100000000000000100000000000000000000000000001000000000000000000 1110111110111
242 1100000000000000000000000000001000000000000001000000000000000000 1110111110110
243 1100000000000000100000000000001100000000000001100000000000000000 1110101010111
244 1000000000001000000000000000000000000000000000000000000000000000 1110101010110
245 1000000000000000000000000001000000000000000000000000000000000000 1110010101111
246 1111111011100000100000001000000010000000100000001000000000000000 1110001101010
247 1111100011110000110000001000000000000000000000000000000000000000 1101110111001
248 1111110011100000111000001000000010000000100000000000000000000000 1101011000000
249 1111111011111000100000001000000000000000000000000000000000000000 1100000011011
250 1110000011100000110000001100000010000000100000000000000000000000 1100000011010
251 1100000000000000000000000000000000000000000000000000100000000000 1011110011100
252 1000000000000000000000000000000000000001000000000000000000000000 1011001101100
253 1111111011100000111000001000000010000000100000001000000000000000 1010110010011
254 1100000000000000100000000000000010000000000000010000000000000000 1010110010010
255 1111111011111100111000001000000000000000000000000000000000000000 1010100100111
256 1000000000000000100000001000000010000000000000000000000000000000 1010100100110
257 1000000011000000110000001100000011000000110000001100000000000000 1001011100111
258 1100000000000000110000000000000010000000000000010000000000000000 1000101111011
259 1000000000000000000000010000000000000000000000000000000000000000 1000101111010
260 1100000000000000100000000000000010000000000000010000000000000000 1000101111000
261 1000000000000000000000000000000000000000000000010000000000000000 1000000001001
262 1111110011100000100000001000000010000000100000000000000000000000 1000000001000
263 1110000011000000110000001100000010000000100000000000000000000000 1111101110100l
264 1000000000000000100000000000000010000000000000010000000000000000 1110010101110O
265 1110000011000000110000001000000010000000100000000000000000000000 1110001101011l
266 1000000000000000000000100000000000000000000000000000000000000000 1110001101011O
267 1000000000000000000000000000000000000011000011000000000000000000 1101110111000O
268 1000000000000000000010000000000000000000000000000000000000000000 1101011000001l
269 1000000000000000000000000000000000000000000000000000000010000000 1100111001101l
270 1000000000000000000000000000000000000000000000010000000000000000 1100111001101O
271 1110000011000000110000001000000010000000100000001000000000000000 1100111001100l
```

272 11111110111000001100000010000000100000001000000010000000000000000 11001110011000
273 11111110111000001000000010000000000000000000000000000000000000000 10111100111011
274 11100000110000001100000011000000100000001000000010000000000000000 10111100111010
275 10000000000000001000000000000000100000000000000010000000000000000 10110011011011
276 11111100111000001000000010000000000000000000000000000000000000000 10010111001101
277 11111110111100001000000010000000000000000000000000000000000000000 10001011110011
278 10000000000000000000000000000000000011000000110000000000000000000 10001011110010
279 11110000111000001000000010000000000000000000000000000000000000000 11111011101000 0
280 10000000000000000000000000000000010000000000000000000000000000000 11100101011101 1
281 10000000000000000000000000000000000000000100000000000000000000000 11011101110001 1
282 11111110111000001110000011100000110000001000000010000000000000000 11010110000010 1
283 10000000000000000000000000000000000000000000000000000000000000000 11010110000010 0
284 11000000000000000100000000000000010000000000000000000000000000000 10110011011010 0
285 11111000111000001000000010000000000000000000000000000000000000000 10010111001100 1
286 10000000000000000000000000000000000000000000000010000000000000000 10010111001100 0
287 11111000111000001000000010000000100000000000000000000000000000000 11111011101000 10
288 11000000000000000100000000000000110000000000000100000000000000000 11100101011101 01
289 11111100111000001100000010000000100000001000000000000000000000000 11100101011101 00
290 11111000111000001100000010000000100000000000000000000000000000000 11011101110001 01
291 11000000000000000100000000000000010000000000000011000000000000000 11011101110001 00
292 10000000000000000000000000000100000000000000000000000000000000000 10110011011010 11
293 10000000000000000000000000000001000000000000000000000000000000000 10110011011010 10
294 11000000000000000100000000000000010000000000000010000000000000000 11111011010001 11
295 10000000000000000000000000000000000000000000000000000000100000000 11111011010001 00
296 10000000000000000000100000000000000000000000000000000000000000000 11111011010001 1011
297 10000000000000000000000000000000000000000000000000000000100000000 11111011010001 1010

INTRA-LUMA SELECTION VECTORS and VLC TABLE:

Vector	VLC
0	10000000000000000000000000000000000000000000000000000000000000000 101
1	11000000110000001100000011000000110000001100000011000000 10010
2	10000000100000000000000000000000000000000000000000000000000000000 01001
3	10000000100000001000000010000000100000001000000010000000 00011
4	11100000110000000000000000000000000000000000000000000000000000000 00001
5	11000000111000001110000011100000111000001110000011100000 00000
6	11110000111000001110000011100000111000001100000011000000 111100
7	11111110111111101111111011111110111111101111111010000000 110001
8	10000000100000001000000010000000100000000000000000000000000000000 110000
9	11100000110000001100000011000000100000000000000000000000000000000 100110
10	11110000111100001111000011110000111100001111000011110000 100011
11	11111100111111001111110011111100111111001111110000000000001000000 100010
12	11111000111100001111000011110000111100001111000011100000 100000
13	11000000110000001000000010000000100000000000000000000000000000000 011101
14	11111100111111001111110011111100111111001111110011111100 011011
15	11110000111000001110000011100000111000001110000011100000 011001
16	11000000110000001000000010000000100000001000000010000000 010110
17	11111110111111101111111011111110111111001111110000000000 010101
18	11111000111110001111100011111000111110001111100011100000 010000
19	11111000111110001111100011111000111110001111100011111000 001111
20	11100000111000001110000011100000110000001000000010000000000000 001110
21	11111000111100001110000011100000111000001110000011100000 001000

22 1111111011111000111000001110000011000000110000001100000011000000 000101
23 1110000011100000110000001100000011000000110000001100000011000000 000100
24 1111100011110000111100001111000011110000111000001100000010000000 1111111
25 1111100011100000110000001000000010000000100000001000000010000000 1111110
26 1111000011100000111000001000000000000000000000000000000000000000 1111100
27 1110000011000000110000001100000010000000100000001000000010000000 1111010
28 1111111111111110111111101111111011111110111111101111111010000000 1110111
29 1100000000000000000000000000000000000000000000000000000000000000 1110101
30 1111100111110001111100011111000111100001111000011100000110000000 1110100
31 1111000111100001110000011100000100000000000000000000000000000000 1110011
32 1111000011100000110000001100000011010000110000001100000011000000 1110010
33 1111000111100001101000011000000110000001100000011000000110000000 1110000
34 1111111011111100111110001111100011110000111000001100000010000000 1101111
35 1111100011000000000000000000000000000000000000000000000000000000 1101110
36 1111111111111111111111111111111111111111111111111111111011111110 1101100
37 1111110111111101111111011111100111111100111111001110000110000000 1101011
38 1110000011100000111100001111000011100000111100001110000011110000 1101001
39 1111100111100001111000011110000111000001110000011100000110000000 1100111
40 1110000011100000110000001100000010000000100000000000000000000000 1100110
41 1111111011111100111100001100000000000000000000000000000000000000 1100100
42 1111111011110000100000000000000000000000000000000000000000000000 1001111
43 1111000111110001111100011111000111100000000000000000000000000000 1000011
44 1111111011111100111111001111110011111000111100001110000011100000 0111001
45 1111100111111001111110011111100111111001111110000010000000000000 0111000
46 1111111111111111111111111111111111111111111111111111111111111111 0110100
47 1111000011100000111000001110000011100000111000001100000011000000 0110000
48 1111111011111100111110001111100011111000111100001110000011100000 0100011
49 1101000000001000010000001000110000001100000000000000000000000000 0011010
50 1111111111111111111111111111111101111010001110000001000000000000 0011001
51 1111100111110001111000011100000110000001000000000000000000000000 0011000
52 1111100111111001111110011111100011111000111000000000000000000000 0010111
53 1111111111111111111111111111111101111110011100000000000000000000 0010101
54 1111000111110001111100011111000011100000000000000000000000000000 0010100
55 1110000011000000110000001100000011000000110000001000000010000000 0010011
56 1111111100000000000000000000000000000000000000000000000000000000 0010010
57 1111100011100000111000011110000011000000110000001100000011000000 11111011
58 1111111011111100111110001111100011111100111110001111110011111000 11110111
59 1111110011111000111110001111100011111000111100001111000011110000 11110110
60 1111111111111111111111111000010000000000100010000000000000000000 11101101
61 1110000011000000100000001000000010000000100000001000000011000000 11100011
62 1111111111111111111111111111111011111100111111000111111001111000 11011011
63 1000000000011100000000000011110000000000001111100011111011111111 11011010
64 1111111011111100111111001111110011111000111111001111100011111000 11010100
65 1111000011110000111100001111000011110000111100001111100011111000 11010001
66 1111110011111100111111001111110011111000111110001111000011110000 11010000
67 1111111111111111000000100000101000000100000000000000000000000000 11001010
68 1111110011010000111000001100100010000000101000001000000010000000 10011101
69 1111111011111100111111001111110011111100111111001000000000011000 10011100
70 1111111011111100111111001111100011111100111111001111000011100000 10000101
71 1111100111110001111000011110000111100001111000011110000111100000 01111111
72 1111111111111110111110101000000000000000000000000000000000000000 01111110
73 1111111111111110111111011111110011110100111000001100000010010000 01111101
74 1111100111000001110000011100000110000001000000000000000000000000 01111100
75 1110000011110000111100001111000001100000010000000000000000000000 01111010

76 1111111011111100111110001111000011100000110000001000000000010000 01111001
77 1111111111111111111111101111111011111110111111101111111011000000 01111000
78 1111111011111110111111101111111001111110011111100111111001111100 01101011
79 1111110011111000100000001000000000000000000000000000000000000000 01100011
80 1111110011100000111001001111100011101000110100001100000011000000 01100010
81 1111111011111110111111101111111111111111011111110111111100011100 01011111
82 1100000011100000110000001100000001000000100000001000000010000000 01011110
83 1100000011100000111000001110000011100000111000001110000011100000 01011101
84 1111111111111010111100001111111011011000100010011001000010000000 01010011
85 1111111111111110110010000001110000101000001000001000000010000000 01010001
86 1111111111111111111111101110111111111110000000000100010011010000 01000100
87 1111111111111111111111111111111111111110111111001111100011110000 00110111
88 1111111111111101111100011110000111000001100000011000000111000000 00110110
89 1000000011100000111000001110000011100000111100001110000000000000 00101101
90 1111110011101000111010001110000011100000111000001110000011110000 11111010 0
91 1111111111111111111111111101111100000111000000110000000100000000 11101100 1
92 1111111111111100111010001111000010010100100000001000000010000000 11100010 1
93 1110000011000000111000001111000011110000111101001111000011110000 11100010 0
94 1111111111111111111111101111111011110000000100000001000000010000 11010101 1
95 1111110111111101111110111111100111111100111111001111100011110000 11001011 1
96 1111110111010001101000010001010100000000100100001001000010000000 11001011 0
97 1111111111111111100010011001000111111000000000000000000000000000 10000100 1
98 1111111111110001000001010101010100101000010101000000000010000000 01111011 1
99 1111111111111111000111111111111101111111000011110100011100000011 01101010 1
100 1111111011100000111000011110000011110010100000001000000010000000 01101010 0
101 1111101111100000111000001100000011101100010100000100000010000000 01011100 1
102 1000110000001100000011000000110000001100000011000000110000001100 01011100 0
103 1111111111111111111111101111110101110100000100000010000010000000 01010010 1
104 1111111111111100111101001110111011100000111101000111010001110000 01010000 1
105 1111111111110000110101001011010010000100111000001000000011010000 01010000 0
106 1111111111111111111111101111110011110000111100001110000011001100 01000101 1
107 1111111011001000111100001100010011000100110010001100000011011000 01000101 0
108 1111101111101011110101111111000101011101001110110010000010010010 00101100 1
109 1111111111110100111111101110010011100000111000001111000011110000 00101100 0
110 1000000000000000000011100000111000001110000011100000010000000000 11110101 1
111 1111111111111111111111111111111101111011011100010100010001111000 11110101 0
112 1111111011111110111111100111111001111100011110000111110001110010 0 11101100 1
113 1111111111111000111100101111010010101100100000011001000010001100 11010101 1
114 1111000010010000100000000100000010000110010101000001000011100000 11010101 0
115 1111111111111110111110001110000000000000101000000000000001000000 10000100 1
116 1111100011010000110101001101110011001000110100001110000011010001 01111011 1
117 1011000000011000001111001111111101111111001111000000100000000000 01111011 0
118 1111111111111111111111101011111001101111100111100100110010001110 01010010 0
119 1111111111111111111000001100010011100100111100100100010010100000 11101100 0
120 1111000011000000110100001100000011010000110000001100100011100000 10000100 1
121 1111111111111110111111001111110111110110111101100111010000100000 01010010 1
122 1111110011001110111001101110100011000101110111011111000011110000 01010010 0
123 1111111011111000110010101101110011100000110100001111110011110000 11101100 1
124 1111101111101100111011011110100011101111111011001010001010111111 11101100 0
125 1111111111111101111110001111100011111000111111001111000011110100 10000100 0
126 1110000111110000111100001000001100101000111010001110000011100000 10000100 1
127 1111111111111111111110000111101011000110111100000100100011110100 1 10000100 0

INTER-CHROMA SELECTION VECTORS and VLC TABLE:

Vector	VLC
0	1
1	0110
2	00111
3	00101
4	00010
5	010111
6	010101
7	010011
8	000001
9	0111111
10	0111101
11	0111000
12	0101100
13	0101000
14	0100101
15	0100001
16	0100000
17	0011010
18	0011000
19	0001110
20	0001101
21	0000111
22	0000100
23	0000001
24	01111100
25	01110110
26	01110101
27	01110100
28	01110010
29	01001001
30	01001000
31	01000111
32	01000101
33	00110110
34	00110010
35	00100111
36	00100101
37	00100011
38	00100001
39	00011111
40	00011001
41	00001101
42	00001100
43	00001010
44	011110011
45	011110001
46	011110000
47	011101110
48	010110101
49	010100111
50	010100100

51 1000000001110000011100000111000000000000000000000000000000000000 001101111
52 1111100011111000111110001111100011111000111110001111100011111000 001100111
53 1110000011000000110000001100000010000000000000000000000000000000 001001100
54 1000000000000000000000010000000000000000000000000000000000000000 001001000
55 1000000011000000110000001100000011000000000000000000000000000000 001000001
56 1111000011110000111100001111000011110000111100001111000011110000 000110000
57 1000000000000000011000000110000000000000000000000000000000000000 000000011
58 1111100011100000111000001000000010000000000000000000000000000000 000000010
59 1000000000000000000000001100000011000000000000000000000000000000 000000001
60 1111110011100000111000001000000010000000100000000000000000000000 0111110110
61 1000111100000000000000000000000000000000000000000000000000000000 0111110101
62 1000000011100000111000001110000011100000000000000000000000000000 0111110100
63 1100000000000000010000000000000000000000000000000000000000000000 0111100101
64 1110000011000000100000001000000010000000000000000000000000000000 0111011110
65 1000000000000000000000001000000010000000000000000000000000000000 0111001101
66 1000000000000000000000001100000011000000000000000000000000000000 0101101101
67 1111100011110000110000001000000000000000000000000000000000000000 0101101100
68 1000000000000000100000000000000010000000000000000000000000000000 0101101001
69 1111110011100000111000001110000010000000100000000000000000000000 0101001101
70 1001110000011100000111000000000000000000000000000000000000000000 0101001100
71 1111110011000000100000001000000010000000100000000000000000000000 0100011000
72 1000000010000000100000001000000010000000100000000000000000000000 0100010011
73 1000000000000000000000000000000010000000000000000000000000000000 0100010010
74 1000000000011000000110000000000000000000000000000000000000000000 0100010001
75 1000000000000000111000001110000011100000000000000000000000000000 0011011101
76 1000000000000000100000001000000010000000100000000000000000000000 0011011100
77 1000000011100000111000001110000000000000000000000000000000000000 0011001100
78 1111110011111000111000001000000000000000000000000000000000000000 0010011010
79 1110011001100110000000000000000000000000000000000000000000000000 0010001011
80 1000000000110000001100000000000000000000000000000000000000000000 0010001001
81 1000000010000000000000001000000000000000100000000000000010000000 0010001000
82 1000000011100000111000001110000011100000111000000000000000000000 0010000000
83 1111111011110000100000001000000000000000000000000000000000000000 0001111011
84 1000000000000000011000000110000000000000000000000000000000000000 0001111010
85 1000000000000000011100000111000001110000000000000000000000000000 0001100011
86 1110000011000000100000001000000010000000100000000000000000000000 0001100010
87 1000000000000000000000001000000010000000100000000000000000000000 0000101111
88 1000000010000000100000001000000010000000100000001000000000000000 0000101101
89 1000000000000000111000001110000000000000000000000000000000000000 0111110111 1
90 1000000000000000000000000000000010000000100000000000000000000000 0111100100 1
91 1111100011110000111000001000000000000000000000000000000000000000 0111100100 0
92 1000000000000000110000001100000011000000000000000000000000000000 0111011111 1
93 1111110011111100111111001111110011111100111111001111110011111100 0111001111 1
94 1000000000000000000000000000000011000000110000000000000000000000 0111001111 0
95 1000000000000000111000001110000011100000111000000000000000000000 0111001110 0
96 0011000000110000001100000011000000110000001100000011000000110000 0111001100 0
97 1110000011100000110000001100000010000000000000000000000000000000 0101101111 1
98 1000000000111000001110000011100000000000000000000000000000000000 0101101111 0
99 1100000000000000010000000000000010000000000000001000000000000000 0101101110 1
100 1000000011000000110000001100000011000000110000000000000000000000 0101101110 0
101 1111100011000000100000001000000010000000000000000000000000000000 0101101000 1
102 1100000010000000010000001000000001000000100000000100000010000000 0101101000 0
103 1100000011000000110000001100000001000000100000000100000010000000 0101001011 0
104 1111110011100000111000001110000011000000100000000000000000000000 0101001010 0

105 10001100000011000000000000000000000000000000000000000000000000000000 01000110111
106 10000000000000000000000001110000011100000111000000000000000000000000 01000110110
107 11000000000000001100000000000000110000000000000100000000000000000000 01000110101
108 11111110111111001111000011000000000000000000000000000000000000000000 01000110010
109 10000000000000000000000000000000000000010000000000000000000000000000 01000100001
110 11111110111000001000000010000000000000000000000000000000000000000000 00110011011
111 11000000000000000000000000000000010000000000000010000000000000000000 00110011010
112 11111110110000001000000010000000100000001000000010000000000000000000 00100110110
113 11111000111000001110000011000000100000000000000000000000000000000000 00100100111
114 10000000000000000000000000000000000000010000000100000000000000000000 00100100110
115 10000100000000000000000000000000000000000000000000000000000000000000 00100100101
116 11110000111000001100000010000000100000001000000010000000100000000000 00100010101
117 11111110111000001110000011100000111000001000000010000000000000000000 00100000011
118 10000000000000001100000011000000110000001100000000000000000000000000 00100000010
119 10000000000000000000000011000000110000001100000000000000000000000000 00011110011
120 11100000110000001100000011000000110000001000000010000000000000000000 00011110001
121 10000000000000000000000000000001100000011000000000000000000000000000 00011110000
122 10000000000000000111000001110000011100000000000000000000000000000000 00001011001
123 11111000111100001110000011000000100000000000000000000000000000000000 00001011000
124 10000000000000000100000000000000010000000000000000000000000000000000 00000000011
125 11111100111110001100000010000000000000000000000000000000000000000000 00000000010
126 10000000000000000010000000000000000000000000000000000000000000000000 00000000001
127 11111100111110001110000011000000000000000000000000000000000000000000 01111011101
128 10000000000000000000000001100000011000000000000000000000000000000000 01110111100
129 10000000100000010000000010000001000000001000000100000000100000001110 01110011010
130 11000000110000001100000011000000110000001100000011000000110000001110 01110010010
131 10000000000000000000000011000000110000001100000011000000000000000000 01010010111
132 11000000110000001100000011000000100000001000000100000000100000001110 01000110100
133 10000000000000000000000000000000000000011000000110000000000000000000 01000110100
134 10000000000000001000000010000000100000001000000010000000100000001110 01000110011
135 10000000000000000011000000110000000000000000000000000000000000000000 01000110011
136 11111110111000001110000011100000111000001100000010000000000000000000 01000100000
137 10000000000000000000000000000000000000000000000010000000100000001110 00100110111
138 10000000000000000000000011100000111000001110000000000000000000000000 00100100100
139 10000000000000000000000000000000011000000110000000000000000000000000 00100100100
140 10000000000000000000000000000000111000001110000011100000000000000000 00100010100
141 10000000000000000000000011100000111000000000000000000000000000000000 00100010100
142 11100000111000001100000011000000100000001000000000000000000000000000 00001011101
143 10001111000011111000000000000000000000000000000000000000000000000000 00001011101
144 10000000000000000000000010000000100000001000000010000000000000000000 00001011100
145 11100000111000001110000000000000000000001110000011100000111000000000 00001011100
146 10000000000000001110000011100000111000001110000011100000000000000000 00000000001
147 10000000000000000000000000000000000000010000000000000000000000000000 00000000000
148 11111000111100001110000011000000000000000000000000000000000000000000 01111101110 00
149 10000000111000001110000011100000111000001110000011100000000000000000 01110011101 10
150 11100000110000001100000010000000100000001000000000000000000000000000 01110011001 10
151 11111110111000001110000011100000100000001000000010000000000000000000 01010010111 10
152 11100000111000001110000011000000110000001000000010000000000000000000 01010010101 01
153 11100000111000001100000011000000110000001000000010000000000000000000 01010010101 00
154 11111000111000001000000010000000100000000000000000000000000000000000 01010010101 11
155 10001110000011100000111000000000000000000000000000000000000000000000 01010010101 10
156 10000000000111000001110000011100000000000000000000000000000000000000 01000100000 11
157 10000000100000001000000010000000100000001000000010000000100000000000 01000100000 10
158 10000000000000000000000000000011000000110000000000000000000000000000 00011110010 11

159 1000000000000000000000000000000001000000000000000000000000000000 0001111001010
160 1000000011000000110000001100000011000000110000001100000000000000 01111101110011
161 1100000011000000000000000000000000000000000000011000000011000000 01111101110010
162 1000000000000000000000000000000111000001110000000000000000000000 01110111110111
163 1111111011100000111000001000000010000000100000001000000000000000 01110111110110
164 1000000000000000110000001100000011000000110000001100000000000000 01110111110101
165 1000000000000000000000000000000100000001000000010000000100000000 01110011001110
166 1100000000000000010000000000000010000000000000000000000000000000 01010010111111
167 1000001000000000000000000000000000000000000000000000000000000000 01010010111110
168 1110000011000000100000001000000010000000100000001000000000000000 00100110111010
169 1110000011100000110000001000000000000000000000000000000000000000 00100110111001
170 1000000000000000000000000000000011000000110000001100000011000000 00100110111000
171 1000000000000000000000011100000111000001110000011100000000000000 00011110010011
172 1000000000000000011100000111000001110000000000000000000000000000 00011110010010
173 1100001111000011000000000000000000000000000000000000000000000000 00011110010001
174 1111111011100000100000001000000010000000100000001000000000000000 01110011011101
175 1000111100001111000011110000000000000000000000000000000000000000 01110011011100
176 1000000000000000000000000000000011100000111000001110000000000000 01110011011111
177 1000000000000000001100000011000000000000001100000011000000000000 01110011011110
178 1100000010000000100000001000000010000000100000001000000010000000 01110111101001
179 1000000000000000000000000000000000000000110000001100000000000000 01110111101000
180 1000000000000000000000000000000000000000100000000000000000000000 01110011001111
181 1110000011000000110000001000000010000000100000001000000000000000 00100110111011
182 1000000000000000000000011100000111000001110000000000000000000000 00100110111010
183 1000000000000000000011000000110000000000000000000000000000000000 00011110010001
184 1000000000000000000000000000000000000011100000111000000000000000 00011110010000
185 1100000000000000110000000000000011000000000000001100000000000000 01110011001111001
186 1000000000000110000001100000000000000000000000000000000000000000 01110011001111000
187 1000000000000000000000000010000000000000000000000000000000000000 01110011001111011
188 0000000000000000000000000000000000000000000000010000000000000000 01110011001111010

INTRA-CHROMA SELECTION VECTORS and VLC TABLE:

Vector	VLC
0 1000000000000000000000000000000000000000000000000000000000000000	1
1 1000000010000000000000000000000000000000000000000000000000000000	0110
2 1000000010000000100000000000000000000000000000000000000000000000	0000
3 1100000000000000000000000000000000000000000000000000000000000000	00111
4 1000000010000000100000001000000000000000000000000000000000000000	00110
5 1000000010000000100000001000000010000000000000000000000000000000	011110
6 1110000011100000110000000000000000000000000000000000000000000000	010110
7 1100000011000000100000000000000000000000000000000000000000000000	010101
8 1100000011000000000000000000000000000000000000000000000000000000	010011
9 1100000011000000100000001000000010000000000000000000000000000000	000110
10 1100000010000000000000000000000000000000000000000000000000000000	000101
11 1110000011000000110000001100000000000000000000000000000000000000	000100
12 1110000011000000000000000000000000000000000000000000000000000000	0111010
13 1100000011000000100000001000000000000000000000000000000000000000	0101111
14 1100000011000000110000001000000010000000000000000000000000000000	0101001
15 1000000010000000100000001000000010000000100000000000000000000000	0100010
16 1110000011100000110000001100000010000000000000000000000000000000	0010101
17 1111000011000000000000000000000000000000000000000000000000000000	0010100
18 1110000011000000110000001100000010000000000000000000000000000000	0010010

19 111000001100000011000000110000001100000010000000100000000000000 01111111
20 101000000000000000000000000000000000000000000000000000000000000 01110111
21 100000000100000000000000000000000000000000000000000000000000000 01110010
22 111100001111000011100000110000000000000000000000000000000000000 01110000
23 111110001111000010000000100000000000000000000000000000000000000 01011101
24 111111001110000011100000111000001000000010000000000000000000000 01000011
25 111100001111000011110000111000000000000000000000000000000000000 01000010
26 111000001110000011000000100000000000000000000000000000000000000 01000001
27 111000001100000010000000000000000000000000000000000000000000000 00101111
28 100000001100000011000000110000000000000000000000000000000000000 00101101
29 110000001100000010000000100000001000000010000000000000000000000 00100010
30 111110001111000011000000110000001000000000000000000000000000000 00100000
31 100000001000000010000000100000001000000010000000100000000000000 00011111
32 111110001100000000000000000000000000000000000000000000000000000 00011110
33 100000001100000011000000000000000000000000000000000000000000000 00011100
34 111000001100000011110000111000000100000001000000000000000000000 011111101
35 111000001111000011110000111100000110000000000000000000000000000 011111011
36 111000001100000011000000100000001000000010000000000000000000000 011111010
37 111111001100000000000000000000000000000000000000000000000000000 011111000
38 111111101110000011100000111000001110000010000000100000000000000 011101100
39 111000001110000011000000110000001100000010000000100000000000000 011100111
40 111111001111110011111000111000001110000011000000100000000000000 011100110
41 111110001111000011110000111100001000000000000000000000000000000 010111001
42 111000001110000011100000110000001100000010000000100000000000000 010111000
43 111000001100000011000000110000001000000010000000000000000000000 010100011
44 111110001110000010000000000000000000000000000000000000000000000 010100010
45 111111001111100011111000111100001111000011100000110000001000000 010100001
46 111000001110000011100000000000000000000000000000000000000000000 010100000
47 111000001100000010000000100000001000000000000000000000000000000 010010111
48 111111001111100010000000100000000000000000000000000000000000000 010010101
49 111110001111000011110000111100001111000011110000111000001100000 010010100
50 111100001110000011100000111000001110000011000000110000001000000 010010011
51 111111101111111001000000100000000000000000000000000000000000000 010010001
52 110000001110000011000000110000000100000000000000100000000000000 010010000
53 111111001111100011111000111100001111000010000000000000000000000 010001110
54 111110001111100011111000111100001110000000000000000000000000000 010001101
55 111111111111110011101000100100001000000010000000100000001000000 010000000
56 111000001110000011000000110000001000000010000000000000000000000 001011101
57 111100001110000011000000100000000000000000000000000000000000000 001011100
58 111111001110000011100000111000001100000010000000000000000000000 001011001
59 110000001000000000000000100000000000000000000000000000000000000 001011000
60 110000001100000010000000100000001000000010000000100000000000000 001001111
61 101000001000000000000000000000000000000000000000000000000000000 001001101
62 111110001111000011110000111100001111000011100000110000001000000 001001100
63 111110001111100011111000111110001111000000000000000000000000000 001000111
64 111110001110000011100000100000001000000000000000000000000000000 001000110
65 111111110000000000000000000000000000000000000000000000000000000 0111111000
66 111100001110000011100000100000000000000000000000000000000000000 0111110011
67 111110001111100011110000111100001100000000000000000000000000000 0111011011
68 111111001111100011100000110000001100000010000000000000000000000 0111011010
69 111100001100000010000000000000000000000000000000000000000000000 0111000101
70 111111111111000011110000110000001000000010000000100000001000000 0100101100
71 111111001111100011110000111000001100000010000000000000000000000 0100100101
72 111111001110000010000000000000000000000000000000000000000000000 0100100100

73 1111111011111100111100001000000000000000000000000000000000000000000000 0100011111
74 1111000011100000111000001100000011010000110000001100000011000000 0100011000
75 1111110011000000100000001000000010000000100000000000000000000000 0100000011
76 1111110011110000111000001100000010000000100000000000000000000000 0100000010
77 1111100011100000110000001000000010000000000000000000000000000000 0010011101
78 1111111011110000100000000000000000000000000000000000000000000000 0010000111
79 1111100011010000110000001100000010000000100000001000000010000000 0010000100
80 1000000011000000110000001100000011000000000000000000000000000000 0001110111
81 1100000011000000101000001000000000000000000000000000000000000000 0001110110
82 1111100011100000100000001000000010000000000000000000000000000000 0001110101
83 1111100011100000111100001111000011000000110000001100000011000000 0001110100
84 1110000011000000110000001100000011000000110000001000000010000000 01111110010
85 1110000011000000110000001100000010000000100000001000000010000000 01111100101
86 1000000010000000100000001000000010000000100000001000000010000000 01111100100
87 1000000011100000111000001110000000000000000000000000000000000000 01110001111
88 1000000011100000111000001110000011100000000000000000000000000000 01110001110
89 1000000011000000000000001100000000000000000000000000000000000000 01110001100
90 1111000011100000111000001110000011100000110000001100000011000000 01110001001
91 1111110011111000110000001000000000000000000000000000000000000000 01001011011
92 1111100011110000111000001100000010000000000000000000000000000000 01000110011
93 1111110011100000100000001000000010000000100000000000000000000000 01000110010
94 1100000011000000110000001100000010000000100000001000000001000000 01000111101
95 1111111011000000000000000000000000000000000000000000000000000000 01000111100
96 1111110011110000111000001110000011000000110000001100000011000000 00100111001
97 1110000011100000110000001100000011000000110000001100000011000000 00100111000
98 1111110011110000111000001110000011100000111000001100000010000000 00100001011
99 1110000011000000110000001100000011000000110000001100000010000000 00100001010
100 1000000011100000111000001110000011100000111000000000000000000000 00100001101
101 1111110011111000111000001000000000000000000000000000000000000000 00100001100
102 1111100011110000110000001100000011000000110000001100000011000000 011111100110
103 1111100011100000111000001100000010000000000000000000000000000000 011100011010
104 1100000010000000110000001110000011000000100000001000000000000000 011100010001
105 1100000010000000010000001010000000000000100000000000000000000000 011100010000
106 1000000011000000110000001100000011000000110000000000000000000000 010010110100
107 1111110011111000111000001100000000000000000000000000000000000000 0111111001110
108 1111100011100000100000001000000010000000100000001000000010000000 0111000110111
109 1111000011100000111000000000000000000000000000000000000000000000 0111000110110
110 1111000011000000110000001100000011000000110000001100000011000000 0100101101010
111 1111110011110000111100001110000011100000111000001100000010000000 0111111001111
112 1111111011000000111000001000000001000000010000000000000000000000 0111111001110
113 1111100011000000100000000010000000000000000000000000000000000000 0100101101011
114 1111111011000000100000001000000000000000000000000000000000000000 0100101101011
115 1111000010000000000000000100000010000010010000000000000000000000 0100101101011

1-D COEFFICIENT VLC TABLE (for INTRA AC, and INTER DC and AC):

-255 to
-11 0000001xxxxxxxx1
-10 000000001111101
-9 00000000111101
-8 0000000011101
-7 000000001101

-6	00000000101
-5	000000011
-4	0000011
-3	000011
-2	00011
-1	001
0	1
1	01
2	00010
3	000010
4	0000010
5	000000010
6	00000000100
7	000000001100
8	0000000011100
9	00000000111100
10	000000001111100
11	0000001xxxxxxxx0

to 255

The magnitude of the coefficient is coded in 8 bits where the "x"s appear.
Larger ranges can be accommodated by increasing the "x" field to 9 bits etc.
The last bit is always the sign bit except for -1, 0 and +1.

Q.7.2 Assessment Criteria

Comparsions proposed to be done as discussed in Tarrytown
:
1) Fixed Quantizer (MQUANT=10), compare # of bits:
Also repeat for other MQUANT values of interest.

- TM2 + Coefficient_drop_with_VQ_rule + Zigzag_runlength_coding
vs.
- TM2 + Coefficient_drop_with_VQ_rule + VQ_coding
vs.
- TM2 (no coefficient dropping)

This compares the coding efficiency of VQ_coding
(used in an optimum way, i.e., with VQ coefficient dropping),
with Zigzag_runlength_coding of the same coefficients.

The last comparison provides a measure of the "inefficiency" of TM2.

2) Fixed Bitrate (4 Mb/s), compare subjective quality:
Also repeat for other bitrates of interest.

- TM2 + Coefficient_drop_with_VQ_rule + VQ_coding
vs.
- TM2 + Coefficient_drop_with_VQ_rule + Zigzag_runlength_coding
vs.
- TM2 + Zigzag_runlength_coding (normal TM-2)

This will compare the subjective quality that can be achieve with

VQ with the best subjective quality that can be achieve with the current TM-2 syntax (with and without Coefficient_drop_with_VQ_rule).

If (TM2 + Coefficient_drop_with_VQ_rule + VQ_coding) is significantly better than the two other, this would indicate that VQ_coding is efficient.

If (TM2 + Coefficient_drop_with_VQ_rule + Zigzag_runlength_coding) is better than (normal TM-2), this shows that Coefficient_drop_with_VQ_rule is a good encoder idea to eliminate some DCT coefficients.

If (TM2 + Coefficient_drop_with_VQ_rule + Zigzag_runlength_coding) is NOT better than (normal TM-2), this shows that Coefficient_drop_with_VQ_rule only works well with VQ because it is dropping coefficient that make VQ coding more efficient.