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

Document, AVC-326
July 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/.....

Title: **Test Model 2 Erratum**

Status: **Draft**

Source: **Test Model Editing Committee**

Editorial note

This document lists additions and corrections on Test Model 2 Draft Revision 1 of omissions and errors found so far. Readers are invited to make further suggestions for improvement.

The sections below will replace 4.4.2 to 4.4.4. In TM2 sections 4.4.5 and 4.4.6 will be renumbered to 4.4.6 and 4.4.7.

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

Odd Field Slice Multiplexing

SIF Odd
CCIR 601 Odd
SIF Odd
CCIR 601 Odd

Even Field Slice Multiplexing

SIF Even
CCIR 601 Even
SIF Even
CCIR 601 Even

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

Odd Field Slice Multiplexing

SIF Odd
CCIR 601 Odd
SIF Odd
CCIR 601 Odd

Even Field Slice Multiplexing

CCIR 601 Even
CCIR 601 Even

4.4.4 Slices in a PictureCompatibility Experiment G.3

Frame Slice Multiplexing

HHR Frame
CCIR 601 Frame
HHR Frame
CCIR 601 Frame

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

I.8 Core Experiment on 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 = last 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=0; 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++) {
      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 0000')
  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

This text will replace the text in Appendix K: Motion Compensation for Simplified FAMC. Figures have been added and the explanation is improved.

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.

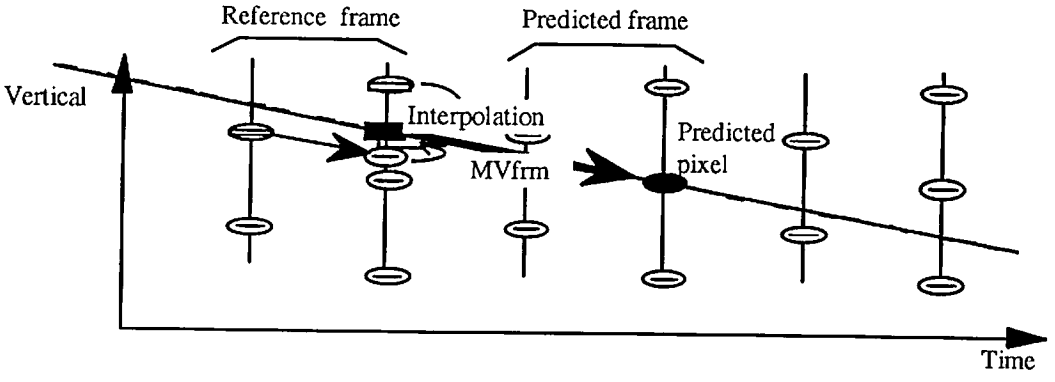

Figure K.0 Principle of FAMC (Vertical direction)

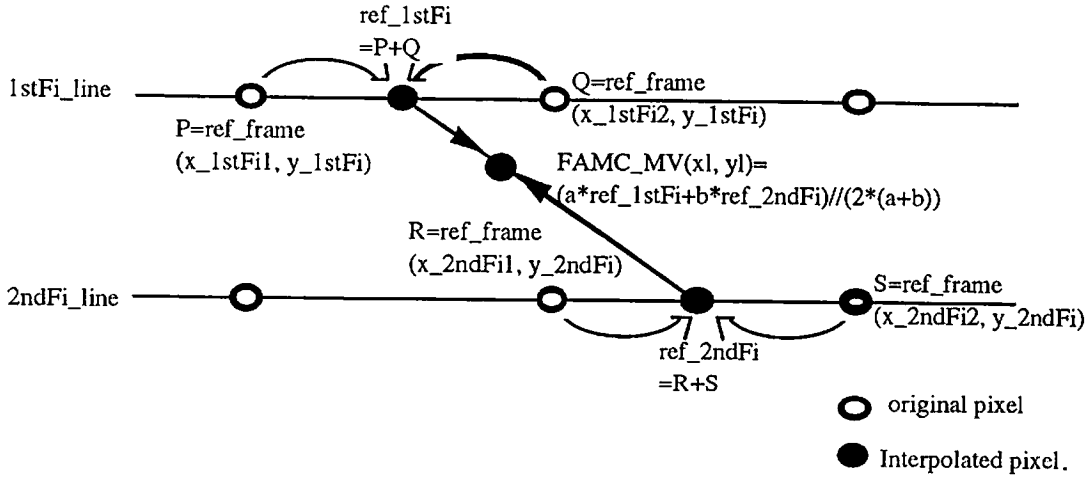

Figure K.1: FAMC prediction for 1stFi line

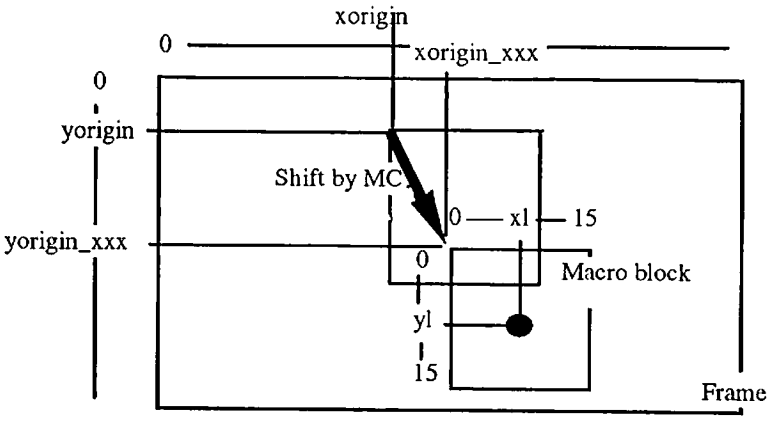

Figure K.2: The relation of variables

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.

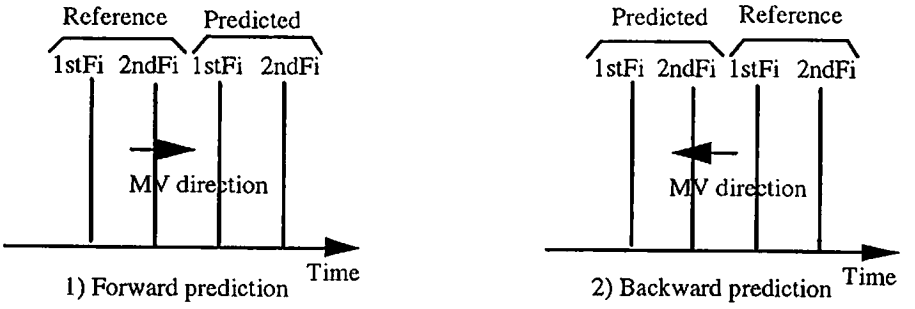

Figure K.3 Difference of forward and backward prediction in FAMC

```
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 + (N*FAMC_MVx)//8)/2
xorigin_2ndFi2 = xorigin + (2*FAMC_MVx + (N*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 - (N*FAMC_MVx)//8)/2
xorigin_1stFi2 = xorigin +(2 * FAMC_MVx - (N*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)
    }
  }
}
```

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	0	4	4
4.5	0	6	4
5.0	0	6	4
5.5	4 + 0	6	4
6.0	2	6	6
6.5	4	6	8
7.0	4	8	8
7.5	4	8	8
8.0	0	0	8
8.5	0	0	10
9.0	0	0	10
9.5	8 + 0	8 + 2	10
10.0	2	2	10
10.5	4	2	10
11.0	4	2	12
11.5	4	2	12
12.0	0	4	
12.5	0	6	
13.0	0	6	
13.5	12+ 0	6	12+
14.0	2	6	
14.5	4	6	
15.0	4	8	
15.5	4	8	
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	1	3	3
4.5	1	3	5
5.0	1	3	5
5.5	2 + 1	3	5
6.0	1	5	5
6.5	1	5	5
7.0	1	5	5
7.5	1	5	5
8.0	1	1	7
8.5	1	1	7
9.0	1	1	7
9.5	4 + 1	6 + 1	7
10.0	1	1	9
10.5	1	3	9
11.0	1	3	9
11.5	1	3	9
12.0	1	3	
12.5	1	3	
13.0	1	3	
13.5	6 + 1	3	10+
14.0	1	5	
14.5	1	5	
15.0	1	5	
15.5	1	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	1	5	5
4.5	1	7	5
5.0	1	7	5
5.5	6 + 1	7	5
6.0	3	7	7
6.5	5	9	9
7.0	5	9	9
7.5	5	9	9
8.0	1	1	9
8.5	1	1	11
9.0	1	1	11
9.5	12 + 1	10 + 1	11
10.0	3	3	11
10.5	5	3	13
11.0	5	3	13
11.5	5	3	13
12.0	1	5	
12.5	1	7	
13.0	1	7	
13.5	18 + 1	7	14+
14.0	3	7	
14.5	5	9	
15.0	5	9	
15.5	5	9	
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	0	4	4
4.5	0	4	6
5.0	2	4	6
5.5	4 + 2	4	6
6.0	2	6	6
6.5	2	6	6
7.0	2	6	6
7.5	4	8	6
8.0	0	0	8
8.5	0	0	8
9.0	2	2	8
9.5	8 + 2	8 + 2	10
10.0	2	2	10
10.5	2	4	10
11.0	2	4	10
11.5	4	4	12
12.0	0	4	
12.5	0	4	
13.0	2	4	
13.5	12 + 2	4	12+
14.0	2	6	
14.5	2	6	
15.0	2	6	
15.5	4	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, b	2 a, b	3 a, b
-1.0	3, 5	2, 6	1, 7
-0.5	5, 3	4, 4	4, 4
0.0	8, 0	8, 0	8, 0
0.5	5, 3	4, 4	4, 4
1.0	3, 5	2, 6	1, 7
1.5	1, 7	2, 6	3, 5
2.0	8, 0	8, 0	8, 0
2.5	1, 7	6, 2	5, 3
3.0	3, 5	3, 5	3, 5
3.5	5, 3	2, 6	0, 8
4.0	8, 0	8, 0	8, 0
4.5	5, 3	2, 6	6, 2
5.0	3, 5	3, 5	4, 4
5.5	1, 7	6, 2	2, 6
6.0	8, 0	8, 0	8, 0
6.5	1, 7	2, 6	2, 6
7.0	3, 5	2, 6	4, 4
7.5	5, 3	4, 4	6, 2
8.0	8, 0	8, 0	8, 0
8.5	5, 3	4, 4	0, 8
9.0	3, 5	2, 6	3, 5
9.5	1, 7	2, 6	5, 3
10.0	8, 0	8, 0	8, 0
10.5	1, 7	6, 2	3, 5
11.0	3, 5	3, 5	1, 7
11.5	5, 3	2, 6	4, 4
12.0	8, 0	8, 0	
12.5	5, 3	2, 6	
13.0	3, 5	3, 5	
13.5	1, 7	6, 2	
14.0	8, 0	8, 0	
14.5	1, 7	2, 6	
15.0	3, 5	2, 6	
15.5	5, 3	4, 4	
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, b	2 a, b	3 a, b
-1.0	3, 5	2, 6	1, 7
-0.5	3, 5	3, 5	4, 4
0.0	8, 0	8, 0	8, 0
0.5	3, 5	3, 5	4, 4
1.0	3, 5	2, 6	1, 7
1.5	6, 2	5, 3	5, 3
2.0	8, 0	8, 0	8, 0
2.5	6, 2	1, 7	1, 7
3.0	3, 5	3, 5	3, 5
3.5	3, 5	6, 2	5, 3
4.0	8, 0	8, 0	8, 0
4.5	3, 5	6, 2	1, 7
5.0	3, 5	3, 5	4, 4
5.5	6, 2	1, 7	6, 2
6.0	8, 0	8, 0	8, 0
6.5	6, 2	5, 3	6, 2
7.0	3, 5	2, 6	4, 4
7.5	3, 5	3, 5	1, 7
8.0	8, 0	8, 0	8, 0
8.5	3, 5	3, 5	5, 3
9.0	3, 5	2, 6	3, 5
9.5	6, 2	5, 3	1, 7
10.0	8, 0	8, 0	8, 0
10.5	6, 2	1, 7	5, 3
11.0	3, 5	3, 5	1, 7
11.5	3, 5	6, 2	4, 4
12.0	8, 0	8, 0	
12.5	3, 5	6, 2	
13.0	3, 5	3, 5	
13.5	6, 2	1, 7	
14.0	8, 0	8, 0	
14.5	6, 2	5, 3	
15.0	3, 5	2, 6	
15.5	3, 5	3, 5	
16.0			
16.5	REPEATED		
17.0			

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

This text will replace the text in Appendix L of Core Experiment No.2

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.

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.

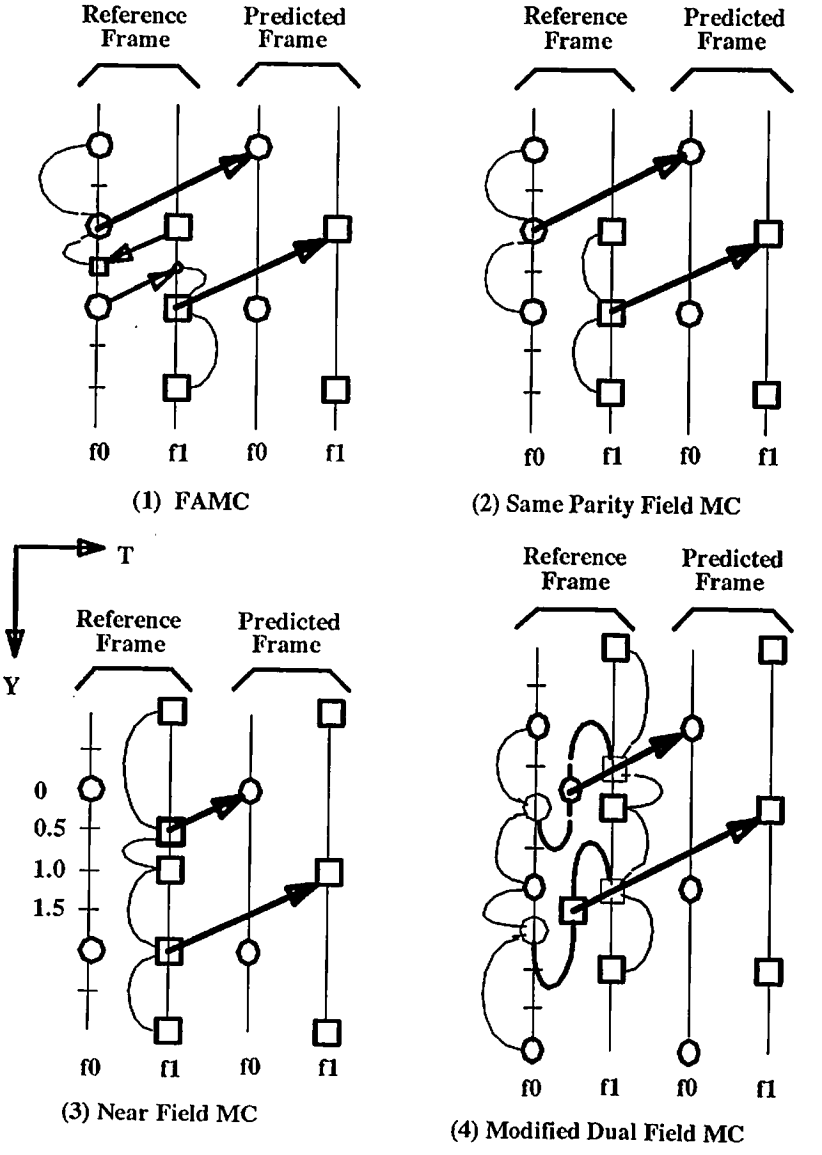

Figure L1 SVMC(Single Vector Motion Compensation) for frame picture

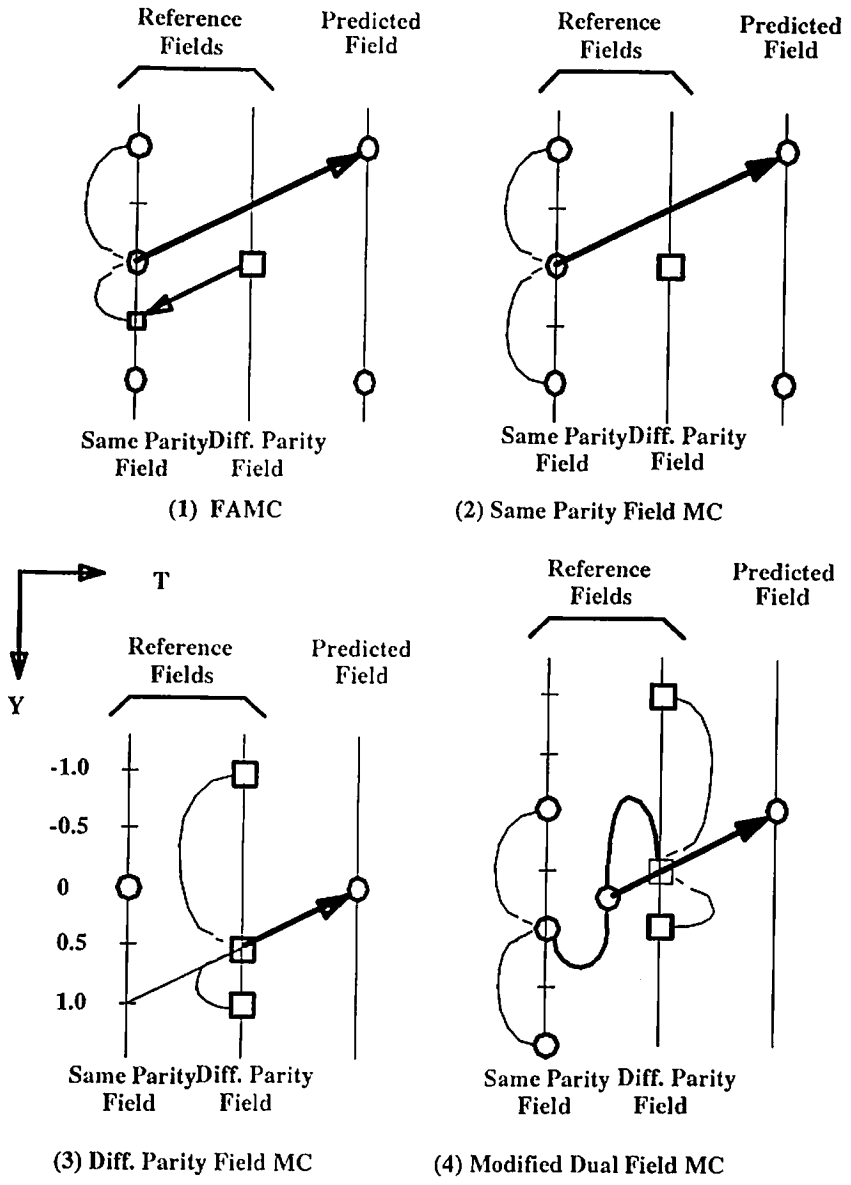

Figure L2 SVMC(Single Vector Motion Compensation) for field picture

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

Corrections on the following pages have to be made.

Page 18: The order of blocks in a 4:2:2 MB is incorrect. The proper text should read:

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

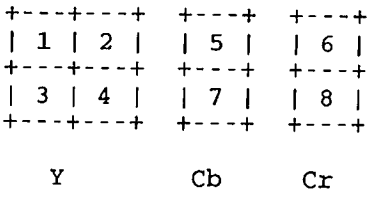

Figure 4.4b: 4:2:2 Macroblock structure

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

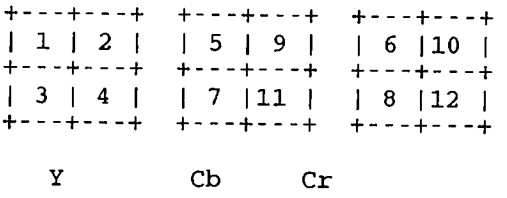

Figure 4.4c: 4:4:4 Macroblock structure

Page 29, a typo was found. The text should read:

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

Page 31, a type was found. The text should read:

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

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

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

if (rec (i,j) is an 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

On page 42 and 43 an syntax extension is necessary to fullfill the horizontal and vertical array size of 64K.

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
compatible	3	uimsbf
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.

Page 47, an ommision was found. The text should read, **the issue of the default value is still not resolved.**

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: ??, 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: ??, what is the meaning of this field for non-interlaced pictures

On page 51, macroblock should be 1-7 vlcbf, according to the extended VLC tables of page 62, and maybe even larger if the tables of page 84 are used. Text should now read

macroblock_type	**1-??**	**vlclbf**

Page 52: "compatible_type"
It seems that the description for the flags "compatible_type[i]" only works for 4:2:0 (6 blocks in a MB). It should be extended to 4:2:2 and 4:4:4. Text should read:

compatible_type - This is a one or two-bit integer defined in the following tables.

binary value	Prediction	compatible_type [0,1,2,3,4,5]
00	Non Compatible	0,0,0,0,0,0
01	FIELD1 Compatible prediction	1,1,0,0,1,1
10	FIELD2 Compatible prediction	0,0,1,1,1,1
11	Field1&2 Compatible prediction	1,1,1,1,1,1

For 4:2:0 and picture_structure == frame_structure

binary value	Prediction	compatible_type [0,1,2,3,4,5,6,7,8,9,10,11]
00	Non Compatible	0,0,0,0,0,0,0,0,0,0, 0, 0
01	FIELD1 Compatible prediction	1,1,0,0,1,1,0,0,1,1, 0, 0
10	FIELD2 Compatible prediction	0,0,1,1,0,0,1,1,0,0, 1, 1
11	Field1&2 Compatible prediction	1,1,1,1,1,1,1,1,1,1, 1, 1

For 4:2:2,4:4:4 and picture_structure == frame_structure

binary value	Prediction	compatible_type [0,1,2,3,4,5,6,7,8,9,10,11]
0	Non Compatible	0,0,0,0,0,0,0,0,0,0, 0, 0
1	Compatible prediction	1,1,1,1,1,1,1,1,1,1, 1, 1

For 4:2:0,4:2:2,4:4:4 and picture_structure == field_structure

compatible_type[i] - This flag is used to indicate the compatible prediction per block, see also the block layer.

And maybe for simplicity we could merge the first two tables.

Page 53: "slave_macroblock()" "6" should be replaced by "block_count"
Text should read:

9.3.7.1 Slave Macroblock Layer

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

Page 71: Insert figure D1 and D2

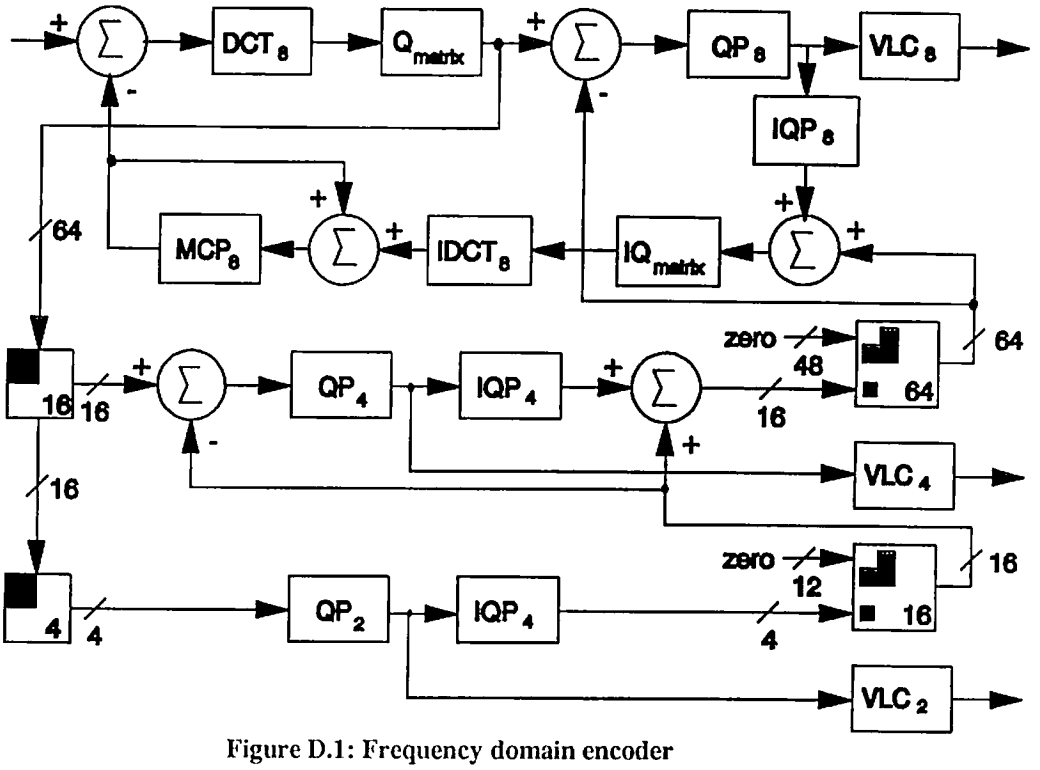

Figure D.1: Frequency domain encoder

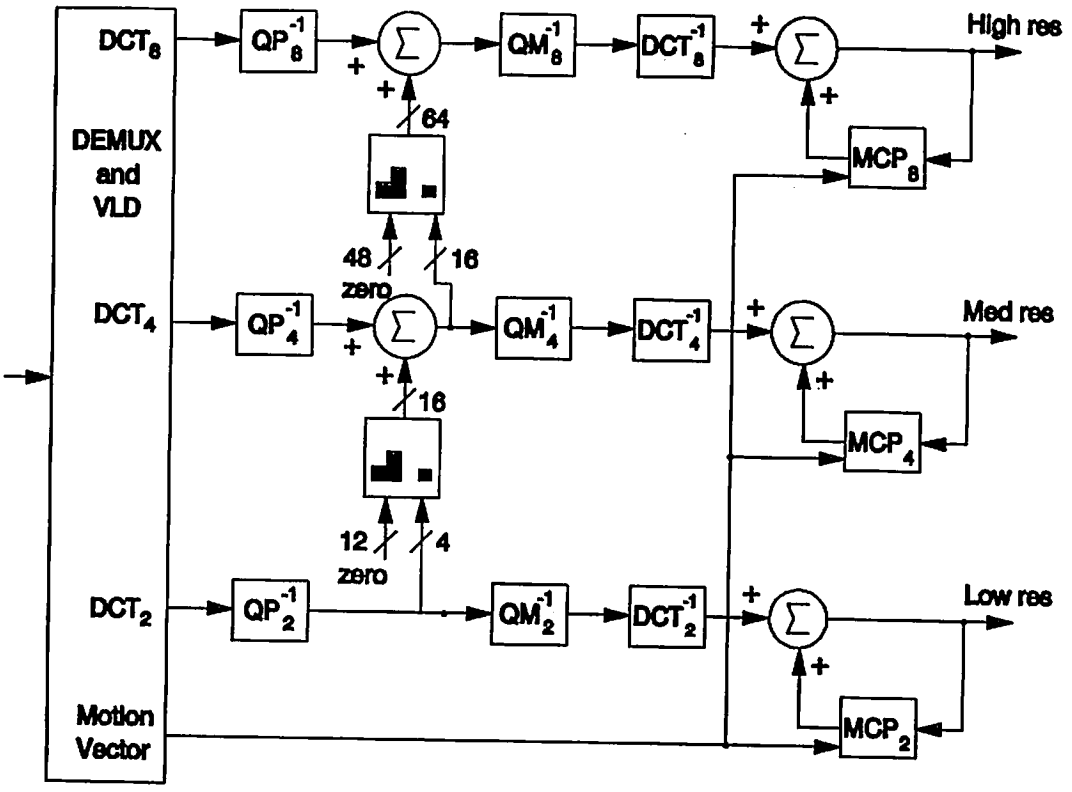

Figure D.2: Frequence domain scalable decoder

Page 101: The denominators of Equation (3) and (4) should be Number_of_slices instead of MB_cnt. Text should read:

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

$$T_{p_P} = \frac{Number\_of\_P\_slice}{Number\_of\_Slices} * T_P \qquad (3)$$

$$T_{p_I} = \frac{Number\_of\_I\_slice}{Number\_of\_Slices} * T_I \qquad (4)$$

Page 103: Insert figure I.1 and I.2

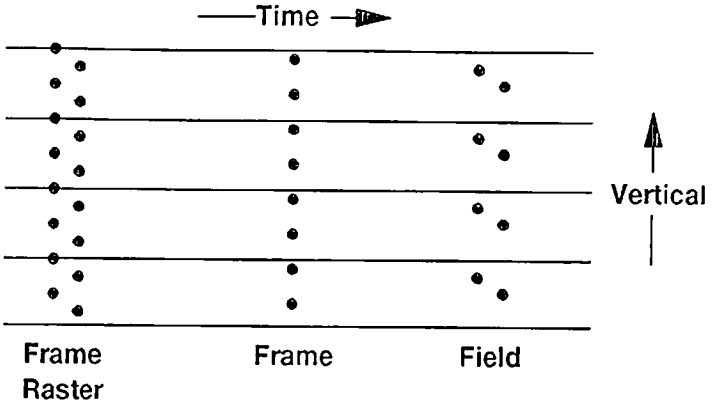

Figure I.1:Raster mis-match between frame and field based 4x4 extraction.

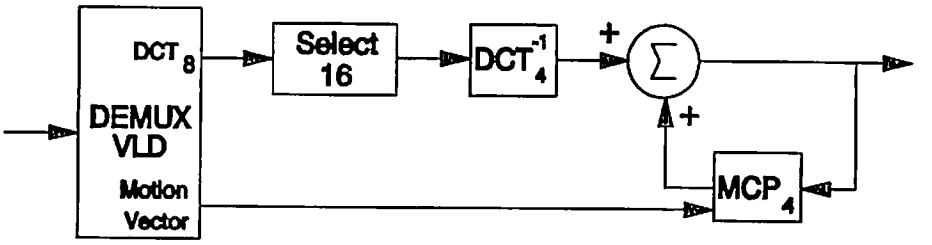

Figure I.2: Scale-4 decoder for core experiment I.1

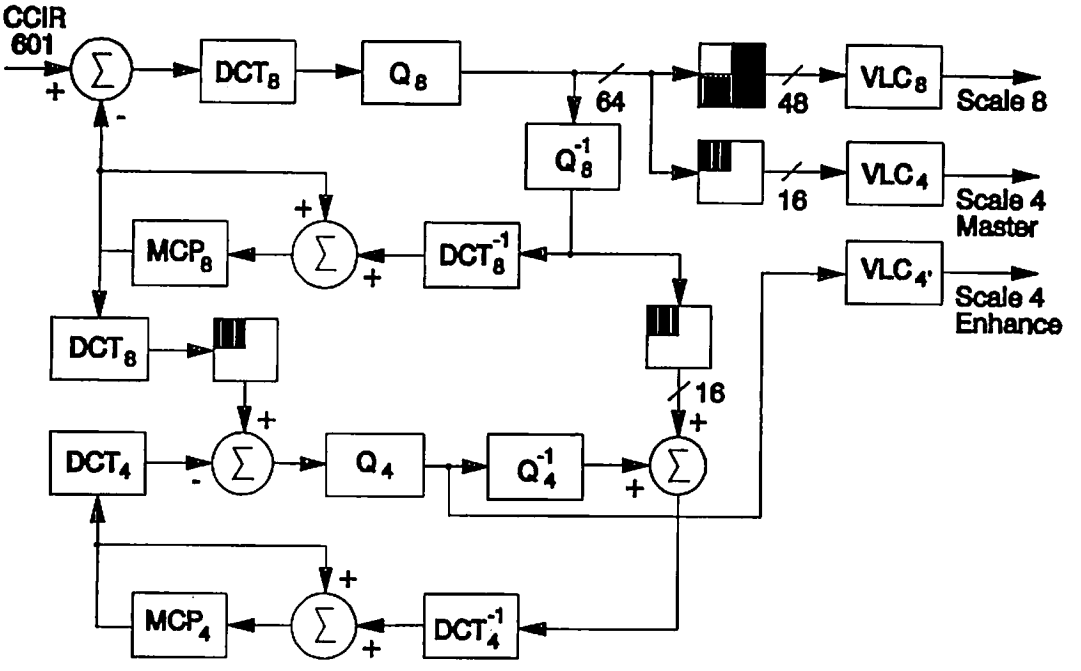

Figure I.5: Drift correcting encoder

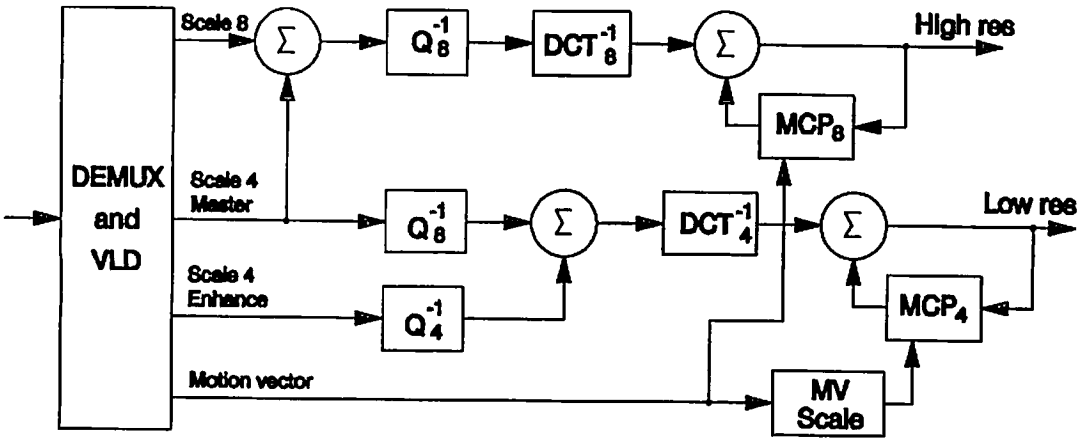

Figure I.6: Drift correcting decoder

Page 133: Core experiment No 1. There is a remark regarding the case M=1. The word "backward" should be removed, since there is no backward prediction. Text should read.

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.