INTERNATIONAL ORGANIZATION FOR STANDARDIZATION
ORGANISATION INTERNATIONALE DE NORMALISATION
ISO/IEC JTC1/SC29/WG11
CODING OF MOVING PICTURES AND ASSOCIATED AUDIO

ISO/IEC JTC1/SC29/WG11
MPEG96/M1557
Maceió, Brazil

Title: **The MPEG-4 Video VM 4.0 Syntax Using MSDL-S**
Source: **A. Kopansky, K. Wu, J.-B. Lee, Y. Fang, A. Eleftheriadis (Columbia University)**
Group: **MPEG-4 System**
Date: **November 12, 1996**

Introduction

This document provides a sample description of the entire MPEG-4 Video VM 4.0 using MSDL-S. Our purpose is to illustrate how MSDL-S is used, as well as provide evidence of the broad capabilities of MSDL-S.

Note that the definition FLC and VLC tables has been excluded from the document in order to keep it relatively short. The entire representation of VM 4.0 bitstream syntax, including the definition of various FLC and VLC tables is available at http://www.itnm.columbia.edu/~fang/MSDL/VM40/.

Known Problems

Note that this code passes through our MSDL-S translator (the code syntax is thus correct). The following are known problems with the syntactic description as given below.

1. In VideoObjectPlane, (4.6.1), use of byte-aligned look ahead is not yet supported.
2. Parameter types are not used yet, because our MSDL-S translator currently doesn't support it.
3. Array handling has some known problems. These are isolated to the loading of array values when these values are VLCs and they contain the flag on if the loaded value is the last one or not (e.g., TcoefClass). The code below uses temporary variables to side-step the problem.
4. Alignment with arbitrary bit pattern stuffing is not yet supported. Several possibilities are considered, but we have not determined the best approach yet.

Solutions to these problems are considered, and we hope that they can be resolved in the Maceio meeting.

Video VM 4.0 Code

- **Block-combined**

```
class BlockIntraLum_Bitstream
{
```

```
public:
  short    data[ 64 ];

  TcoefClass tcoeff[ 63 ];
  TcoefClass tmp;
  short i=0;

  short( DC_SIZE_LUM_TABLE8AP ) DCsize;
  if( DCsize != 0 )
    short( DIFF_DC_TABLE10AP ) intraDC;

  do
  {
    tmp  = tcoeff[ i ];
    TcoefClass( TCOEF_TABLE31AP ) tmp;
    tcoeff[ i ] = tmp;
    i++;
  } while( tmp.last != 1 );
};

class BlockIntraChrom_Bitstream
{
public:
  short    data[ 64 ];

  TcoefClass tcoeff[ 63 ];
  TcoefClass tmp;
  short i;

  short( DC_SIZE_CHROM_TABLE9AP ) DCsize;
  if( DCsize != 0 )
    short( DIFF_DC_TABLE10AP ) intraDC;

  i = 0;
  do
  {
    tmp = tcoeff[ i ];
    TcoefClass( TCOEF_TABLE32AP ) tmp;
    tcoeff[ i ] = tmp;
    i++;
  } while( tmp.last != 1 );
};


class BlockInterLum_Bitstream
{
public:
  short    data[ 64 ];

  TcoefClass tcoeff[ 64 ];
  TcoefClass tmp;
  short i;

  i = 0;
  do
  {
    tmp = tcoeff[ i ];
```

```
    TcoefClass( TCOEF_TABLE32AP ) tmp;
    tcoeff[ i ] = tmp;
    i++;
  } while( tmp.last != 1 );
};

class BlockInterChrom_Bitstream
{
public:
  short    data[ 64 ];

  TcoefClass tcoeff[ 64 ];
  TcoefClass tmp;
  short i;

  i = 0;
  do
  {
    tmp = tcoeff[ i ];
    TcoefClass( TCOEF_TABLE32AP ) tmp;
    tcoeff[ i ] = tmp;
    i++;
  } while( tmp.last != 1 );
};
```

- **Block-texture**

```
class BlockIntraLum_Bitstream
{
public:
  short    data[ 64 ];

  TcoefClass tcoeff[ 63 ];
  TcoefClass tmp;
  short i=0;

  short( DC_SIZE_LUM_TABLE8AP ) DCsize;
  if( DCsize != 0 )
    short( DIFF_DC_TABLE10AP ) intraDC;

  do
  {
    tmp  = tcoeff[ i ];
    TcoefClass( TCOEF_TABLE31AP ) tmp;
    tcoeff[ i ] = tmp;
    i++;
  } while( tmp.last != 1 );
};

class BlockIntraChrom_Bitstream
{
public:
  short    data[ 64 ];

  TcoefClass tcoeff[ 63 ];
```

```
  TcoefClass tmp;
  short i;

  short( DC_SIZE_CHROM_TABLE9AP ) DCsize;
  if( DCsize != 0 )
    short( DIFF_DC_TABLE10AP ) intraDC;

  i = 0;
  do
  {
    tmp = tcoeff[ i ];
    TcoefClass( TCOEF_TABLE32AP ) tmp;
    tcoeff[ i ] = tmp;
    i++;
  } while( tmp.last != 1 );
};


class BlockInterLum_Bitstream
{
public:
  short    data[ 64 ];

  TcoefClass tcoeff[ 64 ];
  TcoefClass tmp;
  short i;

  i = 0;
  do
  {
    tmp = tcoeff[ i ];
    TcoefClass( TCOEF_TABLE32AP ) tmp;
    tcoeff[ i ] = tmp;
    i++;
  } while( tmp.last != 1 );
};

class BlockInterChrom_Bitstream
{
public:
  short    data[ 64 ];

  TcoefClass tcoeff[ 64 ];
  TcoefClass tmp;
  short i;

  i = 0;
  do
  {
    tmp = tcoeff[ i ];
    TcoefClass( TCOEF_TABLE32AP ) tmp;
    tcoeff[ i ] = tmp;
    i++;
  } while( tmp.last != 1 );
};
```

- **Macroblock-combined**

```
class MacroBlock_Bitstream
{
public:
  short vop_prediction_type; // pass from VOP
  short f_code; // forward_f_code or backward_f_code
  int video_object_layer_shape;
  // from VideoObjectLayer_Bitstream

  if ( vop_prediction_type == 0 )                    // I-VOP
  {
    short( first_MMR_code_map_I ) first_MMR_code;
    MCBPC_Class( MCBPC_TABLE24_MAP ) mcbpc; //tableB.1,get
mbType
    unsigned int( 1 ) ACpred_flag;
    if ( mcbpc.mbType == 3 )
      {
     short( CBPY_TABLE29_I_MAP ) cbpy;          // table 29
      }
    else if ( mcbpc.mbType == 4 )
      {
     short( CBPY_TABLE29_I_MAP ) cbpy;          // table 29
     short( DQUANT_TABLE27_MAP ) dquant;        // table 27
      }
    // shape coding here
    if( video_object_layer_shape != 0 )
      {
     //first_MMR_code == "00", multilevel
     if( first_MMR_code == 0 )
       {
         int( VLC_CR ) CR;
         int(1) a0_color;

         do{
           int( VLC_TABLE_Modified_MMR ) VLC_binary;
           if( VLC_binary == 11 ) // Mode == H
          {
            int Vertical_pass_mode;
            if( Vertical_pass_mode == TRUE )
              {
                // Use 4 bits if CR == 1 (0)
                // Use 3 bits if CR == 1/2 (1)
                // Use 2 bits if CR == 1/4 (2)
                int( 4-CR ) RLB;
              }else
                {
               int( 4-CR ) ULB;
                }
          }
         }while( VLC_binary != 12 ); // Mode != EOMB

       }
     // video_object_layer_shape == '10', gray-scale
     if( video_object_layer_shape == 2 )
       {
         // Gray_shape_coding();
```

```
          // 0b00 -- I-VOP
          // 0b01 -- P-VOP
          // 0b10 -- B-VOP
          short(1) CODA;
          // CODA == 0b1,   the alpha block is all "255"
          // CODA == 0b0,   otherwise

          if ( CODA == 0 )
            {
           short( CBPY_TABLE29_I_MAP ) CBPA;

           if ( CBPA & 8 )
             BlockIntraLum_Bitstream block1;
           if ( CBPA & 4 )
             BlockIntraLum_Bitstream block2;
           if ( CBPA & 2 )
             BlockIntraLum_Bitstream block3;
           if ( CBPA & 1 )
             BlockIntraLum_Bitstream block4;
            }
        }
       }

    if ( cbpy & 8 )
      BlockIntraLum_Bitstream block1;
    if ( cbpy & 4 )
      BlockIntraLum_Bitstream block2;
    if ( cbpy & 2 )
      BlockIntraLum_Bitstream block3;
    if ( cbpy & 1 )
      BlockIntraLum_Bitstream block4;
    if ( mcbpc.cbpc &  2 )
      BlockIntraChrom_Bitstream block5;
    if ( mcbpc.cbpc & 1 )
      BlockIntraChrom_Bitstream block6;
  }
  else if ( vop_prediction_type == 1 )      // P-VOP
  {
    short( first_MMR_code_map_PB ) first_MMR_code;
    unsigned int( 1 ) cod;
    if ( cod == 0 )                         // coded, not skipped
      {
     MCBPC_Class( MCBPC_TABLE25_MAP ) mcbpc; //table 25,get
mbType
     unsigned int( 1 ) ACpred_flag;
     if ( mcbpc.mbType == 0 )
       {
        short( CBPY_TABLE29_P_MAP ) cbpy;      // table 29
        float( MVD_TABLE31_MAP ) mvdx;         // table 31
        if( ( mvdx != 0 ) && ( f_code != 1 ) )
          unsigned int( f_code - 1 ) residualx;
        float( MVD_TABLE31_MAP ) mvdy;         // table 31
        if( ( mvdy != 0 ) && ( f_code != 1 ) )
          unsigned int( f_code - 1 ) residualy;

       }
     else if ( mcbpc.mbType == 1 )
```

```
  {
    short( CBPY_TABLE29_P_MAP ) cbpy;      // table 29
    short( DQUANT_TABLE27_MAP ) dquant;    // table 27
    float( MVD_TABLE31_MAP ) mvdx;         // table 31
    if( ( mvdx != 0 ) && ( f_code != 1 ) )
      unsigned int( f_code - 1 ) residualx;
    float( MVD_TABLE31_MAP ) mvdy;         // table 31
    if( ( mvdy != 0 ) && ( f_code != 1 ) )
      unsigned int( f_code - 1 ) residualy;
  }
else if ( mcbpc.mbType == 2 )
  {
    short( CBPY_TABLE29_P_MAP ) cbpy;      // table 29
    float( MVD_TABLE31_MAP ) mvdx;         // table 31
    if( ( mvdx != 0 ) && ( f_code != 1 ) )
      unsigned int( f_code - 1 ) residualx;
    float( MVD_TABLE31_MAP ) mvdy;         // table 31
    if( ( mvdy != 0 ) && ( f_code != 1 ) )
      unsigned int( f_code - 1 ) residualy;
    float( MVD_TABLE31_MAP ) mvdx2;        // table 31
    if( ( mvdx2 != 0 ) && ( f_code != 1 ) )
      unsigned int( f_code - 1 ) residualx2;
    float( MVD_TABLE31_MAP ) mvdy2;        // table 31
    if( ( mvdy2 != 0 ) && ( f_code != 1 ) )
      unsigned int( f_code - 1 ) residualy2;
    float( MVD_TABLE31_MAP ) mvdx3;        // table 31
    if( ( mvdx3 != 0 ) && ( f_code != 1 ) )
      unsigned int( f_code - 1 ) residualx3;
    float( MVD_TABLE31_MAP ) mvdy3;        // table 31
    if( ( mvdy3 != 0 ) && ( f_code != 1 ) )
      unsigned int( f_code - 1 ) residualy3;
    float( MVD_TABLE31_MAP ) mvdx4;        // table 31
    if( ( mvdx4 != 0 ) && ( f_code != 1 ) )
      unsigned int( f_code - 1 ) residualx4;
    float( MVD_TABLE31_MAP ) mvdy4;        // table 31
    if( ( mvdy3 != 0 ) && ( f_code != 1 ) )
      unsigned int( f_code - 1 ) residualy4;
  }
else if ( mcbpc.mbType == 3 )
  {
    short( CBPY_TABLE29_P_MAP ) cbpy;      // table 29
  }
else if ( mcbpc.mbtype == 4 )
  {
    short( CBPY_TABLE29_P_MAP ) cbpy;      // table 29
    short( DQUANT_TABLE27_MAP ) dquant;    // table 27
  }
// shape coding here
//********************************

if( video_object_layer_shape != 0 )
  {

    //first_MMR_code == "00", multilevel
    if( first_MMR_code == 0 )
      {
     int( VLC_CR ) CR;
```

```
    int(1) a0_color;

    do{
      int( VLC_TABLE_Modified_MMR ) VLC_binary;
      if( VLC_binary == 11 ) // Mode == H
        {
          int Vertical_pass_mode;
          if( Vertical_pass_mode == TRUE )
         {
           // Use 4 bits if CR == 1 (0)
           // Use 3 bits if CR == 1/2 (1)
           // Use 2 bits if CR == 1/4 (2)
           int( 4-CR ) RLB;
         }else
           {
             int( 4-CR ) ULB;
           }
        }
    }while( VLC_binary != 12 ); // Mode != EOMB

     }
  // video_object_layer_shape == '10', gray-scale
  if( video_object_layer_shape == 2 )
     {
    // Gray_shape_coding();
    // 0b00 -- I-VOP
    // 0b01 -- P-VOP
    // 0b10 -- B-VOP
    short( CODA_table_P ) CODA;

    if( CODA == 0 )
      {
        short( CBPY_TABLE29_P_MAP ) CBPA;
        if ( CBPA & 8 )
          if( noOfMVectors == 0 )
         BlockIntraLum_Bitstream block1;
          else
         BlockInterLum_Bitstream block1;
        if ( CBPA & 4 )
          if( noOfMVectors == 0 )
         BlockIntraLum_Bitstream block2;
          else
         BlockInterLum_Bitstream block2;
        if ( CBPA & 2 )
          if( noOfMVectors == 0 )
         BlockIntraLum_Bitstream block3;
          else
         BlockInterLum_Bitstream block3;

        if ( CBPA & 1 )
          if( noOfMVectors == 0 )
         BlockIntraLum_Bitstream block4;
          else
         BlockInterLum_Bitstream block4;
      }
     }
}
```

```
    if ( cbpy & 8 )
      {
        if( ( mcbpc.mbtype == 3 ) || (  mcbpc.mbtype == 4 ))
          BlockIntraLum_Bitstream block1;
        else
          BlockInterLum_Bitstream block1;
      }
    if ( cbpy & 4 )
      {
        if( ( mcbpc.mbtype == 3 ) || (  mcbpc.mbtype == 4 ))
          BlockIntraLum_Bitstream block2;
        else
          BlockInterLum_Bitstream block2;
      }
    if ( cbpy & 2 )
      {
        if( ( mcbpc.mbtype == 3 ) || (  mcbpc.mbtype == 4 ))
          BlockIntraLum_Bitstream block3;
        else
          BlockInterLum_Bitstream block3;
      }
    if ( cbpy & 1 )
      {
        if( ( mcbpc.mbtype == 3 ) || (  mcbpc.mbtype == 4 ))
          BlockIntraLum_Bitstream block4;
        else
          BlockInterLum_Bitstream block4;
      }
    if ( mcbpc.cbpc &  2 )
      {
        if( ( mcbpc.mbtype == 3 ) || (  mcbpc.mbtype == 4 ))
          BlockIntraChrom_Bitstream block5;
        else
          BlockInterChrom_Bitstream block5;
      }
    if ( mcbpc.cbpc &  1 )
      {
        if( ( mcbpc.mbtype == 3 ) || (  mcbpc.mbtype == 4 ))
          BlockIntraChrom_Bitstream block6;
        else
          BlockInterChrom_Bitstream block6;
      }
     }
}
else     // B-VOP
{
  short( first_MMR_code_map_PB ) first_MMR_code;
  short( MODB_TABLE32_MAP ) modb;               // table 32
  if ( modb == 0 )
     {
    float( MVD_TABLE31_MAP ) mvdx3;             // table 31
    if( ( mvdx3 != 0 ) && (  f_code != 1 ) )
      unsigned int( f_code - 1 ) residualx3;
    float( MVD_TABLE31_MAP ) mvdy3;             // table 31
    if( ( mvdy3 != 0 ) && (  f_code != 1 ) )
```

```
      unsigned int( f_code - 1 ) residualy3;
     }
  else if ( modb == 1 )
     {
    short( MODB_TABLE33_MAP ) mbTypeB;          // table 33
    if ( mbTypeB == 1 )
      {
        short( DQUANT_TABLE28_MAP ) dquant;    // table 28
        float( MVD_TABLE31_MAP ) mvdx;         // table 31
        if( ( mvdx != 0 ) && ( f_code != 1 ) )
          unsigned int( f_code - 1 ) residualx;
        float( MVD_TABLE31_MAP ) mvdy;         // table 31
        if( ( mvdy != 0 ) && ( f_code != 1 ) )
          unsigned int( f_code - 1 ) residualy;
        float( MVD_TABLE31_MAP ) mvdx2;        // table 31
        if( ( mvdx2 != 0 ) && ( f_code != 1 ) )
          unsigned int( f_code - 1 ) residualx2;
        float( MVD_TABLE31_MAP ) mvdy2;        // table 31
        if( ( mvdy2 != 0 ) && ( f_code != 1 ) )
          unsigned int( f_code - 1 ) residualy2;
      }
    else if ( mbTypeB == 2 )
      {
        short( DQUANT_TABLE28_MAP ) dquant;    // table 28
        float( MVD_TABLE31_MAP ) mvdx2;        // table 31
        if( ( mvdx2 != 0 ) && ( f_code != 1 ) )
          unsigned int( f_code - 1 ) residualx2;
        float( MVD_TABLE31_MAP ) mvdy2;        // table 31
        if( ( mvdy2 != 0 ) && ( f_code != 1 ) )
          unsigned int( f_code - 1 ) residualy2;
      }
    else if ( mbTypeB == 3 )
      {
        short( DQUANT_TABLE28_MAP ) dquant;    // table 28
        float( MVD_TABLE31_MAP ) mvdx;         // table 31
        if( ( mvdx != 0 ) && ( f_code != 1 ) )
          unsigned int( f_code - 1 ) residualx;
        float( MVD_TABLE31_MAP ) mvdy;         // table 31
        if( ( mvdy != 0 ) && ( f_code != 1 ) )
          unsigned int( f_code - 1 ) residualy;
      }
    else // mbtypeB == 0
      {
        float( MVD_TABLE31_MAP ) mvdx3;        // table 31
        if( ( mvdx3 != 0 ) && ( f_code != 1 ) )
          unsigned int( f_code - 1 ) residualx3;
        float( MVD_TABLE31_MAP ) mvdy3;        // table 31
        if( ( mvdy3 != 0 ) && ( f_code != 1 ) )
          unsigned int( f_code - 1 ) residualy3;
      }
     }
  else // modB == 2
     {
    short( MODB_TABLE33_MAP ) mbTypeB;          // table 33
    unsigned int ( 6 ) cbpb;
    if ( mbTypeB == 1 )
      {
```

```
    short( DQUANT_TABLE28_MAP ) dquant;    // table 28
    float( MVD_TABLE31_MAP ) mvdx;         // table 31
    if( ( mvdx != 0 ) && ( f_code != 1 ) )
      unsigned int( f_code - 1 ) residualx;
    float( MVD_TABLE31_MAP ) mvdy;         // table 31
    if( ( mvdy != 0 ) && ( f_code != 1 ) )
      unsigned int( f_code - 1 ) residualy;
    float( MVD_TABLE31_MAP ) mvdx2;        // table 31
    if( ( mvdx2 != 0 ) && ( f_code != 1 ) )
      unsigned int( f_code - 1 ) residualx2;
    float( MVD_TABLE31_MAP ) m vdy2;       // table 31
    if( ( mvdy2 != 0 ) && ( f_code != 1 ) )
      unsigned int( f_code - 1 ) residualy2;
  }
else if ( mbTypeB == 2 )
  {
    short( DQUANT_TABLE28_MAP ) dquant;    //table 28
    float( MVD_TABLE31_MAP ) mvdx2;        //table 31
    if( ( mvdx2 != 0 ) && ( f_code != 1 ) )
      unsigned int( f_code - 1 ) residualx2;
    float( MVD_TABLE31_MAP ) mvdy2;        //table 31

  }
else if ( mbTypeB == 3 )
  {
    short( DQUANT_TABLE28_MAP ) dquant;    //table 28
    float( MVD_TABLE31_MAP ) mvdx;         //table 31
    if( ( mvdx != 0 ) && ( f_code != 1 ) )
      unsigned int( f_code - 1 ) residualx;
    float( MVD_TABLE31_MAP ) mvdy;         //table 31
    if( ( mvdy != 0 ) && ( f_code != 1 ) )
      unsigned int( f_code - 1 ) residualy;
  }
else // mbtypeB == 0
  {
    float( MVD_TABLE31_MAP ) mvdx;         //table 31
    if( ( mvdx != 0 ) && ( f_code != 1 ) )
      unsigned int( f_code - 1 ) residualx;
    float( MVD_TABLE31_MAP ) mvdy;         //table 31
    if( ( mvdy != 0 ) && ( f_code != 1 ) )
      unsigned int( f_code - 1 ) residualy;
  }
// shape coding here

if( video_object_layer_shape != 0 )
  {
    //first_MMR_code == "00", multilevel
    if( first_MMR_code == 0 )
      {
     int( VLC_CR ) CR;
     int(1) a0_color;

     do{
       int( VLC_TABLE_Modified_MMR ) VLC_binary;
       if( VLC_binary == 11 ) // Mode == H
         {
           int Vertical_pass_mode;
```

```
            if( Vertical_pass_mode == TRUE )
            {
              // Use 4 bits if CR == 1 (0)
              // Use 3 bits if CR == 1/2 (1)
              // Use 2 bits if CR == 1 /4 (2)
              int( 4-CR ) RLB;
            }
            else
            {
              int( 4-CR ) ULB;
            }
          }
      }while( VLC_binary != 12 ); // Mode != EOMB

      }
    // video_object_layer_shape == '10', gray-scale
    if( video_object_layer_shape == 2 )
      {
      // Gray_shape_coding();
      // 0b00 -- I-VOP
      // 0b01 -- P-VOP
      // 0b10 -- B-VOP
      short(1) CODA;
      // CODA == 0b1,   the alpha block is all "255"
      // CODA == 0b0,   otherwise

      if( CODA == 0 ) // 0b0
        {
          short( MODB_TABLEB8_MAP ) MODBA;
          if( MODBA == 2 )
            {
            short(4) CBPBA;
            // CBPBA is like CBPB in texture coding in
            //  B-VOP but only have 4 bits.
            // CBPBAn=1, if any coefficient is present
            // for B-block n,
            // else 0.
            if( CBPBA & 8 )
              BlockInterLum_Bitstream block1;
            if( CBPBA & 4 )
              BlockInterLum_Bitstream block2;
            if( CBPBA & 2 )
              BlockInterLum_Bitstream block3;
            if( CBPBA & 1 )
              BlockInterLum_Bitstream block4;
            }
        }
      }
  }
if ( cbpb & 32 )
  BlockInterLum_Bitstream block1;
if ( cbpb & 16 )
  BlockInterLum_Bitstream block2;
if ( cbpb & 8 )
  BlockInterLum_Bitstream block3;
if ( cbpb & 4 )
  BlockInterLum_Bitstream block4;
```

```
      if ( cbpb &  2 )
        BlockInterChrom_Bitstream block5;
      if ( cbpb & 1 )
        BlockInterChrom_Bitstream block6;
       }
  }
};
```

- **Macroblock-motion**

```
class MacroBlock_Motion_Bitstream
{
public:
  if ( VideoObjectPlane.VOP_prediction_type == 1 )    //  P VOP
  {
    short( NUMBER_OF_VECTORS_MAP ) numberOfVectors;
    if ( numberOfVectors == 1 )
      {
      float( MVD_TABLE31_MAP ) mvdx;
      float( MVD_TABLE31_MAP ) mvdy;
      }
    else if ( numberOfVecors == 4 )
      {
      float( MVD_TABLE31_MAP ) mvdx;  // table 31
      float( MVD_TABLE31_MAP ) mvdy;  // table 31
      float( MVD_TABLE31_MAP ) mvdx2; // table 31
      float( MVD_TABLE31_MAP ) mvdy2; // table 31
      float( MVD_TABLE31_MAP ) mvdx3; // table 31
      float( MVD_TABLE31_MAP ) mvdy3; // table 31
      float( MVD_TABLE31_MAP ) mvdx4; // table 31
      float( MVD_TABLE31_MAP ) mvdy4; // table 31
      }
  }
  else if ( VideoObjectPlane.VOP_prediction_type == 2 )    // B-
VOP
  {
    short( MODB_TABLE32_MAP ) modb;              // table 32
    if ( modb == 0 )
      {
      float( MVD_TABLE31_MAP ) mvd3x;            // table 31
      float( MVD_TABLE31_MAP ) mvd3y;            // table 31
      }
    else if ( modb == 1 )
      {
      short( MBTYPE_TABLE33_MAP ) mbTypeB;       // table 33
      if ( mbTypeB == 1 )
        {
          float( MVD_TABLE31_MAP ) mvdx;         // table 31
          float( MVD_TABLE31_MAP ) mvdy;         // table 31
          float( MVD_TABLE31_MAP ) mvd2x;        // table 31
          float( MVD_TABLE31_MAP ) mvd2y;        // table 31
        }
      else if ( mbTypeB == 2 )
        {
          float( MVD_TABLE31_MAP ) mvd2x;        // table 31
          float( MVD_TABLE31_MAP ) mvd2y;        // table 31
```

```
        }
      else if ( mbTypeB == 3 )
        {
          float( MVD_TABLE31_MAP ) mvdx;         // table 31
          float( MVD_TABLE31_MAP ) mvdy;         // table 31
        }
      else // mbtypeB == 0
        {
          float( MVD_TABLE31_MAP ) mvdx;         // table 31
          float( MVD_TABLE31_MAP ) mvdy;         // table 31
        }
      }
    else // modB == 2
      {
      short( MBTYPE_TABLE33_MAP ) mbTypeB;  // table 33
      if ( mbTypeB == 1 )
        {
          float( MVD_TABLE31_MAP ) mvdx;         // table 31
          float( MVD_TABLE31_MAP ) mvdy;         // table 31
          float( MVD_TABLE31_MAP ) mvd2x;        // table 31
          float( MVD_TABLE31_MAP ) mvd2y;        // table 31
        }
      else if ( mbTypeB == 2 )
        {
          float( MVD_TABLE31_MAP ) mvd2x;        // table 31
          float( MVD_TABLE31_MAP ) mvd2y;        // table 31
        }
      else if ( mbTypeB == 3 )
        {
          float( MVD_TABLE31_MAP ) mvdx;         // table 31
          float( MVD_TABLE31_MAP ) mvdy;         // table 31
        }
      else // mbtypeB == 0
        {
          float( MVD_TABLE31_MAP ) mvdx;         // table 31
          float( MVD_TABLE31_MAP ) mvdy;         // table 31
        }
      }
  }

  MacroBlock_Motion_Bitstream();
  ~MacroBlock_Motion_Bitstream();
};
```

- **Macroblock-shape**

```
class AlphaMacroblock_Bitstream
{
  //temporary variables
  int video_object_layer_shape; // from
VideoObjectLayer_Bitstream
  short first_MMR_code; // from VideoObjectPlane_Bitstream
  unsigned int VOP_prediction_type; // from
VideoObjectPlane_Bitstream
  short noOfMVectors; // number of motion vectors; if 0 - INTRA
coded
```

```
// video_object_layer_shape != ’00’, rectangular
if( video_object_layer_shape != 0 )
{

  //first_MMR_code == "00", multilevel
  if( first_MMR_code == 0 )
    {
    int( VLC_CR ) CR;
    int(1) a0_color;

    do{
      int( VLC_TABLE_Modified_MMR ) VLC_binary;
      if( VLC_binary == 11 ) // Mode == H
        {
          int Vertical_pass_mode;
          if( Vertical_pass_mode == TRUE )
         {
           // Use 4 bits if CR == 1 (0)
           // Use 3 bits if CR == 1/2 (1)
           // Use 2 bits if CR == 1/4 (2)
           int( 4-CR ) RLB;
         }else
           {
             int( 4-CR ) ULB;
           }
        }
    }while( VLC_binary != 12 ); // Mode != EOMB

    }
  // video_object_layer_shape == ’10’, gray-scale
  if( video_object_layer_shape == 2 )
    {
    // Gray_shape_coding();
    // 0b00 -- I-VOP
    // 0b01 -- P-VOP
    // 0b10 -- B-VOP
    if( VOP_prediction_type == 0 ) // I-VOP
      {
        short(1) CODA;
        // CODA == 0b1,   the alpha block is all "255"
        // CODA == 0b0,   otherwise

        if ( CODA == 0 )
          {
         short( CBPY_TABLE17_I_MAP ) CBPA;

         if ( CBPA & 8 )
           BlockIntraLum_Bitstream block1;
         if ( CBPA & 4 )
           BlockIntraLum_Bitstream block2;
         if ( CBPA & 2 )
           BlockIntraLum_Bitstream block3;
         if ( CBPA & 1 )
           BlockIntraLum_Bitstream block4;
          }
      }
```

```
else if( VOP_prediction_type == 1 ) // P-VOP
  {
    short( CODA_table_P ) CODA;

    if( CODA == 0 )
      {
     short( CBPY_TABLE17_P_MAP ) CBPA;

     if ( CBPA & 8 )
       if( noOfMVectors == 0 )
         BlockIntraLum_Bitstream block1;
       else
         BlockInterLum_Bitstream block1;
     if ( CBPA & 4 )
       if( noOfMVectors == 0 )
         BlockIntraLum_Bitstream block2;
       else
         BlockInterLum_Bitstream block2;
     if ( CBPA & 2 )
       if( noOfMVectors == 0 )
         BlockIntraLum_Bitstream block3;
       else
         BlockInterLum_Bitstream block3;

     if ( CBPA & 1 )
       if( noOfMVectors == 0 )
         BlockIntraLum_Bitstream block4;
       else
         BlockInterLum_Bitstream block4;

      }
  }
else // B-VOP
  {
    short(1) CODA;
    // CODA == 0b1,   the alpha block is all "255"
    // CODA == 0b0,   otherwise

    if( CODA == 0 )
      {
     short( MODB_TABLEB8_MAP ) MODBA;
     if( MODBA == 2 )
       {
         short(4) CBPBA;
         // CBPBA is like CBPB in texture coding in
         //  B-VOP but only have 4 bits.
         // CBPBAn=1, if any coefficient is present
         // for B-block n,
         // else 0.
         if( CBPBA & 8 )
           BlockInterLum_Bitstream block1;
         if( CBPBA & 4 )
           BlockInterLum_Bitstream block2;
         if( CBPBA & 2 )
           BlockInterLum_Bitstream block3;
         if( CBPBA & 1 )
           BlockInterLum_Bitstream block4;
```

```
              }
            }
        }
      }
  }
};
```

- **Video Object Plane**

```
class VideoObjectPlane_Bitstream
{
  int numOfMBx, numOfMBy, numOfMB;
  short bitLengthOfMB;

  aligned unsigned int( 32 ) VOP_start_code;

  do
  {
    unsigned int( 1 ) modulo_time_base;
  } while( modulo_time_base != 0 );

  unsigned int( 10 ) VOP_time_increment;

  unsigned int( 2 ) VOP_prediction_type;

  if( VideoObjectLayer.video_object_layer_shape != 0 )
  {
    unsigned int( 10 ) VOP_width;
    unsigned int( 10 ) VOP_height;

    // calculation of number of  Macroblocks
    // this part can't be parsed for now.

    if ( ( VOP_width % 16 ) != 0 )
      numOfMBx = VOP_width/16 + 1;
    else
      numOfMBx = VOP_width/16;

    if ( ( VOP_height % 16 ) != 0 )
      numOfMBy = VOP_height/16 + 1;
    else
      numOfMBy = VOP_height/16;

    numOfMB = numOfMBx * numOfMBy;

    //*********************************************

    unsigned int( 10 ) VOP_horizontal_mc_spatial_ref;
    const bit( 1 ) marker_bit = 1;
    unsigned int( 10 ) VOP_vertical_mc_spatial_ref;
    if( VideoObjectLayer.scalability &&
VideoObjectLayer.enhancement_type)
      unsigned int( 1 ) background_composition;
  }

  if( VOP_prediction_type == 2 )
```

```
    unsigned int( 2 ) VOP_dbquant;
  else
  unsigned int( 5 ) VOP_quant;

  if( !VideoObjectLayer.scalability )
  {
    if( !VideoObjectLayer.separate_motion_shape_texture )
      if( VideoObjectLayer.error_resilient_disable )
      // combined_motion_shape_texture_coding
      MacroBlock_Bitstream Macroblock[ numOfMB ]
        else
          do
            {
            do
              {
                // combined_motion_shape_texture_coding
                MacroBlock_Bitstream Macroblock;
              } while( ![ 0b0000000000000000 ] ) // must be
byte_aligned
                  if ( ![ 0b00000000000000000000000 ] ) // must
be byte_aligned
                  {
                    // next_resync_marker()
                    const bit( 17 ) resync_marker =
0b00000000000000001;

                    // calculation of the length of macroblock
number representation in bits

                    if ( numOfMB < 3 )
                      bitLengthOfMB = 1;
                    else if( numOfMB < 5 )
                      bitLengthOfMB = 2;
                    else if( numOfMB < 9 )
                      bitLengthOfMB = 3;
                    else if( numOfMB < 17 )
                      bitLengthOfMB = 4;
                    else if( numOfMB < 33 )
                      bitLengthOfMB = 5;
                    else if( numOfMB < 65 )
                      bitLengthOfMB = 6;
                    else if( numOfMB < 129 )
                      bitLengthOfMB = 7;
                    else if( numOfMB < 257 )
                      bitLengthOfMB = 8;
                    else if( numOfMB < 513 )
                      bitLengthOfMB = 9;
                    else if( numOfMB < 1025 )
                      bitLengthOfMB = 10;
                    else if( numOfMB < 2049 )
                      bitLengthOfMB = 11;
                    else
                      bitLengthOfMB = 12;

                    unsigned int( bitLengthOfMB )
                                   macroblock_number;
                    unsigned int( 5 ) quant_scale;
```

```
                }
            } while ( ![ 0b00000000000000000000000 ] );
                        // must be byte_aligned
    else // separate_motion_shape_texture
      {
      short countOfMB;
      if( VideoObjectLayer.video_object_layer_shape != 0 )
        {

          if( VOP_prediction_type == 0 )    // I-VOP
            short( first_MMR_code_map_I )
                  first_MMR_code[numOfMB];
          else
            short( first_MMR_code_map_PB )
                   first_MMR_code[numOfMB];
        }

      if( VideoObjectLayer.error_resilient_disable )
        {
          // motion coding
          MacroBlock_Motion_Bitstream Macroblock_Motion[ numOfMB
];

          if( VideoObjectLayer.video_object_layer_shape != 0 )
            // shape_coding
            AlphaMacroblock_Bitstream AlphaMacroblock[ numOfMB
];
          // texture coding ( ? )
          Block_Texture_Bitstream Block_Texture[ numOfMB * 4 ];
        }
      else
        {
          do
            {
           // motion coding
           MacroBlock_Motion_Bitstream Macroblock_Motion;
           if( VideoObjectLayer.video_object_layer_shape != 0 )
             {
               // shape_coding
               AlphaMacroblock_Bitstream AlphaMacroblock;

             }
           // texture coding
           Block_Texture_Bitstream Block_Texture;
           if( ![ 0b00000000000000000000000 ] )
           // must be byte_aligned
             {
               // next_resync_marker()
               const bit( 17 ) resync_marker
                                 = 0b00000000000000001;

               // calculation of the length of macroblock
               // number representation in bits

               if ( numOfMB < 3 )
                 bitLengthOfMB = 1;
               else if( numOfMB < 5 )
```

```
                bitLengthOfMB = 2;
              else if( numOfMB < 9 )
                bitLengthOfMB = 3;
              else if( numOfMB < 17 )
                bitLengthOfMB = 4;
              else if( numOfMB < 33 )
                bitLengthOfMB = 5;
              else if( numOfMB < 65 )
                bitLengthOfMB = 6;
              else if( numOfMB < 129 )
                bitLengthOfMB = 7;
              else if( numOfMB < 257 )
                bitLengthOfMB = 8;
              else if( numOfMB < 513 )
                bitLengthOfMB = 9;
              else if( numOfMB < 1025 )
                bitLengthOfMB = 10;
              else if( numOfMB < 2049 )
                bitLengthOfMB = 11;
              else
                bitLengthOfMB = 12;
              unsigned int( bitLengthOfMB ) macroblock_number;
              unsigned int( 5 ) quant_scale;

            }

          } while( ![ 0b00000000000000000000000 ] );
         // must be byte aligned
       }
      }
   }
   else
   {
     if( background_composition )
      {
      unsigned int( 1 ) load_backward_shape;
      if ( load_backward_shape )
        {
         // backward_shape_coding
         AlphaMacroblock_Bitstream AlphaMacroblockBack[numOfMB];
         unsigned int( 1 ) load_forward_shape;
         if( load_forward_shape )
           {
          // forward_shape_coding
          AlphaMacroblock_Bitstream AlphaMacroblockFor[numOfMB
];
           }
        }
      }

     unsigned int( 2 ) ref_select_code;

     if( ( VOP_prediction_type == 1 ) || ( VOP_prediction_type ==
2 ) )
      {
      unsigned int( 10 ) forward_temporal_ref;
```

```
      if( VOP_prediction_type == 2 )
        {
          const bit( 1 ) marker_bit = 1;
          unsigned int( 10 ) backward_temporal_ref;
        }
       }

    // combined_motion_shape_texture_coding
    MacroBlock_Bitstream Macroblock[ numOfMB ];
  }
};
```

- **Video Object Layer**

```
class VideoObjectLayer_Bitstream
{
  aligned unsigned int( 28 ) video_object_layer_start_code;
  unsigned int( 4 ) video_object_layer_id;
  unsigned int( 2 ) video_object_layer_shape;
  // "01"-binary, "10"-grayscale

  if( video_object_layer_shape == 0 )      // 00-rectangular
  {
    unsigned int( 10 )     video_object_layer_width;
    unsigned int( 10 )     video_object_layer_height;
  }

  unsigned int( 1 ) video_object_layer_quant_type;

  if ( video_object_layer_quant_type )
  {
    unsigned int( 1 ) load_intra_quant_mat;
    if( load_intra_quant_mat )
      unsigned int( 8 )    intra_quant_mat[ 64 ];
    unsigned int( 1 ) load_nonintra_quant_mat;
    if( load_nonintra_quant_mat )
      unsigned int( 8 ) nonintra_quant_mat[ 64 ];
  }

  unsigned int( 1 ) error_resilient_disable;
  unsigned int( 1 ) intra_acdc_pred_disable;
  unsigned int( 1 ) deblocking_filter_disable;
  unsigned int( 2 ) video_object_layer_fcode_forward;
  unsigned int( 2 ) video_object_layer_fcode_backward;
  unsigned int( 1 ) separate_motion_shape_texture;
  unsigned int( 1 ) scalability;
  if( scalability )
  {
    unsigned int( 4 ) ref_layer_id;
    unsigned int( 1 ) ref_layer_sampling_direc;
    unsigned int( 5 ) hor_sampling_factor_n;
    unsigned int( 5 ) hor_sampling_factor_m;
    unsigned int( 5 ) vert_sampling_factor_n;
    unsigned int( 5 ) vert_sampling_factor_m;
    unsigned int( 1 ) enhancement_type;
  }
```

```
    do
    {
      VideoObjectPlane_Bitstream  VideoObjectPlane;
    } while( [ VideoObjectPlane.video_object_plane_start_code ] );
  };
```

- **Video Object**

```
  class VideoObject_Bitstream
  {
    aligned unsigned int( 27 )    video_object_start_code;
    unsigned int( 5 )        video_object_id;
    do
    {
      VideoObjectLayer_Bitstream  VideoObjectLayer;
    } while( [ VideoObjectLayer.video_object_layer_start_code ] );
  };
```

- **Video Session**

```
class VideoSession_Bitstream
{
  const bit( 32 ) session_start_code = 0x000001B0;
  do
  {
    VideoObject_Bitstream VideoObject;
  } while ( [ VideoObject.video_object_start_code ] );
  const bit( 32 ) session_end_code = 0x000001B1;

  VideoSession_Bitstream();
  ~VideoSession_Bitstream();
};
```

References

1. Video Verification Model Version 4.0, ISO/IEC JTC1/SC29/WG11 N1380, Chicago, Illinois, October 1996.
2. MSDL Working Draft, Version 1.3, ISO/IEC JTC1/SC29/WG11 N1401, Chicago, Illinois, October 1996.