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
MPEG97/M1819
Seville, ES

Title: MPEG-4 Video WD 1.0 Syntax Using Flavor (MSDL-S)

Source: Yihan Fang and Alexandros Eleftheriadis
Dept. of Electrical Eng., Columbia University, New York, NY 10027, USA
{fang,eleft}@ee.columbia.edu

Group: MPEG-4 Video, MPEG-4 System
Date: February 10, 1997

Abstract

This document describes the bitstream syntax detailed in the MPEG-4 Video Working Draft Version 1.0 using Flavor. Flavor stands for Formal Language for Audio-Visual Object Representation, and it was formerly known as MSDL-S. This document should also appear as Appendix G in the Working Draft input document to the Seville meeting.

1. MPEG-4 Video WD 1.0 Syntax

In this section, we use a bottom-up representation scheme. For simplicity, “unsigned int” is denoted by “unit”. Only some of the map definitions are shown here.

1.1. Block

```
// Table 11-13: Variable length codes for dct_dc_size_luminance
map T_11_13 (uint) {
    0b100           0,
    0b00            1,
    0b01            2,
    0b101           3,
    0b110           4,
    0b1110          5,
    0b1111.0        6,
    0b1111.10       7,
    0b1111.110      8,
    0b1111.1110     9,
    0b1111.1111.0   10,
    0b1111.1111.1   11
};

// Table 11-14: Variable length codes for dct_dc_size_chrominance
map T_11_14 (uint) {
    0b00            0,
    0b01            1,
    0b10            2,
    0b110           3,
    0b1110          4,
    0b1111.0        5,
    0b1111.10       6,
    0b1111.110      7,
    0b1111.1110     8,
    0b1111.1111.0   9,
    0b1111.1111.10  10,
    0b1111.1111.11  11
};

// mb_intra should be figured out from mb type
// pattern_code is not defined yet in VM5.0
class Block(int i, uint mb_intra, uint pattern_code[12])
{
    if ( pattern_code[i] ) {
        if ( mb_intra ) {
            if ( i<4 ) {
                uint (T_11_13) dct_dc_size_luminance;
                if ( dct_dc_size_luminance !=0 )
                    uint(dct_dc_size_luminance) dct_dc_differential;
            } else {
                uint (T_11_14) dct_dc_size_chrominance;
                if ( dct_dc_size_chrominance !=0 )
                    uint(dct_dc_size_chrominance) dct_dc_differential;
            }
        } else {
            // First DCT coefficient 2-24 bits
        }
        while (![lastcoef])     // End of Block
            uint (sub_dct_coeff_table) sub_DCT_coeff;
```

```
	}
};
```

1.2. Macroblock

```
class MCPBC {
	int mb_type;
	int cbpc;
};

// Table 11-7 VLC table for MCBPC for I-VOPs in combined-motion-shape-texture coding
map T_11_7_mcpbc_I ( MCPBC )
{
	0b1,			{3,0},
	0b001,			{3,1},
	0b010,			{3,2},
	0b011,			{3,3},
	0b0001,			{4,0},
	0b0000.01,		{4,1},
	0b0000.10,		{4,2},
	0b0000.11,		{4,3},
	0b000000001,		{-1,-1}		// stuffing
};

// Table 11-8 VLC table for MCBPC for P-VOPs in combined-motion-shape-texture coding
map T_11_8_mcpbc_P ( MCPBC )
{
	0b1,			{0,0},
	0b0011,			{0,1},
	0b0010,			{0,2},
	0b0001.01,		{0,3},
	0b011,			{1,0},
	0b0000.111,		{1,1},
	0b0000.110,		{1,2},
	0b0000.0010.1,		{1,3},
	0b010,			{2,0},
	0b0000.101,		{2,1},
	0b0000.100,		{2,2},
	0b0000.0101,		{2,3},
	0b0001.1,		{3,0},
	0b0000.0100,		{3,1},
	0b0000.0011,		{3,2},
	0b0000.011,		{3,3},
	0b0001.00,		{4,0},
	0b0000.0010.0,		{4,1},
	0b0000.0001.1,		{4,2},
	0b0000.0001.0,		{4,3},
	0b0000.0000.1,		{-1,-1}		// stuffing
};

class CBPY
{
	int I;
```

```
    int P;
};

// Table 11-9 VLC table for CBPY in combined motion-shape-texture coding
map T_11_9_cbpy( CBPY )
{
    0b0011,         {0,15},
    0b00101,        {1,14},
    0b00100,        {2,13},
    0b1001,         {3,12},
    0b00011,        {4,11},
    0b0111,         {5,10},
    0b000010,       {6,9},
    0b1011,         {7,8},
    0b00010,        {8,7},
    0b000011,       {9,6},
    0b0101,         {10,5},
    0b1010,         {11,4},
    0b0100,         {12,3},
    0b1000,         {13,2},
    0b0110,         {14,1},
    0b11,           {15,0}
};

map T_modb( short )
{
    0b0,    0, // use index for mapping
    0b10,   1,
    0b11,   2
};

map T_mb_type( short )
{
    0b1,        0, // use index for mapping
    0b01,       1,
    0b001,      2,
    0b0001,     3
};

class MacroBlock(uint vop_coding_type, uint video_object_layer_shape)
{
    short f_code; // forward_f_code or backward_f_code

    if (vop_coding_type==0 || vop_coding_type==1)    {          // I or P
        if (video_object_layer_shape!=0)             // rectangular
            short( first_MMR_code_map_I ) first_mmr_code;
        if (vop_coding_type==1)
            int(1) not_cod;
        if (!not_cod || cop_coding_type ==0) {
            MCBPC(T_11_7_mcpbc_I ) mcbpc;
            if (mcbpc.mb_type==3 || mcbpc.mb_type==4)
                int( 1 ) ac_pred_flag;
            if ( mcbpc.mb_type != -1 ) {          // stuffing
```

```
                CBPY ( T_11_9_cbpy ) cbpy;
            if (mcbpc.mb_type==1 || mcbpc.mb_type==4) {
                uint(2) dquant;
            if (mcbpc.mb_type==0 || mcbpc.mb_type==1 || mcbpc.mb_type==2)
                MotionVector motion_vector0;
            if (mcbpc.mb_type==2){
                MotionVector motion_vector1;
                MotionVector motion_vector2;
                MotionVector motion_vector3;
            }
            if (video_object_layer_shape != 0 && first_mmr_code==0)  // rectangle, multilevel
                MbBinaryShapeCoding mb_binary_shape_coding;
            if (video_object_layer_shape == 2 && first_mmr_code==0)  // greyscale, multilevel
                MbGreyShapeCoding mb_grey_shape_coding;
            for (int i=0; i<block_count; i++)
                Block block(i);
        }
    }
    if (vop_coding_type==2) {                                    // B vop
        if (video_object_layer_shape!=0)              // rectangular
            short( first_MMR_code_map_I ) first_mmr_code;
        short( T_modb ) modb;                     // table 32 in vm 4.0
        if (modb>0)
            short (T_mb_type) mb_type;
        if (modb==2)
            int(6) cbpb;
        if (ref_select_code !=0 || scalability ==0) {
            if (mb_type !=1)
                uint(2) dquant;
            if (mb_type==1 || mb_type==3)             // "01" || "0001"
                MotionVector motion_vector;           // forward
            if (mb_type==1 || mb_type==2)             // "01" || "001"
                MotionVector motion_vector;           // backward
            if (mb_type==0)                           // '1'
                MotionVector motion_vector;           // direct
        }
**      if (ref_select_code ==0 && scalability !=0) {
            uint(2) dquant;
            if (mb_type==1 || mb_type==0)             // "01" || "1"
                MotionVector motion_vector;           // forward
            if (mb_type==1 || mb_type==2)             // "01" || "001"
                MotionVector motion_vector;           // backward
        }
        if (video_object_layer_shape != 0 && first_mmr_code==0)  // rectangle, multilevel
            MbBinaryShapeCoding mb_binary_shape_coding;
        if (video_object_layer_shape == 2 && first_mmr_code==0)  // greyscale, multilevel
            MbGreyShapeCoding mb_grey_shape_coding;
        for (int i=0; i<block_count; i++)
            Block block(i);
    }
};
```

1.2.1. Motion Vector

1.3. Video Object Plane

```
class VideoObjectPlane (uint video_object_layer_shape, uint scalability,
                        uint enhancement_type, uint separate_motion_shape_texture,
                        uint error_resilience_disable):
    aligned bit(32) vop_start_code = 0x000001B6
{
    do {
        bit( 1 ) modulo_time_base;
    } while( modulo_time_base != 0 );

    bit( 10 ) vop_time_increment;
    uint(2) vop_coding_type;

    if( video_object_layer_shape != 0 ) {  // != "rectangular"
        uint(10) vop_width;
        uint(10) vop_height;
        uint(10) vop_horizontal_mc_spatial_ref;
        bit(1) marker_bit;
        uint(10) vop_vertical_mc_spatial_ref;
        if (scalability && enhancement_type)
            uint(1) background_composition;
    }

    if (vop_coding_type == 2)
        uint(2) vop_dbquant;
    else
        uint(5) vop_quant;

    // calculation of number of Macroblocks

    int num_mbx, num_mby, num_mb;

    if ( ( vop_width % 16 ) != 0 )
        num_mbx = vop_width/16 + 1;
    else
        num_mbx = vop_width/16;

    if ( ( vop_height % 16 ) != 0 )
        num_mby = vop_height/16 + 1;
    else
        num_mby = vop_height/16;

    num_mb = num_mbx * num_mby;

     // calculation of the length of macroblock number representation in bits
    short bitLengthOfMB;
    if ( num_mb < 3 )             bitLengthOfMB = 1;
    else if( num_mb < 5 )         bitLengthOfMB = 2;
    else if( num_mb < 9 )         bitLengthOfMB = 3;
    else if( num_mb < 17 )        bitLengthOfMB = 4;
    else if( num_mb < 33 )        bitLengthOfMB = 5;
    else if( num_mb < 65 )        bitLengthOfMB = 6;
    else if( num_mb < 129 )       bitLengthOfMB = 7;
    else if( num_mb < 257 )       bitLengthOfMB = 8;
```

```
else if( num_mb < 513 )          bitLengthOfMB = 9;
else if( num_mb < 1025 )         bitLengthOfMB = 10;
else if( num_mb < 2049 )         bitLengthOfMB = 11;
else if( num_mb < 4097 )         bitLengthOfMB = 12;

if( !scalability ) {
    if( !separate_motion_shape_texture )
    if( error_resilient_disable )
        // combined_motion_shape_texture_coding
        Macroblock mb[num_mb];
    else
        do {
            do {
                // combined_motion_shape_texture_coding
                Macroblock mb[num_mb];
            } while( ![ 0b0000.0000.0000.0000 ] )                // byte_aligned
            if ( [ !0b000.0000.0000.0000.0000.0000 ] ) {       // byte_aligned
                // next_resync_marker()
                const bit( 17 ) resync_marker = 0b0.0000.0000.0000.0001;
                uint( bitLengthOfMB ) macroblock_number;
                uint( 5 ) quant_scale;
            }
        } while ( ![ 0b000.0000.0000.0000.0000.0000 ] )       // byte_aligned
    // separate motion texture
    else {
        if ( video_object_layer_shape != 0 ) {
            if (vop_coding_type == 1)  // I
                short( first_MMR_code_map_I ) first_mmr_code[num_mb];
            else if (vop_coding_type == 2 || vop_coding_type == 3) // P, B
                short( first_MMR_code_map_PB ) first_mmr_code[num_mb];
        }

        if( error_resilient_disable ) {
            // motion coding
            MotionCoding motion_coding(vop_coding_type, ref_seclect_code, scalability);
            if( video_object_layer_shape == 2 )           // binary shape coding
                BinaryShapeCoding binary_shape_coding(video_object_layer_shape,
                                                first_mmr_code[num_mb], num_mb);
            if( video_object_layer_shape == 3 )           // grey-scale shape coding
                GreyscaleShapeCoding greyscale_shape_coding(video_object_layer_shape,
                                                vop_coding_type, num_mb);
            TextureCoding texture_coding(video_coding_type, num_mb); // texture coding
        } else {
            do {
                // motion coding
                MotionCoding motion_coding(vop_coding_type,
                                            ref_seclect_code, scalability);
                if( video_object_layer_shape == 2 )           // binary shape coding
                    BinaryShapeCoding binary_shape_coding(video_object_layer_shape,
                                                first_mmr_code[num_mb], num_mb);
                if( video_object_layer_shape == 3 )           // grey-scale shape coding
                    GreyscaleShapeCoding
```

```
                                        greyscale_shape_coding(video_object_layer_shape,
                                                                        vop_coding_type, num_mb);
                                TextureCoding texture_coding(video_coding_type, num_mb);
                                if( [! 0b000.0000.0000.0000.0000.0000 ] ) {
                                // next_resync_marker
                                const bit( 17 ) resync_marker = 0b0.0000.0000.0000.0001;
                                uint( bitLengthOfMB ) macroblock_number;
                                uint( 5 ) quant_scale;
                                }
                        } while( ![ 0b000.0000.0000.0000.0000.0000 ] );
                }
        }
    } else {        // if scalability
        if( background_composition ) {
            uint( 1 ) load_backward_shape;
            if ( load_backward_shape ) {
                // backward_shape_coding
                AlphaMacroblock alpha_macroblock_back[num_mb];
                uint( 1 ) load_forward_shape;
                if( load_forward_shape ) {
                // forward_shape_coding
                AlphaMacroblock alpha_macroblock_for[num_mb ];
                }
            }
        }

        uint( 2 ) ref_select_code;
        if( ( vop_coding_type == 1 ) || ( vop_coding_type == 2 ) ) {
            uint( 10 ) forward_temporal_ref;
            if( VOP_prediction_type == 2 ) {
                bit( 1 ) marker_bit;
                uint( 10 ) backward_temporal_ref;
            }
        }

        // combined_motion_shape_texture_coding
        Macroblock mb[num_mb];
    }
};
```

1.3.1. Combined Motion Shape Texture Coding

1.3.2. Motion Coding

```
map T_number_of_vectors ( short )
{
    0b11,   0,
    0b0,    1,
    0b10,   4
};

class MotionCoding (uint vop_coding_type, uint ref_select_code, uint scalability)
{
```

```
    if (vop_codiing_type == 1 ) {                     //  P VOP
        short( T_number_of_vectors ) number_of_vectors;
        repeat(num_mb) {
            MotionVector motion_vector;
        }
    }
    if ( vop_coding_type == 2 ) {                     // B-VOP
        short( T_modb ) modb;                         // table 32 in vm 4.0
        if (modb>0)
            short (T_mb_type) mb_type;
        if (ref_select_code !=0 || scalability ==0) {
            if (mb_type ==1 || mb_type==3)            // '01' || '0001'
                MotionVector motion_vector;           // forward
            if (mb_type ==1 || mb_type==2)            // '01' || '001'
                MotionVector motion_vector;           // backward
};
```

1.3.3. Shape Coding

1.3.3.1. Binary Shape

```
class BinaryShapeCoding(uint video_object_layer_shape, short first_mmr_code[] , int num_mb)
{
    if (video_object_layer_shape != 0) {          // "rectangular"
        for (int i=0; i++;  i<num_mb) {
            if (first_mmr_code[i] ==  0) {         // "multilevel"
                MbBinaryShapeCoding mb_binary_shape_coding;
        }
    }
};

map T6_9_conversion_ratio (uint) {
    0b0,     0,       // 1
    0b01,    1,       // 1/2
    0b10,    2        // 1/4
];

map T6_10_modified_mmr (uint) {
    0b1,                0,    // V(0)
    0b01,               1,    // V(1)
    0b0001,             2,    // V(2)
    0b0000.1,           3,    // V(3)
    0b0000.01,          4,    // V(4)
    0b0000.001,         5,    // V(5)
    0b001,              6,    // Horizontal
    0b0001,             7,    // End of  Macroblock
};

class  MbBinaryShapeCoding
{
    uint(T6_9_conversion_ratio) conversion_ratio;
    uint(1) a0_color;
    do {
```

```
		uint (T6_10_modified_mmr) vlc_binary;
		if (vlc_binary ==6) {		// mode = horizontal
			uint(1) vertical_pass_mode;		//added here
			if (vertical_pass_mode)
				// Use 4 bits if CR = 1 (0),3 bits if CR = 1/2 (1), 2 bits if CR = 1/4 (2)
				int( 4-conversion_ratio ) residual_length_binary_shape;
			else
				int( 4- conversion_ratio ) unchanged_length_binary_shape;
		}
	}while( vlc_binary != 7 );			// Mode != EOMB
};
```

1.3.3.2. Grey Scale Shape Coding

```
class GreyShapeCoding (uint video_object_layer_shape, uint vop_coding_type, int num_mb)
{
	if(video_ object_layer_shape == 2) {		// "grey-scale"
		for (int i=0; i++; i<num_mb) {
			if (first_mmr_code[i] ==  0) {		// "multilevel"
				MbGreyShapeCoding mb_grey_shape_coding(vop_coding_type);
		}
	}
};

class MbGreyShapeCoding (uint vop_coding_type)
{
	if (vop_coding_type==0 || vop_coding_type==1) {		// I or P
		uint(vlc_table…undefined) cod_alpha;
		if (cod_alpha==0) {
			uint(vlc….) cbp_alpha;
			for(int i=0; i<4; i++)
				Block block(i);
		}
	}
	if (vop_coding_type==2) {				// B vop
		uint(1) cod_alpha;
		if (cod_alpha==0) {
			uint(vlc…) modb_alpha;
			if (modb_alpha==2)
				bit(4) cbpb_alpha;
			for(int i=0; i<4; i++)
				Block block(i);
		}
	}
};
```

1.3.4. Texture Coding

```
class TextureCoding (uint vop_coding_type)
{
	if (vop_coding_type==0) {			// I vop
		repeat (num_mb) {
			MCPBC(T_11_7_mcpbc_I) mcpbc;
```

```
                if (mcpbc.mb_type==3 || mcpbc.mb_type==4)
                      int(1) ac_pred_flag;
                if (mcpbc.mb_type != -1)              // != stuffing
                      CPBY(T_11_9_cbpy) cbpy;
                if (mcpbc.mb_type==1 ||mcpbc.mb_type==4)
                      uint(2) dquant;
                for ( i=0; i<block_count; i++)
                      Block block(i);
          }
    }
    if (vop_coding_type==1) {                         // P vop
          repeat (num_mb) {
                int(1) no_dct_flag;
                MCPBC(T_…) mcpbc;
                if(mcpbc.mb_type != -1)
                      CBPY(T_11_9_cbpy) cbpy;
                if (mcpbc.mb_type==1 ||mcpbc.mb_type==4)
                      uint(2) dquant;
                for ( i=0; i<block_count; i++)
                      Block block(i);
          }
    }
    if (vop_coding_type==2) {                         // B vop
          repeat (num_mb) {
                int(6) cbpb;
                uint(Table…) dbquant;
                for ( i=0; i<block_count; i++)
                      Block block(i);
          }
    }
};
```

1.4. Video Object Layer

```
class VideoObjectLayer:
    aligned bit(32) video_object_layer_start_code = 0x00000120..0x0000012F
{
    uint( 2 ) video_object_layer_shape;
    // "01"-binary, "10"-grayscale

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

    uint( 1 ) quant_type;

    if (quant_type) {
        uint( 1 ) load_intra_quant_mat;
        if( load_intra_quant_mat )
            uint( 8 ) intra_quant_mat[ 64 ];
        uint( 1 ) load_nonintra_quant_mat;
        if( load_nonintra_quant_mat )
            uint( 8 ) nonintra_quant_mat[ 64 ];
    }

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

    do {
        VideoObjectPlane video_object_plane(video_object_layer_shape,
                        scalability, enhancement_type, separate_motion_shape_texture,
                        error_resilience_disable);
    } while( [ VideoObjectPlane.video_object_plane_start_code ] );
};
```

1.5. Video Object

```
class VideoObject :
    aligned bit(32) video_object_start_code = 0x00000100..0x0000011F
{
    do {
        VideoObjectLayer video_object_layer;
    } while( [ VideoObjectLayer.video_object_layer_start_code ] );
};
```

1.6. Video Session

```
class VideoSession: aligned bit(32) session_start_code = 0x000001B0;
{
    do {
        VideoObject video_object;
    } while ( [ VideoObject.video_object_start_code ] );
    const bit( 32 ) session_end_code = 0x000001B1;
};
```