

```

1  /*
2  ****
3  * COPYRIGHT AND WARRANTY INFORMATION
4  *
5  * Copyright 2001, International Telecommunications Union, Geneva
6  *
7  * DISCLAIMER OF WARRANTY
8  *
9  * These software programs are available to the user without any
10 * license fee or royalty on an "as is" basis. The ITU disclaims
11 * any and all warranties, whether express, implied, or
12 * statutory, including any implied warranties of merchantability
13 * or of fitness for a particular purpose. In no event shall the
14 * contributor or the ITU be liable for any incidental, punitive, or
15 * consequential damages of any kind whatsoever arising from the
16 * use of these programs.
17 *
18 * This disclaimer of warranty extends to the user of these programs
19 * and user's customers, employees, agents, transferees, successors,
20 * and assigns.
21 *
22 * The ITU does not represent or warrant that the programs furnished
23 * hereunder are free of infringement of any third-party patents.
24 * Commercial implementations of ITU-T Recommendations, including
25 * shareware, may be subject to royalty fees to patent holders.
26 * Information regarding the ITU-T patent policy is available from
27 * the ITU Web site at http://www.itu.int.
28 *
29 * THIS IS NOT A GRANT OF PATENT RIGHTS - SEE THE ITU-T PATENT POLICY.
30 ****
31 */
32
33
34 /*!
35 ****
36 * \file
37 *   parset.c
38 * \brief
39 *   Parameter Sets
40 * \author
41 *   Main contributors (see contributors.h for copyright, address and affiliation details)
42 *   - Stephan Wenger      <stewe@cs.tu-berlin.de>
43 *
44 ****
45 */
46
47 #include <stdlib.h>
48 #include <assert.h>
49 #include <string.h>
50
51 #include "parsetcommon.h"

```

```

52 #include "parset.h"
53 #include "nalu.h"
54 #include "global.h"
55 #include "memalloc.h"
56 #include "fmo.h"
57
58 #if TRACE
59 #define SYMTRACESTRING(s) strncpy(sym->tracestring,s,TRACESTRING_SIZE)
60 #else
61 #define SYMTRACESTRING(s) // do nothing
62 #endif
63
64 static int se_v (char *tracestring, DataPartition *part);
65 static int ue_v (char *tracestring, DataPartition *part);
66 static int u_1 (char *tracestring, DataPartition *part);
67 static int u_v (int LenInBits, char *tracestring, DataPartition *part);
68
69 static int UsedBits;    // for internal statistics, is adjusted by se_v, ue_v, u_1
70
71 static seq_parameter_set_rbsp_t SeqParSet[MAXSPS];
72 static pic_parameter_set_rbsp_t PicParSet[MAXPPS];
73
74 // fill sps with content of p
75
76 int InterpretSPS (DataPartition *p, seq_parameter_set_rbsp_t *sps)
77 {
78     unsigned i;
79
80     assert (p != NULL);
81     assert (p->bitstream != NULL);
82     assert (p->bitstream->streamBuffer != 0);
83     assert (sps != NULL);
84
85     UsedBits = 0;
86
87     sps->profile_idc          = ue_v ("SPS: profile_idc"          , p);
88     sps->level_idc           = ue_v ("SPS: level_idc"            , p);
89     sps->seq_parameter_set_id = ue_v ("SPS: seq_parameter_set_id" , p);
90     sps->log2_max_frame_num_minus4 = ue_v ("SPS: log2_max_frame_num_minus4" , p);
91     sps->pic_order_cnt_type    = ue_v ("SPS: pic_order_count_type" , p);
92     if (sps->pic_order_cnt_type == 1)
93         sps->num_ref_frames_in_pic_order_cnt_cycle = ue_v ("SPS:
94 num_ref_frames_in_pic_order_cnt_cycle", p);
95     if (sps->pic_order_cnt_type < 2)
96     {
97         sps->delta_pic_order_always_zero_flag = u_1 ("SPS: delta_pic_order_always_zero_flag"
98 , p);
99         sps->offset_for_non_ref_pic          = se_v ("SPS: offset_for_non_ref_pic"          , p);
100         sps->offset_for_top_to_bottom_field = se_v ("SPS: offset_for_top_to_bottom_field" ,
101 p);
102     }

```

```

103     if (sps->pic_order_cnt_type == 0)
104         sps->offset_for_ref_frame[0] = se_v ("SPS: offset_for_ref_frame[0]" , p);
105     else if (sps->pic_order_cnt_type == 1)
106     {
107         for(i=0; i<sps->num_ref_frames_in_pic_order_cnt_cycle; i++)
108             sps->offset_for_ref_frame[i] = se_v ("SPS: offset_for_ref_frame[i]" , p);
109     }
110     sps->num_ref_frames = ue_v ("SPS: num_ref_frames" , p);
111     sps->required_frame_num_update_behaviour_flag = u_1 ("SPS:
112 required_frame_num_update_behaviour_flag", p);
113     sps->frame_width_in_mbs_minus1 = ue_v ("SPS: frame_width_in_mbs_minus1"
114 , p);
115     sps->frame_height_in_mbs_minus1 = ue_v ("SPS: frame_height_in_mbs_minus1"
116 , p);
117     sps->frame_coding_only_flag = u_1 ("SPS: frame_coding_flag" , p);
118     sps->mb_adaptive_frame_field_flag = u_1 ("SPS: mb_adaptive_frame_field_flag"
119 , p);
120     sps->direct_temporal_constrained_flag = u_1 ("SPS: direct_temporal_constrained_flag"
121 , p);
122     sps->vui_seq_parameters_present_flag = u_1 ("SPS: vui_seq_parameters_present_flag"
123 , p);
124     if (sps->vui_seq_parameters_present_flag)
125     {
126         printf ("VUI sequence parameters present but not supported, ignored, proceeding to next
127 NALU\n");
128     }
129     sps->Valid = TRUE;
130     return UsedBits;
131 }
132
133
134 int InterpretPPS (DataPartition *p, pic_parameter_set_rbsp_t *pps)
135 {
136     unsigned i;
137     int NumberBitsPerSliceGroupId;
138
139     assert (p != NULL);
140     assert (p->bitstream != NULL);
141     assert (p->bitstream->streamBuffer != 0);
142     assert (pps != NULL);
143
144     UsedBits = 0;
145
146     pps->pic_parameter_set_id = ue_v ("PPS: pic_parameter_set_id" , p);
147     pps->seq_parameter_set_id = ue_v ("PPS: seq_parameter_set_id" , p);
148     pps->entropy_coding_mode = ue_v ("PPS: entropy_coding_mode" , p);
149
150     /// Note: as per JVT-F078 the following bit is unconditional. If F078 is not accepted, then
151     /// one has to fetch the correct SPS to check whether the bit is present (hopefully there is
152     /// no consistency problem :-()
153     /// The current encoder code handles this in the same way. When you change this, don't forget

```

```

154    /// the encoder! StW, 12/8/02
155    pps->pic_order_present_flag          = u_1 ("PPS: pic_order_present_flag"          , p);
156
157    pps->num_slice_groups_minus1        = ue_v ("PPS: num_slice_groups_minus1"
158    , p);
159
160    /// InterpretSPS is able to process all different FMO modes, but UseParameterSet() is not (yet
161    /// Hence, make sure FMO is not used.
162
163    assert (pps->num_slice_groups_minus1 == 0);
164
165    // FMO stuff begins here
166    if (pps->num_slice_groups_minus1 > 0)
167    {
168        pps->mb_slice_group_map_type      = ue_v ("PPS: mb_slice_group_map_type"
169    , p);
170        if (pps->mb_slice_group_map_type == 0)
171            for (i=0; i<pps->num_slice_groups_minus1; i++)
172                pps->run_length [i]      = ue_v ("PPS: run_length [i]"          , p);
173        else if (pps->mb_slice_group_map_type == 2)
174            for (i=0; i<pps->num_slice_groups_minus1; i++)
175            {
176                /// JVT-F078: avoid reference of SPS by using ue(v) instead of u(v)
177                pps->top_left_mb [i]      = ue_v ("PPS: top_left_mb [i]"          , p);
178                pps->bottom_right_mb [i]  = ue_v ("PPS: bottom_right_mb [i]"      , p);
179            }
180        else if (pps->mb_slice_group_map_type == 3 ||
181                pps->mb_slice_group_map_type == 4 ||
182                pps->mb_slice_group_map_type == 5)
183        {
184            pps->slice_group_change_direction_flag = u_1 ("PPS:
185    slice_group_change_direction_flag"          , p);
186            pps->slice_group_change_rate_minus1    = ue_v ("PPS:
187    slice_group_change_rate_minus1"          , p);
188        }
189        else if (pps->mb_slice_group_map_type == 6)
190        {
191            if (pps->num_slice_groups_minus1 >= 4)
192                NumberBitsPerSliceGroupId = 3;
193            else if (pps->num_ref_idx_l0_active_minus1 >= 2)
194                NumberBitsPerSliceGroupId = 2;
195            else if (pps->num_ref_idx_l0_active_minus1 >= 1)
196                NumberBitsPerSliceGroupId = 1;
197            else
198                NumberBitsPerSliceGroupId = 0;
199            /// JVT-F078, explicitly signal number of MBs in the map
200            pps->slice_group_id_cnt          = ue_v ("PPS: slice_group_id_cnt"          , p);
201            for (i=0; i<pps->slice_group_id_cnt; i++)
202                pps->slice_group_id[i] = u_v (NumberBitsPerSliceGroupId, "slice_group_id[i]", p);
203        }
204    }

```

```

205 // End of FMO stuff
206
207 pps->num_ref_idx_l0_active_minus1 = ue_v ("PPS: num_ref_idx_l0_active_minus1"
208 , p);
209 pps->num_ref_idx_l1_active_minus1 = ue_v ("PPS: num_ref_idx_l1_active_minus1"
210 , p);
211 pps->weighted_pred_flag = u_1 ("PPS: wheighted prediction flag" , p);
212 pps->weighted_bipred_explicit_flag = u_1 ("PPS: weighted_bipred_explicit_flag" ,
213 p);
214 pps->weighted_bipred_implicit_flag = u_1 ("PPS: weighted_bipred_ixplicit_flag" ,
215 p);
216 pps->pic_init_qp_minus26 = se_v ("PPS: pic_init_qp_minus26" , p);
217 pps->pic_init_qs_minus26 = se_v ("PPS: pic_init_qs_minus26" , p);
218 pps->chroma_qp_index_offset = se_v ("PPS: chroma_qp_index_offset" ,
219 p);
220 pps->deblocking_filter_parameters_present_flag = u_1 ("PPS:
221 deblocking_filter_parameters_present_flag", p);
222 pps->constrained_intra_pred_flag = u_1 ("PPS: constrained_intra_pred_flag" ,
223 p);
224 pps->redundant_slice_flag = u_1 ("PPS: redundant_slice_flag" , p);
225 pps->vui_pic_parameters_flag = u_1 ("PPS: vui_pic_parameters_flag" , p);
226
227 if (pps->vui_pic_parameters_flag)
228 {
229     printf ("VUI picture parameters present but not supported, ignored, proceeding to next
230 NALU\n");
231 }
232 pps->Valid = TRUE;
233 return UsedBits;
234 }
235
236 static int se_v (char *tracestring, DataPartition *part)
237 {
238     SyntaxElement symbol, *sym=&symbol;
239
240     assert (part->bitstream->streamBuffer != NULL);
241     sym->type = SE_HEADER;
242     sym->mapping = linfo_dquant; // Mapping rule: signed integer
243     SYMTRACESTRING(tracestring);
244     readSyntaxElement_UVLC (sym,img,input,part);
245     UsedBits+=sym->len;
246     return sym->value1;
247 }
248
249
250 static int ue_v (char *tracestring, DataPartition *part)
251 {
252     SyntaxElement symbol, *sym=&symbol;
253
254     assert (part->bitstream->streamBuffer != NULL);
255     sym->type = SE_HEADER;

```

```

256     sym->mapping = linfo; // Mapping rule
257     SYMTRACESTRING(tracestring);
258     readSyntaxElement_UVLC (sym,img,input,part);
259     UsedBits+=sym->len;
260     return sym->value1;
261 }
262
263
264 static int u_1 (char *tracestring, DataPartition *part)
265 {
266     return u_v (1, tracestring, part);
267 }
268
269
270 static int u_v (int LenInBits, char*tracestring, DataPartition *part)
271 {
272     SyntaxElement symbol, *sym=&symbol;
273
274     assert (part->bitstream->streamBuffer != NULL);
275     sym->type = SE_HEADER;
276     sym->mapping = linfo; // Mapping rule
277     sym->len = LenInBits;
278     SYMTRACESTRING(tracestring);
279     readSyntaxElement_FLC (sym,part);
280     UsedBits+=sym->len;
281     return sym->inf;
282 };
283
284
285 void DumpSPS (seq_parameter_set_rbsp_t *sps)
286 {
287     printf ("Dumping a sequence parset, to be implemented\n");
288 };
289
290 void DumpPPS (pic_parameter_set_rbsp_t *pps)
291 {
292     printf ("Dumping a picture parset\n");
293 }
294
295 void PPSConsistencyCheck (pic_parameter_set_rbsp_t *pps)
296 {
297     printf ("Consistency checking a picture parset\n");
298     // if (pps->seq_parameter_set_id invalid then do something)
299 }
300
301 void SPSConsistencyCheck (seq_parameter_set_rbsp_t *sps)
302 {
303     printf ("Consistency checking a sequence parset\n");
304 }
305
306 void MakePPSAvailable (int id, pic_parameter_set_rbsp_t *pps)

```

```

307 {
308     assert (pps->Valid == TRUE);
309
310     if (PicParSet[id].Valid == TRUE && PicParSet[id].slice_group_id != NULL)
311         free (PicParSet[id].slice_group_id);
312
313     memcpy (&PicParSet[id], pps, sizeof (pic_parameter_set_rbsp_t));
314     if ((PicParSet[id].slice_group_id = calloc (PicParSet[id].slice_group_id_cnt, sizeof(int))) ==
315     NULL)
316         no_mem_exit ("MakePPSavailable: Cannot calloc slice_group_id");
317
318     memcpy (PicParSet[id].slice_group_id, pps->slice_group_id, pps-
319     >slice_group_id_cnt*sizeof(int));
320 }
321
322 void MakeSPSavailable (int id, seq_parameter_set_rbsp_t *sps)
323 {
324     assert (sps->Valid == TRUE);
325     memcpy (&SeqParSet[id], sps, sizeof (seq_parameter_set_rbsp_t));
326 }
327
328
329 void ProcessSPS (NALU_t *nalu)
330 {
331     DataPartition *dp = AllocPartition(1);
332     seq_parameter_set_rbsp_t *sps = AllocSPS();
333     int dummy;
334
335     memcpy (dp->bitstream->streamBuffer, &nalu->buf[1], nalu->len-1);
336     dp->bitstream->code_len = dp->bitstream->bitstream_length = RBSPtoSODB (dp->bitstream-
337     >streamBuffer, nalu->len-1);
338     dp->bitstream->ei_flag = 0;
339     dp->bitstream->read_len = dp->bitstream->frame_bitoffset = 0;
340     dummy = InterpretSPS (dp, sps);
341     // DumpSPS (sps);
342     // SPSConsistencyCheck (pps);
343     MakeSPSavailable (sps->seq_parameter_set_id, sps);
344     FreePartition (dp, 1);
345     FreeSPS (sps);
346 }
347
348
349 void ProcessPPS (NALU_t *nalu)
350 {
351     DataPartition *dp;
352     pic_parameter_set_rbsp_t *pps = AllocPPS();
353     int dummy;
354
355     dp = AllocPartition(1);
356     pps = AllocPPS();
357     memcpy (dp->bitstream->streamBuffer, &nalu->buf[1], nalu->len-1);

```

```

358     dp->bitstream->code_len = dp->bitstream->bitstream_length = RBSPtoSODB (dp->bitstream-
359 >streamBuffer, nalu->len-1);
360     dp->bitstream->ej_flag = 0;
361     dp->bitstream->read_len = dp->bitstream->frame_bitoffset = 0;
362     dummy = InterpretPPS (dp, pps);
363     // DumpPPS (pps);
364     // PPSConsistencyCheck (pps);
365     MakePPSAvailable (pps->pic_parameter_set_id, pps);
366     FreePartition (dp, 1);
367     FreePPS (pps);
368 }
369
370
371
372
373
374
375 void UseParameterSet (int PicParsetId)
376 {
377     seq_parameter_set_rbsp_t *sps =
378 &SeqParSet[PicParSet[PicParsetId].seq_parameter_set_id];
379     pic_parameter_set_rbsp_t *pps = &PicParSet[PicParsetId];
380     static unsigned int ExpectedDeltaPerPicOrderCntCycle;
381     int i;
382
383
384     if (PicParSet[PicParsetId].Valid != TRUE)
385         printf ("Trying to use an invalid (uninitialized) Picture Parameter Set with ID %d, expect the
386 unexpected...\n", PicParsetId);
387     if (SeqParSet[PicParSet[PicParsetId].seq_parameter_set_id].Valid != TRUE)
388         printf ("PicParset %d references an invalid (uninitialized) Sequence Parameter Set with ID
389 %d, expect the unexpected...\n", PicParsetId, PicParSet[PicParsetId].seq_parameter_set_id);
390
391     sps = &SeqParSet[PicParSet[PicParsetId].seq_parameter_set_id];
392
393     active_sps = sps;
394     active_pps = pps;
395
396     // In theory, and with a well-designed software, the lines above
397     // are everything necessary. In practice, we need to patch many values
398     // in img-> (but no more in inp-> -- these have been taken care of)
399
400
401     // Sequence Parameter Set Stuff first
402
403     // printf ("Using Picture Parameter set %d and associated Sequence Parameter Set %d\n",
404 PicParsetId, PicParSet[PicParsetId].seq_parameter_set_id);
405
406     // img->log2_max_frame_num_minus4 = sps->log2_max_frame_num_minus4;
407     img->MaxFrameNum = 1<<(sps->log2_max_frame_num_minus4+4);
408

```

```

409     img->pic_order_cnt_type = sps->pic_order_cnt_type;
410     if (img->pic_order_cnt_type != 1)
411     {
412         printf ("sps->pic_order_cnt_type %d, expected 1, exoect the unexpected...\n", sps-
413 >pic_order_cnt_type);
414         assert (sps->pic_order_cnt_type == 1);
415         error ("pic_order_cnt_type != 1", -1000);
416     }
417
418     img->num_ref_frames_in_pic_order_cnt_cycle = sps-
419 >num_ref_frames_in_pic_order_cnt_cycle;
420     if(img->num_ref_frames_in_pic_order_cnt_cycle != 1)
421         error("num_ref_frames_in_pic_order_cnt_cycle != 1",-1001);
422     if(img->num_ref_frames_in_pic_order_cnt_cycle >= MAX_LENGTH_POC_CYCLE)
423         error("num_ref_frames_in_pic_order_cnt_cycle too large",-1011);
424
425     img->delta_pic_order_always_zero_flag = sps->delta_pic_order_always_zero_flag;
426     if(img->delta_pic_order_always_zero_flag != 0)
427         error ("delta_pic_order_always_zero_flag != 0",-1002);
428
429     img->offset_for_non_ref_pic = sps->offset_for_non_ref_pic;
430
431     img->offset_for_top_to_bottom_field = sps->offset_for_top_to_bottom_field;
432
433
434     ExpectedDeltaPerPicOrderCntCycle=0;
435     if (sps->num_ref_frames_in_pic_order_cnt_cycle)
436         for(i=0;i<(int)sps->num_ref_frames_in_pic_order_cnt_cycle;i++)
437         {
438             img->offset_for_ref_frame[i] = sps->offset_for_ref_frame[i];
439             ExpectedDeltaPerPicOrderCntCycle += sps->offset_for_ref_frame[i];
440         }
441
442     img->width = (sps->frame_width_in_mbs_minus1+1) * MB_BLOCK_SIZE;
443     img->width_cr = img->width /2;
444     img->height = (sps->frame_height_in_mbs_minus1+1) * MB_BLOCK_SIZE;
445     img->height_cr = img->height / 2;
446
447     // Picture Parameter Stuff
448
449     img->weighted_pred_flag = pps->weighted_pred_flag;
450     img->weighted_bipred_explicit_flag = pps->weighted_bipred_explicit_flag;
451     img->weighted_bipred_implicit_flag = pps->weighted_bipred_implicit_flag;
452
453     img->pic_order_present_flag = pps->pic_order_present_flag;
454     if(img->pic_order_present_flag != 0)
455         error ("pic_order_present_flag != 0",-1004);
456
457     img->constrained_intra_pred_flag = pps->constrained_intra_pred_flag;
458
459

```

```

460 // currSlice->dp_mode is set by read_new_slice (NALU first byte available there)
461 switch (pps->entropy_coding_mode)
462 {
463 case UVLC:
464     nal_startcode_follows = uvlc_startcode_follows;
465     for (i=0; i<3; i++)
466     {
467         img->currentSlice->partArr[i].readSyntaxElement = readSyntaxElement_UVLC;
468     }
469     break;
470 case CABAC:
471     nal_startcode_follows = cabac_startcode_follows;
472 //     currSlice->readSlice = readSliceCABAC;
473     for (i=0; i<3; i++)
474     {
475         img->currentSlice->partArr[i].readSyntaxElement = readSyntaxElement_CABAC;
476     }
477     break;
478 default:
479     printf ("Unknown PPS:entropy_coding_mode %d\n", pps->entropy_coding_mode);
480     assert (0==1);
481 }
482
483 }

```