

```

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 * \mainpage
36 * This is the H.26L decoder reference software. For detailed documentation
37 * see the comments in each file.
38 *
39 * \author
40 * The main contributors are listed in contributors.h
41 *
42 * \version
43 * JM 5.0f
44 *
45 * \note
46 * tags are used for document system "doxygen"
47 * available at http://www.doxygen.org
48 *
49 * \par Limitations:
50 * Using different NAL's the assignment of partition-id to containing
51 * syntax elements may got lost, if this information is not transmitted.

```

```

52  *   The same has to be stated for the partitionlength if partitions are
53  *   merged by the NAL.
54  * \par
55  *   The presented solution in Q15-K-16 solves both of this problems as the
56  *   departitioner parses the bitstream before decoding. Due to syntax element
57  *   dependencies both, partition bounds and partitionlength information can
58  *   be parsed by the departitioner.
59  *
60  * \par Handling partition information in external file:
61  *   As the TML is still a work in progress, it makes sense to handle this
62  *   information for simplification in an external file, here called partition
63  *   information file, which can be found by the extension .dp extending the
64  *   original encoded H.26L bitstream. In this file partition-ids followed by its
65  *   partitionlength is written. Instead of parsing the bitstream we get the
66  *   partition information now out of this file.
67  *   This information is assumed to be never sent over transmission channels
68  *   (simulation scenarios) as it's information we allways get using a
69  *   "real" departitioner before decoding
70  *
71  * \par Extension of Interim File Format:
72  *   Therefore a convention has to be made within the interim file format.
73  *   The underlying NAL has to take care of fulfilling these conventions.
74  *   All partitions have to be bytealigned to be readable by the decoder,
75  *   So if the NAL-encoder merges partitions, >>this is only possible to use the
76  *   VLC structure of the H.26L bitstream<<, this bitaligned structure has to be
77  *   broken up by the NAL-decoder. In this case the NAL-decoder is responsible to
78  *   read the partitionlength information from the partition information file.
79  *   Partitionlosses are signaled with a partition of zero length containing no
80  *   syntax elements.
81  *
82  */
83 /*!
84  * \file
85  *   ldecod.c
86  * \brief
87  *   TML decoder project main()
88  * \author
89  *   Main contributors (see contributors.h for copyright, address and affiliation details)
90  *   - Inge Lille-Lang y    <inge.lille-langoy@telenor.com>
91  *   - Rickard Sjoberg     <rickard.sjoberg@era.ericsson.se>
92  *   - Stephan Wenger      <stewe@cs.tu-berlin.de>
93  *   - Jani Lainema        <jani.lainema@nokia.com>
94  *   - Sebastian Purreiter  <sebastian.purreiter@mch.siemens.de>
95  *   - Byeong-Moon Jeon    <jeonbm@lge.com>
96  *   - Gabi Blaettermann   <blaetter@hhi.de>
97  *   - Ye-Kui Wang         <wyk@ieee.org>
98  *
99  *****
100 */
101
102 #include "contributors.h"

```

```

103
104 #include <stdlib.h>
105 #include <stdio.h>
106 #include <string.h>
107 #include <time.h>
108 #include <sys/timeb.h>
109
110 #if defined WIN32
111     #include <conio.h>
112 #endif
113
114 #include <assert.h>
115
116 #include "global.h"
117 #include "rtp.h"
118 #include "memalloc.h"
119 #include "mbuffer.h"
120 #include "leaky_bucket.h"
121 #include "decodeiff.h"
122 #include "fmo.h"
123 #include "annexb.h"
124
125 #include "erc_api.h"
126
127 #define JM          "5"
128 #define VERSION     "5.0f"
129
130 #define LOGFILE      "log.dec"
131 #define DATADECFILE "dataDec.txt"
132 #define TRACEFILE    "trace_dec.txt"
133
134 extern objectBuffer_t *erc_object_list;
135 extern ercVariables_t *erc_errorVar;
136
137
138 /// I have started to move the inp and img structures into global variables.
139 /// They are declared in the following lines. Since inp is defined in conio.h
140 /// and cannot be overridden globally, it is defined here as input
141 ///
142 /// Everywhere, input-> and img-> can now be used either globally or with
143 /// the local override through the formal parameter mechanism
144
145 extern FILE* bits;
146 struct inp_par  *input;    // input parameters from input configuration file
147 struct snr_par  *snr;      // statistics
148 struct img_par  *img;      // image parameters
149
150 /*!
151 *****
152 * \brief
153 *   main function for TML decoder

```

```

154 *****
155 */
156 int main(int argc, char **argv)
157 {
158     // allocate memory for the structures
159     if ((input = (struct inp_par *)calloc(1, sizeof(struct inp_par)))==NULL) no_mem_exit("main:
160 input");
161     if ((snr = (struct snr_par *)calloc(1, sizeof(struct snr_par)))==NULL) no_mem_exit("main: snr");
162     if ((img = (struct img_par *)calloc(1, sizeof(struct img_par)))==NULL) no_mem_exit("main:
163 img");
164
165     // Read Configuration File
166     if (argc != 2)
167     {
168         sprintf(errortext, ET_SIZE, "Usage: %s <config.dat> \n\t<config.dat> defines decoder
169 parameters",argv[0]);
170         error(errortext, 300);
171     }
172     isBigEndian = testEndian();
173     init_conf(input, argv[1]);
174
175
176     switch (input->FileFormat)
177     {
178     case 0:
179         OpenBitstreamFile (input->infile);
180         break;
181     case 1:
182         OpenRTPFile (input->infile);
183         break;
184     default:
185         printf ("Unsupported file format %d, exit\n", input->FileFormat);
186     }
187
188     // Allocate Slice data struct
189     malloc_slice(input,img);
190
191     init(img);
192     img->number=0;
193     img->type = INTRA_IMG;
194     img->tr_old = -1; // WYK: Oct. 8, 2001, for detection of a new frame
195     img->refPicID = -1; // WYK: for detection of a new non-B frame
196     img->imgtr_last_P = 0;
197     img->imgtr_next_P = 0;
198     img->mmco_buffer=NULL;
199     img->last_decoded_pic_id = -1; // JVT-D101
200
201     // B pictures
202     Bframe_ctr=0;
203     init_poc();
204

```

```

205 // time for total decoding session
206 tot_time = 0;
207 while (decode_one_frame(img, input, snr) != EOS)
208     ;
209
210 // B PICTURE : save the last P picture
211 write_prev_Pframe(img, p_out);
212
213 report(input, img, snr);
214 free_slice(input, img);
215 FmoFinit();
216 free_frame_buffers(input, img);
217 free_global_buffers(input, img);
218
219 CloseBitstreamFile();
220
221 fclose(p_out);
222 if (p_ref)
223     fclose(p_ref);
224 #if TRACE
225     fclose(p_trace);
226 #endif
227
228 ercClose(erc_errorVar);
229
230 free (input);
231 free (snr);
232 free (img);
233
234 //while( !kbhit() );
235 return 0;
236 }
237
238 /*!
239 *****
240 * \brief
241 *   Initializes the POC structure with appropriate parameters.
242 *
243 *****
244 */
245 void init_poc()
246 {
247     int i;
248
249     for(i=0; i<MAX_NO_POC_FRAMES; i++){toprefpoc[i] = bottomrefpoc[i] = 1<<29;}
250 //init with large
251
252 }
253
254 /*!
255 *****

```

```

256  * \brief
257  *   Pushes values onto the POC ref arrays  toprefpoc[] & bottomrefpoc[]
258  *
259  *****
260  */
261  void push_poc(unsigned int topvalue, unsigned int bottomvalue, unsigned int ref_frame_ind )
262  {
263      int i;
264      static int current_is_ref = 0;          //indicates if current top value is for a ref frame
265
266      if(current_is_ref){
267          for(i=MAX_NO_POC_FRAMES-1; i>0; i--){          //move all the data down by one
268              toprefpoc[i] = toprefpoc[i-1] ;
269              bottomrefpoc[i] = bottomrefpoc[i-1] ;
270          }
271      }
272
273      toprefpoc[0] = topvalue;          //put new data
274      bottomrefpoc[0] = bottomvalue;
275
276      current_is_ref = ref_frame_ind;    //new data type
277
278  }
279
280
281
282  /*!
283  *****
284  * \brief
285  *   Initilize some arrays
286  *****
287  */
288  void init(struct img_par *img) //!< image parameters
289  {
290      int i;
291
292      // initilize quad matrix used in snr routine
293      for (i=0; i < 256; i++)
294      {
295          img->quad[i]=i*i; // fix from TML 1, truncation removed
296      }
297  }
298
299  /*!
300  *****
301  * \brief
302  *   Read input from configuration file
303  *
304  * \par Input:
305  *   Name of configuration filename
306  *

```

```

307  * \par Output
308  *   none
309  ****
310  */
311  void init_conf(struct inp_par *inp,
312               char *config_filename)
313  {
314      FILE *fd;
315      int NAL_mode;
316
317      // read the decoder configuration file
318      if((fd=fopen(config_filename,"r")) == NULL)
319      {
320          snprintf(errortext, ET_SIZE, "Error: Control file %s not found\n",config_filename);
321          error(errortext, 300);
322      }
323
324      fscanf(fd,"%s",inp->infile);           // H.26L compressed input bitsream
325      fscanf(fd,"%*[^\\n]");
326
327      fscanf(fd,"%s",inp->outfile);           // YUV 4:2:2 input format
328      fscanf(fd,"%*[^\\n]");
329
330      fscanf(fd,"%s",inp->reffile);           // reference file
331      fscanf(fd,"%*[^\\n]");
332
333
334      // Frame buffer size
335      fscanf(fd,"%d",&inp->buf_cycle); // may be overwritten in case of RTP NAL
336      fscanf(fd,"%*[^\\n]");
337      if (inp->buf_cycle < 1)
338      {
339          snprintf(errortext, ET_SIZE, "Frame Buffer Size is %d. It has to be at least 1",inp->buf_cycle);
340          error(errortext,1);
341      }
342
343      fscanf(fd,"%d",&(NAL_mode));           // NAL mode
344      fscanf(fd,"%*[^\\n]");
345
346      switch(NAL_mode)
347      {
348      case 0:
349          inp->FileFormat = PAR_OF_ANNEXB;
350          break;
351      case 1:
352          inp->FileFormat = PAR_OF_RTP;
353          break;
354      default:
355          snprintf(errortext, ET_SIZE, "NAL mode %i is not supported", NAL_mode);
356          error(errortext,400);
357      }

```

```

358
359 #ifdef _LEAKYBUCKET_
360     fscanf(fd,"%ld",&inp->R_decoder);           // Decoder rate
361     fscanf(fd, "%*[^\\n]");
362     fscanf(fd,"%ld",&inp->B_decoder);           // Decoder buffer size
363     fscanf(fd, "%*[^\\n]");
364     fscanf(fd,"%ld",&inp->F_decoder);           // Decoder initial delay
365     fscanf(fd, "%*[^\\n]");
366     fscanf(fd,"%s",inp->LeakyBucketParamFile); // file where Leaky Bucket params (computed
367 by encoder) are stored
368     fscanf(fd,"%*[^\\n]");
369 #endif
370
371
372 // Loop Filter parameters flag
373     fscanf(fd,"%d",&inp->LFPParametersFlag); // 0: No Params 1: Read Filter Params, may be
374 overwritten in case of RTP NAL
375     fscanf(fd,"%*[^\\n]");
376     if(inp->LFPParametersFlag != 0 && inp->LFPParametersFlag != 1)
377     {
378         snprintf(errortext, ET_SIZE, "Unsupported value=%d on loop filter parameters flag",inp-
379 >LFPParametersFlag);
380         error(errortext,1);
381     }
382
383
384     fclose (fd);
385
386
387 #if TRACE
388     if ((p_trace=fopen(TRACEFILE,"w"))==0)           // append new statistic at the end
389     {
390         snprintf(errortext, ET_SIZE, "Error open file %s!",TRACEFILE);
391         error(errortext,500);
392     }
393 #endif
394
395
396     if ((p_out=fopen(inp->outfile,"wb"))==0)
397     {
398         snprintf(errortext, ET_SIZE, "Error open file %s ",inp->outfile);
399         error(errortext,500);
400     }
401
402     fprintf(stdout,"-----\\n");
403     fprintf(stdout," Decoder config file           : %s \\n",config_filename);
404     fprintf(stdout,"-----\\n");
405     fprintf(stdout," Input H.26L bitstream           : %s \\n",inp->infile);
406     fprintf(stdout," Output decoded YUV 4:2:0           : %s \\n",inp->outfile);
407     fprintf(stdout," Output status file             : %s \\n",LOGFILE);
408     if ((p_ref=fopen(inp->reffile,"rb"))==0)

```

```

409     {
410         fprintf(stdout," Input reference file           : %s does not exist \n",inp->reffile);
411         fprintf(stdout,"                               SNR values are not available\n");
412     }
413     else
414         fprintf(stdout," Input reference file           : %s \n",inp->reffile);
415
416     if ( inp->abt )
417     {
418         fprintf(stdout," Adaptive Block Transforms           : Used ");
419         if (inp->abt==INTER_ABT)
420             fprintf(stdout,"(Inter only)\n");
421         else
422             fprintf(stdout,"(Inter and Intra)\n");
423     }
424     else
425         fprintf(stdout," Adaptive Block Transforms           : Not used \n");
426     fprintf(stdout,"-----\n");
427 #ifdef _LEAKYBUCKET_
428     fprintf(stdout," Rate_decoder           : %8ld \n",inp->R_decoder);
429     fprintf(stdout," B_decoder             : %8ld \n",inp->B_decoder);
430     fprintf(stdout," F_decoder             : %8ld \n",inp->F_decoder);
431     fprintf(stdout," LeakyBucketParamFile: %s \n",inp->LeakyBucketParamFile); // Leaky Bucket
432     Param file
433     calc_buffer(inp);
434     fprintf(stdout,"-----\n");
435 #endif
436     fprintf(stdout,"Frame  TR  QP  SnrY  SnrU  SnrV  Time(ms)\n");
437 }
438
439 /*!
440 *****
441 * \brief
442 *   Reports the gathered information to appropriate outputs
443 *
444 * \par Input:
445 *   struct inp_par *inp,
446 *   struct img_par *img,
447 *   struct snr_par *stat
448 *
449 * \par Output:
450 *   None
451 *****
452 */
453 void report(struct inp_par *inp, struct img_par *img, struct snr_par *snr)
454 {
455     #define OUTSTRING_SIZE 255
456     char string[OUTSTRING_SIZE];
457     FILE *p_log;
458
459     #ifndef WIN32

```

```

460     time_t now;
461     struct tm *_time;
462 #else
463     char timebuf[128];
464 #endif
465
466     fprintf(stdout,"----- Average SNR all frames -----\\n");
467     fprintf(stdout," SNR Y(dB)      : %5.2f\\n",snr->snr_ya);
468     fprintf(stdout," SNR U(dB)      : %5.2f\\n",snr->snr_ua);
469     fprintf(stdout," SNR V(dB)      : %5.2f\\n",snr->snr_va);
470     fprintf(stdout," Total decoding time : %.3f sec \\n",tot_time*0.001);
471     fprintf(stdout,"-----\\n");
472     fprintf(stdout," Exit JM %s decoder, ver %s ",JM,VERSION);
473 #if ( INI_CTX == 0 )
474     fprintf(stdout,"No CABAC Initialization. ");
475     fprintf(stdout," ABT_max_count %d ",INICNT_ABT);
476 #endif
477     fprintf(stdout,"\\n");
478     // write to log file
479
480     snprintf(string, OUTSTRING_SIZE, "%s", LOGFILE);
481     if ((p_log=fopen(string,"r"))==0)          // check if file exist
482     {
483         if ((p_log=fopen(string,"a"))==0)
484         {
485             snprintf(errortext, ET_SIZE, "Error open file %s for appending",string);
486             error(errortext, 500);
487         }
488         else
489             // Create header to new file
490             {
491                 fprintf(p_log," -----\\n");
492                 fprintf(p_log,"| Decoder statistics. This file is made first time, later runs are appended
493 |\\n");
494                 fprintf(p_log," ----- \\n");
495                 fprintf(p_log,"| Date | Time | Sequence    |#Img|Format|SNRY 1|SNRU 1|SNRV
496 1|SNRY N|SNRU N|SNRV N|\\n");
497                 fprintf(p_log," -----\\n");
498             }
499         else
500         {
501             fclose(p_log);
502             p_log=fopen(string,"a");          // File exist,just open for appending
503         }
504
505 #ifdef WIN32
506     _strdate( timebuf );
507     fprintf(p_log,"| %1.5s |",timebuf );
508
509     _strtime( timebuf );
510     fprintf(p_log,"| %1.5s |",timebuf);

```

```

511  #else
512      now = time ((time_t *) NULL); // Get the system time and put it into 'now' as 'calender time'
513      time (&now);
514      l_time = localtime (&now);
515      strftime (string, sizeof string, "%d-%b-%Y", l_time);
516      fprintf(p_log,"| %1.5s |",string );
517
518      strftime (string, sizeof string, "%H:%M:%S", l_time);
519      fprintf(p_log,"| %1.5s |",string );
520  #endif
521
522      fprintf(p_log,"%20.20s|",inp->infile);
523
524      fprintf(p_log,"%3d |",img->number);
525
526      fprintf(p_log,"%6.3f|",snr->snr_y1);
527      fprintf(p_log,"%6.3f|",snr->snr_u1);
528      fprintf(p_log,"%6.3f|",snr->snr_v1);
529      fprintf(p_log,"%6.3f|",snr->snr_ya);
530      fprintf(p_log,"%6.3f|",snr->snr_ua);
531      fprintf(p_log,"%6.3f|\\n",snr->snr_va);
532
533      fclose(p_log);
534
535      snprintf(string, OUTSTRING_SIZE,"%s", DATADECFILE);
536      p_log=fopen(string,"a");
537
538      if(Bframe_ctr != 0) // B picture used
539      {
540          fprintf(p_log, "%3d %2d %2d %2.2f %2.2f %2.2f %5d "
541              "%2.2f %2.2f %2.2f %5d "
542              "%2.2f %2.2f %2.2f %5d %.3f\\n",
543              img->number, 0, img->qp,
544              snr->snr_y1,
545              snr->snr_u1,
546              snr->snr_v1,
547              0,
548              0.0,
549              0.0,
550              0.0,
551              0,
552              snr->snr_ya,
553              snr->snr_ua,
554              snr->snr_va,
555              0,
556              (double)0.001*tot_time/(img->number+Bframe_ctr-1));
557      }
558      else
559      {
560          fprintf(p_log, "%3d %2d %2d %2.2f %2.2f %2.2f %5d "
561              "%2.2f %2.2f %2.2f %5d "

```

```

562     "%2.2f %2.2f %2.2f %5d %.3f\n",
563     img->number, 0, img->qp,
564     snr->snr_y1,
565     snr->snr_u1,
566     snr->snr_v1,
567     0,
568     0.0,
569     0.0,
570     0.0,
571     0,
572     snr->snr_ya,
573     snr->snr_ua,
574     snr->snr_va,
575     0,
576     (double)0.001*tot_time/img->number);
577 }
578 fclose(p_log);
579 }
580
581 /*!
582  *****
583  * \brief
584  *   Allocates a stand-alone partition structure. Structure should
585  *   be freed by FreePartition();
586  *   data structures
587  *
588  * \par Input:
589  *   n: number of partitions in the array
590  * \par return
591  *   pointer to DataPartition Structure, zero-initialized
592  *****
593  */
594
595 DataPartition *AllocPartition(int n)
596 {
597     DataPartition *partArr, *dataPart;
598     int i;
599
600     partArr = (DataPartition *) calloc(n, sizeof(DataPartition));
601     if (partArr == NULL)
602     {
603         snprintf(errortext, ET_SIZE, "AllocPartition: Memory allocation for Data Partition failed");
604         error(errortext, 100);
605     }
606
607     for (i=0; i<n; i++) // loop over all data partitions
608     {
609         dataPart = &(partArr[i]);
610         dataPart->bitstream = (Bitstream *) calloc(1, sizeof(Bitstream));
611         if (dataPart->bitstream == NULL)
612         {

```

```

613     snprintf(errortext, ET_SIZE, "AllocPartition: Memory allocation for Bitstream failed");
614     error(errortext, 100);
615 }
616 dataPart->bitstream->streamBuffer = (byte *) calloc(MAX_CODED_FRAME_SIZE,
617 sizeof(byte));
618 if (dataPart->bitstream->streamBuffer == NULL)
619 {
620     snprintf(errortext, ET_SIZE, "AllocPartition: Memory allocation for streamBuffer failed");
621     error(errortext, 100);
622 }
623 }
624 return partArr;
625 }
626
627
628
629
630 /*!
631 *****
632 * \brief
633 *   Frees a partition structure (array).
634 *
635 * \par Input:
636 *   Partition to be freed, size of partition Array (Number of Partitions)
637 *
638 * \par return
639 *   None
640 *
641 * \note
642 *   n must be the same as for the corresponding call of AllocPartition
643 *****
644 */
645
646
647 void FreePartition (DataPartition *dp, int n)
648 {
649     int i;
650
651     assert (dp != NULL);
652     assert (dp->bitstream != NULL);
653     assert (dp->bitstream->streamBuffer != NULL);
654     for (i=0; i<n; i++)
655     {
656         free (dp[i].bitstream->streamBuffer);
657         free (dp[i].bitstream);
658     }
659     free (dp);
660 }
661
662
663 /*!

```

```

664 *****
665 * \brief
666 *   Allocates the slice structure along with its dependent
667 *   data structures
668 *
669 * \par Input:
670 *   Input Parameters struct inp_par *inp, struct img_par *img
671 *****
672 */
673 void malloc_slice(struct inp_par *inp, struct img_par *img)
674 {
675     Slice *currSlice;
676     const int buffer_size = MAX_CODED_FRAME_SIZE; // picture size unknown at this time, this
677     value is to check
678
679     img->currentSlice = (Slice *) calloc(1, sizeof(Slice));
680     if ( (currSlice = img->currentSlice) == NULL)
681     {
682         snprintf(errortext, ET_SIZE, "Memory allocation for Slice datastruct in NAL-mode %d failed",
683 inp->FileFormat);
684         error(errortext, 100);
685     }
686     img->currentSlice->rmppi_buffer=NULL;
687     /*! you don't know whether we do CABAC here, hence initialize CABAC anyway
688     // if (inp->symbol_mode == CABAC)
689     if (1)
690     {
691         // create all context models
692         currSlice->mot_ctx = create_contexts_MotionInfo();
693         currSlice->tex_ctx = create_contexts_TextureInfo();
694     }
695     currSlice->max_part_nr = 3; /*! assume data partitioning (worst case) for the following
696     mallocs()
697     currSlice->partArr = AllocPartition(currSlice->max_part_nr);
698 }
699
700
701 /*!
702 *****
703 * \brief
704 *   Memory frees the Slice structure and of its dependent
705 *   data structures
706 *
707 * \par Input:
708 *   Input Parameters struct inp_par *inp, struct img_par *img
709 *****
710 */
711 void free_slice(struct inp_par *inp, struct img_par *img)
712 {
713     Slice *currSlice = img->currentSlice;
714

```

```

715     FreePartition (currSlice->partArr, 3);
716 //     if (inp->symbol_mode == CABAC)
717     if (1)
718     {
719         // delete all context models
720         delete_contexts_MotionInfo(currSlice->mot_ctx);
721         delete_contexts_TextureInfo(currSlice->tex_ctx);
722     }
723     free(img->currentSlice);
724
725     currSlice = NULL;
726 }
727
728 /*!
729 *****
730 * \brief
731 *   Dynamic memory allocation of frame size related global buffers
732 *   buffers are defined in global.h, allocated memory must be freed in
733 *   void free_global_buffers()
734 *
735 * \par Input:
736 *   Input Parameters struct inp_par *inp, Image Parameters struct img_par *img
737 *
738 * \par Output:
739 *   Number of allocated bytes
740 *****
741 */
742 int init_global_buffers(struct inp_par *inp, struct img_par *img)
743 {
744     int i,j;
745
746     int memory_size=0;
747 #ifdef _ADAPT_LAST_GROUP_
748     extern int *last_P_no_frm;
749     extern int *last_P_no fld;
750 #endif
751
752     img->buf_cycle = inp->buf_cycle+1;
753
754     img->buf_cycle *= 2;
755
756     if (img->structure != FRAME)
757     {
758         img->height *= 2;    // set height to frame (twice of field) for normal variables
759         img->height_cr *= 2; // set height to frame (twice of field) for normal variables
760     }
761
762 #ifdef _ADAPT_LAST_GROUP_
763     if ((last_P_no_frm = (int*)malloc(2*img->buf_cycle*sizeof(int))) == NULL)
764         no_mem_exit("get_mem4global_buffers: last_P_no_frm");
765     if ((last_P_no_fld = (int*)malloc(2*img->buf_cycle*sizeof(int))) == NULL)

```

```

766     no_mem_exit("get_mem4global_buffers: last_P_no_fld");
767 #endif
768
769     // allocate memory for encoding frame buffers: imgY, imgUV
770     // byte imgY[288][352];
771     // byte imgUV[2][144][176];
772     memory_size += get_mem2D(&imgY_frm, img->height, img->width); // processing memory in
773     frame mode
774     memory_size += get_mem3D(&imgUV_frm, 2, img->height_cr, img->width_cr); // processing
775     memory in frame mode
776
777     memory_size += get_mem2D(&imgY_top, img->height/2, img->width); // processing memory
778     in field mode
779     memory_size += get_mem3D(&imgUV_top, 2, img->height_cr/2, img->width_cr); // processing
780     memory in field mode
781
782     memory_size += get_mem2D(&imgY_bot, img->height/2, img->width); // processing memory
783     in field mode
784     memory_size += get_mem3D(&imgUV_bot, 2, img->height_cr/2, img->width_cr); // processing
785     memory in field mode
786
787     // allocate memory for multiple ref. frame buffers: mref, mcref
788     // rows and cols for chroma component mcef[ref][croma][4x][4y] are switched
789     // compared to luma mref[ref][4y][4x] for whatever reason
790     // number of reference frames increased by one for next P-frame
791     alloc_mref(img);
792
793     // allocate memory for imgY_prev
794     memory_size += get_mem2D(&imgY_prev, img->height, img->width);
795     memory_size += get_mem3D(&imgUV_prev, 2, img->height_cr, img->width_cr);
796
797     // allocate memory for reference frames of each block: refFrArr
798     // int refFrArr[72][88];
799     memory_size += get_mem2Dint(&refFrArr_frm, img->height/BLOCK_SIZE, img-
800     >width/BLOCK_SIZE);
801     memory_size += get_mem2Dint(&refFrArr_top, img->height/BLOCK_SIZE, img-
802     >width/BLOCK_SIZE);
803     memory_size += get_mem2Dint(&refFrArr_bot, img->height/BLOCK_SIZE, img-
804     >width/BLOCK_SIZE);
805
806     // allocate memory for collocated motion stationarity - int could be replaced with boolean
807     memory_size += get_mem2Dint(&moving_block_frm, img->height/BLOCK_SIZE, img-
808     >width/BLOCK_SIZE);
809     memory_size += get_mem2Dint(&moving_block_top, img->height/BLOCK_SIZE, img-
810     >width/BLOCK_SIZE);
811     memory_size += get_mem2Dint(&moving_block_bot, img->height/BLOCK_SIZE, img-
812     >width/BLOCK_SIZE);
813
814     // allocate memory for reference frame in find_snr
815     // byte imgY_ref[288][352];
816     // byte imgUV_ref[2][144][176];

```

```

817     memory_size += get_mem2D(&imgY_ref, img->height, img->width);
818     memory_size += get_mem3D(&imgUV_ref, 2, img->height_cr, img->width_cr);
819
820     // allocate memory in structure img
821     if(((img->mb_data) = (Macroblock *) calloc((img->width/MB_BLOCK_SIZE) * (img-
822 >height/MB_BLOCK_SIZE),sizeof(Macroblock))) == NULL)
823         no_mem_exit("init_global_buffers: img->mb_data");
824     // if(img->UseConstrainedIntraPred)
825     if (1)    // Always alloc the memory -- we don't know what the parset will tell us later
826     {
827         if(((img->intra_block) = (int**)calloc((j=(img->width/MB_BLOCK_SIZE) * (img-
828 >height/MB_BLOCK_SIZE)),sizeof(int))) == NULL)
829             no_mem_exit("init_global_buffers: img->intra_block");
830         for (i=0; i<j; i++)
831         {
832             if ((img->intra_block[i] = (int*)calloc(4, sizeof(int))) == NULL)
833                 no_mem_exit ("init_global_buffers: img->intra_block");
834         }
835     }
836     // img => int mv[92][72][3]
837     memory_size += get_mem3Dint(&(img->mv_frm),img->width/BLOCK_SIZE +4, img-
838 >height/BLOCK_SIZE,3);
839     memory_size += get_mem3Dint(&(img->mv_top),img->width/BLOCK_SIZE +4, img-
840 >height/BLOCK_SIZE,3);
841     memory_size += get_mem3Dint(&(img->mv_bot),img->width/BLOCK_SIZE +4, img-
842 >height/BLOCK_SIZE,3);
843     // img => int ipredmode[90][74]
844     memory_size += get_mem2Dint(&(img->ipredmode),img->width/BLOCK_SIZE +2 , img-
845 >height/BLOCK_SIZE +2);
846     // int dfMV[92][72][3];
847     memory_size += get_mem3Dint(&(img->dfMV),img->width/BLOCK_SIZE +4, img-
848 >height/BLOCK_SIZE,3);
849     // int dbMV[92][72][3];
850     memory_size += get_mem3Dint(&(img->dbMV),img->width/BLOCK_SIZE +4, img-
851 >height/BLOCK_SIZE,3);
852     // int fw_refFrArr[72][88];
853     memory_size += get_mem2Dint(&(img->fw_refFrArr_frm),img->height/BLOCK_SIZE,img-
854 >width/BLOCK_SIZE);
855     // int bw_refFrArr[72][88];
856     memory_size += get_mem2Dint(&(img->bw_refFrArr_frm),img->height/BLOCK_SIZE,img-
857 >width/BLOCK_SIZE);
858     // int fw_mv[92][72][3];
859     memory_size += get_mem3Dint(&(img->fw_mv),img->width/BLOCK_SIZE +4, img-
860 >height/BLOCK_SIZE,3);
861     // int bw_mv[92][72][3];
862     memory_size += get_mem3Dint(&(img->bw_mv),img->width/BLOCK_SIZE +4, img-
863 >height/BLOCK_SIZE,3);
864
865     memory_size += get_mem3Dint(&colB8mode,3,img->height/B8_SIZE, img->width/B8_SIZE); //
866     collocated ABT block mode
867

```

```

868     // int fw_refFrArr[72][88];
869     memory_size += get_mem2Dint(&(img->fw_refFrArr_top),img->height/BLOCK_SIZE,img-
870 >width/BLOCK_SIZE);
871     // int bw_refFrArr[72][88];
872     memory_size += get_mem2Dint(&(img->bw_refFrArr_top),img->height/BLOCK_SIZE,img-
873 >width/BLOCK_SIZE);
874     // int fw_refFrArr[72][88];
875     memory_size += get_mem2Dint(&(img->fw_refFrArr_bot),img->height/BLOCK_SIZE,img-
876 >width/BLOCK_SIZE);
877     // int bw_refFrArr[72][88];
878     memory_size += get_mem2Dint(&(img->bw_refFrArr_bot),img->height/BLOCK_SIZE,img-
879 >width/BLOCK_SIZE);
880
881     memory_size += get_mem2Dint(&(img->ipredmode_frm),img->width/BLOCK_SIZE + 2 , img-
882 >height/BLOCK_SIZE + 2);
883     memory_size += get_mem2Dint(&(img->ipredmode_top),img->width/BLOCK_SIZE + 2 , (img-
884 >height /2)/BLOCK_SIZE + 2);
885     memory_size += get_mem2Dint(&(img->ipredmode_bot),img->width/BLOCK_SIZE + 2 , (img-
886 >height /2)/BLOCK_SIZE + 2);
887
888     memory_size += get_mem3Dint(&(img->fw_mv_frm),img->width/BLOCK_SIZE + 4, img-
889 >height/BLOCK_SIZE,3);
890     memory_size += get_mem3Dint(&(img->fw_mv_top),img->width/BLOCK_SIZE + 4, (img-
891 >height/2)/BLOCK_SIZE,3);
892     memory_size += get_mem3Dint(&(img->fw_mv_bot),img->width/BLOCK_SIZE + 4, (img-
893 >height/2)/BLOCK_SIZE,3);
894
895     memory_size += get_mem3Dint(&(img->bw_mv_frm),img->width/BLOCK_SIZE + 4, img-
896 >height/BLOCK_SIZE,3);
897     memory_size += get_mem3Dint(&(img->bw_mv_top),img->width/BLOCK_SIZE + 4, (img-
898 >height/2)/BLOCK_SIZE,3);
899     memory_size += get_mem3Dint(&(img->bw_mv_bot),img->width/BLOCK_SIZE + 4, (img-
900 >height/2)/BLOCK_SIZE,3);
901
902     memory_size += get_mem3Dint(&(img->dfMV_top),img->width/BLOCK_SIZE + 4, img-
903 >height/BLOCK_SIZE,3);
904     memory_size += get_mem3Dint(&(img->dbMV_top),img->width/BLOCK_SIZE + 4, img-
905 >height/BLOCK_SIZE,3);
906
907     memory_size += get_mem3Dint(&(img->dfMV_bot),img->width/BLOCK_SIZE + 4, img-
908 >height/BLOCK_SIZE,3);
909     memory_size += get_mem3Dint(&(img->dbMV_bot),img->width/BLOCK_SIZE + 4, img-
910 >height/BLOCK_SIZE,3);
911
912     memory_size += get_mem2Dint(&(img->field_anchor),img->height/BLOCK_SIZE,img-
913 >width/BLOCK_SIZE);
914     memory_size += get_mem2Dint(&(field_mb), img->height/MB_BLOCK_SIZE, img-
915 >width/MB_BLOCK_SIZE);
916
917     memory_size += get_mem3Dint(&(img->wp_weight), 2, MAX_REFERENCE_PICTURES, 3);
918     memory_size += get_mem3Dint(&(img->wp_offset), 2, MAX_REFERENCE_PICTURES, 3);

```

```

919     memory_size += get_mem4Dint(&(img->wbp_weight), 2, MAX_REFERENCE_PICTURES,
920     MAX_REFERENCE_PICTURES, 3);
921
922     // CAVLC mem
923     if((img->nz_coeff = (int****)calloc(img->width/MB_BLOCK_SIZE,sizeof(int****))) == NULL)
924         no_mem_exit("get_mem4global_buffers: nzcoeff");
925     for(j=0;j<img->width/MB_BLOCK_SIZE;j++)
926     {
927         memory_size += get_mem3Dint(&(img->nz_coeff[j]), img->height/MB_BLOCK_SIZE, 4, 6);
928     }
929
930     memory_size += get_mem2Dint(&(img->siblock),img->width/MB_BLOCK_SIZE , img-
931     >height/MB_BLOCK_SIZE);
932
933     if (img->structure != FRAME)
934     {
935         img->height /= 2;    // reset height for normal variables
936         img->height_cr /= 2; // reset height for normal variables
937     }
938
939     img->buf_cycle = inp->buf_cycle+1;
940
941     return (memory_size);
942 }
943
944 /*!
945  *****
946  * \brief
947  *   Free allocated memory of frame size related global buffers
948  *   buffers are defined in global.h, allocated memory is allocated in
949  *   int init_global_buffers()
950  *
951  * \par Input:
952  *   Input Parameters struct inp_par *inp, Image Parameters struct img_par *img
953  *
954  * \par Output:
955  *   none
956  *
957  *****
958  */
959 void free_global_buffers(struct inp_par *inp, struct img_par *img)
960 {
961     int i,j;
962     #ifdef _ADAPT_LAST_GROUP_
963         extern int *last_P_no_frm;
964         extern int *last_P_no fld;
965         free (last_P_no_frm);
966         free (last_P_no fld);
967     #endif
968
969     free_mem2D(imgY_frm);

```

```

970     free_mem2D(imgY_top);
971     free_mem2D(imgY_bot);
972     free_mem3D(imgUV_frm,2);
973     free_mem3D(imgUV_top,2);
974     free_mem3D(imgUV_bot,2);
975     free_mem2D(imgY_prev);
976     free_mem3D(imgUV_prev,2);
977
978     // free multiple ref frame buffers
979     free (mref_frm);
980     free (mcef_frm);
981
982     free (mref_fld);
983     free (mcef_fld);
984
985     free_mem2Dint(refFrArr_frm);
986     free_mem2Dint(refFrArr_top);
987     free_mem2Dint(refFrArr_bot);
988
989     free_mem2Dint(moving_block_frm);
990     free_mem2Dint(moving_block_top);
991     free_mem2Dint(moving_block_bot);
992
993     free_mem2D (imgY_ref);
994     free_mem3D (imgUV_ref,2);
995     // free_mem2D (imgY_tmp);
996     // free_mem3D (imgUV_tmp,2);
997
998     // CAVLC free mem
999     for(j=0;j<img->width/MB_BLOCK_SIZE;j++)
1000     for(i=0;i<img->height/MB_BLOCK_SIZE;i++)
1001     {
1002         if(img->nz_coeff[j][i][0] != NULL) free(img->nz_coeff[j][i][0]);
1003         if(img->nz_coeff[j][i] != NULL) free(img->nz_coeff[j][i]);
1004     };
1005     if (img->nz_coeff !=NULL) free(img->nz_coeff );
1006     free_mem2Dint(img->siblock);
1007
1008     // free mem, allocated for structure img
1009     if (img->mb_data != NULL) free(img->mb_data);
1010
1011     // if(img->UseConstrainedIntraPred)
1012     if (1) // unconditionally allocated since introduction of par sets
1013     {
1014         j = (img->width/16)*(img->height/16);
1015         for (i=0; i<j; i++)
1016         {
1017             free (img->intra_block[i]);
1018         }
1019         free (img->intra_block);
1020     }

```

```

1021     free_mem3Dint(img->mv_frm,img->width/BLOCK_SIZE + 4);
1022     free_mem3Dint(img->mv_top,img->width/BLOCK_SIZE + 4);
1023     free_mem3Dint(img->mv_bot,img->width/BLOCK_SIZE + 4);
1024
1025     free_mem2Dint (img->ipredmode);
1026
1027     free_mem3Dint(img->dfMV,img->width/BLOCK_SIZE + 4);
1028     free_mem3Dint(img->dbMV,img->width/BLOCK_SIZE + 4);
1029
1030     free_mem2Dint(img->fw_refFrArr_frm);
1031     free_mem2Dint(img->bw_refFrArr_frm);
1032     free_mem2Dint(img->fw_refFrArr_top);
1033     free_mem2Dint(img->bw_refFrArr_top);
1034     free_mem2Dint(img->fw_refFrArr_bot);
1035     free_mem2Dint(img->bw_refFrArr_bot);
1036
1037     free_mem3Dint(img->fw_mv,img->width/BLOCK_SIZE + 4);
1038     free_mem3Dint(img->bw_mv,img->width/BLOCK_SIZE + 4);
1039
1040     free_mem3Dint(colB8mode,3); // collocated ABT block mode
1041
1042     free_mem2Dint (img->ipredmode_frm);
1043     free_mem2Dint (img->ipredmode_top);
1044     free_mem2Dint (img->ipredmode_bot);
1045
1046     free_mem3Dint(img->fw_mv_frm,img->width/BLOCK_SIZE + 4);
1047     free_mem3Dint(img->fw_mv_top,img->width/BLOCK_SIZE + 4);
1048     free_mem3Dint(img->fw_mv_bot,img->width/BLOCK_SIZE + 4);
1049
1050     free_mem3Dint(img->bw_mv_frm,img->width/BLOCK_SIZE + 4);
1051     free_mem3Dint(img->bw_mv_top,img->width/BLOCK_SIZE + 4);
1052     free_mem3Dint(img->bw_mv_bot,img->width/BLOCK_SIZE + 4);
1053
1054     free_mem3Dint(img->dfMV_top,img->width/BLOCK_SIZE + 4);
1055     free_mem3Dint(img->dbMV_top,img->width/BLOCK_SIZE + 4);
1056
1057     free_mem3Dint(img->dfMV_bot,img->width/BLOCK_SIZE + 4);
1058     free_mem3Dint(img->dbMV_bot,img->width/BLOCK_SIZE + 4);
1059
1060     free_mem2Dint(img->field_anchor);
1061     free_mem2Dint(field_mb);
1062
1063 }
1064
1065
1066

```