Application Report
SPRA649 - March 2000

MPEG-2 Video Decoder: TMS320C62x Implementation

Ngai-Man Cheung — Texas Instruments Incorporated

ABSTRACT

This application report describes the implementation of the MPEG-2 video decoder on the TMS320C62x DSP. The MPEG-2 video standard specifies the decompression and coded representation for entertainment-quality digital video, and is widely used in different digital video systems including DVB, DTV, DVD, DSS, etc. The decoder software implements all the MPEG-2 main-profile-at-main-level functionality, and conforms to the eXpressDSP™ Algorithm Standard (xDAIS) to enhance reusability. This report describes different aspects of the decoder software, including algorithm overview, coding guidelines, decoder APIs, memory requirement, and performance.

Contents

List of Figures

List of Tables

1 Introduction

This application report describes the implementation of the MPEG-2 video decoder on the TMS320C62x DSP. The decoder software implements all the MPEG-2 main-profile-at-main-level functionality, and conforms to the eXpressDSP Algorithm Standard (xDAIS) to enhance reusability. In the following sections we will describe different aspects of the decoder software, including algorithm overview, coding guidelines, decoder APIs, memory requirement, and performance.

1.1 Algorithm Overview

The *MPEG-2 video* [1,2] standard specifies the decompression and coded representation for entertainment-quality digital video. It is widely used in different digital video systems, including DTV (digital television), DVB (Digital Video Broadcast), DSS (direct satellite system), and DVD (digital versatile disc). The MPEG-2 video decoder plays an important role in consumer electronics like DVD players, set-top boxes, and DSS units. Compared with the hardware implementation, software implementation of the decoder is more flexible, easier to be customize for different applications, and easier to upgrade with new features. Also, the programmability of the device offers the advantage of putting multiple functions (e.g., video decoding, modem function, and speech control interface) in the same hardware platform.

We have implemented the MPEG-2 *main-profile-at-main-level* video decoder, which has the maximum input bit rate of 15 Mbps (megabit per second) and *chrominance* format of 4:2:0. This is the most common format and is being used in many applications.

2 Algorithm Description

Figure 1 shows the MPEG-2 video-decoding algorithm. The MPEG-2 standard employs a number of techniques to achieve high compression ratio while preserving good video quality.

2.1 Interframe Coding Using Motion Compensation

Motion compensation (MC) achieves compression by using the fact that within a short sequence of pictures, the scenes are similar and many objects move only a short distance. By using these temporal redundancies, many parts of the current picture could be predicted by the previously decoded pictures. In MC, the picture is divided into blocks. Two-dimensional motion vectors are computed to tell where to retrieve blocks of pixel values from the previously decoded pictures to predict the block of pixels of the current picture. Compression is achieved by encoding the motion vectors and prediction error instead of the block of pixels. The prediction error has less spatial redundancy and can be compressed effectively by transform coding.

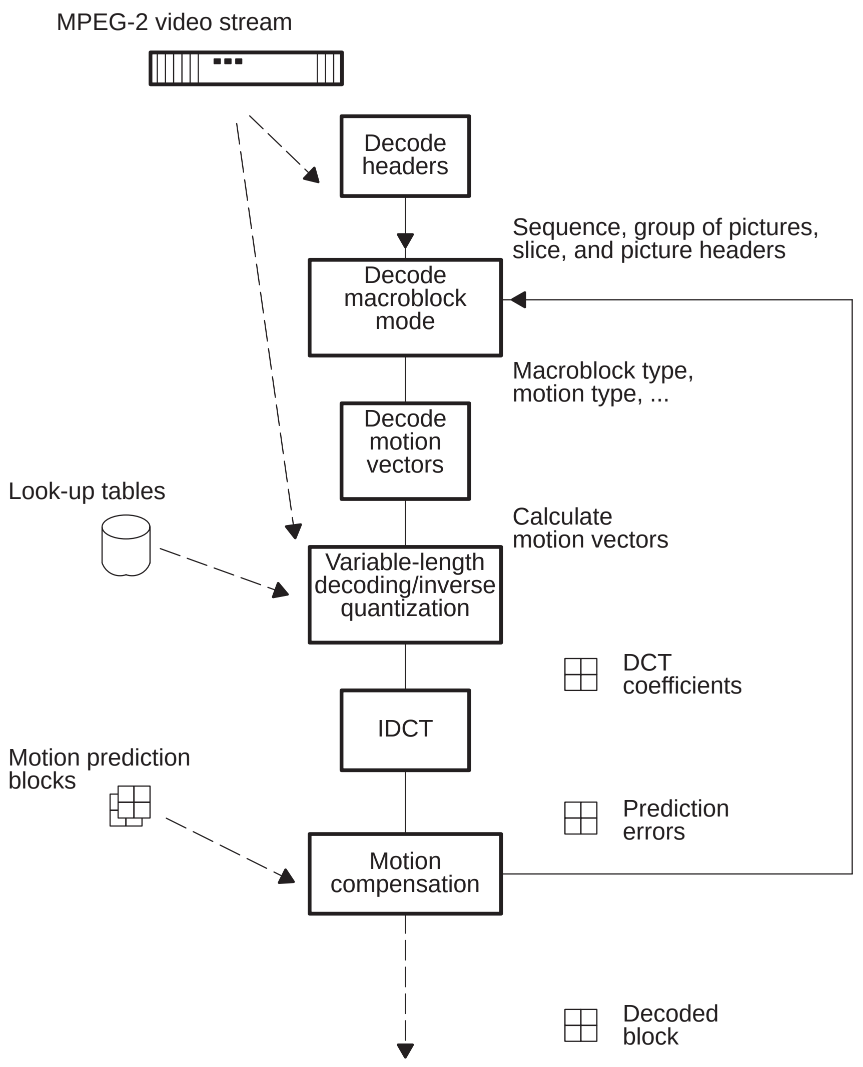

Figure 1. MPEG-2 Video Decoding Algorithm

2.2 Transform Coding Using Discrete Cosine Transform

During encoding, the discrete cosine transform (DCT) is applied to the prediction error in interframe coded macroblock and the pixel values in intraframe coded macroblock. The picture is divided into blocks of 8-by-8 pixels. The DCT transforms the pixel values into another block of the same size, consisting of the horizontal and vertical spatial frequency coefficients representing the detail of the block. While the energy of the image signal and prediction error can be distributed randomly across a block, the energy of the DCT block is concentrated on the low frequency. Compression is achieved by using a quantizer with quantization steps varied by the frequency, according to psycho-visual characteristics such that quantization noise is unlikely to be perceived. Also, many high-frequency coefficients are very small and have a value of zero after quantization. Compression can be achieved by using a zig-zag order to gather the coefficients of value zero, and encoding the block into a series of zero-run and level pairs with run-length encoding.

2.3 Variable-Length Coding

The variable-length coding (VLC) assigns each run-level pair a code word based on the frequency of occurrence of the pair. Pairs that occur more frequently are assigned short code words while those that occur less frequently are assigned long code words. Compression is achieved by the fact that overall, the more frequent shorter code words dominate.

3 Decoder Implementation

In this section, we describe different aspects of our implementation of the video decoder.

3.1 Features

These are the features of the video decoder software:

- The whole video decoding is software-based working on the programmable DSP. There is no hardware assistance of decoding. This ensures maximum flexibility.
- The decoder is completely MPEG-2 main-profile-at-main-level compliant. We have tested the decoder thoroughly with the official MPEG-2 compliance test-streams and have verified that the decoder is completely compliant with the specification. This ensures both the correctness of the algorithm implementation and the quality of the output picture.
- The decoder could also handle MPEG-1 constrained parameters bit-streams (CPB).
- The decoder is xDAIS [3] compliant. We implemented all the xDAIS rules and most of the guidelines. This ensures the decoder algorithm can be easily integrated into different framework systems and environments. Please note that as xDAIS itself may undergo some changes, the software will be updated to reflect these changes.
- The decoder is multichannel enabled. The decoder is reentrant and can handle several different decoding channels simultaneously (this is subject to further testing). The decoder can be interrupted at any place other than the software pipeline code. The interrupt latency shall be less than 10 µs at 250 MHz.

3.2 Decoder Structure

The decoder is divided into the following modules (see Figure 2):

- VLD, which includes functions to perform variable-length decoding, run-length expansion and dequantization
- IDCT, which includes functions to perform inverse discrete cosine transform
- Motion compensation address calculation, which includes functions to calculate the reference blocks location and to fetch the blocks into internal memory
- Motion compensation kernel, which includes functions to calculate the prediction pixels
- Miscellaneous functions to decode header information, motion vectors, etc.
- Implementation of IALG and IRTC interfaces as required in xDAIS

The modules are glued together with the decoder control code. The control code invokes the functions in different modules as well as passes and receives the data.

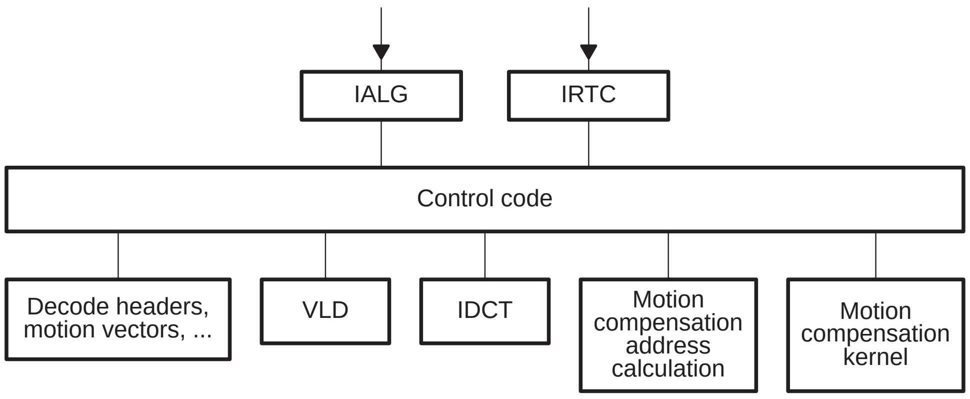
IALG
IRTC
Control code
Decode headers, motion vectors, ...
VLD
IDCT
Motion compensation address calculation
Motion compensation kernel

Figure 2. MPEG-2 Video Decoder Structure

3.3 Coding Guidelines

The decoder program is a mixed C and TMS320C62x assembly language implementation. The coding follows all the xDAIS rules. Some of them are:

- The decoder is reentrant.
- All the data references are fully relocatable. Also, all the decoder code is fully relocatable.
- All external definitions are prefixed with MPEG2VDEC or MPEG2VDEC_ti.

Please refer to the xDAIS document for a complete listing of coding rules.

3.4 Interrupt Issues

The decoder can be interrupted at any place other than the software pipelined loops. The maximum interrupt latency is less than 10 µs at 250 MHz, as recommended in the xDAIS guideline.

3.5 Multichannel Implementation

The decoder is reentrant and can be used in multichannel environment. The decoded information of each channel is retained in the algorithm instance object, MPEG2VDEC_Handle. Client or framework uses the decoder's API (Application Programming Interfaces) MPEG2VDEC_create to create the algorithm instance object for each decoding channel. After that, the framework passes the instance object to the API MPEG2VDEC_apply to decode a picture.

4 Interfacing With the Decoder

The decoder can be configured to run on its own or link to some framework systems. In the latter case, framework can use the decoder's APIs to interface with the decoder. In this section we will describe the decoder APIs. Also, we will give an example framework code to illustrate the interfacing.

4.1 Decoder APIs

The decoder APIs include:

- MPEG2VDEC_init

```
Void MPEG2VDEC_init(Void);
```

Parameters
NULL.

Return Value
NULL.

Description
Decoder initialization. Should be the first call to the decoder.

- MPEG2VDEC_create

```
MPEG2VDEC_Handle MPEG2VDEC_create(
                const IMPEG2VDEC_Fxns *fxns,
                const MPEG2VDEC_Params *prms);
```

Parameters

Parameter	Meaning
const IMPEG2VDEC_Fxns *fxns	Functions table
const MPEG2VDEC_Params *prms	Creation parameter

Return Value
MPEG2VDEC algorithm instance handle.

Description
Create an algorithm instance object. Call this for every decoding channel.

- MPEG2VDEC_delete

```
Void MPEG2VDEC_delete(MPEG2VDEC_Handle handle);
```

Parameters

Parameter	Meaning
MPEG2VDEC_Handle handle	MPEG2VDEC algorithm instance handle

Return Value
NULL.

Description
Delete an algorithm instance object. Call this after completion of a decoding channel.

- MPEG2VDEC_exit

```
Void MPEG2VDEC_exit(Void);
```

Parameters
NULL.

Return Value
NULL.

Description
Decoder finalization.

- MPEG2VDEC_apply

```
Void MPEG2VDEC_apply(MPEG2VDEC_Handle handle,
                      Int *input[],Int *output[]);
```

Parameters

Parameter	Meaning
MPEG2VDEC_Handle handle	MPEG2VDEC algorithm instance handle.
Int *input[1]	Address of the function code, functionCode. The functionCode could be FUNC_DECODE_FRAME or FUNC_START_PARA.
Int *input[2]	Starting of external input bit-stream buffer.
Int *input[3]	Address of the size of the external input bit-stream buffer.
Int *output[1]	Address of the output parameter buffer.
Int *output[2]	Starting of the external output frame buffer.
Others	Reserve for framework specific extension.

Return Value
NULL.

Description
This applies the decoder to the input bit stream and outputs the result in the output buffer. The function code, input[1], should be FUNC_START_PARA at the beginning of a video sequence and FUNC_DECODE_FRAME afterward. The pass-in algorithm instance object identifies the decoding channel.

Framework should call the API MPEG2VDEC_init to initialize the decoder. After that, framework should call the MPEG2VDEC_create to create an algorithm instance object for each channel. The algorithm instance object contains all the status information for each decoding channel. Then, framework can call the MPEG2VDEC_apply to apply the decoder to the input bit stream.

4.1.1 Input Raw Data

Framework passes the MPEG-2 raw input data to the decoder through an input buffer starting at input[2] when calling MPEG2VDEC_apply. The size of the input buffer is pointed by input[3]. The input buffer is organized in a circular fashion. Framework is responsible for filling the buffer and ensuring enough input data to feed the decoding of one picture. Framework can learn how much input data the decoder has consumed by the variable ((DECODE_OUT *) (output[1]) ->next_wptr), which points to the head of the circular input buffer. The input buffer must be a multiple of 4 bytes and aligned on a 4-byte boundary. We recommend the input buffer size to be 512 KB or more. The input buffer should reside in external memory.

4.1.2 Output Decoded Picture

The MPEG2VDEC algorithm returns the decoded picture in the output frame buffer pointed by output[2] at the return of MPEG2VDEC_apply. The algorithm requires keeping four output frames, so the output frame buffer should be of size 4 x Picture_Size at the minimum, where Picture_Size = Picture_Height x Picture_Width x 1.5. Moreover the output frame buffer has to be aligned on a 4-byte boundary. We recommend the output buffer size to be 2440 KB, which can handle 720x576 4:2:0 video properly.

The output picture is stored in the 4:2:0 YU12 format. When algorithm returns, the client should check the variable ((DECODE_OUT *) (output[1]) ->outputting) to see if a decoded picture is ready, and if so the output picture could be found at the memory location ((DECODE_OUT *) (output[1]) ->outframe). The output picture is always in frame format. This is to avoid the confusion in decoding interlaced video when the output picture can be in frame or field format within the same sequence.

4.1.3 Output Parameters

The decoder returns the output parameter in output[1] at the return of MPEG2VDEC_apply. The parameter is either the structure START_OUT at the beginning of a sequence or DECODE_OUT afterward, as shown in Figure 3.

```
/*********************************************************/
/* Output parameter at the beginning of sequence.       */
/*********************************************************/
typedef struct _START_OUT {
  Int fault;                        /* Any problem occur?    */
  Int ld_mpeg2;                     /* MPEG-2 stream?        */
  Int bit_rate;                     /* Input bit rate        */
  Int picture_rate;                 /* Output picture rate   */
  Int vertical_size;                /* Ori. pic. dimension   */
  Int horizontal_size;
  Int coded_picture_width;          /* Coded pic. dimension  */
  Int coded_picture_height;
  Int chroma_format;
  Int chrom_width;
  Int prog_seq;                     /* Progressive seq.?     */
} START_OUT;
/*********************************************************/
/* Output parameter afterward.                          */
/*********************************************************/
typedef struct _DECODE_OUT {
  Int fault;                        /* Any problem occur?    */
  Int pict_type;                    /* I, P or B-pic.       */
  Int pict_struct;
  Int next_wptr;                    /* Head of cir. input    */
  Int topfirst;
  Int end_of_seq;                   /* End of sequence?      */
  Int outputting;                   /* Output any frame?     */
  SmUns outframe;                   /* Starting of out pic. */
} DECODE_OUT;
```

Figure 3. Video Decoder Output Parameters

4.2 Example Framework Code

Figure 4 shows an example framework code to illustrate the usage of the decoder's APIs.

```
#define SHARE_MPEG2_RDBUF_SIZE    (128 * 1024)
unsigned int share_bsbuf_storage[SHARE_MPEG2_RDBUF_SIZE];
/* 512KB input buffer. 4-bytes alignment. */
#define MAX_PICT_SIZE             0x098800
/* 610KB. Enough for 720x576 4:2:0. */
#define NO_OF_FRAME_BUF           4
unsigned char frame_all_storage[NO_OF_FRAME_BUF * MAX_PICT_SIZE];
/* Output buffer. 4-bytes alignment. */
#define MAXPARAM                  5
int *in[MAXPARAM];
int *out[MAXPARAM];
int h_share_mpeg2_rdbuf_size = SHARE_MPEG2_RDBUF_SIZE;
int functionCode;
…
MPEG2VDEC_Handle mpeg2vdec;
/**********************************************************/
/* To do -- fill up the whole input buffer               */
/**********************************************************/
old_ptr = 0; /* head of circular buffer */
MPEG2VDEC_init();
mpeg2vdec = MPEG2VDEC_create(&MPEG2VDEC_TI_IMPEG2VDEC, NULL);
in[1] = &functionCode;                  out[1] = (int *) &out_para[0];
in[2] = (int *) &share_bsbuf_storage[0]; out[2] = (int *) &frame_all_storage[0];
in[3] = &h_share_mpeg2_rdbuf_size;
functionCode = FUNC_START_PARA;
MPEG2VDEC_apply(mpeg2vdec, in, out);          /* decode sequence header */
while (! (decode_out-> end_of_seq) ){         /* not end of sequence    */
  functionCode = FUNC_DECODE_FRAME;
  MPEG2VDEC_apply(mpeg2vdec, in, out);        /* decode one picture     */
  decode_out = (DECODE_OUT *)(out[1]);
  if (decode_out-> outputting) {
    /*********************************************/
    /* To do -- output the frame                  */
    /* starting at location decode_out-> outframe */
    /*********************************************/
  }
  /*************************************************/
  /* To do -- fill the input buffer between        */
  /* old_ptr and decode_out->next_wptr from source */
  /*************************************************/
  old_ptr = decode_out->next_wptr;
} /* while */
MPEG2VDEC_delete(mpeg2vdec);
MPEG2VDEC_exit();
/* End of program */
```

Figure 4. An Example Framework to Interface the Video Decoder

5 Running the Program

This section describes the procedure to build and run the video decoder.

5.1 Build Procedure

To build the code:

1. Compile and assemble individual files with

   cl6x –@cl.cmd

   or use the Code Composer Studio™ (CCS) project file Mpg2vdec.mak.

2. Create the decoder library Mpeg2vdec_ti.lib with

   ar6x @ar.cmd

3. Link the decoder library with the target system.

The code directories contain all the files to compile, assemble, and link the decoder. The CCS project file Mpg2vdec.mak compiles and assembles all the source files, and links them with the linker file Mpg2vdec.cmd to produce an example stand alone executable Mpg2vdec.out. The executable Mpg2vdec.out can be loaded into the DSP and tested with the Windows™ GUI interface program. The object files can also be packaged into the decoder library Mpeg2vdec_ti.lib using the ar6x command and linked to the target system by following the above steps.

5.2 Run the Program

To run the example executable:

1. Load and start the DSP executable Mpg2vdec.out using CCS or evm6xldr.
2. Start the Windows GUI interface program Mplay.exe.
3. In the Mplay.exe program, select File –> Open, and choose an MPEG-2 video file.
4. In the Mplay.exe program, select Control –> Play to start decoding.

If you have Microsoft™ DirectDraw™ software package on your computer, then you would be able to see the decoded video.

5.3 Test and Validation

We have tested the decoder thoroughly with the official MPEG-2 compliance test streams and have verified that the decoder is completely compliant with the requirements. This ensures both the correctness of the algorithm implementation and the quality of the output picture.

Code Composer Studio is a trademark of Texas Instruments.
Windows, Microsoft, and DirectDraw are registered trademarks of Microsoft Corporation.

6 Memory Requirements and Performance

The section reports the memory requirements and performance of the decoder.

Table 1 lists all data and program memory requirements.

Table 1. Memory Requirements of the Decoder

	Internal Memory	External Memory
Data Memory		
Heap data memory	7552 (7.4 KB)	3022848 (2952 KB)
Stack space data memory	16384 (16 KB)	
Static data memory	10616 (10.4 KB)	
Total	34552 (33.8 KB)	3022848 (2952 KB)
Program Memory		
Total	65504 (64 KB)	26432 (26 KB)

The external memory includes the input bit-stream buffer and the output frame buffers. They are allocated by framework and passed to the decoder as arguments of API MPEG2VDEC_apply.

We have benchmarked the video decoder with several MPEG-2 video streams. The results are shown in Table 2. The decoder software was benchmarked on a C6201 DSP with internal memory configured as mapped mode (cache disabled). The performance was interpreted by the cycle count measured in CCS. Please contact Texas Instruments for the latest performance information.

Table 2. Performance of the Decoder

Test Stream	Information	Format	Bit-rate (Mbps)	Numbers of picture	Performance (MHz)	
Public domain MPEG-2 stream	Mobl_080.m2v. Available from http://www.mpeg.org.	704x576 x25fps	7.629	375	Av. Max.	214 239
DVD test stream #1	Vts_05_1.vob in Panasonic DVD Demonstration Disc.	720x480 x30fps	9.346	10000	Av. Max.	204 255
DVD test stream #2	Vts_40_1.vob in Philips DVD Demonstration Disc.	720x576 x25fps	9.346	1000	Av. Max.	190 220
MPEG-2 compliance test stream	Bitstream gi_9.m2v in MPEG-2 test suite.	720x480 x30fps	14.305	16	Av. Max.	261 267

7 References

1. ISO/IEC 11172-2, *Coding of moving pictures and associated audio for digital storage media at up to about 1.5Mbits/s, Part 2: Video (MPEG-1 video standard)*.
2. ISO/IEC 13818-2, *Generic coding of moving pictures and associated audio information, Part 2: Video (MPEG-2 video standard)*.
3. Texas Instruments, *The eXpressDSP Algorithm Standard (xDAIS): Rules and Guidelines*, September 1999 (SPRU352).