Application Report
SPRA704 - December 2000

TMS320C6000 JPEG Implementation

Ajai Narayan
Jungki Min
Vishal Markandey

Digital Signal Processing Solutions

ABSTRACT

This application report describes the implementation of the Joint Photographic Experts Group (JPEG) Encoder and Decoder on TMS320C6000™ digital signal processors (DSPs). The JPEG Encoder and Decoder are eXpressDSP™ Algorithm Standard compliant. This document describes the details of JPEG Encoder and Decoder implementation, APIs, and measured performance.

Contents

List of Figures

List of Tables

1 Introduction

This application report describes the implementation of JPEG, the still image compression standard on TMS320C6000 DSPs. This JPEG implementation is subject to the following constraints:

- DCT-based, Sequential, 8-bit precision samples of image components (Y–Cb–Cr 4:4:4/4:2:2/4:2:0).
- Two quantization tables (one each for luma and chroma). Supports tables K1 and K2 in the standards document. Arbitrary quantization tables are supported, tables may change per image.
- Two DC, 2 AC tables (separate sets for luma and chroma). Supports tables K3, K4, K5, and K6 in the standards document. Support is limited to these tables only.
- The encoder expects the image data as three separate raster scanned components in memory (i.e. non-interleaved Y–Cr–Cb).
- The decoder outputs the image as three separate raster scanned components in contiguous memory.
- Each scan contains a complete image component. A single image component is contained in a scan.
- The implementation does not handle ‘restart intervals’ even though their presence should not affect the decoding.
- The decoder expects the first scan to start within the first DMA packet in the bit-stream. At least the following markers should be included in the first DMA packet.

  0xFFD8: Start of Image

  0xFFC0: Start of Frame (Baseline DCT)

  0xFFDB: Define Quantization Table

  0xFFDA: Start of Scan

The TI JPEG encoder outputs those markers within the first 590 bytes, so the decoder minimum input DMA packet should be 590 bytes and the default is set to 640 bytes.

- Lossless JPEG encode/decode are not supported.
- Progressive image transmission capability is not supported.
- Image component dimensions (rows, columns, Y/C) must be multiples of 8.
- A simple compression ratio control capability is provided in the encoder.
- The decoder can only decode those bit-streams that have a structure identical to that produced by the encoder. Please see Section 5 for a description of the bit-stream structure used.

2 The JPEG (ISO DIS 10918) Standard

It is recommended that this application note be read with reference to the JPEG standards document ISO DIS 10918. The standard is available as Appendix A in the book by William B. Pennebaker and Joan L. Mitchell (Von Nostrand Reinhold, New York ISBN 0–442–01272–1).

3 JPEG Encoder

The JPEG standard is a broad standard encompassing several compression and transmission modes. In order to facilitate future expansion to other modes, this implementation has a very modular construction. A single thread of control code handles all individual routines (kernels) which are called multiple times as required by the application. This control code will always be in ‘C’ to facilitate changes in control architecture.

3.1 JPEG Encoder Algorithms

Figure 1 provides an overview of the processing involved in JPEG Encoder.

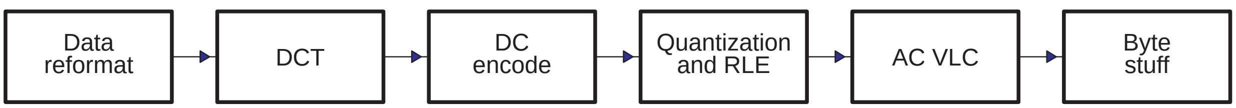

Figure 1. JPEG Encoder

Data Reformat: This operation converts raster scanned image component data into a contiguous set of 8x8 image blocks. Figure 2 shows the raw image data as stored in the memory. All image samples belonging to the same row in the image frame are represented by a single alphabet. Figure 3 shows the reformatted data as required for any block based compression scheme. This operation also converts the dynamic range of the pixel intensity values from (0–255) to (–128, 127) thus eliminating the DC bias of the signal.

x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x
y y y y y y y y y y y y y y y y y y y y y y y y y y y y y y y y y y y y y y y y y y y y y y y y y y y
z z z z z z z z z z z z z z z z z z z z z z z z z z z z z z z z z z z z z z z z z z z z z z z z z z z z
o o o o o o o o o o o o o o o o o o o o o o o o o o o o o o o o o o o o o o o o o o o o o o o o o o o o
p p p p p p p p p p p p p p p p p p p p p p p p p p p p p p p p p p p p p p p p p p p p p p p p p p p p
q q q q q q q q q q q q q q q q q q q q q q q q q q q q q q q q q q q q q q q q q q q q q q q q q q q q
k k k k k k k k k k k k k k k k k k k k k k k k k k k k k k k k k k k k k k k k k k k k k k k k k k k k
n n n n n n n n n n n n n n n n n n n n n n n n n n n n n n n n n n n n n n n n n n n n n n n n n n n n

Figure 2. Raster Scanned Image Data

x x x x x x x x y y y y y y y y z z z z z z z z o o o o o o o o p p p p p p p p q q q q q q q q k k k k
k k k k n n n n n n n n x x x x x x x x y y y y y y y y z z z z z z z z o o o o o o o o p p p p p p p p
q q q q q q q q k k k k k k k k n n n n n n n n x x x x x x x x y y y y y y y y z z z z z z z z o o o o
o o o o p p p p p p p p q q q q q q q q k k k k k k k k n n n n n n n n x x x x x x x x y y y y y y y y
– – – – – – – –
– – – – – – – –

Figure 3. Reformatted Image Data

DCT: This operation performs a 2-D Discrete Cosine Transform (DCT) on the reformatted 8x8 block of image samples and outputs a corresponding 8x8 block of 2-D frequency components. The mathematical expression for the DCT is given below:

$$S_{vu} = \frac{1}{4} C_u C_v \sum_{x=0}^{7} \sum_{y=0}^{7} S_{yx} \cos\left[\frac{(2x+1)u\pi}{16}\right] \cos\left[\frac{(2y+1)u\pi}{16}\right] \quad (1)$$

where

C_u, C_v = 1/ $\sqrt{2}$ for u, v = 0; C_u, C_v = 1 otherwise
S_{vu} is the DCT component at u,v
S_{yx} is the spatial sample value of the image pixel at x,y

The 2D DCT is separated into two 1D operations to reduce the number of processing operations as shown below:

- perform eight 1D DCTs, one for each row of the array (row computation).
- perform eight 1D DCTs, one for each column of the array resulting from the row IDCT computation (column computation).

DC Encode: This step quantizes and Huffman encodes (also called Variable Length Coding, VLC) the DC coefficients obtained from the DCT module. In JPEG, the DC coefficient is differentially encoded i.e, a difference between the present and the preceding DC component is computed and this difference is quantized and encoded. Quantization involves an inherent division operation with an element from the quantizer table. In this implementation, a reciprocal quantizer table, pre-computed from the quantizer table, is used.

Quantization and RLE: This step quantizes the AC coefficients, casts them in a zig-zag pattern and run-level encodes the resulting coefficients. As in the case of the DC coefficient, quantization involves an inherent division operation with an element from the quantizer table. In this implementation, a reciprocal quantizer table, pre-computed from the quantizer table, is used. The result of the zig-zag re-ordering of transformed coefficients is shown in Figure 4.

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63

0	1	5	6	14	15	27	28
2	4	7	13	16	26	29	42
3	8	12	17	25	30	41	43
9	11	18	24	31	40	44	53
10	19	23	32	39	45	52	54
20	22	33	38	46	51	55	60
21	34	37	47	50	56	59	61
35	36	48	49	57	58	62	63

Figure 4. Zig-Zag Reordering of Transformed Coefficients (Input and Output)

AC VLC: This step performs Variable Length Coding (VLC) of the run-level pairs that are output by the quantization routine to construct the entropy coded segments of the image. The variable length codes in JPEG do not map directly to quantized AC coefficients. Instead, they map to a positive integer value. This integer represents the additional number of bits to be appended to the variable length code itself. The value of the additional bits is calculated as part of the encoding process.

Byte Stuff: In the JPEG standard, control markers are flagged by a 0xFF. This flag is followed by one or more bytes of control code. A 0x00 byte following a 0xFF byte signifies that the 0xFF byte is indeed part of the data and not control segments. This step inserts a 0x00 byte after every 0xFF byte within the entropy coded (i.e. VLC) segments.

3.2 JPEG Encoder Control Code

The encoding process consists of several data processing and transmission operations. The encoder has to insert several headers (frame-header, scan-headers etc.) into the JPEG bit-stream to facilitate decoding. The standard specifies that a JPEG file contain all the necessary tables required for decoding. Hence, the encoder has to perform several auxiliary transmission related functions in addition to image compression.

The control function **jpgenc_ti( )** is found in the file **‘jpgenc_ti.c’**. It chronologically calls all the encoder component routines like DCT, quantization, run-level encoding, variable length encoding etc., and performs data translations between the encoder routines. It operates the required double buffering scheme for DMA read-in of image component data from the external memory and DMA write-out of the JPEG bit-stream to the external memory. It takes care of masking the data transfers by the encoding operation itself.

Figure 5 shows the schematic diagram for the control flow for the encoder:

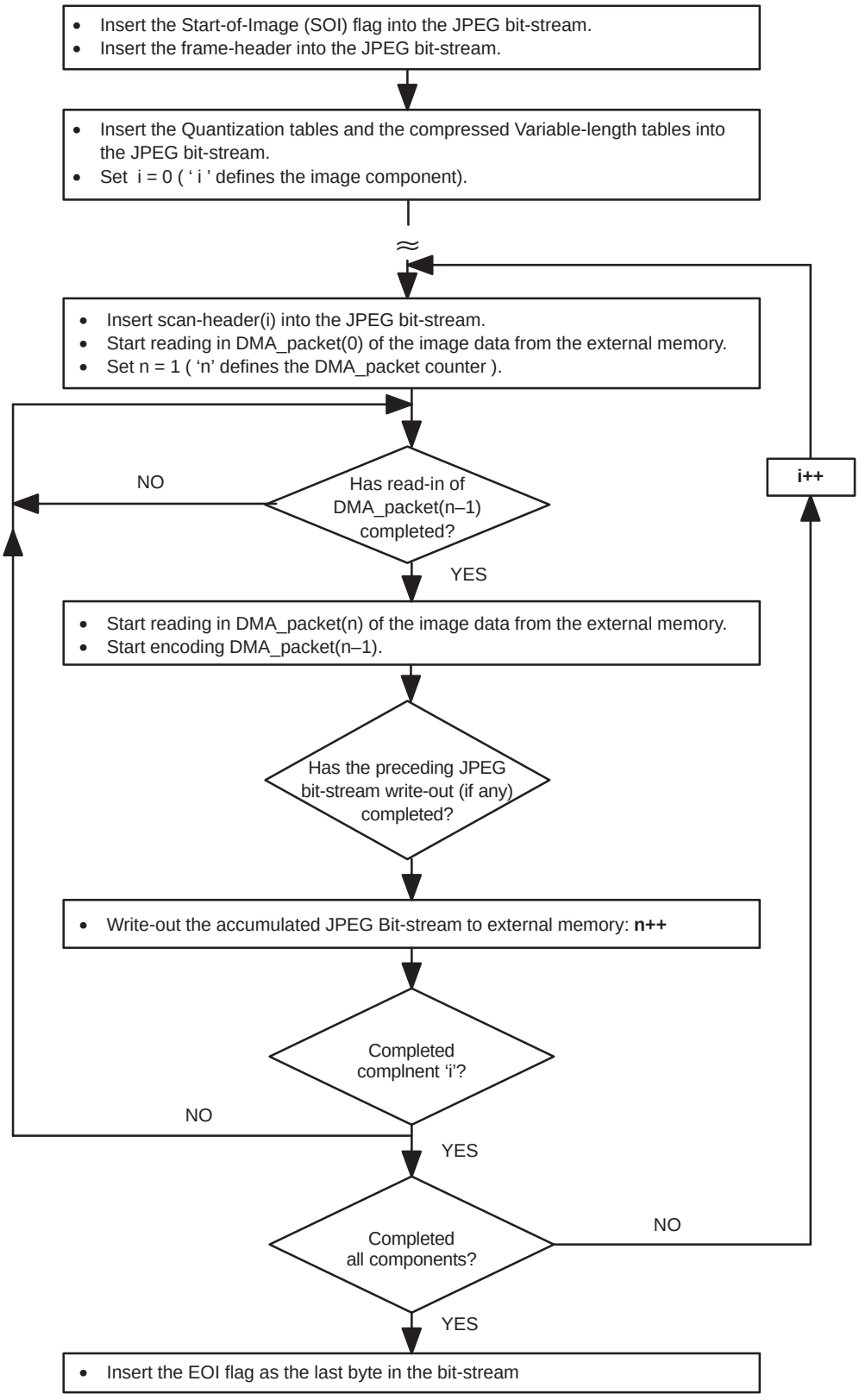

Figure 5. Encoder Control Flow

The function **jpegenc_ti()** contains the overall control logic for the encoder application. It is called from the driver function with all the required parameters defined. A prototype for the driver is shown in the file **'main_interface.c'.** The form of the function jpegenc_ti() is shown below.

```
jpegenc_ti (const unsigned char     sample_prec,
            const unsigned char     num_comps,
            const unsigned char     num_qtables,
            const unsigned char     interleaved,
            const unsigned int      format,
            const unsigned short    *num_lines,
            const unsigned short    *num_samples,
            unsigned char           **raw_img,
            unsigned char           *output_area);

sample_prec:      8–12 bit image samples (only 8–bits currently supported)
num_comps:        3 – color; 1 – grayscale
num_qtables:      2 – color; 1 – grayscale
interleaved:      1 – interleaved; 2 – non–interleaved
format:           0x01110111 – 4:4:4; 0x01120112 – 4:2:0; 0x01110112 – 4:2:2
num_lines:        Vertical dimension of the frame in terms of lines
num_samples:      Horizontal dimension of the frame in terms of samples
raw_img:          Pointer to the three memory areas that hold the image data
```

3.3 JPEG Encoder Auxiliary Functions

Frame-header Specification: Frame header specification is performed in the routine **framehdr_spec( )**. This routine inserts the standard frame header into the JPEG bit-stream. The frame header contains the image specific parameters eg., image height, width, number of image components etc. which are to be transmitted to the decoder.

Quantization Tables Specification: Quantization tables are specified in the routine **quant_table_spec( )**. This routine inserts the quantization tables into the JPEG bitstream.

Variable Length Tables Specification: Variable length tables are specified by the routine **Huffman_tables_spec( )**. This routine inserts compressed forms of huffman tables into the JPEG bit-stream.

Scan-header Specification: Frame header specification is performed in the routine **scanhdr_spec( )**. This inserts a standard scan-header into the bit-stream. The scan header contains image component specific information required by the decoder such as number of image components in this scan, the tables to be used to decode the data in this scan etc.

3.4 JPEG Encoder API

The API wrapper is derived from template material provided in the TMS320 DSP Algorithm Standard documentation. Knowledge of the algorithm standard is essential to understand the API wrapper. A complete discussion on how to make the algorithm eXpressDSP compliant is beyond the scope of this document, however the algorithm interface will be discussed as knowledge of this ensures inter-operability of algorithms. An algorithm is said to be eXpressDSP compliant if it implements the IALG Interface and observes all the programming rules in the algorithm standard. The core of the ALG interface is the IALG_Fxns structure type, in which a number of function pointers are defined. Each eXpressDSP-compliant algorithm **must** define and initialize a variable of type IALG_Fxns. In IALG_fxns, algAlloc(), algInit() and algFree() are required, while other functions are optional.

```
typedef struct IALG_Fxns {
Void    *implementationId;
Void    (*algActivate)(IALG_Handle);
Int     (*algAlloc)(const IALG_Params *, struct IALG_Fxns **, IALG_MemRec *);
Int     (*algControl)(IALG_Handle, IALG_Cmd, IALG_Status *);
Void    (*algDeactivate)(IALG_Handle);
Int     (*algFree)(IALG_Handle, IALG_MemRec *);
Int     (*algInit)(IALG_Handle, const IALG_MemRec *, IALG_Handle, const

         IALG_Params *);
Void    (*algMoved)(IALG_Handle, const IALG_MemRec *, IALG_Handle, const

         IALG_Params *);
Int     (*algNumAlloc)(Void);
} IALG_Fxns;
```

The algorithm implements the algAlloc() function to inform the framework of its memory requirements by filling the memTab structure. It also informs the framework whether there is a parent object for this algorithm. Based on information it obtains by calling algAlloc(), the framework then allocates the requested memory. AlgInit() initializes the instance persistent memory requested in algAlloc(). After the framework has called algInit(), the instance of the algorithm pointed to by handle is ready to be used. To delete an instance of the algorithm pointed to by handle, the framework needs to call algFree(). It is the algorithm's responsibility to set the addresses and the size of each memory block requested in algAlloc() such that the application can delete the instance object without creating memory leaks.

The API for the JPEG Encoder is:

```
/*
 * ======== ijpegenc.h ========
 *  IJPEGENC Interface Header
 */
#ifndef IJPEGENC_
#define IJPEGENC_


#include <std.h>
#include <xdas.h>
#include <ialg.h>
#include <ijpeg.h>


/*
 *  ======== IJPEGENC_Handle ========
 *  This handle is used to reference all JPEGENC instance objects
 */
typedef struct IJPEGENC_Obj *IJPEGENC_Handle;


/*
 *  ======== IJPEGENC_Obj ========
 *  This structure must be the first field of all JPEGENC instance objects
 */
typedef struct IJPEGENC_Obj {
    struct IJPEGENC_Fxns *fxns;
} IJPEGENC_Obj;
/*
 *  ======== IJPEGENC_Params ========
 *  This structure defines the creation parameters for all JPEGENC objects
 */
typedef struct IJPEGENC_Params {
    Int size;     /* must be first field of all params structures */
    unsigned int  sample_prec;
    unsigned int  num_comps;
    unsigned int  num_qtables;
    unsigned int  interleaved;
    unsigned int  format;
    unsigned int  quality;
    unsigned int  num_lines[3];
    unsigned int  num_samples[3];
    unsigned int  output_size;


} IJPEGENC_Params;
typedef IJPEGENC_Params IJPEGENC_Status;
/*
 *  ======== IJPEGENC_PARAMS ========
 *  Default parameter values for JPEGENC instance objects
 */
extern IJPEGENC_Params IJPEGENC_PARAMS;
```

```
/*
 *  ======== IJPEGENC_Fxns ========
 *  This structure defines all of the operations on JPEGENC objects
 */
typedef struct IJPEGENC_Fxns {
    IALG_Fxns ialg;    /* IJPEGENC extends IALG */
      XDAS_Bool    (*control)(IJPEGENC_Handle handle, IJPEG_Cmd cmd, IJPEGENC_Status
*status);
    XDAS_Int32  (*encode)(IJPEGENC_Handle handle, XDAS_Int8* in, XDAS_Int8* out);


} IJPEGENC_Fxns;


#endif /* IJPEGENC_ */
```

3.5 JPEG Encoder Performance

JPEG Encoder performance has been measured on a wide range of test images. The following performance is based on measurements on C6201 EVM and C6211 DSK.

Table 1. JPEG Encoder Performance

Image Resolution	Frames/Sec With 200 MHz C6201	Frames/Sec With 150 MHz C6211†
128x128 (4:2:0)	569 frames/sec	382 frames/sec
256x256 (4:2:0)	156 frames/sec	106 frames/sec
352x288 (4:2:0) [CIF resolution]	104 frames/sec	69 frames/sec
640x480 (4:2:0) [VGA resolution]	36 frames/sec	24 frames/sec
720x480 (4:2:0) [SDTV resolution]	32 frames/sec	21 frames/sec

† C6211 performance data based on [48K cache/16K SRAM] configuration. Recommended for JPEG.

4 JPEG Decoder

The JPEG standard is a broad standard encompassing several compression and transmission modes. In order to facilitate future expansion to other modes, this implementation has a very modular construction. A single thread of control code handles all individual routines (kernels) which are called multiple times as required by the application. This control code will always be in ‘C’ to facilitate changes in control architecture.

4.1 JPEG Decoder Algorithms

Figure 6 provides an overview of the processing involved in JPEG Decoder.

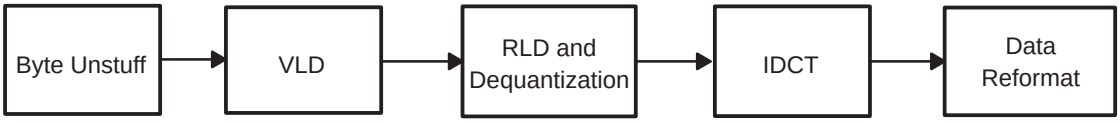

Figure 6. JPEG Decoder

Byte Unstuff: In the JPEG standard, control markers are flagged by a preceding 0x'FF' followed by one or more bytes of control code. A 0x00 byte following a 0xFF byte signifies that the 0xFF byte is indeed part of the data and not control. Thus, every 0xFF byte occurring in the entropy (VL) coded data is followed by a redundant 0x00 byte which has to be stripped off.

Variable Length Decode (VLD): VLD decodes the JPEG bit-stream and generates image data in the DCT domain. The decoding is done in two steps 1) DC coefficient decoding followed by 2) AC (run, level) decoding. The decoding is conceptually implemented as a series of exhaustive look-ups into a predefined table. The C6000 ISA has a single cycle instruction lmbd that can reduce the decoding complexity. It facilitates a faster decoding 1) by decoding several bits during each table look-up and 2) by effectively constraining the search range within the table for each look-up.

The **lmbd** instruction gives the bit-position where a first bit reversal occurs in a register. Many intelligent decoding methods can be designed using this capability. For example, in this implementation, the value returned by the **lmbd** instruction is used to select a sub-table from the entire variable length table for an exhaustive search. VLD using the **lmbd** operation is shown below, register A4 contains valid 32 bits from the JPEG bit-stream. The **lmbd** operation on A4 returns the number of leading 1's in A4 which results in

- Decoding of 5 code-bits in a single cycle.
- Unique identification the sub-table for exhaustive search.
- Identification of the number of additional bits after the five 1's to be extracted from A4 for the exhaustive search.

A4 = 1 1 1 1 1 0 1 1 0 0 1 1 1 1 1 1 1 0 0 1 1 0 0 0 0 1 0 1 1 0 1 0

lmbd (0, A4) = 5 ⇒ Unique sub-table and number of additional bits to be extracted from A4 for further decoding

Such optimizations in the VLD mechanism restrict the use of the algorithm to a specific table. This is because the structure of the table is exploited during the decoding process. The baseline JPEG recommends separate DC and AC tables for luminance and chrominance components. Hence VLD decoding has to be done separately for the two components in order to exploit individual table structures.

Variable length decoding with partial JPEG bit-streams is a non-trivial problem. The DMA packets used for transferring data to DSP generally do not end at block boundaries. Complex structures would be required to track the number of run-level pairs decoded and to ensure that data is not read beyond the end of a DMA packet. To circumvent this problem, the number of bytes that are consumed from the DMA packet when a complete block (8x8) is decoded is monitored. If this number exceeds a threshold value (smaller than the DMA packet size), the VLD is discontinued and the blocks that have been decoded thus far are grouped into a set. This set of blocks is passed down the decoding chain in a single pass. The succeeding DMA packet is concatenated to the remaining bytes in the present packet and the process is repeated.

Run Level Decoding (RLD) and Dequantization: The quantized DC coefficient and the (run, level) pairs that were decoded from the variable length decoder routines are input to this function. This function expands the (run, level) pairs with explicit zeroes and quantized AC coefficients in the same zig-zag pattern as at the encoder. It then performs inverse quantization (i.e, a multiplication with the corresponding element in the quantization tables) of all non-zero coefficients.

Inverse Discrete Cosine Transform (IDCT): This routine performs the inverse DCT on the frequency components of an 8x8 data block and outputs a corresponding 8x8 block of image component samples. The input to this routine is an array of amplitude values corresponding to specific 2D frequencies. The output from it is an array containing a 2D array of amplitude values which correspond to image samples.

$$S_{yx} = \frac{1}{4}\sum_{u=0}^{7}\sum_{v=0}^{7} C_u C_v S_{vu} \cos\left[\frac{2x+1)u\pi}{16}\right]\cos\left[\frac{2y+1v\pi}{16}\right] \quad (2)$$

where C_u, C_v = 1/ $\sqrt{2}$ for u, v = 0; C_u, C_v = 1 otherwise.
S_{vu} is the DCT component at u,v
S_{yx} is the spatial sample value of the image pixel at x,y

The 2D IDCT is separated into two 1D operations to reduce the number of processing operations as shown below:

- perform eight 1D IDCTs, one for each row of the array (row computation).
- perform eight 1D IDCTs, one for each column of the array resulting from the row IDCT computation (column computation).

Data Reformat: Data reformatting converts a contiguous set of 8x8 image blocks into a raster scanned image frame. Figure 7 shows the decoded image data as stored in the memory before reformat.

x x x x x x x x y y y y y y y y z z z z z z z z o o o o o o o o p p p p p p p p q q q q q q q q k k k k
k k k k n n n n n n n n x x x x x x x x y y y y y y y y z z z z z z z z o o o o o o o o p p p p p p p p
q q q q q q q q k k k k k k k k n n n n n n n n x x x x x x x x y y y y y y y y z z z z z z z z o o o o
o o o o p p p p p p p p q q q q q q q q k k k k k k k k n n n n n n n n x x x x x x x x y y y y y y y y
– – – – – – – –
– – – – – – – –

Figure 7. Decoded Image Data Before Reformat

All image data belonging to a single 8x8 block occur contiguously followed by the data for the next block. Successive groups of 8 samples are depicted by a different alphabet. Figure 8 shows the reformatted data as required for display of an image frame. Reformatting also converts the dynamic range of the pixel intensity values from (–128, 127) to (0, 255) as per the JPEG standard.

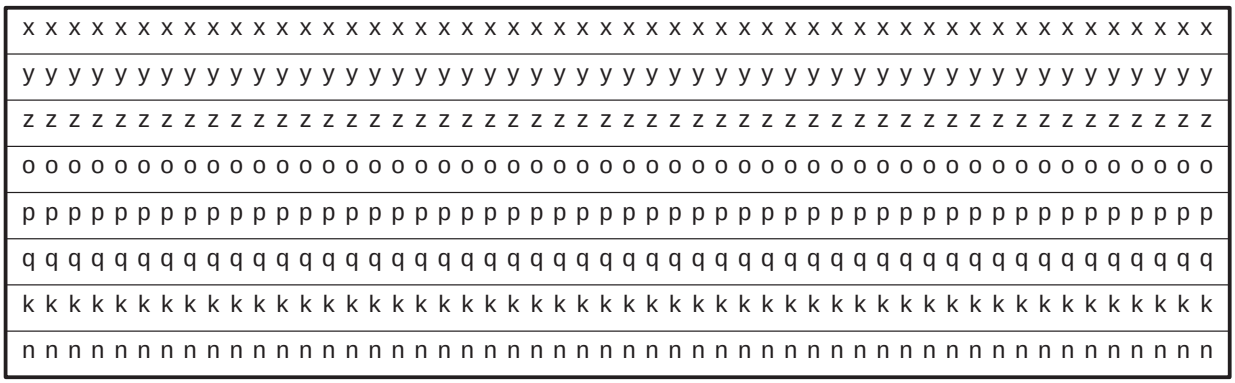

Figure 8. Reformatted Image Data in Raster Scan Format

4.2 JPEG Decoder Control Code

The control function **jpgdec_ti( )** is found in the file **'jpgdec_ti.c'**. It chronologically calls all the decoder component routines like variable length decoding, run-level decoding, inverse quantization, IDCT etc., and performs data translations between the decoder routines. It operates the required double buffering scheme for DMA read-in of the JPEG bit-stream from the external memory and DMA write-out of image data to the external memory. It takes care of masking the data transfers by the decoding operation itself.

The function **jpegdec_ti()** contains the overall control logic for the decoder application. It is called from the driver function with all the required parameters defined. A prototype for the driver is shown in the file **'main_interface.c'.** The form of the function jpegdec_ti() is shown below.

```
jpegdec_ti (unsigned char *jpg_bit_stream,
            unsigned char *output_addr );

jpg_bit_stream: The pointer to the JPEG bit-stream (input)
ouput_addr: The pointer to the output data (Y–Cr–Cb)
```

Figure 9 shows the schematics of the control code for the decoder in this implementation.

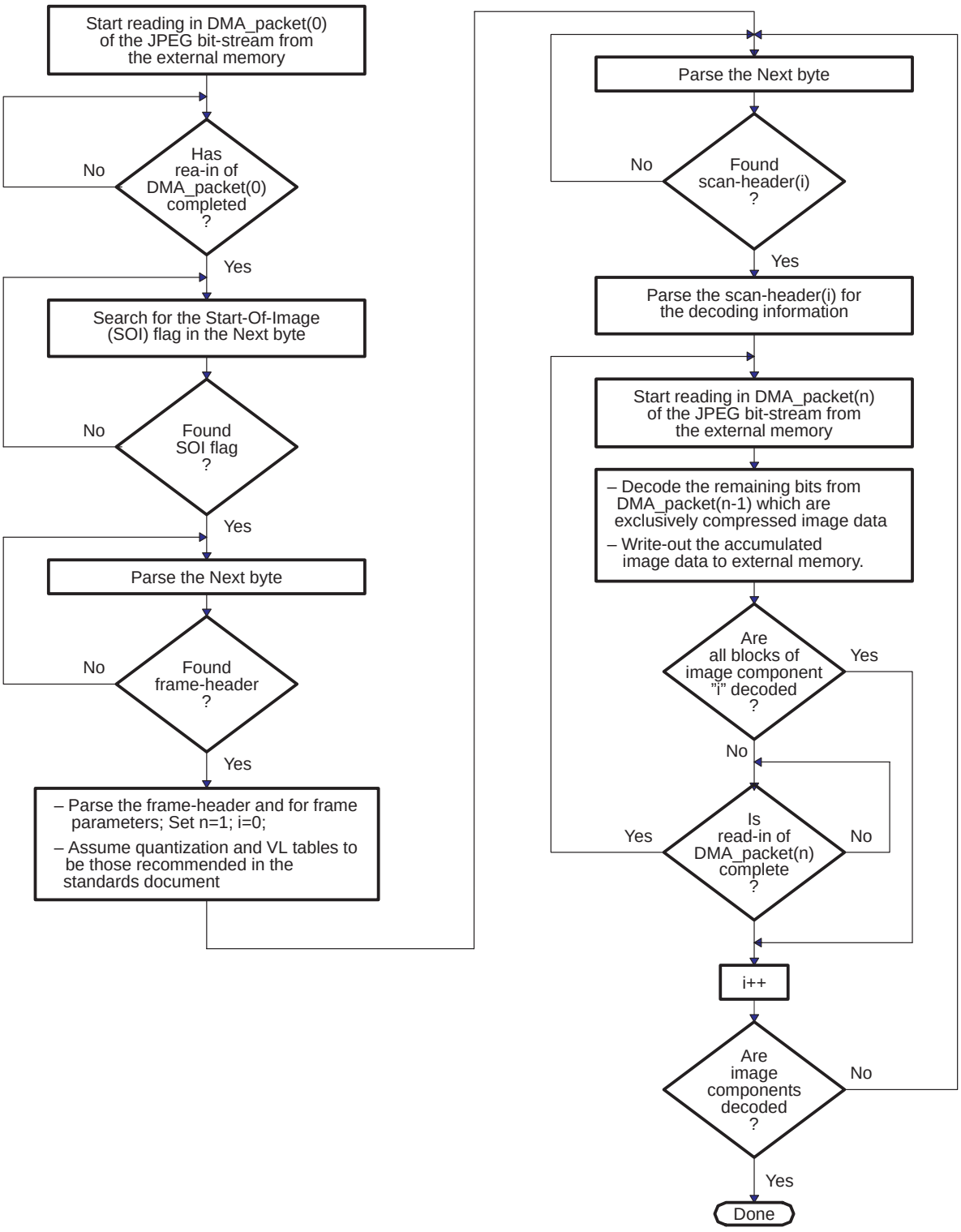

Figure 9. Decoder Control Flow

4.3 JPEG Decoder API

The API wrapper is derived from template material provided in the TMS320 DSP Algoriuthm Standard documentation. Knowledge of the algorithm standard is essential to understand the API wrapper.The API for the JPEG Decoder is:

```
/*
 *  ======== ijpegdec.h ========
 *  IJPEGDEC Interface Header
 */
#ifndef IJPEGDEC_
#define IJPEGDEC_
#include <xdas.h>
#include <ialg.h>
#include <ijpeg.h>
/*
*  ======== IJPEGDEC_Handle ========
 *  This handle is used to reference all JPEG_DEC instance objects
 */
typedef struct IJPEGDEC_Obj *IJPEGDEC_Handle;
/*
 *  ======== IJPEGDEC_Obj ========
 *  This structure must be the first field of all JPEG_DEC instance objects
 */
typedef struct IJPEGDEC_Obj {
struct IJPEGDEC_Fxns *fxns;
} IJPEGDEC_Obj;
/*
*  ======== IJPEGDEC_Params ========
 *  This structure defines the creation parameters for all JPEG_DEC objects
 */
typedef struct IJPEGDEC_Params {
    Int size;  /* must be first field of all params structures */
} IJPEGDEC_Params;
/*
 *  ======== IJPEGDEC_Status ========
 *  This structure defines the status parameters for all JPEG_DEC objects
 */
```

```
typedef struct IJPEGDEC_Status {
    Int size;  /* must be first field of all params structures */
    unsigned int    num_lines[3];
    unsigned int    num_samples[3];
    unsigned int    gray_FLAG;
    unsigned int    outputSize;
} IJPEGDEC_Status;
/*
 *  ======== IJPEGDEC_PARAMS ========
 *  Default parameter values for JPEG_DEC instance objects
 */
extern IJPEGDEC_Params IJPEGDEC_PARAMS;
 /*  ======== IJPEGDEC_Fxns ========
 *  This structure defines all of the operations on JPEG_DEC objects
 */
typedef struct IJPEGDEC_Fxns {
    IALG_Fxns  ialg;    /* IJPEGDEC extends IALG */
    XDAS_Bool  (*control)(IJPEGDEC_Handle handle, IJPEG_Cmd cmd, IJPEGDEC_Status
*status);
    XDAS_Int32 (*decode)(IJPEGDEC_Handle handle, XDAS_Int8 *in, XDAS_Int8 *out);
} IJPEGDEC_Fxns;
#endif  /* IJPEGDEC_ */
```

4.4 JPEG Decoder Performance

JPEG Decoder performance has been measured on a wide range of test images and compression factors. The following performance is based on measurements on C6201 EVM and C6211 DSK.

Table 2. JPEG Decoder Performance

Image Resolution	Frames/Sec With 200 MHz C6201	Frames/Sec With 150 MHz C6211†
128x128 (4:2:0)	528 frames/sec	374 frames/sec
256x256 (4:2:0)	159 frames/sec	108 frames/sec
352x288 (4:2:0) [CIF resolution]	107 frames/sec	72 frames/sec
640x480 (4:2:0) [VGA resolution]	39 frames/sec	26 frames/sec
720x480 (4:2:0) [SDTV resolution]	35 frames/sec	23 frames/sec

† C6211 performance data based on [48K cache/16K SRAM] configuration. Recommended for JPEG.

Appendix A JPEG Bit-Stream Structure

The JPEG Encoder implementation discussed in this document produces a specific type of bit-stream upon encoding an image. This type of bit-stream can be decoded by most JPEG decoders. On the other hand, the JPEG Decoder implementation discussed in this document is limited to decoding bit-streams that have a structure idential to the one produced by the encoder discussed in this document. This bit-stream structure for each compressed frame is as shown in Figure A–1.

The bit-stream for each compressed frame starts with the Start Flag of value FFD8 as specified by the ISO JPEG standard. The start flag is followed by the Frame Header, which specifies the source image characteristics such as sample precision, number of lines, number of samples per line, number of image components in frame, component identifier, horizontal sampling factor, vertical sampling factor, and quantization table selector. See section B.2.2 of the ISO JPEG standard for further details.

The frame header is followed by the Quantization Tables. By default, tables K1 and K2 of the ISO JPEG standard are used. However, the encoder may change the quantization tables per frame. The quantization tables are included in the bit-stream using the syntax provided in tsection B.2.4.1 of the ISO JPEG standard.

The quantization tables are followed by the Huffman Tables. 2 DC and 2 AC component tables (separate table for luminance and chrominance) are supported. The supported tables are tables K3, K4, K5, K6 from the ISO JPEG standard. The Huffman tables are coded, using the coding scheme described in sections K.3.3.1 and K.3.3.2 of the ISO JPEG standard. These coded tables are then included in the bit-stream using the syntax specified in section B.2.4.2 of the ISO JPEG standard. The encoder does not provide the option to use any tables other than the ones from the ISO JPEG standard mentioned above. Please note that while the encoder includes these tables in the bit-stream, as required by the standard, the decoder does not use this information from the bit-stream to reconstruct the Huffman tables. Instead, the decoder assumes that the tables used are the ones mentioned above, and the software for Huffman decoding is specific to those tables only.

The Huffman tables are followed by the Huffman encoded Y data in the bit-stream. This is the Y data for the entire frame or image, and it is considered a single “scan” in the terminology of the JPEG standard. A Scan Header is included before the actual Y data. See section B.2.3 of the ISO JPEG standard for a description of the Scan Header Syntax. The Y data is followed by Cb and Cr data, including corresponding scan headers, in similar form.

The Cr data is followed by the End Flag of value FFD9 as specified by the ISO JPEG standard.

This structure of the bit-stream is repeated for each frame.

Note that the decoder expects the first scan to start within the first DMA packet in the bit-stream. The Quantization and Huffman tables for any frame should be contained within the first DMA packet corresponding to that frame. This restricts the decoder to only the type of bit-streams shown in Figure 5-1. As an example of a bit-stream that the decoder will not decode, one could potentially have bit-streams where the Quantization and Huffman tables for Y and C data were located separately, say contiguous with the Y and C data. The current control code is not set up to handle such variations in bit-stream structure and the decoder will not decode such bit-streams.

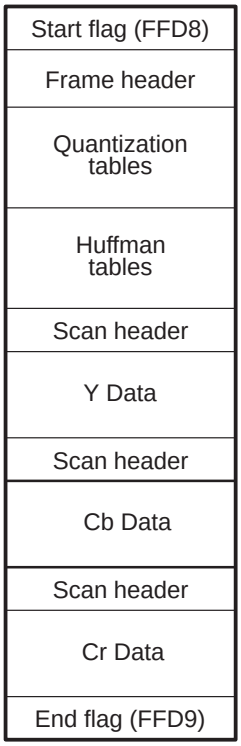

Figure A–1. JPEG Bit-Stream Structure

IMPORTANT NOTICE

Texas Instruments and its subsidiaries (TI) reserve the right to make changes to their products or to discontinue any product or service without notice, and advise customers to obtain the latest version of relevant information to verify, before placing orders, that information being relied on is current and complete. All products are sold subject to the terms and conditions of sale supplied at the time of order acknowledgment, including those pertaining to warranty, patent infringement, and limitation of liability.

TI warrants performance of its semiconductor products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Customers are responsible for their applications using TI components.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards must be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance or customer product design. TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used. TI's publication of information regarding any third party's products or services does not constitute TI's approval, warranty or endorsement thereof.