AN4996
Application note

Hardware JPEG codec peripheral in STM32F76/77xxx and STM32H743/53/45/55/47/57/50xx microcontrollers

Introduction

This application note describes the use of the hardware JPEG codec peripheral for JPEG decoding/encoding applications in STM32F76/77xxx and STM32H743/53/45/55/47/57/50xx microcontrollers.

The STM32F76/77xxx and STM32H743/53/45/55/47/57/50xx microcontrollers embed a dedicated hardware JPEG codec peripheral providing a fast and simple hardware JPEG image compressor and decompressor with:

- Full management of JPEG file headers,
- Fully programmable Huffman tables (two ACs and two DCs),
- Up to four programmable quantization tables,
- Fully programmable minimum coded unit (MCU).

The hardware JPEG codec supports pixel input/output formats in YCbCr or RGB (3 color components), grayscale (1 color component) and CMYK (4 color components) with fully programmable sub-sampling factors for each component.

To fully benefit from this application note, the user must be familiar with:

- The STM32's JPEG codec peripheral as described in the STM32F76/77xxx reference manual (RM0410) and the STM32H743/53/45/55/47/57/50xx reference manuals (RM0399, RM0433) available from the STMicroelectronics website *www.st.com*.
- The JPEG compression standard (JPEG ISO/IEC 10918-1 ITU-T recommendation T.81) and the JFIF file format standard (JPEG file interchange format).

Reference documents

- *STM32F76xxx and STM32F77xxx advanced Arm®-based 32-bit MCUs* reference manual (RM0410),
- *STM32H745/755 and STM32H747/757 advanced Arm®-based 32-bit MCUs* reference manual (RM0399),
- *STM32H742, STM32H743/753 and STM32H750 Value line advanced Arm®-based 32-bit MCUs* reference manual (RM0433),
- Embedded software for STM32F7 Series (STM32CubeF7) and STM32H7 Series (STM32CubeH7).

Table 1. Applicable products

Type	Product lines and part numbers
Microcontrollers	STM32F777BI, STM32F777II, STM32F777NI, STM32F777VI, STM32F777ZI, STM32F7x8 line, STM32F7x9 line
	STM32H743/753 line, STM32H745/755 line, STM32H747/757 line, STM32H750 Value line

Contents

List of tables

List of figures

1 Hardware JPEG codec overview

The hardware JPEG codec peripheral is compliant with the JPEG standard (JPEG ISO/IEC 10918-1 ITU-T recommendation T.81). It can decode/encode JPEG compressed images with 8-bit per sample.

The hardware JPEG codec peripheral provides a hardware acceleration for entropy-codec segments (ECS) encoding and decoding. It supports the JPEG header generation and parsing. The hardware JPEG codec peripheral also supports JFIF (JPEG file interchange format), the de facto standard used to encode JPEG images. However, all application-specific marker segments found in these data streams are ignored. The JPEG codec supports up to four color components, four quantization tables and two sets of DC and AC Huffman tables.

The hardware JPEG codec provides the flexibly to specify which quantization and Huffman tables to be used for each component.

The JPEG encoding and decoding operations, as defined by the JPEG standard, are performed by blocks. The JPEG standard defines the MCU (minimum codec unit) as the minimum number of blocks that can be encoded or decoded. In the hardware JPEG codec peripheral, the MCU composition is programmable. The hardware JPEG codec allows to define how many blocks in each MCU belong to a particular color component. Each block is an 8x8 array of samples where each sample is defined on 8 bits (one byte). Given so each block is a 64-byte array (1 byte per sample).

The hardware JPEG codec supports pixel input/output formats in YCbCr or RGB (3 color components), grayscale (1 color component) and CMYK (4 color components) with fully programmable sub-sampling factors for each component.

Using the STM32H743/53/45/55/47/57/50xx devices for JPEG decoding operations, and when the output color format is YCbCr, the Chrom-Art Accelerator peripheral (also called DMA2D) allows to convert YCbCr blocks (output of the JPEG decoder) to RGB pixels ready for display.

Using the STM32H743/53/45/55/47/57/50xx devices for encoding (all color formats) or for decoding with a color format different than YCbCr (case of the gray scale or the CMYK color format), the conversion from/to RGB pixels is not hardware accelerated and must be performed by the software.

Using the STM32F76/77xxx devices for decoding or encoding, the YCbCr to RGB conversion is not accelerated and must be performed by the software.

The STM32CubeF7/H7 MCU Packages provide a dedicated JPEG utility software with necessary APIs allowing to perform the conversion of JPEG MCU blocks to/from RGB pixels (available under \Firmware\Utilities\JPEG).

The STM32CubeF7/H7 provides the dedicated HAL (hardware abstraction layer) driver for the JPEG codec peripheral:

- STM32CubeF7: stm32f7xx_hal_jpeg.c/ stm32f7xx_hal_jpeg.h
- STM32CubeH7: stm32h7xx_hal_jpeg.c/ stm32h7xx_hal_jpeg.h

This document applies to Arm®[a]-based devices.

a. Arm is a registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere.

2 Hardware JPEG codec settings versus color space

2.1 YCbCr color space

2.1.1 YCbCr to/from RGB conversion

The YCbCr to/from RGB conversions and chroma sampling are described by the JPEG file interchange format (JFIF) standard. The JFIF compliant files have generally the extension .jpg, .jpeg, .JPG, .JPEG.

While the JPEG standard (JPEG ISO/IEC 10918-1 ITU-T recommendation T.81) does not define a color space to be used for the source row image, the JFIF standard defines two possible color space: either grayscale (Y luminance) or colored (YCbCr luminance and chrominance).

The JFIF standard uses the YCbCr color instead of the original RGB color space. This color space allows to separate luminance component (Y) which tells how bright the pixel is (basically a grayscale signal) from the 2 chrominance components Cb and Cr which give the color of the pixel. The conversion matrices to switch from RGB color space to YCbCr and vice versa are as follows:

Figure 1. YCbCr/RGB color conversion

$$\begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.16874 & -0.33126 & 0.5 \\ 0.5 & -0.41869 & -0.08131 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} + \begin{bmatrix} 0 \\ 128 \\ 128 \end{bmatrix}$$

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1.402 \\ 1 & -0.34414 & -0.71414 \\ 1 & 1.77200 & 0 \end{bmatrix} \begin{bmatrix} Y \\ Cb - 128 \\ Cr - 128 \end{bmatrix}$$

Knowing that the human eyes are more sensitive to the brightness variation than the color variation, using YCbCr allows to define two separate quantization tables: one for the luminance and a second for the chrominance (Cb and Cr) components allowing to quantize harder the chrominance (at least for low frequencies).

2.1.2 YCbCr quantization tables

Figure 2 and *Figure 3* show the sample luminance and chrominance quantization tables provided by the JPEG standard. These tables, as described in the standard, give good results on 8-bit per sample luminance and chrominance images (which is the case of the STM32F7/H7 hardware JPEG codec).

The standard also describes these tables as follows: if these quantization values are divided by 2, the resulting reconstructed image is usually nearly indistinguishable from the source image.

Figure 2. YCbCr luminance quantization table

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

Figure 3. YCbCr chrominance quantization table

17	18	24	47	99	99	99	99
18	21	26	66	99	99	99	99
24	26	56	99	99	99	99	99
47	66	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99

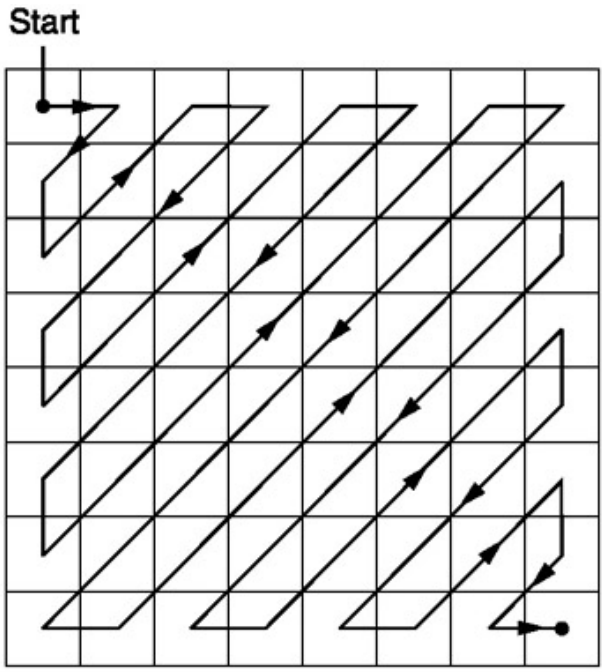
Note: These tables are provided in zig-zag order.

Figure 4 and *Figure 5* provide the zig-zag sequence.

Figure 4. Zig-zag sequence for quantization table

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 5. Zig-zag scanning order of quantization table



The STM32CubeF7 and STM32CubeH7 use these default tables for encoding and decoding in conjunction with a user quality factor.

- In encoding, the STM32CubeF7/H7 JPEG HAL driver allows the user to define a quality factor in percentage from 1% to 100%. The quality factor is then used to scale the above tables as follows:
  - When the quality is in the range 50% to 100%: scaling_factor = 200 - 2 x quality.
  - When the quality is less than 50% then: scaling_factor = 5000 / quality.

As a result, when the quality is set to 100% the scaling factor goes to zero then all the table entries go to 1 (as zero entries are systematically replaced by 1s). This gives a minimum quantization loss.

The quantization tables are then programmed into a dedicated memory table in the hardware JPEG codec: QMEM0 for luminance (Y) and QMEM1 for chrominance (Cb and Cr).

- In decoding, the STM32CubeF7/H7 JPEG HAL driver allows to retrieve the quality as follows. For each value of the quantization table:
  - Read the quantization coefficient and calculate the scaling factor in percentage versus the corresponding value in the reference table.

    Scale = (100 x quantization_coefficient) / reference_table_value).
  - If the quantization_coefficient is equal to 1 then the quality is 100%.
  - Else if the scale is less than100: quality = (200 - scale) /2.
  - Else quality = 5000 / scale.

The encoding quality is calculated as the average of the calculated quality for each coefficient of the quantization table (that is 64 coefficients). Only the luminance table is used to calculate the average quality.

The QMEMx memory tables of the hardware JPEG codec are used to store/retrieve the scaled quantization tables (versus the reference tables). These tables are accessed in Zig-Zag order.

The hardware JPEG codec provides a RAM "QMEM" region to store/retrieve up to 4 quantization tables (respectively for up to 4 color components). The QMEM RAM is located at the offset 0x0050 to 0x014C. Each table size is 64 bytes (that are 16 32-bits words).

Figure 6. Hardware JPEG QMEM RAM

0x0050-0x014C	**QMEM**	QMem RAM																															
	Reset value	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

The STM32CubeF7/H7 JPEG HAL driver, by default, uses only 2 quantization tables for YCbCr color space:

- QMEM0: used for luminance (Y) component. 64 bytes located at the offset 0x0050.
- QMEM1: used for chrominance (Cb and Cr) components. 64 bytes located at the offset 0x0090.

Note: The QMEM RAM is available for read/write only when the hardware JPEG codec is stopped, that is, no ongoing encoding/decoding operation (bit 0 'START' of the JPEG_CONFR0 register set to zero).

The STM32CubeF7/H7 JPEG HAL driver uses by default the table given in *Figure 2* for (Y) luminance component and the table given in *Figure 3* for both Cb and Cr chrominance components. The JPEG HAL driver offers also the possibility for the user to define a quantization table per color components (3 quantization tables in this case). If needed to customize quantization tables, the user must provide 3 quantization tables (one per component). These tables are used (after scaling with the quality factor) to program respectively QMEM0 to QMEM2 RAM tables of the hardware JPEG codec (where QMEM2 table is located at the offset 0x00D0).

The HAL function "HAL_JPEG_SetUserQuantTables" is the API used to customize the user quantization tables.

2.1.3 YCbCr chrominance sub-sampling and minimum codec unit (MCU) construction

In YCbCr color space, the chrominance components can be sub-sampled (information reduced) without significant visual quality reduction. The hardware JPEG codec allows to define the horizontal and vertical sampling factors and the number of 8x8 blocks for each component (up to 4 components) using the JPEG_CONFR4 to JPEG_CONFR7 registers.

For encoding or for decoding with the header parsing disabled, these registers are used to inform the codec with the encoding parameters for each component.

For decoding with the header parsing enabled, these registers are automatically filled by the hardware codec once the JPEG header parsing is done, that is, the bit 6 (header parsing done flag) of the JPEG_SR register goes to 1.

The hardware JPEG codec offers the possibility to define any sampling factor and number of 8x8 blocks for each component using the JPEG CONFR4-7 registers, see below the register description.

JPEG codec configuration register 4-7 (JPEG_CONFR4-7)

Address offset: 0x0010 + 0x4 * i, where “i” is image component from 0 to 3

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
HSF[3:0]				VSF[3:0]				NB[3:0]				QT[1:0]		HA	HD
rw				rw				rw				rw		rw	rw

Bits 31: 16 Reserved

Bits 15: 12 **HSF[3:0]**: horizontal sampling factor
Horizontal sampling factor for component i.

Bits 11: 8 **VSF[3:0]**: vertical sampling factor
Vertical sampling factor for component i.

Bits 7: 4 **NB[3:0]**: number of blocks
Number of data units minus 1 that belong to a particular color in the MCU.

Bits 3: 2 **QT[1:0]**: quantization table
Selects quantization table used for component i.

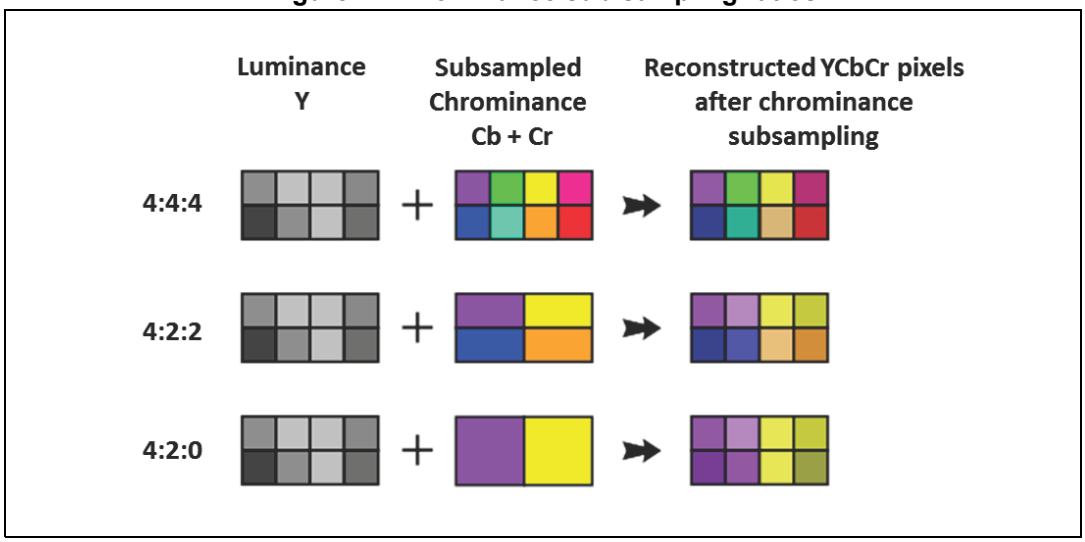
Bit 1 **HA**: Huffman AC
Selects the Huffman table for encoding AC coefficients.

Bit 0 **HD**: Huffman DC
Selects the Huffman table for encoding DC coefficients.

The STM32CubeF7/H7 JPEG HAL driver offers the possibility to select one of the following chrominance sub-sampling ratios:

- 4:4:4 → No chrominance sub-sampling keeps full information for all Y, Cb and Cr components.
- 4:2:2 → Cb and Cr are horizontally sampled at the half compared to the Y component (keeping only the chrominance information of one pixel over two horizontally adjacent pixels).
- 4:2:0 → Cb and Cr are horizontally and vertically sampled at the half compared to the Y component (keeping only the chrominance information of one pixel over four adjacent pixels).

Figure 7. Chrominance sub-sampling ratios

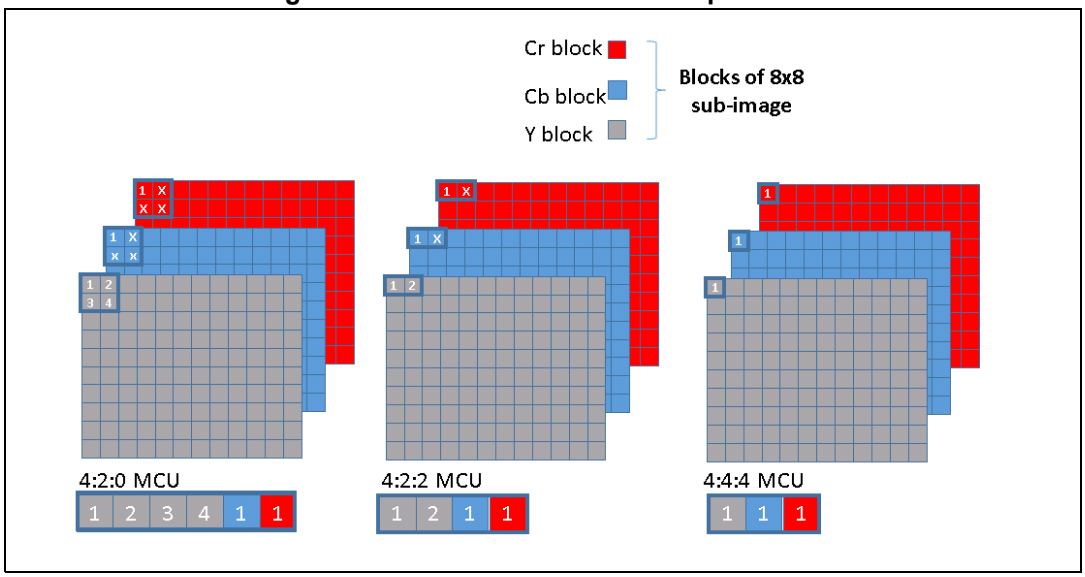


The sub-sampled YCbCr pixels are then encapsulated into blocks of 8x8 called MCU (minimum coded unit).

Each MCU is composed of:

- 4:4:4 → one 8x8 Y block + one 8x8 Cb block + one 8x8 Cr block → a total of 192 bytes.
- 4:2:2 → two 8x8 Y blocks + one 8x8 Cb block + one 8x8 Cr block → a total of 256 bytes.
- 4:2:0 → four 8x8 Y blocks + one 8x8 Cb block + one 8x8 Cr block → a total of 384 bytes.

Figure 8. Minimum coded unit encapsulation



The following are the JPEG codec registers settings in the YCbCr color space depending on the chroma sampling.

2.1.4 CONFR1 register settings

- Bits 31:16 **YSIZE[15:0]**: this field represents the number of lines in the encoded/decoded image. In decoding this register is filled automatically by the hardware JPEG codec from the JPEG file header.
- Bit 8 **HDR:** header processing: this is an optional field used for decoding only. The user can set this bit to zero to disable the JPEG header processing. In this case, other configuration registers and quantization/Huffman tables must be programmed by the user.

These registers and tables can also be programmed by hardware in case of a previous decode with the header parsing enabled at the condition that the next images (to be decoded with the header parsing disabled) have the same quantization and Huffman tables and the same dimensions, color space and chrominance sub-sampling.

- Bits 7:6 **NS[1:0]**: this field represents the number of components minus 1 in the header marker segment. Hence for the YCbCr color space it is set to “2”. This field is filled by the hardware when decoding with the header parsing enabled.
- Bits 5:4 **COLORSPACE[1:0]**: this field represents the number of quantization tables minus 1. Hence for the YCbCr color space it is set to 1 as two quantization tables are required for YCbCr (one for the Y luminance and one for both the Cb and Cr chrominance). This field is filled by the hardware when decoding with the header parsing enabled.

Note: If the user chooses to customize the quantization tables (giving an individual table per component) this field is set to 3 – 1 = 2.

- Bit 3 **DE**: to be set to 1 for decoding and 0 for encoding. This field must be set by the user to select between encoding or decoding.
- Bits 1:0 **NF[1:0]**: this field represents the number of colors components minus 1. Hence for the YCbCr color space it is set to 2. This field is filled by the hardware when decoding with the header parsing enabled.

2.1.5 CONFR2 register settings

In the CONFR2 register only the bits 25:0 are useful, the **NMCU[25:0]** field. It represents the number of MCUs minus 1 in the JPEG image.

For encoding and for decoding with the header parsing disabled, this field must be set as follows for YCbCr color space:

NMCU = (hMCU * vMCU) - 1

Where hMCU and vMCU are respectively the number of MCUs per horizontal lines and vertical columns.

- 4:4:4 chroma sampling:

  hMCU = scaled_Image_width / 8

  vMCU = scaled_Image_ Height / 8

Where: scaled_Image_width represents the image width round to the next multiple of 8 and scaled_Image_Height represents the round image height to the next multiple of 8.

- 4:2:2 sampling:

  hMCU = scaled_Image_width / 16

  vMCU = scaled_Image_ Height / 8

Where: scaled_Image_width represents the image width round to the next multiple of 16 and scaled_Image_Height represents the round image height to the next multiple of 8.

- 4:2:0 sampling:

  hMCU = scaled_Image_width / 16

  vMCU = scaled_Image_ Height / 16

Where: scaled_Image_width represents the image width round to the next multiple of 16 and scaled_Image_Height represents the round image height to the next multiple of 16.

When decoding with the header parsing enabled, this field is filled by the hardware. However it gives the number of complete MCUs, it does not take into account the incomplete MCUs at the end of lines and columns. The above formula must be used to get the exact number of MCUs.

2.1.6 CONFR3 register settings

In this register only bits 15:0 are useful, the **XSIZE [15:0]** field. It represents the number of pixels per line.

2.1.7 CONFR4-7 registers settings

In the YCbCr color space, three color components are used: Y for luminance, Cb for blue chrominance and Cr for red chrominance. As consequence only three registers CONFR4 to CONFR6 are used.

The CONFR4 register is used for luminance (Y) component. the CONFR5 and CONFR6 registers are respectively used for Cb and Cr chrominance components.

All the fields of these registers are either set by the user for encoding or for decoding with the header parsing disabled, or set by the hardware when decoding with the header parsing enabled.

- Bits 15:12 **HSF[3:0]**: this field represents the horizontal sampling factor for each component. It is the number of horizontal blocks of 8x8. It is set as follows:
  - For both CONFR5 and CONFR6 the HSF[3:0] field is always set to 1 as Cb and Cr components are always subdivided to blocks of 8x8: 1 per MCU.
  - The CONFR4 HSF[3:0] field is set depending on the chroma sampling as follows:

    4:4:4: set to 1 as each MCU has one block (Y) of 8x8.

    4:2:2: set to 2 as each MCU has 2 horizontally adjacent 8x8 (Y) blocks.

    4:2:0: set to 2 as each MCU has 2 horizontally adjacent 8x8 (Y) blocks in this case (and 2 vertically adjacent blocks so the VSF field is set to 2 also).
- Bits 11:8 **VSF[3:0]**: this field represents the vertical sampling factor for each component. It is the number of vertical blocks of 8x8. It is set as follows:
  - For both CONFR5 and CONFR6 the VSF[3:0] field is always set to 1 as Cb and Cr components are always subdivided to blocks of 8x8: 1 per MCU.
  - The CONFR4 VSF[3:0] field is set depending on the chroma sampling as follows:

    4:4:4: set to 1 as each MCU has one (Y) block of 8x8.

    4:2:2: set to 1 as each MCU has only 1 vertically adjacent 8x8 (Y) block in this case.

    4:2:0: set to 2 as each MCU has 4 (Y) blocks of 8x8 (2 horizontally adjacent and 2 vertically adjacent).
- Bits 7:4 **NB[3:0]**: this field represents the number of 8x8 blocks for each component minus 1. Hence in the YCbCr color space and for both CONFR5 and CONFR6 registers, it is set to 1 as both Cb and Cr components are always subdivided to blocks of 8x8: 1 per MCU.
  - CONFR4 **NB[3:0]** field is set depending on the chroma sampling as follows:

    4:4:4: set to 0 as each MCU has one (Y) block of 8x8.

    4:2:2: set to 1 as each MCU has 2 (Y) blocks of 8x8.

    4:2:0: set to 3 as each MCU has 4 (Y) blocks of 8x8.

- Bits 3:2 **QT[1:0]**: this field represents the quantization table associated with the given component.
  - For CONFR4 register, it is set to 0 as (Y) component uses QMEM0. It is set to 1 for both CONFR5 and CONFR6 registers as both Cb and Cr components use the same QMEM1 table.
  - Note that when the user chooses to customize the quantization tables (giving one table per component) the QT[1:0] field of the CONFR6 register is set to “2” so the Cr component uses the QMEM2 quantization table.
- Bit 1 **HA[1]** and Bit[2] **HD[1]** are both set to:
  - 0 for the CONFR4 register as (Y) component uses the AC Huffman table zero and the DC Huffman table zero.
  - 1 for CONFR5 and CONFR6 as both Cb and Cr components use the AC Huffman table one and the DC Huffman table one.

2.2 Grayscale color space

The grayscale color space uses only one color component which represents the luminance (Y). Hence the chrominance sub-sampling is not applicable in this case and a MCU is always composed of a single 8x8 (Y) block.

2.2.1 RGB to grayscale conversion

The conversion from RGB color space to grayscale can be obtained by using the following formula:

$$Y = 0.299 \times R + 0.587 \times G + 0.114 \times B$$

Note: This formula is a subset from Figure 1 for (Y) component.

2.2.2 Grayscale quantization table

As the grayscale color space is represented by a single component which is the (Y) luminance, only one quantization table is required. The table given in *Figure 3* is used with the same techniques of quality factor and zig-zag scanning. The obtained scaled quantization table (according to the quality factor) is then used to fill the hardware JPEG codec QMEM0 table.

The following are the JPEG codec registers settings in grayscale color space.

2.2.3 CONFR1 register settings

- Bits 7:6 **NS[1:0]**: this field represents the number of components for scan minus 1 in the JPEG header. Hence for the grayscale color space it is set to “0”. This field is filled by the hardware when decoding with the header parsing enabled.
- Bits 5:4 **COLORSPACE[1:0]**: this field represents the number of quantization tables minus 1. Hence for the grayscale color space it is set to “0” (1 quantization table required)
- Bits 1:0 **NF[1:0]**: this field represents the number of colors components minus 1. Hence for the grayscale color space it is set to “0”. This field is filled by the hardware when decoding with the header parsing enabled.

For all other fields of this register, same rules described in YCbCr section are applicable (YSIZE, HDR and DE fields).

2.2.4 CONFR2 register settings

As in the grayscale MCUs are always composed of 8x8 (Y) blocks the NMCU field is set as follows:

NMCU = (hMCU * vMCU) - 1

Where hMCU and vMCU are respectively the number of MCUs per horizontal lines and vertical columns.

hMCU = scaled_Image_width / 8

vMCU = scaled_Image_ Height / 8

Where scaled_Image_width represents the image width rounded to the next multiple of 8 and scaled_Image_Height represents the image height rounded to the next multiple of 8.

2.2.5 CONFR3 register settings

In this register only bits 15:0 are useful: the XSIZE [15:0] field. It represents the number of pixels per line.

2.2.6 CONFR4-7 registers settings

In grayscale only the CONFR4 register is relevant, that represents the settings for the unique (Y) component. the settings of this register in grayscale are similar to the case YCbCr 4:4:4. that are:

- HSF [3:0] and VSF [3:0]: both set to 1.
- NB [3:0]: set to 0.
- QT[1:0]: set to 0.
- HA[1] and HD[1] both set to 0.

2.3 CMYK color space

The CMYK is a color space intended for printing application. A CMYK image is represented by 3 colors cyan, magenta, yellow and a key color that is the amount of black ink. Given so, for JPEG encoding/decoding the CMYK corresponds to 4 color components. No component sub-sampling is required as per YCbCr. The same quantization table can be used for all 4 components.

For JPEG encoding/decoding, a CMYK MCU is composed of 4 blocks of 8x8 in the following order: one 8x8 cyan block followed by one 8x8 magenta block followed by one 8x8 yellow block and finally one 8x8 key block. The MCU total size is then 8x8 x 4 = 256 bytes.

2.3.1 CMYK quantization table

The same quantization table can be used for all 4 components. Nevertheless the hardware JPEG codec offers the possibility to define an individual table per each component. The STM32CubeF7/H7 JPEG HAL driver uses by default the table given in *Figure 3* for all CMYK components. The JPEG HAL driver offers also the possibility for the user to define a quantization table per color components (4 quantization tables in this case).

If needed to customize quantization tables, the user must provide 4 quantization tables (one per component). These tables are used (after scaling with the quality factor) to program respectively QMEM0 to QMEM3 RAM tables of the hardware JPEG codec (where QMEM2 table is located at the offset 0x00D0 and QMEM3 table located at the offset 0x0110).

The HAL function “HAL_JPEG_SetUserQuantTables” is the API used to customize the user quantization tables.

Same techniques of quality factor and zig-zag scanning are applicable. The obtained scaled quantization tables (according to the quality factor) are then used to fill the hardware JPEG codec QMEM0 table (or QMEM0 to QMEM3).

The following are the JPEG codec registers settings in CMYK color space.

2.3.2 CONFR1 register settings

- Bits 7:6 **NS[1:0]**: this field represents the number of components for scan minus 1 in the JPEG file header. Hence for the CMYK color space it is set to “3”. This field is filled by the hardware when decoding with the header parsing enabled.
- Bits 5:4 **COLORSPACE[1:0]**: this field represents the number of quantization tables minus 1. Hence for the CMYK color space it is set by default to “0” (1 quantization table used for all 4 components).

When the user chooses to customize the quantization tables (giving one table per component) this field is set to 3 (4 quantization tables are used).

- Bits 1:0 **NF[1:0]**: this field represents the number of colors components minus 1. For the CMYK color space it is set to “3”. This field is filled by the hardware when decoding with the header parsing enabled.

For all other fields of this register, same rules described in YCbCr section are applicable (YSIZE, HDR and DE fields).

2.3.3 CONFR2 register settings

In CMYK the NMCU field is set as follows:

NMCU = (hMCU * vMCU) - 1

Where hMCU and vMCU are respectively the number of MCUs per horizontal lines and vertical columns

hMCU = scaled_Image_width / 8

vMCU = scaled_Image_Height / 8

Where scaled_Image_width represents the image width rounded to the next multiple of 8 and scaled_Image_Height represents the image height rounded to the next multiple of 8.

2.3.4 CONFR3 register settings

In this register only bits 15:0 are useful: the XSIZE [15:0] field. It represents the number of pixels per line.

2.3.5 CONFR4-7 registers settings

In CMYK color space, 4 color components are used. Hence the registers CONFR4 to CONFR7 are set (one register respectively for each component).

All the fields of these registers are either set by the user for encoding or for decoding with the header parsing disabled. Either set by the hardware when decoding with the header parsing enabled.

Each CMYK MCU is composed by 4 blocks of 8x8: 1 block per component:

- HSF [3:0] and VSF [3:0]: both set to 1 in CMYK.
- NB [3:0]: set to 0.

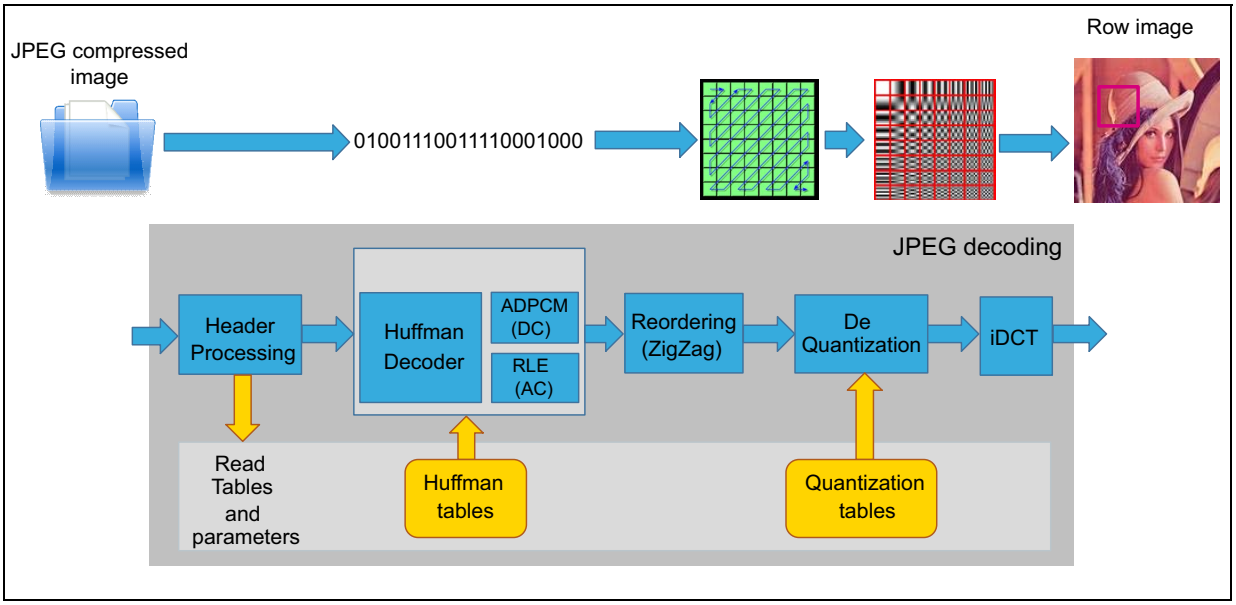
The QT[1:0] field is set to 0 by default so all 4 components use the same quantization QMEM0 table. When the user chooses to select one table per component, this field is set to 0, 1, 2 and 3 respectively for the registers CONFR4 to CONFR7. (So each component uses respectively QMEM0 to QMEM3 quantization tables).

HA[1] and HD[1] both set to 0: all components use the same Huffman AC and DC tables.

3 JPEG decoding

The hardware JPEG codec allows the decoding of a JPEG compressed image as defined in the JPEG standard (in the *ISO/IEC 10918-1)*. It can parse the JPEG header and update the codec registers (CONFR1 to CONFR7 registers), the quantization table (QMEM) and the Huffman tables.

Figure 9. JPEG decoding flow



In decoding, the JPEG codec output data are organized in MCU blocks. A MCU is composed of a number of 8x8 blocks (of the image) depending on the color space and the chroma sampling as detailed in the previous sections.

The application must then reorganize these blocks, remove the chroma sampling and convert the colors to RGB in order to display the decoded image.

To summarize, the MCUs must be organized as follows:

Table 2. JPEG MCU organization

Color space	Chroma sampling	MCU organization	MCU size in bytes
YCbCr	4:4:4	One Y 8x8 block + one Cb 8x8 block + one Cr 8x8 block	192
	4:2:2	Two Y 8x8 blocks + one Cb 8x8 block + one Cr 8x8 block	256
	4:2:0	Four Y 8x8 blocks + one Cb 8x8 block + one Cr 8x8 block	384
Grayscale	N.A	One Y 8x8 block	64
CMYK	N.A	One Cyan 8x8 block + one magenta 8x8 block + one Yellow 8x8 block + one Key 8x8 block	256

The application can wait for the hardware JPEG codec to end the decoding operation and output all the MCUs then transform these blocks to RGB pixels. Or it can start the MCUs to RGB conversion as soon as some MCUs are available.

In the STM32CubeF7/H7, the JPEG HAL driver allows to get the output data from the hardware JPEG codec by chunks with size defined by the user. If the application needs to convert the output MCUs as soon as they are available, and if the application has to deal with different color spaces and chroma sampling, it is recommended to set the output chunk size to a multiple of 768 bytes. Getting data from the JPEG codec output codec by chunks multiple of 768 bytes allows to have complete MCUs in a chunk:

Depending on the color space and the chroma sampling, 768 bytes correspond to:

- YCbCr 4:4:4 → 4 MCUs
- YCbCr 4:2:2 → MCUs
- YCbCr 4:2:0 → 2 MCUs
- Grayscale → 12 MCUs
- CMYK: → 3 MCUs

Note: The hardware JPEG codec always outputs complete MCUs. If the original image width and/or height is not multiple of 8 or 16 (depending on the color space and the chroma sampling) then the MCUs at the end of lines and/or columns are completed by dummy data (generally a duplication of previous pixels data). When converting the output MCUs to RGB pixels these extra data must be removed.

3.1 MCUs reordering and conversion

3.1.1 On the STM32H743/53/45/55/47/57/50xx devices

The Chrom-Art Accelerator peripheral also called DMA2D, implemented on the STM32H743/53/45/55/47/57/50xx devices, offers a new feature allowing to convert and reorder YCbCr MCUs (as output from the hardware JPEG codec) to RGB pixels, all with chrominance up-sampling. The DMA2D supported chrominance sampling factors are: 4:4:4, 4:2:2 and 4:2:0.

The DMA2D can handle up to 2 graphical layer foreground and background. On the STM32H743/53/45/55/47/57/50xx devices only, the foreground layer provides the capability to convert YCbCr MCU blocks to RGB pixels.

To configure the DMA2D for YCbCr MCUs to RGB pixels the following register settings are required:

FGPFCCR register:

- Bits 19:18 **CSS[1:0]**: chroma sub-sampling selection
  - 00: 4:4:4 (no chroma sub-sampling)
  - 01: 4:2:2
  - 10: 4:2:0
- Bits 3:0 **CM[3:0]**: input color mode selection
  1011: YCbCr

FGOR register:

This register allows to select the DMA2D foreground input line offset. It must be programmed as follows:

- Chroma sampling 4:4:4
  - If image width multiple of 8 pixels FGOR is set to 0
  - else FGOR = scaled_Image_width - Image_width

    With scaled_Image_width is the image width (in pixels) rounded to the next multiple of 8.
- Chroma sampling 4:2:2 or 4:2:0
  - If image width multiple of 16 pixels FGOR is set to 0

    else FGOR = scaled_Image_width - Image_width

    With scaled_Image_width is the image width (in pixels) rounded to the next multiple of 16.

    The setting of FGOR register allows to remove extra data in the MCUs covering the end of lines regions when the image dimensions are not multiple of 8 or 16.

Others DMA2D register configurations must be done as usual for pixel format conversion.

Others color spaces (grayscale and CMYK) must be handled by software. The next paragraph describes how to use the JPEG utility provided in the STM32CubeF7/H7 to perform the MCUs to RGB conversion. It is applicable for the STM32F76/77xxx devices (all color spaces) and the STM32H743/53/45/55/47/57/50xx devices (grayscale and CMYK).

Several JPEG decoding examples are available in the STM32CubeH7 showing how to use the hardware JPEG codec peripheral to:

- Decode and display JPEG compressed files using the hardware JPEG codec peripheral. The Chrom-Art Accelerator (DMA2D) is used for the YCbCr to RGB conversion.
- Decode and display MJPEG video files: using the hardware JPEG codec peripheral and the Chrom-Art Accelerator (DMA2D) for the YCbCr to RGB conversion.

Table 3 summarizes the JPEG decoding examples available in the STM32CubeH7 MCU Package:

Table 3. List of JPEG decoding examples in the STM32CubeH7 MCU Package

Example	Description
JPEG_DecodingFromFLASH_DMA	To decode and display a compressed JPEG image stored on the internal Flash memory using the hardware JPEG decoder in DMA model and the DMA2D for the YCbCr to RGB conversion.
JPEG_DecodingUsingFs_DMA	To decode and display a compressed JPEG image stored on the SD card memory using the hardware JPEG decoder in DMA model and the DMA2D for the YCbCr to RGB conversion.
JPEG_DecodingUsingFs_Interrupt	To decode and display a compressed JPEG image stored on the SD Card memory using the hardware JPEG decoder in Interrupt model and the DMA2D for the YCbCr to RGB conversion.
JPEG_DecodingUsingFs_Polling	To decode and display a compressed JPEG image stored on the SD Card memory using the hardware JPEG decoder in polling model and the DMA2D for the YCbCr to RGB conversion.

Table 3. List of JPEG decoding examples in the STM32CubeH7 MCU Package (continued)

Example	Description
JPEG_MJPEG_VideoDecoding	To decode and display an MJPEG video file stored on the SD Card memory using the hardware JPEG decoder. The YCbCr to RGB conversions are performed using the DMA2D.
JPEG_MJPEG_VideoDecodingFromQSPI	To decode and display an MJPEG video file stored on the external Quad-SPI Flash memory using the hardware JPEG decoder. The YCbCr to RGB conversions are performed using the DMA2D.

3.1.2 On the STM32F76/77xxx devices

A dedicated software layer is used to convert the MCU blocks to RGB pixels that can be presented for display. The MCUs to RGB pixel conversion includes the chrominance up-sampling and the YCbCr to RGB color conversion. This software layer is provided within the STM32CubeF7/H7 under \Utilities\JPEG.

Several JPEG decoding examples are available in the STM32CubeF7 showing how to use the hardware JPEG peripheral to:

- Decode and display JPEG compressed files using the hardware JPEG codec peripheral. The conversion from YCbCr blocks to RGB pixels is performed by the JPEG utility software.
- Decode and display MJPEG video files: using the JPEG decoder peripheral. The conversion from YCbCr blocks to RGB pixels is performed by the JPEG utility software.

Table 4 summarizes the JPEG decoding examples available in the STM32CubeF7 MCU Package:

Table 4. List of JPEG decoding examples in the STM32CubeF7 MCU Package

Example	Description
JPEG_DecodingFromFLASH_DMA	To decode and display a compressed JPEG image stored on the internal Flash memory using the hardware JPEG decoder in DMA model and the JPEG utility software for the YCbCr to RGB conversion
JPEG_DecodingUsingFs_DMA	To decode and display a compressed JPEG image stored on the SD Card memory using the hardware JPEG decoder in DMA model and the JPEG utility software for the YCbCr to RGB conversion.
JPEG_DecodingUsingFs_Interrupt	To decode and display a compressed JPEG image stored on the SD Card memory using the hardware JPEG decoder in Interrupt model and the JPEG utility software for the YCbCr to RGB conversion.
JPEG_DecodingUsingFs_Polling	To decode and display a compressed JPEG image stored on the SD Card memory using the hardware JPEG decoder in polling model and the JPEG utility software for the YCbCr to RGB conversion.
JPEG_MJPEG_VideoDecoding	To decode and display an MJPEG video file stored on the SD Card memory using the hardware JPEG decoder. The YCbCr to RGB conversion are performed by the JPEG utility software.

The following steps are required to use this utility for decoding.

- Copy the jpeg_utils_conf_template.h file under the user application folder and modify it as follows:
  - Rename it to 'jpeg_utils_conf.h'.
  - Uncomment include lines (#include "stm32fXxx_hal.h" and #include "stm32fXxx_hal_jpeg.h) and modify it respectively to (#include "stm32f7xx_hal.h" and #include "stm32f7xx_hal_jpeg.h).
  - Select the output RGB format between ARGB8888, RGB888 or RBG565 using the #define JPEG_RGB_FORMAT.
  - Optionally the red and blue swap can be selected using the #define JPEG_SWAP_RB (set to 1 to swap red and blue order in pixels)
- In the application call function JPEG_InitColorTables to initialize the red, green and blue color lookup table. This function allows to initialize 4 lookup tables (CR_RED_LUT, CB_BLUE_LUT, CR_GREEN_LUT and CB_GREEN_LUT) used to avoid multiplications and a floating point calculation during the YCbCr to RGB color conversions (according to formula given in *Figure 1*). This step must be done only one time in the application even if multiple YCbCr to RGB conversions must be done and/or multiple JPEG images must be converted.
- Next step is to select the YCbCr conversion function according to the color space and the chroma sampling by calling function JPEG_GetDecodeColorConvertFunc. This function initializes also necessary internal variables according to the image settings (dimensions color space and chroma sampling). The parameters of this function are as follows:
  - JPEG_ConfTypeDef *pJpegInfo: pointer to a JPEG_ConfTypeDef structure that contains the JPEG image information (color space, chroma sub-sampling, image height and width). These info are available once the JPEG header parsing is done by the hardware JPEG codec, that is, under the HAL driver callback HAL_JPEG_InfoReadyCallback. These info can also be retrieved (after the header parsing or at the end of the JPEG decode operation) using function HAL_JPEG_GetInfo.
  - JPEG_YCbCrToRGB_Convert_Function *pFunction: this parameter returns the pointer to the function that is used to convert JPEG codec output MCUs to RGB pixels in the destination image frame buffer.
  - uint32_t *ImageNbMCUs: this parameter is used to return to the user the total number of MCUs according to image dimensions, color space and chroma sampling.
- The conversion function can then be called to convert YCbCr MCUs to RGB pixel into the destination RGB frame buffer. The conversion function parameters are as follows:
  - uint8_t *pInBuffer: a buffer containing a number of complete MCUs (output of the hardware JPEG codec)
  - uint8_t *pOutBuffer: the RGB destination buffer where the RGB image is stored.
  - uint32_t BlockIndex: the index of the first MCU in the current input buffer (pInBuffer) versus the total number of MCUs.
  - uint32_t DataCount: the input buffer (pInBuffer) size in bytes.

– uint32_t *ConvertedDataCount: reserved for future use (to be used to return the number of converted bytes from input buffer).

  The conversion function returns the number of converted MCUs from the input buffer to the output RGB buffer so it can be used for the parameter BlockIndex in the next call of this function if conversion is done by chunks (not in one shot).

For information, *Table 5* provides the conversion function for each color space. These functions are implemented as static in the “jpeg_utils.c” source file. The application does not need to directly call these functions, instead need to call “JPEG_GetDecodeColorConvert Func()” to retrieve a pointer to the function that corresponds to the given image color space and chroma sampling.

Table 5. List of MCU to RGB internal conversion functions

Color space	Chroma sampling	MCU to RGB conversion function
YCbCr	4:4:4	JPEG_MCU_YCbCr444_ARGB_ConvertBlocks()
	4:2:2	JPEG_MCU_YCbCr422_ARGB_ConvertBlocks()
	4:2:0	JPEG_MCU_YCbCr420_ARGB_ConvertBlocks()
Grayscale	N.A	JPEG_MCU_Gray_ARGB_ConvertBlocks()
CMYK	N.A	JPEG_MCU_YCCK_ARGB_ConvertBlocks()

The MCU blocks to RGB conversion functions work on complete MCUs and suppose that the image width and height are multiple of 8 or 16 (depending on the color space and the chroma sampling). At the same time the hardware JPEG codec always outputs complete MCUs and when converted to RGB pixels, gives an image with dimensions (height and width) multiple of 8 or 16.

In order to use the JPEG utility layer when decoding images with dimensions (width and height) not multiple of 8 or 16 the following technique can be used:

- Before calling the “JPEG_GetDecodeColorConvertFunc()” update the ImageWidth and ImageHeight of the structure pJpegInfo depending on the color space and chroma sampling as follows:
  - YCbCr 4:4:4, grayscale of CMYK: rounds both ImageWidth and ImageHeight to the next multiple of 8.
  - YCbCr 4:2:2: rounds both ImageWidth to the next multiple of 16 and ImageHeight to the next multiple of 8.
  - YCbCr 4:2:0, grayscale of CMYK: rounds both ImageWidth and ImageHeight to the next multiple of 16.
- Precede with the MCUs conversion. The output RGB image has height and width extended to the next multiple of 8 or 16 as above.

- Use the DMA2D to crop the obtained image to the original dimensions: by programming the DMA2D input line offset (FGOR register) as per the STM32H743xx conversion case. That is:

  FGOR register: allows to select the DMA2D foreground input line offset. It must be programmed as follows:

  – YCbCr 4:4:4, grayscale of CMYK:

    - FGOR = scaled_Image_width - Image_width

    With scaled_Image_width is the image width (in pixels) round to the next multiple of 8.

  – Chroma sampling 4:2:2 or 4:2:0:

    - FGOR = scaled_Image_width - Image_width

    With scaled_Image_width is the image width (in pixels) round to the next multiple of 16.

The setting of the DMA2D FGOR register allows to remove extra pixels due to the dimension (height and width) rounding. The DMA2D can be configured in memory to memory or pixel format conversion (to change the output image color format).

3.2 JPEG decoding performances

The following tables provide the decoding performances for:

- STM32H743/53/45/55/47/57/50xx: using the hardware JPEG peripheral and the DMA2D peripheral for YCbCr to RGB conversion.
- STM32F76/77xxx: using the hardware JPEG peripheral and the software utility for YCbCr to RGB conversion.

Note: These performance measurements are given with the JPEG buffers (RGB and YCbCr) located on the external SDRAM.

Table 6. STM32H743/53/45/55/47/57/50xx JPEG decoding performances

Product	Image resolution	Decoding (ms)		
		Hardware decoding	DMA2D YCbCr to RGB conversion	Total time
STM32H743I	VGA: 640 x 480	4	6	**10** (100 fps)
	QVGA: 320 x 240	1	1.5	**2.5** (400 fps)

Table 7. STM32F76/77xxx JPEG decoding performances

Product	Image resolution	Decoding (ms)		
		Hardware decoding	Software YCbCr to RGB conversion	Total time
STM32F769I	VGA: 640 x 480	4	22	**26** (38 fps)
	QVGA: 320 x 240	1	5	**6** (166 fps)

The above measurement has been performed with the conditions given in *Table 8*.

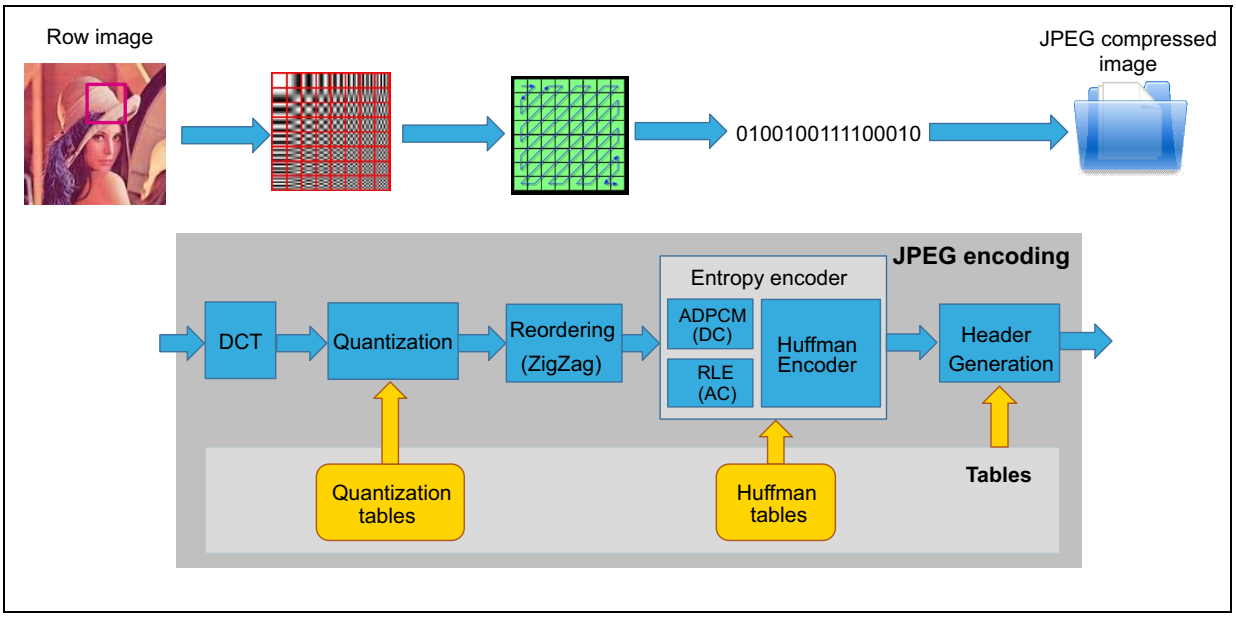
Table 8. JPEG decoding performance measurement conditions

Product	STM32F76/77xxx	STM32H743/53/45/55/47/57/50xx
Board	STM32F769I-EVAL rev.B	STM32H743I-EVAL rev.B
CPU frequency	200 MHz	400 MHz
Hardware JPEG codec frequency	200 MHz	200 MHz
IDE/compiler	IAR embedded workbench for Arm version 7.80	IAR embedded workbench for Arm version 7.80
Compiler optimization	High speed	High speed
SDRAM external memory	Ref: IS42S32800G-6BLI Clock frequency: 100 MHz Access: ROW access (not file system access)	Ref: IS42S32800G-6BLI Clock frequency: 200 MHz Access: ROW access (not file system access)
JPEG image	Resolution: 640 x 480 Color format: YCbCr Chroma sampling: 4:2:0	Resolution: 640 x 480 Color format: YCbCr Chroma sampling: 4:2:0

4 JPEG encoding

The hardware JPEG codec allows to compress images to jpeg files compliant with the JPEG file interchange format (JFIF) including necessary headers and segments.

Figure 10. JPEG encoding flow



The JPEG HAL driver available in the STM32CubeF7/H7 provides necessary functions to perform encoding operations including the initialization of the codec with default Huffman tables.

In encoding mode, the JPEG codec input data are expected to be organized in MCU blocks depending on the color space and the chroma sampling as explained in *Table 2*.

The application must reorganize and convert the input RGB pixels to MCU blocks. The chroma sub-sampling must also be applied in case of the YCbCr color space. The hardware JPEG codec expects complete MCUs. If the RGB image dimensions (height and width) are not multiple of 8 or 16 then extra pixels must be added at the end of lines and columns in order to generate complete MCUs with blocks of 8x8. Nevertheless in the hardware JPEG codec registers CONFR1 and CONFR3, the original images dimensions must be set (in the YSIZE and XSIZE fields).

The software utility provided with the STM32CubeF7/H7 can be used to perform the necessary conversion from input RGB pixels to MCU blocks that can be used to feed the hardware JPEG codec. The STM32CubeF7/H7 provides examples showing how to encode RGB images into JPEG compressed files (using this software utility for MCUs generation).

The examples are available under:

- STM32CubeF7: \Firmware\Projects\STM32F769I_EVAL\Examples\JPEG
- STM32CubeH7: \Firmware\Projects\STM32H743I_EVAL\Examples\JPEG

Table 9 summarizes the available encoding examples:

Table 9. List of JPEG encoding examples in the STM32CubeF7/H7 MCU Packages

Example	Description
JPEG_EncodingFromFLASH_DMA	To encode an RGB image stored on the internal Flash memory using the hardware JPEG codec in DMA model and store the result compressed jpeg file to the SD Card memory. The RGB to YCbCr conversion (required prior to the encoding) is performed by the JPEG utility software.
JPEG_EncodingUsingFs_DMA	To encode a bmp image stored in SD Card memory using the hardware JPEG codec in DMA model and store the result compressed jpeg file to the SD Card. The RGB to YCbCr conversion (required prior to the encoding) is performed by the JPEG utility software.

The following steps are required to use the JPEG utility for encoding.

- Copy the jpeg_utils_conf_template.h file under the user application folder and modify it as follows:
  - Rename it to 'jpeg_utils_conf.h'.
  - Uncomment include lines: #include "stm32fXxx_hal.h" and #include "stm32fXxx_hal_jpeg.h and modify them respectively to:
    - Using the STM32CubeF7: #include "stm32f7xx_hal.h" and #include "stm32f7xx_hal_jpeg.h.
    - Using the STM32CubeH7: #include "stm32h7xx_hal.h" and #include "stm32h7xx_hal_jpeg.h.
  - Select the output RGB format between ARGB8888, RGB888 or RBG565 using the #define JPEG_RGB_FORMAT.
  - Optionally red and blue swap can be select using the #define JPEG_SWAP_RB (set to 1 to invert red and blue order in pixels)
- In the user application call function JPEG_InitColorTables to initialize the red, green and blue colors lookup table. This function allows to initialize different lookup tables used to avoid multiplications and floating point calculation during the colors conversions (according to formula given in *Figure 1*). This step must be done only one time in the application even if multiple images must be encoded.
- The next step is to select the RGB to YCbCr conversion function according to the color space and chroma sampling. This is done by calling the function JPEG_GetEncodeColorConvertFunc. This function also initializes necessary internals variable for the RGB to YCbCr MCU conversion according to the image settings dimensions color space and chroma sampling). The parameters of this function ares as follows:
  - JPEG_ConfTypeDef *pJpegInfo: pointer to a JPEG_ConfTypeDef structure that contains the image information (color space, chroma sub-sampling, image height and width). These info must be filled by the user for encoding.
  - PEG_RGBToYCbCr_Convert_Function *pFunction: this parameter returns the pointer to the function that is used to convert the RGB pixels to MCUs.
  - uint32_t *ImageNbMCUs: this parameter is used to return to the user the total number of MCUs according to image dimensions, color space and chroma sampling.

- The conversion function can then be called to convert input image RGB pixel to YCbCr MCUs. The conversion function parameters are as follows:
  - uint8_t *pInBuffer: a buffer containing RGB pixels to be converted to MCUs. Due to the fact that MCUs correspond to 8x8 blocks of the original images, the input buffer must correspond to a multiple of:

    - 8 lines of the input RGB image in case of YCBCR 4:4:4, YCbCr 4:2:2, grayscale or CMYK.

    - 16 lines of the input RGB image in case of YCbCr 4:2:0.
  - uint8_t *pOutBuffer: the MCUs destination buffer. This buffer can then be used to feed the hardware JPEG codec.
  - uint32_t BlockIndex: the index of the first MCU in the current input buffer (pInBuffer) versus the total number of MCUs.
  - uint32_t DataCount: the input buffer (pInBuffer) size in bytes.
  - uint32_t *ConvertedDataCount: returns the number of converted bytes from input buffer.

    The conversion function returns the number of converted MCUs from the input buffer to the output MCUs buffer so it can be used for the parameter BlockIndex in the next call of this function if the conversion is done by chunks (not in one shot).

For information, *Table 10* provides the conversion function for each color space. These functions are implemented as static in the “jpeg_utils.c” source file. The application does not need to directly call these function, instead need to call “JPEG_GetEncodeColorConvert Func ()” to retrieve a pointer to the function that corresponds to the given image color space and chroma sampling.

Table 10. List of RGB to MCU internal conversion functions

Color space	Chroma sampling	RGB to MCU conversion function
YCbCr	4:4:4	JPEG_ARGB_MCU_YCbCr444_ConvertBlocks ()
	4:2:2	JPEG_ARGB_MCU_YCbCr422_ConvertBlocks ()
	4:2:0	JPEG_ARGB_MCU_YCbCr420_ConvertBlocks ()
Grayscale	N.A	JPEG_ARGB_MCU_Gray_ConvertBlocks ()
CMYK	N.A	JPEG_ARGB_MCU_YCCK_ConvertBlocks ()

The HAL driver function “HAL_JPEG_ConfigEncoding” must be called to fill the hardware JPEG codec registers with the parameters of the image to be encoded before starting the encoding operation using one of the 3 available models:

- Pooling model: using HAL driver function HAL_JPEG_Encode
- Interrupt model: using HAL driver function HAL_JPEG_Encode_IT
- DMA model: using HAL driver function HAL_JPEG_Encode_DMA

The MCUs retrieved with the conversion utility function must then be used as input for the above HAL conversion functions.

4.1 JPEG encoding performances

The following tables provide the encoding performances for STM32H743/53/45/55/47/57/50xx and STM32F76/F77xxx: using the hardware JPEG peripheral and the software utility for the RGB to YCbCr conversion.

Note: These performance measurements are given with the JPEG buffers (RGB and YCbCr) located on the external SDRAM.

Table 11. STM32H743/53/45/55/47/57/50xx JPEG encoding performances

Product	Image resolution	Encoding (ms)		
		Software RGB to YCbCr conversion	Hardware encoding	Total time
STM32H743I	VGA: 640 x 480	58	4	**62** (16 fps)
	QVGA: 320 x 240	14	1	**15** (66 fps)

Table 12. STM32F76/77xxx JPEG encoding performances

Product	Image resolution	Encoding (ms)		
		Software RGB to YCbCr conversion	Hardware encoding	Total time
STM32F769I	VGA: 640 x 480	103	4	**107** (9 fps)
	QVGA: 320 x 240	27	1	**28** (35 fps)

The above measurement has been performed with the conditions given in *Table 13*.

Table 13. JPEG encoding performance measurement conditions

Product	STM32F76/77xxx	STM32H743/53/45/55/47/57/50xx
Board	STM32F769I-EVAL rev.B	STM32H743I-EVAL rev.B
CPU frequency	200 MHz	400 MHz
Hardware JPEG codec frequency	200 MHz	200 MHz
IDE/Compiler	IAR embedded workbench for Arm version 7.80	IAR embedded workbench for Arm version 7.80
Compiler optimization	High speed	High speed
SDRAM external memory	Ref: IS42S32800G-6BLI Clock frequency: 100 MHz Access: ROW access (not file system access)	Ref: IS42S32800G-6BLI Clock frequency: 200 MHz Access: ROW access (not file system access)
JPEG image	Resolution: 640 x 480 Color format: YCbCr Chroma sampling: 4:2:0	Resolution: 640 x 480 Color format: YCbCr Chroma sampling: 4:2:0

5 Conclusion

The STM32F7/H7 hardware JPEG codec peripheral provides a hardware acceleration for JPEG encoding/decoding operations with significant performance improvement. It allows also to reduce the firmware footprint (RAM and ROM) for a JPEG based application overcoming the use of a software JPEG encoding/decoding (example libjpeg).

The hardware JPEG codec is compliant with the JPEG standard (JPEG ISO/IEC 10918-1 ITU-T recommendation T.81). A software processing is provided with the STM32CubeF7/H7 MCU Packages to deal with the YCbCr MCU block conversion from/to RGB pixels in order to be compliant with the JPEG file interchange format (JFIF).

Using the STM32H743/53/45/55/47/57/50xx devices, and in case of decoding images in the YCbCr color space, the MCUs to RGB conversion can be accelerated using the DMA2D peripheral.

Several examples for encoding/decoding are available in the STM32CubeF7/H7 showing how to use the JPEG HAL driver with the JPEG software utility or with the DMA2D peripheral.

This application note describes the different register settings of the hardware JPEG codec depending on the image parameters (the register settings are covered by the JPEG HAL driver). It provides guides on how to use the JPEG software utility to perform the necessary conversion of RGB pixels from/to MCU blocks used by the hardware JPEG codec. This application note provides also the necessary DMA2D settings when using this peripheral to convert the hardware JPEG codec output MCUs to RGB pixels in case of decoding a YCbCr JPEG compressed image. This feature of the DMA2D is available on the STM32H743/53/45/55/47/57/50xx devices only.

6 Revision history

Table 14. Document revision history

Date	Revision	Changes
14-Nov-2017	1	Initial release.
28-May-2019	2	Updated the whole document applicable to the STM32H743/53/45/55/47/57/50xx devices. Updated *Table 1: Applicable products*.