

# JE100, JE110, JE150 and JE160

JPEG Baseline Encoder IP-Core

## Users Manual

Rev 3.0

## Table of Contents

1.	The JPEG Baseline Encoding Standard .....	5
1.1	Level Shift.....	5
1.2	Blocks.....	5
1.3	Components .....	6
1.4	DCT .....	7
1.5	Quantization .....	7
1.6	Zigzag Order.....	8
1.7	Differential DC Encoding .....	8
1.8	Run / Size Symbols Encoding .....	9
1.9	Huffman Coding.....	10
2.	The JPEG File Format .....	11
2.1	SOI Start of Image.....	11
2.2	DQT Define Quantization Table.....	11
2.3	SOF Start of Frame .....	12
2.4	DHT Define Huffman Table .....	12
2.5	SOS Start of Scan .....	13
2.6	EOI End Of Image .....	13
3.	Encoding Example.....	14
4.	The JE100, JE110, JE150 and JE160 JPEG Encoder .....	16
4.1	Technical Features .....	16
4.2	Difference between JE100, JE110, JE150 and JE160 .....	16
4.3	Needed Resource.....	17
4.4	Functional Description .....	18
4.5	Typical Application.....	19
4.6	Core Entitys .....	20
4.6.1	JE100 Core Entity .....	20
4.6.2	JE110 Core Entity .....	20
4.6.3	JE150 Core Entity .....	21
4.6.4	JE160 Core Entity .....	21
4.7	Core Interface.....	22
4.7.1	Pixel Input Interface.....	22
4.7.2	Compressed Data Output Interface.....	24
4.7.3	Tables Programming Interface .....	25
5.	Using with Xilinx ISE .....	27
5.1	NGC File.....	27
5.2	Component Declarations .....	29
5.2.1	JE100 Component Declaration.....	29
5.2.2	JE110 Component Declaration.....	30
5.2.3	JE150 Component Declaration.....	31
5.2.4	JE160 Component Declaration.....	32
5.3	Component Instantiations .....	33
5.3.1	JE100 Component Instantiation .....	33
5.3.2	JE110 Component Instantiation .....	34
5.3.3	JE150 Component Instantiation .....	35
5.3.4	JE160 Component Instantiation .....	36
6.	Quantization Tables.....	37
6.1	Default Quantization Table for Luminance .....	38
6.2	Default Quantization Table for Chrominance.....	38

6.3	Changing the Compression Rate.....	38
7.	Huffman Tables .....	40
7.1	Huffman Tables Generation.....	41
7.2	Default Huffman Tables for JE100 and JE110.....	41
7.2.1	Huffman Table for Luminance .....	41
7.2.2	Huffman Table for Chrominance .....	42
7.3	Default Huffman Tables for JE150 and JE160.....	44
7.3.1	Huffman Table for Luminance .....	44
7.3.2	Huffman Table for Chrominance .....	45
8.	Literature and Links .....	47
8.1	Documents from the Internet.....	47
8.2	Hard Book.....	47
8.3	Internet Links.....	47
9.	Revision Info.....	48

## List of Figures:

Figure 1:	JPEG Encoding Structure.....	5
Figure 2:	Dividing Image in Blocks.....	6
Figure 3:	Color Image as Blocks and Components.....	6
Figure 4:	Color Components.....	7
Figure 5:	Subsampled Color Components .....	7
Figure 6:	Zigzag Order from 8x8 Block .....	8
Figure 7:	DC Coefficient Size, VLI Symbols.....	9
Figure 8:	AC Coefficients Run/Size, VLI Symbols .....	9
Figure 9:	Block with Original Samples .....	14
Figure 10:	Block with Level Shifted Samples .....	14
Figure 11:	Coefficients After DCT .....	14
Figure 12:	Quantization Table.....	14
Figure 13:	Quantized Coefficients.....	14
Figure 14:	Values in Zigzag Order .....	14
Figure 15:	Huffman Encoding .....	15
Figure 16:	Encoder Structure.....	18
Figure 17:	Block Diagram of a Typical Application.....	19
Figure 18:	ISE Project.....	27
Figure 19:	ISE Implementation Process Properties .....	28

## List of Tables:

Table 1: Size Symbols .....	9
Table 2: AC Run / Size Symbols.....	10
Table 3: Difference between Cores .....	16
Table 4: Needed Resource for JE100.....	17
Table 5: Needed Resource for JE110.....	17
Table 6: Needed Resource for JE150.....	17
Table 7: Needed Resource for JE160.....	17
Table 8: Core Entity Signals .....	22
Table 9: Zigzag Order .....	37
Table 10: Default Quantization Table for Luminance.....	38
Table 11: Default Quantization Table for Chrominance .....	38
Table 12: DC and AC Codes Position Within the Huffman Table .....	40
Table 13: Luminance Number of DC Codes .....	41
Table 14: Luminance DC Symbol to Code Assignment .....	41
Table 15: Luminance Number of AC Codes .....	41
Table 16: Luminance AC Symbol to Code Assignment .....	42
Table 17: Default Huffman Codes for Luminance .....	42
Table 18: Chrominance Number of DC Codes .....	42
Table 19: Chrominance DC Symbol to Code Assignment .....	43
Table 20: Chrominance Number of AC Codes.....	43
Table 21: Chrominance AC Symbol to Code Assignment.....	43
Table 22: Default Huffman Codes for Chrominance .....	43
Table 23: Luminance Number of DC Codes .....	44
Table 24: Luminance DC Symbol to Code.....	44
Table 25: Luminance Number of AC Codes .....	44
Table 26: Luminance AC Symbol to Code Assignment .....	44
Table 27: Default Huffman Codes for Luminance .....	45
Table 28: Chrominance Number of DC Codes .....	45
Table 29: Chrominance DC Symbol to Code Assignment .....	45
Table 30: Chrominance Number of AC Codes.....	45
Table 31: Chrominance AC Symbol to Code Assignment.....	46
Table 32: Default Huffman Codes for Chrominance .....	46

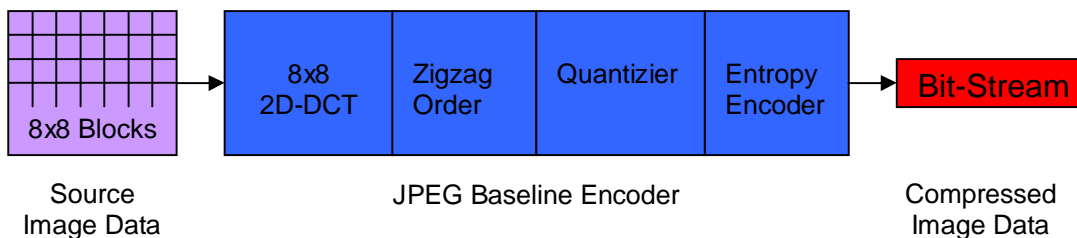
# 1. The JPEG Baseline Encoding Standard

The JPEG standard defines four modes of operation:

- Sequential DCT-based
- Progressive DCT-based
- Hierarchical
- Sequential lossless

The most used mode is the Sequential DCT-based, also known as Baseline mode. The following overview describes the Baseline encoding standard.

**Figure 1: JPEG Encoding Structure**



Typical compress rates are values from 10 to 15.

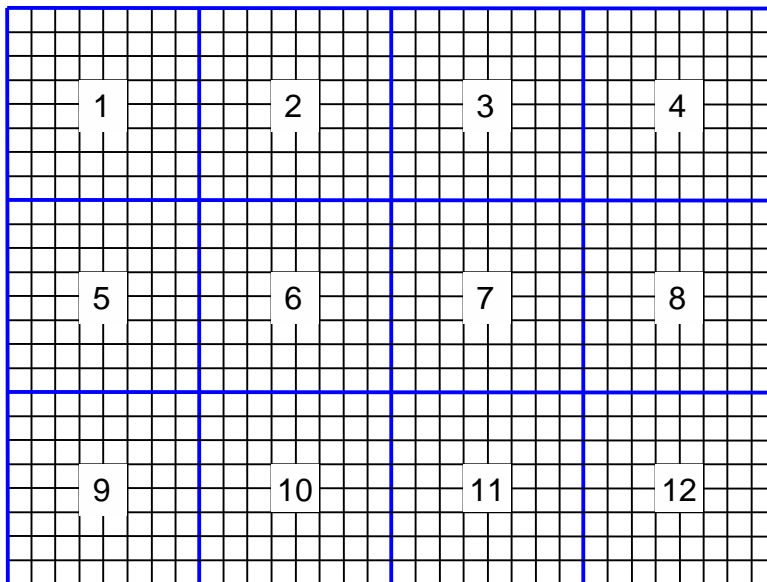
## 1.1 Level Shift

The image samples must be in the two's complement format with a size of 8 bit. If the samples are in integer format, a value of 80h (800h with 12 bit precision) must be subtracted.

## 1.2 Blocks

The complete image is divided into 8x8 blocks, starting in the top left row. The blocks are read out in the same order, beginning in the top left corner to the top right corner, and then continuing the next eight rows until all done. If the number of pixels / lines are not a multiple of 8, the last pixel / line is repeated until the block is completed. Any extra pixels / lines are discarded by the decoding process.

**Figure 2: Dividing Image in Blocks**

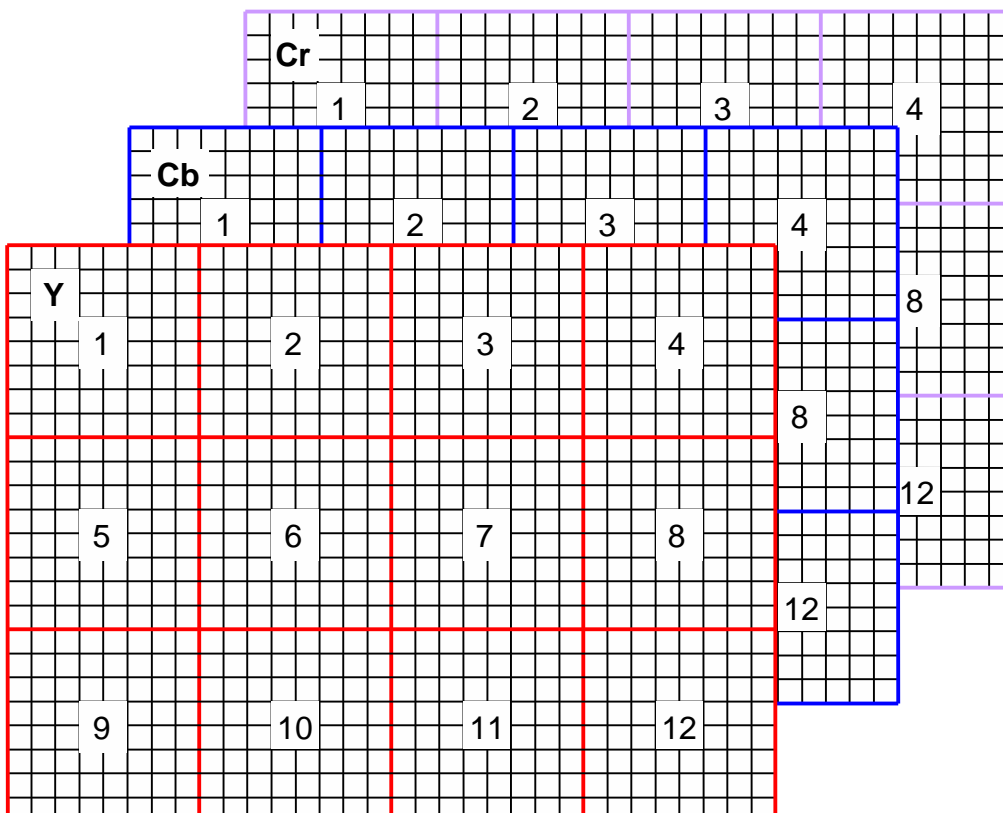


### 1.3 Components

The parts of a pixel are named components. A mono chrome image has only one and a color image three components. One ore more components may be sub-sampled but usually this is done, only with the chrominance components.

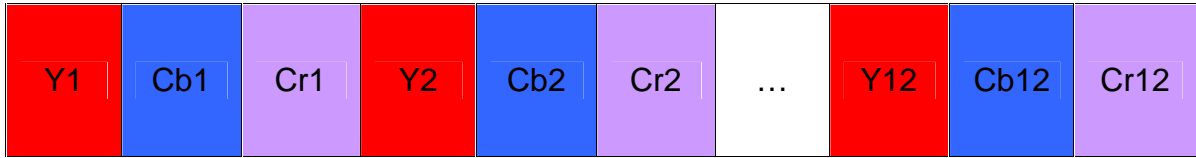
Here a color image with the components Y, Cb and Cr, no subsampling.

**Figure 3: Color Image as Blocks and Components**



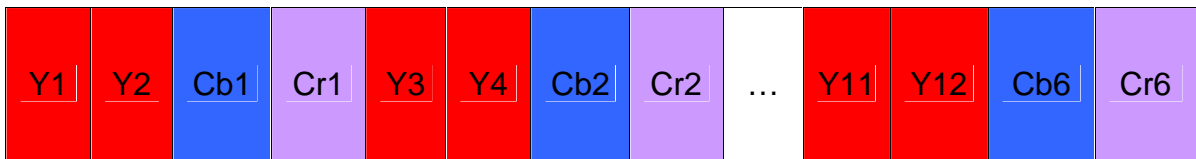
The blocks are processed in the order components and then the indexes.

**Figure 4: Color Components**



When the chrominance components (Cb and Cr) are subsampled with a value of 2 then one chrominance sample pair is used for two luminance samples.

**Figure 5: Subsampled Color Components**



## 1.4 DCT

The Discrete Cosine Transformation (DCT) transforms the 64 samples array of an 8x8 block into an 8x8 array of coefficients. Doing this by using the following equation:

$$S(v,u) = \frac{C(v)}{2} * \frac{C(u)}{2} * \sum_{y=0}^7 \sum_{x=0}^7 S(y,x) * \cos\left(\frac{(2x+1)u\pi}{16}\right) * \cos\left(\frac{(2y+1)v\pi}{16}\right)$$

$$C(u) = \frac{1}{\sqrt{2}} \text{when } u = 0 \text{ else } 1$$

$$C(v) = \frac{1}{\sqrt{2}} \text{when } v = 0 \text{ else } 1$$

The two indices x and y represent the sample placement, the indices u and v represent the coefficients frequencies. The top left element (S(0,0)) is the DC coefficient, the bottom right left element (S(7,7)) is the coefficient with the highest horizontal and vertical frequencies.

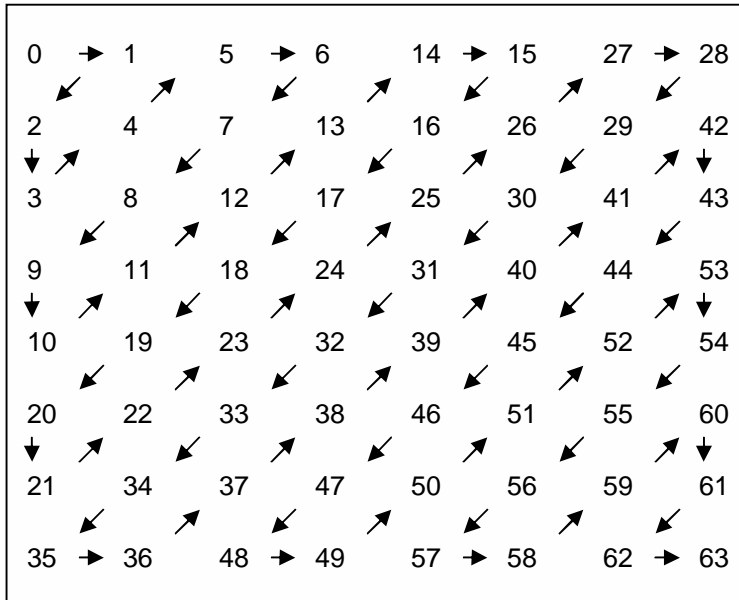
## 1.5 Quantization

The Quantization reduces the accuracy of the coefficients. This is done by dividing each coefficient by the value in the Quantization table with the same indices. When using larger values in the table consequence a higher compression rate, but also more artifacts. On the other hand, the using of lower values results in less compression and lossy. The higher frequently coefficients are lower and the table values for these coefficients are larger, so much of quantized coefficients become zero. This is important for a high compression rate. The Quantization stage is the mean reason for lossy, all other stages are lossless (the DCT is a little lossy cause the rounding errors).

## 1.6 Zigzag Order

The quantized coefficients of a block are rearranged in that way that there are sorted from DC to the highest frequencies. The goal is to have a large number of subsequent zeros for the following Run / Size encoding.

Figure 6: Zigzag Order from 8x8 Block



## 1.7 Differential DC Encoding

The DC coefficient represents the average value of all 64 samples. Because the average differs only slightly from one block to the next, the DC coefficient is differential encoded. From the DC coefficient, is subtracted the DC coefficient from the previous block of the same component. For the first block in the image, is a predicted value of zero defined. The difference is usually a short value and results so in a short integer in the following symbol encoding.

## 1.8 Run / Size Symbols Encoding

The Zigzag ordered values are transformed in a Run/Size (DC only Size) symbol and variable length integers bits (VLI). The Size symbol is taken from the given table:

**Table 1: Size Symbols**

Size	Values range
0	0
1	-1, 1
2	-3 ... -1, 2 ... 3
3	-7 ... -4, 4 ... 7
4	-15 ... -8, 8 ... 15
5	-31 ... -16, 16 ... 31
6	-63 ... -32, 32 ... 63
7	-127 ... -64, 64 ... 127
8	-255 ... -128, 128 ... 255
9	-511 ... -256, 256 ... 511
10	-1023 ... -512, 512 ... 1023
11	-2047 ... -1024, 1024 ... 2047
12*	-4095 ... -2048, 2048 ... 4095
13*	-8191 ... -4096, 4096 ... 8191
14*	-16383 ... -8192, 8192 ... 16383
15*	-32767 ... -16384, 16384 ... 32767

\*Defined only for 12 bit precision.

The Size symbol represents the number of the VLI bits. The VLI bits are generated by adding a one, if the value is negative. Then the VLI bits are truncate to n lower bits, where n is the Size value.

**Figure 7: DC Coefficient Size, VLI Symbols**

(Size), VLI

The AC values are coded in a Run/Size and VLI symbol, but only the non-zero values. If the value is zero, then only the Run part of the next non-zero value is incremented. Because the Run part is four bit, only 15 consecutive zeros can be coded in the Run/Size symbol. If 16 consecutive zeros appear, the special Zero Run Length (ZRL) symbol is inserted. When at one point all remaining values are zero, then the end of block (EOB) symbol is inserted, and the block is finished.

**Figure 8: AC Coefficients Run/Size, VLI Symbols**

(Run/Size), VLI

## 1.9 Huffman Coding

The Run/Size symbols get a code from the Huffman table. The idea behind the Huffman coding is, to use codes with variable length. Symbols with a large number of occurs get short codes and symbols with less occurs get longer codes.

**Table 2: AC Run / Size Symbols**

		Size														
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E
Run	0	EOB	0/1	0/2	0/3	0/4	0/5	0/6	0/7	0/8	0/9	0/10	0/11*	0/12*	0/13*	0/14*
	1		1/1	1/2	1/3	1/4	1/5	1/6	1/7	1/8	1/9	1/10	1/11*	1/12*	1/13*	1/14*
	2		2/1	2/2	2/3	2/4	2/5	2/6	2/7	2/8	2/9	2/10	2/11*	2/12*	2/13*	2/14*
	3		3/1	3/2	3/3	3/4	3/5	3/6	3/7	3/8	3/9	3/10	3/11*	3/12*	3/13*	3/14*
	4		4/1	4/2	4/3	4/4	4/5	4/6	4/7	4/8	4/9	4/10	4/11*	4/12*	4/13*	4/14*
	5		5/1	5/2	5/3	5/4	5/5	5/6	5/7	5/8	5/9	5/10	5/11*	5/12*	5/13*	5/14*
	6		6/1	6/2	6/3	6/4	6/5	6/6	6/7	6/8	6/9	6/10	6/11*	6/12*	6/13*	6/14*
	7		7/1	7/2	7/3	7/4	7/5	7/6	7/7	7/8	7/9	7/10	7/11*	7/12*	7/13*	7/14*
	8		8/1	8/2	8/3	8/4	8/5	8/6	8/7	8/8	8/9	8/10	8/11*	8/12*	8/13*	8/14*
	9		9/1	9/2	9/3	9/4	9/5	9/6	9/7	9/8	9/9	9/10	9/11*	9/12*	9/13*	9/14*
	10		10/1	10/2	10/3	10/4	10/5	10/6	10/7	10/8	10/9	10/10	10/11*	10/12*	10/13*	10/14*
	11		11/1	11/2	11/3	11/4	11/5	11/6	11/7	11/8	11/9	11/10	11/11*	11/12*	11/13*	11/14*
	12		12/1	12/2	12/3	12/4	12/5	12/6	12/7	12/8	12/9	12/10	12/11*	12/12*	12/13*	12/14*
	13		13/1	13/2	13/3	13/4	13/5	13/6	13/7	13/8	13/9	13/10	13/11*	13/12*	13/13*	13/14*
	14		14/1	14/2	14/3	14/4	14/5	14/6	14/7	14/8	14/9	14/10	14/11*	14/12*	14/13*	14/14*
	15	ZRL	15/1	15/2	15/3	15/4	15/5	15/6	15/7	15/8	15/9	15/10	15/11*	15/12*	15/13*	15/14*

\*Defined only for 12 bit precision.

The codes from the Huffman table and the belonging VLI bits (and all following) are threaded to a bit-stream. The bit-stream is divided into bytes with a length of 8 bit. If a byte contain only 1 (FFh), then a zero byte is inserted behind, to avoid the misinterpretation with a marker. After the very last block, the remaining bits are filled up with ones, to have a complete byte.

## 2. The JPEG File Format

The JPEG file is assembled of segments. Each segment has a Header with the marker and when further parameter, a segment length. A marker is a two byte code with the first byte always FFh and the second byte unequal zero. Integer values are high significant byte first arranged.

A typical JPEG file segment structure is SOI, DQT, SOF, DHT, SOS and EOI.

The following description shows only those segments, where needed by the mono chrome and color (4:2:2) baseline encoding. Some others modes have more segments, or the segment parameters has another meaning.

### 2.1 SOI Start of Image

The **Start Of Image** segment defines the beginning of an image and has no further parameter.

SOI marker	16 bit	xx, xx	FFh, D8h
------------	--------	--------	----------

### 2.2 DQT Define Quantization Table

The **Define Quantization Table** segment defines the Quantization tables, used by the encoder. The table elements are in Zigzag order.

DQT marker	16 bit	xx, xx	FFh, DBh
Segment length	16 bit	xx, xx	2 + num tables * 65

For each table:

Table precision	4 bit	x	0
Table identifier	4 bit	xx	0 for luminance, 1 for chrominance
Table elements	64 x 8 bit	xx, ... , xx	

## 2.3 SOF Start of Frame

The **Start Of Frame** segment defines the geometric parameters of the image

SOF marker	16 bit	xx, xx	FFh, C0h
Segment length	16 bit	xx, xx	32 + num components * 24
Sample precision	8 bit	xx	08h or 0Ch
Number of lines	16 bit	xx, xx	
Number of samples / line	16 bit	xx, xx	
Number of components	8 bit	xx	1 for mono chrome, 3 for color

For each component:

Component identifier	8 bit	xx	0 for Y, 1 for Cb, 2 for Cr
Horizontal sampling factor	4 bit	x	1 for luminance, 2 for chrominance
Vertical sampling factor	4 bit	x	1
Quantization table selector	8 bit	xx	0 for luminance, 1 for chrominance

## 2.4 DHT Define Huffman Table

The **Define Huffman Table** segment defines the Huffman tables, used by the encoder.

DHT marker	16 bit	xx, xx	FFh, C4h
Segment length	16 bit	xx	2 + Num Tables * 175

For each table:

Table class	4 bit	x	0 for DC table, 1 for AC table
Table identifier	4 bit	x	0 for luminance, 1 for chrominance
Number of codes length	16 x 8 bit	xx,..., xx	
Codes	n x 8 bit	xx,..., xx	n = 12 for DC, 162 for AC

## 2.5 SOS Start of Scan

The **Start Of Scan** segment defines the scan parameter, followed by the compressed bit-stream.

SOS marker	16 bit	xx, xx	FFh, DAh
Segment length	16 bit	xx, xx	6 + Num Components * 2
Number of component	8 bit	xx	1 for mono chrome, 3 for color

For each component:

Component selector	8 bit	xx	0 for Y, 1 for Cb, 2 for Cr
DC table selector	4 bit	x	0 for Y, 1 for Cb and Cr
AC table selector	4 bit	x	0 for Y, 1 for Cb and Cr
Start of spectral selection	8 bit	xx	00
End of spectral selection	8 bit	xx	3Fh
Successive appr. bit pos high	4 bit	x	0
Successive appr. bit pos low	4 bit	x	0
Image bit stream	n x 8 bit	xx, ..., xx	

## 2.6 EOI End Of Image

The **End Of Image** segment defines the end of an image and has no further parameter.

EOI marker	16 bit	xx, xx	FFh, D9h
------------	--------	--------	----------

### 3. Encoding Example

This example shows the way of an 8x8 block with sampling values to a JPEG compressed bit-stream.

Here an 8x8 block with the original luminance sampling, as unsigned values.

**Figure 9: Block with Original Samples**

139	144	149	153	155	155	155	155
144	151	153	156	159	156	156	156
150	155	160	163	158	156	156	156
159	161	162	160	160	159	159	159
159	160	161	162	162	155	155	155
161	161	161	161	160	157	157	157
162	162	161	163	162	157	157	157
162	162	161	161	163	158	158	158

Now, the sampling values are transformed from the time domain to the frequency domain by a discrete cosine transformation.

**Figure 11: Coefficients After DCT**

236	-1	-12	-5	2	-2	-3	1
-23	-18	-6	-3	-3	0	0	-1
-11	-9	-2	2	0	-1	-1	0
-7	-2	0	2	1	0	0	0
-1	-1	2	2	0	-1	1	1
2	0	2	0	-1	2	1	-1
-1	0	0	-2	-1	2	1	-1
-3	2	-4	-2	2	1	-1	0

After Quantization, most of the higher frequency coefficients are zero.

**Figure 13: Quantized Coefficients**

15	0	-1	0	0	0	0	0
-2	-1	0	0	0	0	0	0
-1	-1	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

As first, the unsigned sampling values are level shifted to signed values, by subtracting 128:

**Figure 10: Block with Level Shifted Samples**

11	16	21	25	27	27	27	27
16	23	25	28	31	28	28	28
22	27	32	35	30	28	28	28
31	33	34	32	32	31	31	31
31	32	33	34	34	27	27	27
33	33	33	33	32	29	29	29
34	34	33	35	34	29	29	29
34	34	33	33	35	30	30	30

For quantization, the default luminance table is used.

**Figure 12: 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

The quantized values are rearranged in Zigzag order.

**Figure 14: Values in Zigzag Order**

15, 0, -2, -1, -1, -1, 0, 0, -1, 0 ... 0

Now, the values are transformed in Run/Size symbols and variable length integers (VLI). For the DC coefficient we assumed, that the previous DC coefficient was 12. The codes for the symbols are taken from the default luminance Huffman table.

**Figure 15: Huffman Encoding**

DC-Value	Difference	(Size),VLI	Code,VLI
15	15 - 12 = 3	(2), 11	011 11
AC-Values		(Run, Size),VLI	Code,VLI
0,-2		(1, 2), 01	11011, 01
-1		(0, 1), 0	00, 0
-1		(0, 1), 0	00, 0
-1		(0, 1), 0	00, 0
0, 0, -1		(2, 1), 0	11100, 0
0 ... 0		EOB	1010

The code and VLI bits are threaded to a finally bit-stream of 31 bit, where represents the 64 byte block.

01111 1101101 000 000 000 111000 1010

The compression rate for this example is  $(64 * 8 \text{ bit} / 31 \text{ bit}) 16.5$ .

## 4. The JE100, JE110, JE150 and JE160 JPEG Encoder

### 4.1 Technical Features

- Optimized for Xilinx Spartan and Virtex FPGA
- Baseline Encoder
- Compliant with Baseline ISO/IEC 10918-1
- Motion-JPEG capability
- 8-bit/pixel or 12-bit/pixel input (Core dependent)
- 8x8 block format pixel input
- Up to 16 components
- Up to 4 Quantization tables
- Up to 8 Huffman tables (four DC and four AC)
- Predefined luminance and chrominance tables
- Fully synchronous design
- Fully stall able design
- Simple CPU interface for table reprogramming
- Different clocks for encoder and CPU interface
- Single clock cycle per pixel encoding
- No pause cycles between blocks

### 4.2 Difference between JE100, JE110, JE150 and JE160

The four cores are very similar, but optimized for different FPGA families and input sample precision:

**Table 3: Difference between Cores**

FPGA Family	8 Bit Sample Precision	12 Bit Sample Precision
Spartan-II Spartan-IIE Virtex Virtex-E	JE100	JE150
Spartan-III Virtex-II Virtex-IIP	JE110	JE160

### 4.3 Needed Resource

Table 4: Needed Resource for JE100

Family	Device	Flip Flops	4 Input Luts	Block RAM	TBufs	Clock IOBs	IOBs	Performance (MHz)
Spartan-IIe	XC2S200E-6	2030	3451	6	154	2	65	80
Spartan-IIe	XC2S200E-7	2030	3451	6	154	2	65	90
Virtex	XCV200-4	2030	3451	6	154	2	65	60
Virtex-E	XCV200E-8	2030	3451	6	154	2	65	100

Table 5: Needed Resource for JE110

Family	Device	Flip Flops	4 Input Luts	Block RAM	TBufs	Clock IOBs	IOBs	Performance (MHz)	Multiplier Blocks
Spartan-III	XC3S1000-4	1646	2157	3	-	2	65	100	16
Spartan-III	XC3S1000-5	1646	2157	3	-	2	65	110	16
Virtex-II	XC2V250-4	1646	2157	3	-	2	65	125	16
Virtex-IIP	XC2VP4-5	1646	2157	3	-	2	65	145	16

Table 6: Needed Resource for JE150

Family	Device	Flip Flops	4 Input Luts	Block RAM	TBufs	Clock IOBs	IOBs	Performance (MHz)
Spartan-IIe	XC2S600E-6	2789	5998	6	208	2	69	55
Spartan-IIe	XC2S600E-7	2789	5998	6	208	2	69	65
Virtex	XCV600-4	2789	5998	6	208	2	69	45
Virtex-E	XCV600E-8	2789	5998	6	208	2	69	75

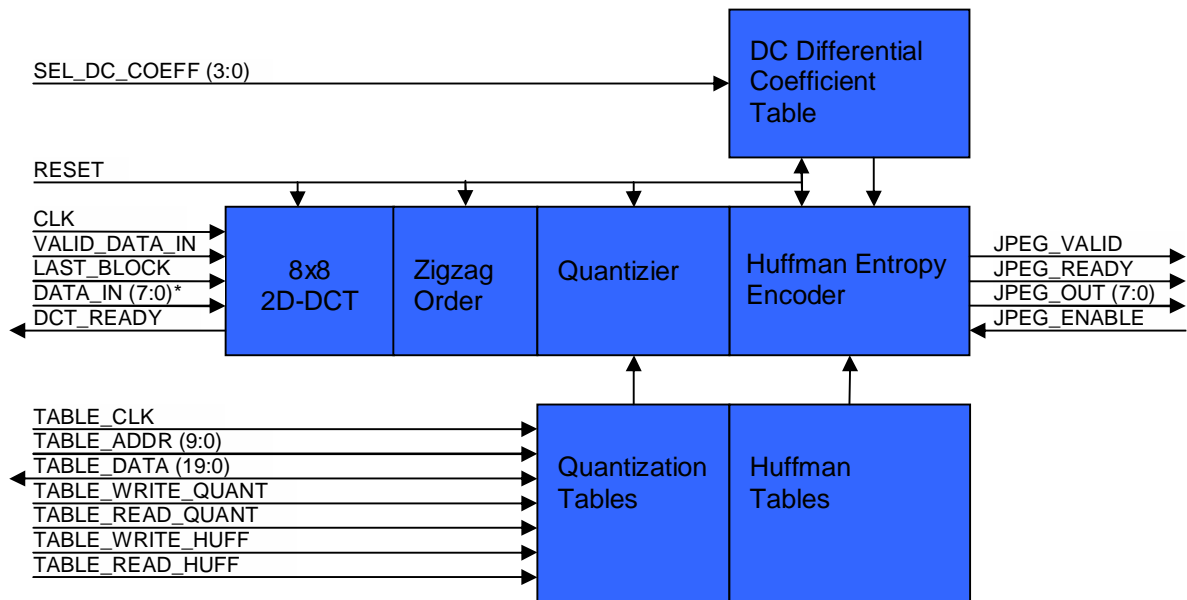
Table 7: Needed Resource for JE160

Family	Device	Flip Flops	4 Input Luts	Block RAM	TBufs	Clock IOBs	IOBs	Performance (MHz)	Multiplier Blocks
Spartan-III	XC3S400-4	2056	2634	3	-	2	69	90	16
Spartan-III	XC3S400-5	2056	2634	3	-	2	69	95	16
Virtex-II	XC2V250-4	2056	2634	3	-	2	69	100	16
Virtex-IIP	XC2VP4-5	2056	2634	3	-	2	69	120	16

### 4.4 Functional Description

The JE100 is an image compressor using the JPEG “baseline system”. The core is a fully synchronous design and has the capability to be stalled from the pixel input and from the compressed image output side.

Figure 16: Encoder Structure



\* Vector size (11:0) on JE150 and JE160

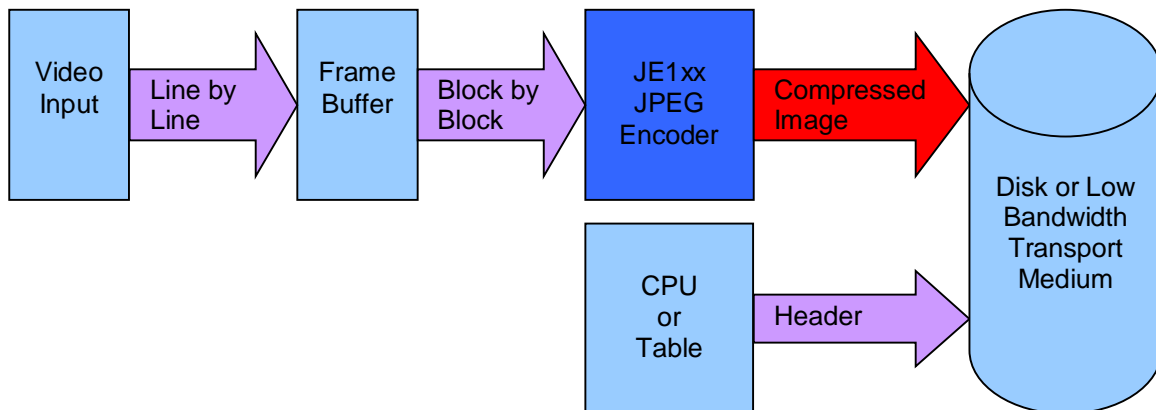
An additional interface, the “Table Programming Interface”, is used to change the Quantization and Huffman tables. The “Table Programming Interface” has an own clock to minimize the glue logic to connect the core to a controller. Usually there is no need to change the tables, only if there are custom tables, optimized for a special kind of images.

Each incoming 8x8 pixel block goes through the two-dimensional discrete cosine transformation (2D-DCT) and will be transformed into the 64 coefficients of the frequency domain. Now, the coefficients will be read out in a Zigzag order and quantized by dividing through the selected Quantization table. The first (DC) quantized coefficient is differentially coded, using the most recently DC coefficient from the same component. Now all coefficients are run length coded to Run/Size symbols (the DC coefficient only to a Size symbol). Finally, the symbols are Huffman coded, using the code from the selected Huffman table. The bit-stream of Huffman codes is divided into parts with 8 bit and stored in the output register.

## 4.5 Typical Application

In a typical application, first the video signal is digitized (if analog). Because the encoding process is done block by block, but the pixels are ordered line by line, the image must be stored in a memory. The pixel are read out in 8x8 blocks and passed to the encoder. The encoder outputs the bit-stream of the compressed image. The bit-stream can be transmitted over a system with limited bandwidth like a network or USB bus. In the case, the bit-stream is stored as a JPEG file to a disk, a header with information about the image size and the used tables must be stored with the bit-stream. The header can be generated by a microcontroller, or if fixed stored in a table.

Figure 17: Block Diagram of a Typical Application



## 4.6 Core Entitys

This is the core entity like defined in the VHDL source code:

### 4.6.1 JE100 Core Entity

```
entity JE100 is Port (
-- Pixel Input
CLK                : in std_logic;                -- Main clock for encoder
RESET              : in std_logic;                -- Reset jpeg logic
DATA_IN            : in std_logic_vector (7 downto 0); -- Sample data
VALID_DATA_IN      : in std_logic;                -- Sample data is valid
LAST_BLOCK         : in std_logic;                -- Last block in this picture
DCT_READY          : out std_logic;               -- Encoder is ready for new pixel
SEL_TABLE_DC_COEFF : in std_logic_vector (3 downto 0); -- DC coefficient table select
SEL_TABLE_QUANT    : in std_logic_vector (1 downto 0); -- Quantization table select
SEL_TABLE_HUFF     : in std_logic_vector (1 downto 0); -- Huffman table select

-- Compressed data Output
JPEG_OUT           : out std_logic_vector (7 downto 0); -- Jpeg stream byte output
JPEG_VALID         : out std_logic;                -- Jpeg stream byte is valid
JPEG_ENABLE        : in std_logic;                -- Application read the jpeg byte
JPEG_READY         : out std_logic;               -- Jpeg stream is finished

-- Tables Programming
TABLE_CLK          : in std_logic;                -- Clock for table reprogramming
TABLE_ADDR         : in std_logic_vector (9 downto 0); -- Table address
TABLE_DATA         : inout std_logic_vector (19 downto 0); -- Table data
TABLE_WRITE_QUANT  : in std_logic;                -- Write value in Quantization tab
TABLE_READ_QUANT   : in std_logic;                -- Read Quantization table
TABLE_WRITE_HUFF   : in std_logic;                -- Write value in Huffman table
TABLE_READ_HUFF    : in std_logic;                -- Read Huffman table
);
end JE100;
```

### 4.6.2 JE110 Core Entity

```
entity JE110 is Port (
-- Pixel Input
CLK                : in std_logic;                -- Main clock for encoder
RESET              : in std_logic;                -- Reset jpeg logic
DATA_IN            : in std_logic_vector (7 downto 0); -- Sample data
VALID_DATA_IN      : in std_logic;                -- Sample data is valid
LAST_BLOCK         : in std_logic;                -- Last block in this picture
DCT_READY          : out std_logic;               -- Encoder is ready for new pixel
SEL_TABLE_DC_COEFF : in std_logic_vector (3 downto 0); -- DC coefficient table select
SEL_TABLE_QUANT    : in std_logic_vector (1 downto 0); -- Quantization table select
SEL_TABLE_HUFF     : in std_logic_vector (1 downto 0); -- Huffman table select

-- Compressed data Output
JPEG_OUT           : out std_logic_vector (7 downto 0); -- Jpeg stream byte output
JPEG_VALID         : out std_logic;                -- Jpeg stream byte is valid
JPEG_ENABLE        : in std_logic;                -- Application read the jpeg byte
JPEG_READY         : out std_logic;               -- Jpeg stream is finished

-- Tables Programming
TABLE_CLK          : in std_logic;                -- Clock for table reprogramming
TABLE_ADDR         : in std_logic_vector (9 downto 0); -- Table address
TABLE_DATA         : inout std_logic_vector (19 downto 0); -- Table data
TABLE_WRITE_QUANT  : in std_logic;                -- Write value in Quantization tab
TABLE_READ_QUANT   : in std_logic;                -- Read Quantization table
TABLE_WRITE_HUFF   : in std_logic;                -- Write value in Huffman table
TABLE_READ_HUFF    : in std_logic;                -- Read Huffman table
);
end JE110;
```

### 4.6.3 JE150 Core Entity

```

entity JE150 is Port (
-- Pixel Input
  CLK           : in std_logic;           -- Main clock for encoder
  RESET         : in std_logic;          -- Reset jpeg logic
  DATA_IN      : in std_logic_vector (11 downto 0); -- Sample data
  VALID_DATA_IN : in std_logic;          -- Sample data is valid
  LAST_BLOCK    : in std_logic;          -- Last block in this picture
  DCT_READY     : out std_logic;         -- Encoder is ready for new pixel
  SEL_TABLE_DC_COEFF : in std_logic_vector (3 downto 0); -- DC coefficient table select
  SEL_TABLE_QUANT : in std_logic_vector (1 downto 0); -- Quantization table select
  SEL_TABLE_HUFF : in std_logic_vector (1 downto 0); -- Huffman table select

-- Compressed data Output
  JPEG_OUT      : out std_logic_vector (7 downto 0); -- Jpeg stream byte output
  JPEG_VALID    : out std_logic;          -- Jpeg stream byte is valid
  JPEG_ENABLE   : in std_logic;          -- Application read the jpeg byte
  JPEG_READY    : out std_logic;         -- Jpeg stream is finished

-- Tables Programming
  TABLE_CLK    : in std_logic;          -- Clock for table reprogramming
  TABLE_ADDR   : in std_logic_vector (9 downto 0); -- Table address
  TABLE_DATA   : inout std_logic_vector (19 downto 0); -- Table data
  TABLE_WRITE_QUANT : in std_logic;     -- Write value in Quantization tab
  TABLE_READ_QUANT : in std_logic;     -- Read Quantization table
  TABLE_WRITE_HUFF : in std_logic;     -- Write value in Huffman table
  TABLE_READ_HUFF : in std_logic;     -- Read Huffman table
);
end JE150;

```

### 4.6.4 JE160 Core Entity

```

entity JE160 is Port (
-- Pixel Input
  CLK           : in std_logic;           -- Main clock for encoder
  RESET         : in std_logic;          -- Reset jpeg logic
  DATA_IN      : in std_logic_vector (11 downto 0); -- Sample data
  VALID_DATA_IN : in std_logic;          -- Sample data is valid
  LAST_BLOCK    : in std_logic;          -- Last block in this picture
  DCT_READY     : out std_logic;         -- Encoder is ready for new pixel
  SEL_TABLE_DC_COEFF : in std_logic_vector (3 downto 0); -- DC coefficient table select
  SEL_TABLE_QUANT : in std_logic_vector (1 downto 0); -- Quantization table select
  SEL_TABLE_HUFF : in std_logic_vector (1 downto 0); -- Huffman table select

-- Compressed data Output
  JPEG_OUT      : out std_logic_vector (7 downto 0); -- Jpeg stream byte output
  JPEG_VALID    : out std_logic;          -- Jpeg stream byte is valid
  JPEG_ENABLE   : in std_logic;          -- Application read the jpeg byte
  JPEG_READY    : out std_logic;         -- Jpeg stream is finished

-- Tables Programming
  TABLE_CLK    : in std_logic;          -- Clock for table reprogramming
  TABLE_ADDR   : in std_logic_vector (9 downto 0); -- Table address
  TABLE_DATA   : inout std_logic_vector (19 downto 0); -- Table data
  TABLE_WRITE_QUANT : in std_logic;     -- Write value in Quantization tab
  TABLE_READ_QUANT : in std_logic;     -- Read Quantization table
  TABLE_WRITE_HUFF : in std_logic;     -- Write value in Huffman table
  TABLE_READ_HUFF : in std_logic;     -- Read Huffman table
);
end JE160;

```

## 4.7 Core Interface

The Core interface is subdivided into three parts: the “Pixel Input Interface”, the “Compressed Data Output Interface” and the “Tables Programming Interface”. The “Tables Programming Interface” is used, only when the tables are programmed with custom values.

**Table 8: Core Entity Signals**

Signal Name	Direction	Description
<b>Pixel Input Interface</b>		
CLK	In	Encoder clock
RESET	In	Reset encoder state machines
VALID_DATA_IN	In	Pixel is valid
LAST_BLOCK	In	Last block of field
DATA_IN (7 : 0)	In	Pixel input, (11:0) for JE150 and JE160
DCT_READY	Out	Encoder is ready for new pixel
SEL_DC_COEFF (3:0)	In	Select the component
SEL_TABLE_QUANT (1:0)	In	Select a Quantization table for this component
SEL_TABLE_HUFF (1:0)	In	Select a Huffman table for this component
<b>Compressed Data Output Interface</b>		
JPEG_ENABLE	In	Data handshake
JPEG_VALID	Out	Output data is valid
JPEG_READY	Out	Last data for this field
JPEG_OUT (7:0)	Out	Output data
<b>Tables Programming Interface</b>		
TABLE_CLK	In	Clock for table reprogramming
TABLE_ADDR (9:0)	In	Table select and addressing
TABLE_DATA (19:0)	In/Out	Table read- or write-data
TABLE_WRITE_QUANT	In	Write data into Quantization table
TABLE_READ_QUANT	In	Read data from Quantization table
TABLE_WRITE_HUFF	In	Write data into Huffman table
TABLE_READ_HUFF	In	Read data from Huffman table

### 4.7.1 Pixel Input Interface

#### CLK

```
CLK          : in std_logic;          -- Main clock for encoder
```

This is the main clock for the entire encoder, except the Tables Programming Interface. Input signals must be valid at the rising edge, output signals are updated with the rising edge too.

## RESET

```
RESET          : in std_logic;          -- Reset jpeg logic
```

Synchron reset of the encoder core, but Quantization tables and Huffman tables are not affected. Assert this signal for at least one clock cycle, if a image compression is canceled.

## DATA\_IN

```
DATA_IN        : in std_logic_vector (7 downto 0);  -- Sample data for JE100 and JE110
DATA_IN        : in std_logic_vector (11 downto 0); -- Sample data for JE150 and JE160
```

This is the pixel input port. The format is 8 (12) bit signed with values from -128 (-2048) to 127 (2047). Subdivide the image in 8x8 blocks and read out the blocks from left to right and top to bottom. The data will be accepted, when "VALID\_DATA\_IN" and "DCT\_READY" are asserted.

## VALID\_DATA\_IN

```
VALID_DATA_IN  : in std_logic;          -- Sample data is valid
```

This is the pixel qualifier. Assert this signal, when a new pixel is valid. Pixel data will be accepted, when "DCT\_READY" is asserted too.

## LAST\_BLOCK

```
LAST_BLOCK     : in std_logic;          -- Last block in this picture
```

This signal must be valid, when the last pixel of a block is stored. Assert this signal, when the very last block of the last component from the entire image is stored. When the last Huffman code of a block leaves the encoder and the block is marked as last, then bit stream is filled up to a complete byte and JPEG\_READY is asserted to mark the image as finished.

## DCT\_READY

```
DCT_READY      : out std_logic;         -- Encoder is ready for new pixel
```

This signal is asserted, when the core is ready to receive new data. When "JPEG\_ENABLE" stays always asserted, then this signal will never be asserted.

This signal is asserted, when the core is ready to receive new data. When "JPEG\_ENABLE" stays always asserted, then usually this signal will not be deasserted. Only, if the compression rate is lower then three on the JE100 and JE110 or lower as four on the JE150 and JE160 then this signal is deasserted, even if "JPEG\_ENABLE" stays always asserted.

## SEL\_DC\_COEFF

```
SEL_TABLE_DC_COEFF : in std_logic_vector (3 downto 0);    -- DC coefficient table select
```

Select a component, used to build the differential coding for the DC coefficient from this block, and must be valid when the last pixel of a block is stored.

For a mono chrome image, let it fixed at zero.

For a 4:4:4 color image, count with every block 0, 1, 2, 0, 1, 2 ...

For a 4:2:2 color image, count with every block 0, 0, 1, 2, 0, 0, 1, 2 ...

## SEL\_TABLE\_QUANT

```
SEL_TABLE_QUANT    : in std_logic_vector (1 downto 0);    -- Quantization table select
```

Select a Quantization table, used to quantize the coefficients from this block, and must be valid when the last pixel of a block is stored. Two tables are initialized after power-up, the first (index 0) is optimized for luminance components and the second (index 1) is optimized for chrominance components.

For a mono chrome image, let it fixed at zero.

For a 4:4:4 color image, count with every block 0, 1, 1, 0, 1, 1 ...

For a 4:2:2 color image, count with every block 0, 0, 1, 1, 0, 0, 1, 1 ...

## SEL\_TABLE\_HUFF

```
SEL_TABLE_HUFF     : in std_logic_vector (1 downto 0);    -- Huffman table select
```

Select a Huffman table, used to get the Huffman code for Run/Size symbols from this block, and must be valid when the last pixel of a block is stored. Two tables are initialized after power-up, the first (index 0) is optimized for luminance components and the second (index 1) is optimized for chrominance components.

For a mono chrome image, let it fixed at zero.

For a 4:4:4 color image, count with every block 0, 1, 1, 0, 1, 1 ...

For a 4:2:2 color image, count with every block 0, 0, 1, 1, 0, 0, 1, 1 ...

## 4.7.2 Compressed Data Output Interface

### JPEG\_OUT

```
JPEG_OUT           : out std_logic_vector (7 downto 0);    -- Jpeg stream byte output
```

Compressed data output stream with a size of 8 bit. Data is valid, when the signal "JPEG\_VALID" is asserted. Stay unchanged until "JPEG\_ENABLE" is asserted. When the very last coefficient is encoded and the byte is not completed, the reminded bits are filled up with ones.

## JPEG\_ENABLE

```
JPEG_VALID          : out std_logic;          -- Jpeg stream byte is valid
```

Assert this signal, when the application is ready to get the data byte. A valid data byte stays on the output until this signal is asserted. If not asserted, while “JPEG\_VALID” is asserted, the output interface will be stalled, but the remain encoder still works until all internal FiFo’s are filled up.

## JPEG\_VALID

```
JPEG_ENABLE        : in std_logic;          -- Application read the jpeg byte
```

This is the output data qualifier. When asserted, the output data is valid and stay valid, until “JPEG\_ENABLE” is asserted. If “JPEG\_ENABLE” is already asserted, the output data is valid, only for one clock cycle.

## JPEG\_READY

```
JPEG_READY         : out std_logic;          -- Jpeg stream is finished
```

This signal will be asserted, when the last byte from the last block of the entire image, is valid. Will say, the image is completely compressed.

### 4.7.3 Tables Programming Interface

#### TABLE\_CLK

```
TABLE_CLK          : in std_logic;          -- Clock for table reprogramming
```

This is the clock for the Tables Programming Interface. Input signals must be valid at the rising edge, output signals are updated with the rising edge too.

#### TABLE\_ADDR

```
TABLE_ADDR         : in std_logic_vector (9 downto 0); -- Table address
```

These are the address lines for the access to the Quantization and Huffman tables.

For Quantization table access:

A5.. A0 select one of the 64 elements from the table.

A7.. A6 select one of the four tables.

A9 and A8 are not used.

For Huffman table access:

A7.. A0 select one of the 256 elements from the table.

A9.. A8 select one of the four tables.

For more information about the tables contents, see in the chapter “6. Tables” section.

## TABLE\_DATA

```
TABLE_DATA      : inout std_logic_vector (19 downto 0); -- Table data
```

Bidirectional data bus to writing into the tables and read there out. Data contents differ in Quantization and Huffman tables. For detail information see the chapter “6. Tables” section.

## TABLE\_WRITE\_QUANT

```
TABLE_WRITE_QUANT : in std_logic; -- Write value in Quantization tab
```

Write enable for the Quantization tables. Assert this signal for one clock cycle. Address and data must be valid when asserted.

## TABLE\_READ\_QUANT

```
TABLE_READ_QUANT : in std_logic; -- Read Quantization table
```

Read enable for the Quantization tables. Assert this signal for two cycles at least. Data is valid one cycle after the last address change.

## TABLE\_WRITE\_HUFF

```
TABLE_WRITE_HUFF : in std_logic; -- Write value in Huffman table
```

Write enable for the Huffman tables. Assert this signal for one clock cycle. Address and data must be valid when asserted.

## TABLE\_READ\_HUFF

```
TABLE_READ_HUFF : in std_logic; -- Read Huffman table
```

Read enable for the Huffman tables. Assert this signal for two cycles at least. Data is valid one cycle after the last address change.

## 5. Using with Xilinx ISE

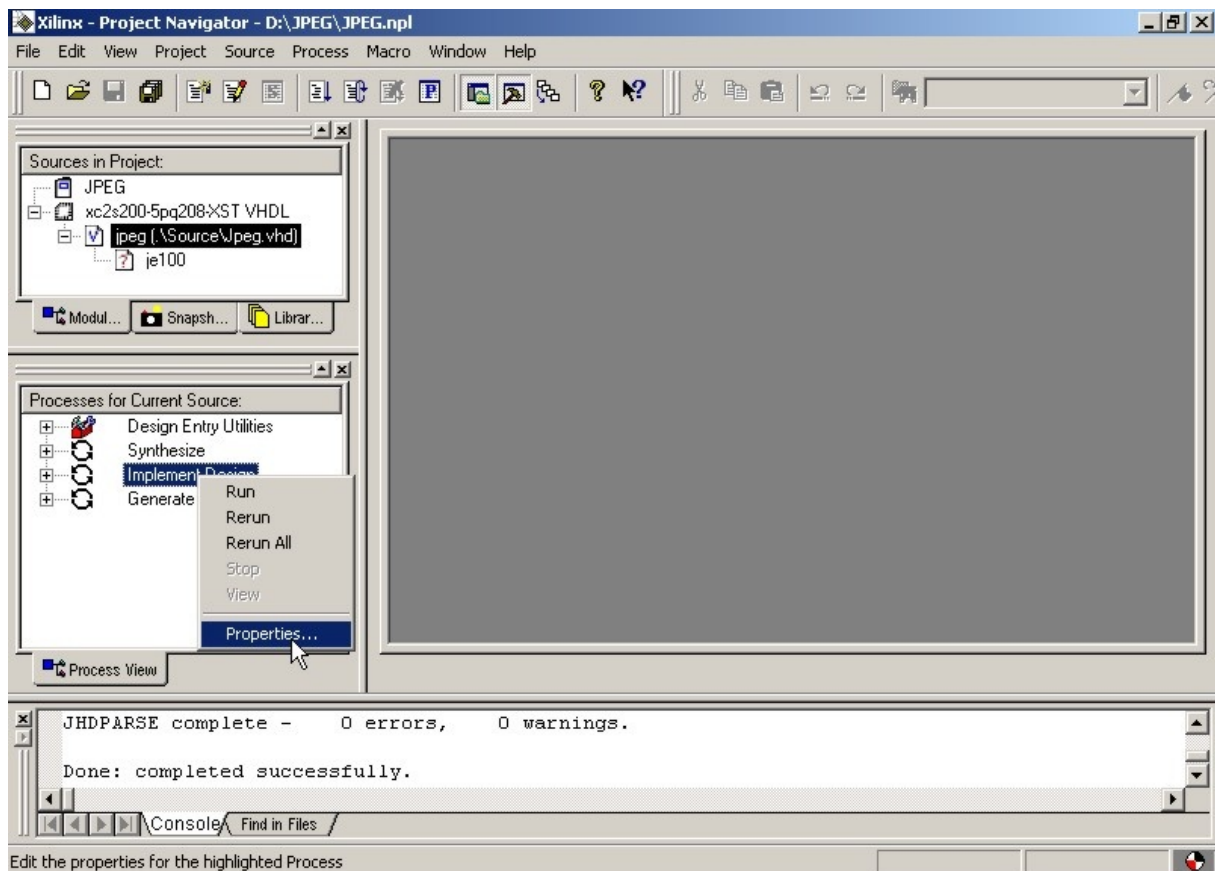
Including the JE1xx into the VHDL project is easy and need only three steps:

- set the “Macro Search Path” to the directory, that contain the NGC file
- copy the JE1xx Component declaration in the source file
- copy the JE1xx Instantiation in your source file and map the signals

### 5.1 NGC File

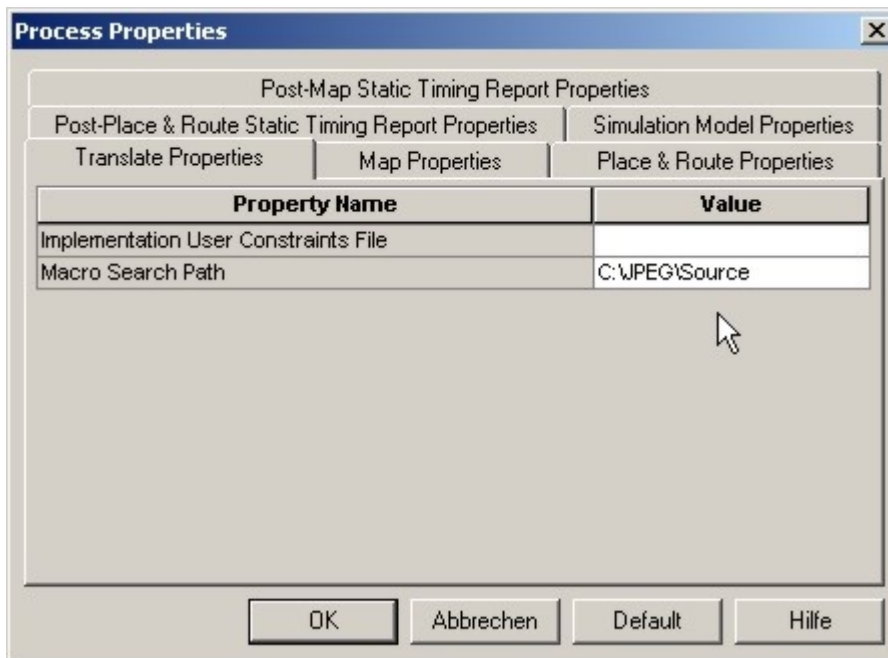
It is a good practice to create a new directory “Source” in your project directory and copy the NGC file into. The ISE must know where to search for the NGC file. This is done by setting the “Macro Search Path”:

Figure 18: ISE Project



Select in the source window the top level source file. In the process window beneath appear all available processes. Select the “Implement Design” process and do a right click. In the pop-up menu click to “Properties...”.

Figure 19: ISE Implementation Process Properties



Set the “Macro Search Path” in the Process Properties to the directory where the NGC file (JE1xx.NGC) is inside.

The ISE shows the JE1xx module with a red question mark as absent. This can be ignored.

## 5.2 Component Declarations

Copy the JE1xx Component declaration into the architecture body:

### 5.2.1 JE100 Component Declaration

```

entity MyEntity is Port (
    .
    .
    .
);
end MyEntity;

architecture RTL of MyEntity is
    .
    .
    .

    component JE100 is Port (
        -- Pixel Input

        CLK                : in std_logic;                -- Main clock for encoder
        RESET               : in std_logic;                -- Reset jpeg logic
        DATA_IN            : in std_logic_vector (7 downto 0); -- Sample data
        VALID_DATA_IN      : in std_logic;                -- Sample data is valid
        LAST_BLOCK         : in std_logic;                -- Last block in this picture
        DCT_READY          : out std_logic;                -- Encoder is ready for new pixel
        SEL_TABLE_DC_COEFF : in std_logic_vector (3 downto 0); -- DC coefficient table select
        SEL_TABLE_QUANT    : in std_logic_vector (1 downto 0); -- Quantization table select
        SEL_TABLE_HUFF     : in std_logic_vector (1 downto 0); -- Huffman table select

        -- Compressed data Output

        JPEG_OUT           : out std_logic_vector (7 downto 0); -- Jpeg stream byte output
        JPEG_VALID        : out std_logic;                -- Jpeg stream byte is valid
        JPEG_ENABLE       : in std_logic;                -- Application read the jpeg byte
        JPEG_READY        : out std_logic;                -- Jpeg stream is finished

        -- Tables Programming

        TABLE_CLK         : in std_logic;                -- Clock for table reprogramming
        TABLE_ADDR        : in std_logic_vector (9 downto 0); -- Table address
        TABLE_DATA        : inout std_logic_vector (19 downto 0); -- Table data
        TABLE_WRITE_QUANT : in std_logic;                -- Write value in Quantization tab
        TABLE_READ_QUANT  : in std_logic;                -- Read Quantization table
        TABLE_WRITE_HUFF  : in std_logic;                -- Write value in Huffman table
        TABLE_READ_HUFF   : in std_logic;                -- Read Huffman table
    );
    end component;

    .
    .
    .

begin

    .
    .
    .

end RTL;

```

## 5.2.2 JE110 Component Declaration

```

entity MyEntity is Port (
    .
    .
    .
);
end MyEntity;

architecture RTL of MyEntity is
    .
    .
    .

    component JE110 is Port (

-- Pixel Input

        CLK                : in std_logic;                -- Main clock for encoder
        RESET              : in std_logic;                -- Reset jpeg logic
        DATA_IN           : in std_logic_vector (7 downto 0); -- Sample data
        VALID_DATA_IN     : in std_logic;                -- Sample data is valid
        LAST_BLOCK        : in std_logic;                -- Last block in this picture
        DCT_READY         : out std_logic;               -- Encoder is ready for new pixel
        SEL_TABLE_DC_COEFF : in std_logic_vector (3 downto 0); -- DC coefficient table select
        SEL_TABLE_QUANT    : in std_logic_vector (1 downto 0); -- Quantization table select
        SEL_TABLE_HUFF     : in std_logic_vector (1 downto 0); -- Huffman table select

-- Compressed data Output

        JPEG_OUT           : out std_logic_vector (7 downto 0); -- Jpeg stream byte output
        JPEG_VALID        : out std_logic;                -- Jpeg stream byte is valid
        JPEG_ENABLE       : in std_logic;                -- Application read the jpeg byte
        JPEG_READY        : out std_logic;               -- Jpeg stream is finished

-- Tables Programming

        TABLE_CLK        : in std_logic;                -- Clock for table reprogramming
        TABLE_ADDR       : in std_logic_vector (9 downto 0); -- Table address
        TABLE_DATA       : inout std_logic_vector (19 downto 0); -- Table data
        TABLE_WRITE_QUANT : in std_logic;               -- Write value in Quantization tab
        TABLE_READ_QUANT  : in std_logic;               -- Read Quantization table
        TABLE_WRITE_HUFF  : in std_logic;               -- Write value in Huffman table
        TABLE_READ_HUFF   : in std_logic;               -- Read Huffman table
    );
end component;

    .
    .
    .

begin

    .
    .
    .

end RTL;

```

## 5.2.3 JE150 Component Declaration

```

entity MyEntity is Port (
    .
    .
    .
);
end MyEntity;

architecture RTL of MyEntity is
    .
    .
    .

    component JE150 is Port (

-- Pixel Input

        CLK                : in std_logic;                -- Main clock for encoder
        RESET              : in std_logic;                -- Reset jpeg logic
        DATA_IN           : in std_logic_vector (11 downto 0); -- Sample data
        VALID_DATA_IN     : in std_logic;                -- Sample data is valid
        LAST_BLOCK        : in std_logic;                -- Last block in this picture
        DCT_READY         : out std_logic;               -- Encoder is ready for new pixel
        SEL_TABLE_DC_COEFF : in std_logic_vector (3 downto 0); -- DC coefficient table select
        SEL_TABLE_QUANT    : in std_logic_vector (1 downto 0); -- Quantization table select
        SEL_TABLE_HUFF     : in std_logic_vector (1 downto 0); -- Huffman table select

-- Compressed data Output

        JPEG_OUT           : out std_logic_vector (7 downto 0); -- Jpeg stream byte output
        JPEG_VALID        : out std_logic;                -- Jpeg stream byte is valid
        JPEG_ENABLE       : in std_logic;                -- Application read the jpeg byte
        JPEG_READY        : out std_logic;               -- Jpeg stream is finished

-- Tables Programming

        TABLE_CLK        : in std_logic;                -- Clock for table reprogramming
        TABLE_ADDR       : in std_logic_vector (9 downto 0); -- Table address
        TABLE_DATA       : inout std_logic_vector (19 downto 0); -- Table data
        TABLE_WRITE_QUANT : in std_logic;               -- Write value in Quantization tab
        TABLE_READ_QUANT  : in std_logic;               -- Read Quantization table
        TABLE_WRITE_HUFF  : in std_logic;               -- Write value in Huffman table
        TABLE_READ_HUFF   : in std_logic;               -- Read Huffman table
    );
end component;

    .
    .
    .

begin

    .
    .
    .

end RTL;

```

## 5.2.4 JE160 Component Declaration

```

entity MyEntity is Port (
    .
    .
    .
);
end MyEntity;

architecture RTL of MyEntity is
    .
    .
    .

    component JE100 is Port (

-- Pixel Input

        CLK                : in std_logic;                -- Main clock for encoder
        RESET              : in std_logic;                -- Reset jpeg logic
        DATA_IN           : in std_logic_vector (11 downto 0); -- Sample data
        VALID_DATA_IN      : in std_logic;                -- Sample data is valid
        LAST_BLOCK         : in std_logic;                -- Last block in this picture
        DCT_READY          : out std_logic;               -- Encoder is ready for new pixel
        SEL_TABLE_DC_COEFF : in std_logic_vector (3 downto 0); -- DC coefficient table select
        SEL_TABLE_QUANT     : in std_logic_vector (1 downto 0); -- Quantization table select
        SEL_TABLE_HUFF      : in std_logic_vector (1 downto 0); -- Huffman table select

-- Compressed data Output

        JPEG_OUT           : out std_logic_vector (7 downto 0); -- Jpeg stream byte output
        JPEG_VALID         : out std_logic;                -- Jpeg stream byte is valid
        JPEG_ENABLE        : in std_logic;                -- Application read the jpeg byte
        JPEG_READY         : out std_logic;               -- Jpeg stream is finished

-- Tables Programming

        TABLE_CLK         : in std_logic;                -- Clock for table reprogramming
        TABLE_ADDR        : in std_logic_vector (9 downto 0); -- Table address
        TABLE_DATA        : inout std_logic_vector (19 downto 0); -- Table data
        TABLE_WRITE_QUANT : in std_logic;                -- Write value in Quantization tab
        TABLE_READ_QUANT  : in std_logic;                -- Read Quantization table
        TABLE_WRITE_HUFF  : in std_logic;                -- Write value in Huffman table
        TABLE_READ_HUFF   : in std_logic;                -- Read Huffman table
    );
end component;

    .
    .
    .

begin

    .
    .
    .

end RTL;

```

## 5.3 Component Instantiations

Copy the JE1xx Component instantiation into the RTL and assign your signals to the entity signals.

### 5.3.1 JE100 Component Instantiation

```

entity MyEntity is Port (
    .
    .
    .
);
end MyEntity;

architecture RTL of MyEntity is
    .
    .
    .
begin
    .
    .
    .

    UJE100: JE100 Port map (

-- Pixel Input

    CLK                => ... ,           -- Main clock for encoder
    RESET              => ... ,           -- Reset jpeg logic
    DATA_IN           => ... ,           -- Sample data
    VALID_DATA_IN      => ... ,           -- Sample data is valid
    LAST_BLOCK         => ... ,           -- Last block in this picture
    DCT_READY          => ... ,           -- Encoder is ready for new pixel
    SEL_TABLE_DC_COEFF => ... ,           -- DC coefficient table select
    SEL_TABLE_QUANT     => ... ,           -- Quantization table select
    SEL_TABLE_HUFF      => ... ,           -- Huffman table select

-- Compressed data Output

    JPEG_OUT           => ... ,           -- Jpeg stream byte output
    JPEG_VALID         => ... ,           -- Jpeg stream byte is valid
    JPEG_ENABLE        => ... ,           -- Application read the jpeg byte
    JPEG_READY         => ... ,           -- Jpeg stream is finished

-- Tables Programming

    TABLE_CLK         => ... ,           -- Clock for table reprogramming
    TABLE_ADDR        => ... ,           -- Table address
    TABLE_DATA        => ... ,           -- Table data
    TABLE_WRITE_QUANT => ... ,           -- Write value in Quantization tab
    TABLE_READ_QUANT  => ... ,           -- Read Quantization table
    TABLE_WRITE_HUFF  => ... ,           -- Write value in Huffman table
    TABLE_READ_HUFF   => ... ,           -- Read Huffman table
    );

    .
    .
    .
end RTL;

```

### 5.3.2 JE110 Component Instantiation

```

entity MyEntity is Port (
    .
    .
    .
);
end MyEntity;

architecture RTL of MyEntity is
    .
    .
    .

begin

    .
    .
    .

UJE110: JE110 Port map (

-- Pixel Input

    CLK                => ... ,                -- Main clock for encoder
    RESET              => ... ,                -- Reset jpeg logic
    DATA_IN           => ... ,                -- Sample data
    VALID_DATA_IN      => ... ,                -- Sample data is valid
    LAST_BLOCK         => ... ,                -- Last block in this picture
    DCT_READY          => ... ,                -- Encoder is ready for new pixel
    SEL_TABLE_DC_COEFF => ... ,                -- DC coefficient table select
    SEL_TABLE_QUANT     => ... ,                -- Quantization table select
    SEL_TABLE_HUFF      => ... ,                -- Huffman table select

-- Compressed data Output

    JPEG_OUT           => ... ,                -- Jpeg stream byte output
    JPEG_VALID         => ... ,                -- Jpeg stream byte is valid
    JPEG_ENABLE        => ... ,                -- Application read the jpeg byte
    JPEG_READY         => ... ,                -- Jpeg stream is finished

-- Tables Programming

    TABLE_CLK         => ... ,                -- Clock for table reprogramming
    TABLE_ADDR        => ... ,                -- Table address
    TABLE_DATA        => ... ,                -- Table data
    TABLE_WRITE_QUANT => ... ,                -- Write value in Quantization tab
    TABLE_READ_QUANT  => ... ,                -- Read Quantization table
    TABLE_WRITE_HUFF  => ... ,                -- Write value in Huffman table
    TABLE_READ_HUFF   => ... ,                -- Read Huffman table
);

    .
    .
    .

end RTL;

```

### 5.3.3 JE150 Component Instantiation

```

entity MyEntity is Port (
    .
    .
    .
);
end MyEntity;

architecture RTL of MyEntity is
    .
    .
    .

begin

    .
    .
    .

UJE150: JE150 Port map (

-- Pixel Input

    CLK           => ... ,           -- Main clock for encoder
    RESET         => ... ,           -- Reset jpeg logic
    DATA_IN      => ... ,           -- Sample data
    VALID_DATA_IN => ... ,           -- Sample data is valid
    LAST_BLOCK    => ... ,           -- Last block in this picture
    DCT_READY     => ... ,           -- Encoder is ready for new pixel
    SEL_TABLE_DC_COEFF => ... ,     -- DC coefficient table select
    SEL_TABLE_QUANT => ... ,         -- Quantization table select
    SEL_TABLE_HUFF => ... ,         -- Huffman table select

-- Compressed data Output

    JPEG_OUT      => ... ,           -- Jpeg stream byte output
    JPEG_VALID    => ... ,           -- Jpeg stream byte is valid
    JPEG_ENABLE   => ... ,           -- Application read the jpeg byte
    JPEG_READY    => ... ,           -- Jpeg stream is finished

-- Tables Programming

    TABLE_CLK    => ... ,           -- Clock for table reprogramming
    TABLE_ADDR   => ... ,           -- Table address
    TABLE_DATA   => ... ,           -- Table data
    TABLE_WRITE_QUANT => ... ,     -- Write value in Quantization tab
    TABLE_READ_QUANT => ... ,      -- Read Quantization table
    TABLE_WRITE_HUFF => ... ,      -- Write value in Huffman table
    TABLE_READ_HUFF => ... ,       -- Read Huffman table
);

    .
    .
    .

end RTL;

```

### 5.3.4 JE160 Component Instantiation

```

entity MyEntity is Port (
    .
    .
    .
);
end MyEntity;

architecture RTL of MyEntity is
    .
    .
    .

begin

    .
    .
    .

UJE160: JE160 Port map (

-- Pixel Input

    CLK           => ... ,           -- Main clock for encoder
    RESET         => ... ,           -- Reset jpeg logic
    DATA_IN      => ... ,           -- Sample data
    VALID_DATA_IN => ... ,           -- Sample data is valid
    LAST_BLOCK    => ... ,           -- Last block in this picture
    DCT_READY     => ... ,           -- Encoder is ready for new pixel
    SEL_TABLE_DC_COEFF => ... ,     -- DC coefficient table select
    SEL_TABLE_QUANT => ... ,        -- Quantization table select
    SEL_TABLE_HUFF => ... ,         -- Huffman table select

-- Compressed data Output

    JPEG_OUT      => ... ,           -- Jpeg stream byte output
    JPEG_VALID    => ... ,           -- Jpeg stream byte is valid
    JPEG_ENABLE   => ... ,           -- Application read the jpeg byte
    JPEG_READY    => ... ,           -- Jpeg stream is finished

-- Tables Programming

    TABLE_CLK    => ... ,           -- Clock for table reprogramming
    TABLE_ADDR   => ... ,           -- Table address
    TABLE_DATA   => ... ,           -- Table data
    TABLE_WRITE_QUANT => ... ,     -- Write value in Quantization tab
    TABLE_READ_QUANT => ... ,     -- Read Quantization table
    TABLE_WRITE_HUFF => ... ,     -- Write value in Huffman table
    TABLE_READ_HUFF => ... ,     -- Read Huffman table
);

    .
    .
    .

end RTL;

```

## 6. Quantization Tables

The Quantization table contains the divisors for the coefficient quantization. There are four tables implemented, the first two tables are predefined after power-up. The RESET signal doesn't affect the table's contents.

The address lines "TABLE\_ADDR" are used to select a value in one of the four tables:

A9 – A8	A7 – A6	A5 – A0
Unused	Table	Zigzag Index

Because the values in the table are in Zigzag order, a table is needed to convert the indices from a Quantization table in coefficient order. The following table can be used to do this.

**Table 9: Zigzag Order**

	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xA	xB	xC	xD	xE	xF
0x	0	1	8	16	9	2	3	10	17	24	32	25	18	11	04	05
1x	12	19	26	33	40	48	41	34	27	20	13	06	07	14	21	28
2x	35	42	49	56	57	50	43	36	29	22	15	23	30	37	44	51
3x	58	59	52	45	38	31	39	46	53	60	61	54	47	55	62	63

To fill the JE1xx Quantization table from a Divisor table in coefficient order, use the following C code:

```
int Zigzag_Table [64] = {
    0, 1, 8,16, 9, 2, 3,10,
    17,24,32,25,18,11,04,05,
    12,19,26,33,40,48,41,34,
    27,20,13,06,07,14,21,28,
    35,42,49,56,57,50,43,36,
    29,22,15,23,30,37,44,51,
    58,59,52,45,38,31,39,46,
    53,60,61,54,47,55,62,63
};

for (i = 0; i < 64; i++) {
    Quantization_Table [i] = Divisor_Table [ Zigzag_Table [i] ];
}
```

Each table entry has a size of 8 bit and contains the unsigned divisor for quantization. Allowed values are 1 to 255.

D19 – D8	D7 – D0
Unused	Divisor

## 6.1 Default Quantization Table for Luminance

This Quantization table is predefined in the core as index 0. The table is shown in the coefficient order.

Table 10: Default Quantization Table for Luminance

	x0	x1	x2	x3	x4	x5	x6	x7
0x	16	11	10	16	24	40	51	61
1x	12	12	14	19	26	58	60	55
2x	14	13	16	24	40	57	69	56
3x	14	17	22	29	51	87	80	62
4x	18	22	37	56	68	109	103	77
5x	24	35	55	64	81	104	113	92
6x	49	64	78	87	103	121	120	101
7x	72	92	95	98	112	100	103	99

## 6.2 Default Quantization Table for Chrominance

This Quantization table is predefined in the core as index 1. The table is shown in the coefficient order.

Table 11: Default Quantization Table for Chrominance

	x0	x1	x2	x3	x4	x5	x6	x7
0x	17	18	24	47	99	99	99	99
1x	18	21	26	66	99	99	99	99
2x	24	26	56	99	99	99	99	99
3x	47	66	99	99	99	99	99	99
4x	99	99	99	99	99	99	99	99
5x	99	99	99	99	99	99	99	99
6x	99	99	99	99	99	99	99	99
7x	99	99	99	99	99	99	99	99

## 6.3 Changing the Compression Rate

To change the compression rate, change the values of the Quantization table. There is no rule how to do this, but usually the default values are scaled. A Quality parameter (Q) is used to determinate the scaling. A low value results in a low quality image with a high compression rate and a high value results in a high quality image with a low compression rate. Allowed values are 1 to 100; see at our web pages for sample images with various quality values.

Use the following formula to calculate a scaled Quantization table:

Quality : Q (1..100)  
Scaling factor : S  
Item from the default table :  $XD_i$   
Item from the scaled table :  $XS_i$

$S = 50/Q$  for  $Q < 50$

$S = 2-Q/50$  for  $Q \geq 50$

$XS_i = XD_i * S$

The following C code illustrates the scaling of the Quantization table:

```
int    Q = 75;
double S;
int    i;

if (Q <= 50) {
    S = 50.0 / Level;
} else {
    S = 2.0 - Q / 50.0;
}

for (i = 0; i < 64; i++) {
    Scaled_Quantization_Table[i] = (BYTE)(Default_Quantization_Table[i] * S);
}
```

## 7. Huffman Tables

The Huffman table contains the Huffman code words for the DC and AC coefficients, there are different codes for the DC and the AC coefficients. The both logical tables for the Size symbol (DC) and the Run/Size symbols (AC) are integrated in one physical table. There are four tables implemented, the first two tables are initialized with valid values after power-up. The RESET signal doesn't affect the table's contents. The columns 0 to A contain the code words for the AC symbols. The last column contains the code words for the DC symbol. The address line signals "TABLE\_ADDR" are used to select a value in one of the four tables:

To access the DC part, let A0 to A3 at a fixed value of Fh:

A9 – A8	A7 – A4	A3	A2	A1	A0
Table	Size	1	1	1	1

The AC part of the table use all address lines, the Size symbol have a maximal value of ten:

A9 – A8	A7 – A4	A3 – A0
Table	Run Length	Size

The position of codes for the DC Size symbols and AC Run/Size symbols is showed in this table:

**Table 12: DC and AC Codes Position Within the Huffman Table**

	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xA	xB*	xC*	xD*	xE*	xF
0x	EOB	0/1	0/2	0/3	0/4	0/5	0/6	0/7	0/8	0/9	0/10	0/11*	0/12*	0/13*	0/14*	DS0
1x		1/1	1/2	1/3	1/4	1/5	1/6	1/7	1/8	1/9	1/10	1/11*	1/12*	1/13*	1/14*	DS1
2x		2/1	2/2	2/3	2/4	2/5	2/6	2/7	2/8	2/9	2/10	2/11*	2/12*	2/13*	2/14*	DS2
3x		3/1	3/2	3/3	3/4	3/5	3/6	3/7	3/8	3/9	3/10	3/11*	3/12*	3/13*	3/14*	DS3
4x		4/1	4/2	4/3	4/4	4/5	4/6	4/7	4/8	4/9	4/10	4/11*	4/12*	4/13*	4/14*	DS4
5x		5/1	5/2	5/3	5/4	5/5	5/6	5/7	5/8	5/9	5/10	5/11*	5/12*	5/13*	5/14*	DS5
6x		6/1	6/2	6/3	6/4	6/5	6/6	6/7	6/8	6/9	6/10	6/11*	6/12*	6/13*	6/14*	DS6
7x		7/1	7/2	7/3	7/4	7/5	7/6	7/7	7/8	7/9	7/10	7/11*	7/12*	7/13*	7/14*	DS7
8x		8/1	8/2	8/3	8/4	8/5	8/6	8/7	8/8	8/9	8/10	8/11*	8/12*	8/13*	8/14*	DS8
9x		9/1	9/2	9/3	9/4	9/5	9/6	9/7	9/8	9/9	9/10	9/11*	9/12*	9/13*	9/14*	DS9
Ax		10/1	10/2	10/3	10/4	10/5	10/6	10/7	10/8	10/9	10/10	10/11*	10/12*	10/13*	10/14*	DS10
Bx		11/1	11/2	11/3	11/4	11/5	11/6	11/7	11/8	11/9	11/10	11/11*	11/12*	11/13*	11/14*	DS11
Cx		12/1	12/2	12/3	12/4	12/5	12/6	12/7	12/8	12/9	12/10	12/11*	12/12*	12/13*	12/14*	DS12*
Dx		13/1	13/2	13/3	13/4	13/5	13/6	13/7	13/8	13/9	13/10	13/11*	13/12*	13/13*	13/14*	DS13*
Ex		14/1	14/2	14/3	14/4	14/5	14/6	14/7	14/8	14/9	14/10	14/11*	14/12*	14/13*	14/14*	DS14*
Fx	ZRL	15/1	15/2	15/3	15/4	15/5	15/6	15/7	15/8	15/9	15/10	15/11*	15/12*	15/13*	15/14*	DS15*

\*Defined only for 12 bit precision.

Each table entry has a size of 20 bit. The lower 16 bits contain the code word with variable length. Code words with less than 16 bits are shifted to the MSB and the unused lower bits are set to one. The upper 4 bits are the code length, decreased by one. The code “11111000” with a length of 8 results in a table entry of 7F8FFh.

D19 – D16	D15 – D0
Code Length	Code

## 7.1 Huffman Tables Generation

Huffman tables are generated from two lists. The first list specifies the number of codes for every possible code length. The maximal code length is limited to 16 bit, so we need a list with 16 values. The second list specifies the assignment of symbols to the codes and has a length of 12 (16 for JE150 and JE160) for a DC table and a length of 162 (240 for JE150 and JE160) for an AC table. The demo program contains a C++ method to generate the tables.

## 7.2 Default Huffman Tables for JE100 and JE110

### 7.2.1 Huffman Table for Luminance

This table shows the number of codes for the luminance DC table:

**Table 13: Luminance Number of DC Codes**

Length	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Number	0h	1h	5h	1h	1h	1h	1h	1h	1h	0h	0h	0h	0h	0h	0h	0h

This table shows the assignment of generated codes to the Size symbols for the luminance DC table:

**Table 14: Luminance DC Symbol to Code Assignment**

Code	0	1	2	3	4	5	6	7	8	9	10	11
Symbol	0h	1h	2h	3h	4h	5h	6h	7h	8h	9h	Ah	Bh

This table shows the number of codes for the luminance AC table:

**Table 15: Luminance Number of AC Codes**

Length	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Number	0h	2h	1h	3h	3h	2h	4h	3h	5h	5h	4h	4h	0h	0h	1h	7Dh

This table shows the assignment of generated codes to the Run/Size symbols for the luminance AC table:

**Table 16: Luminance AC Symbol to Code Assignment**

	x0	x1	X2	x3	x4	x5	x6	X7	x8	x9	xA	xB	xC	xD	xE	xF
0x	01h	02h	03h	00h	04h	11h	05h	12h	21h	31h	41h	06h	13h	51h	61h	07h
1x	22h	71h	14h	32h	81h	91h	A1h	08h	23h	42h	B1h	C1h	15h	52h	D1h	F0h
2x	24h	33h	62h	72h	82h	09h	0Ah	16h	17h	18h	19h	1Ah	25h	26h	27h	28h
3x	29h	2Ah	34h	35h	36h	37h	38h	39h	3Ah	43h	44h	45h	46h	47h	48h	49h
4x	4Ah	53h	54h	55h	56h	57h	58h	59h	5Ah	63h	64h	65h	66h	67h	68h	69h
5x	6Ah	73h	74h	75h	76h	77h	78h	79h	7Ah	83h	84h	85h	86h	87h	88h	89h
6x	8Ah	92h	93h	94h	95h	96h	97h	98h	99h	9Ah	A2h	A3h	A4h	A5h	A6h	A7h
7x	A8h	A9h	AAh	B2h	B3h	B4h	B5h	B6h	B7h	B8h	B9h	BAh	C2h	C3h	C4h	C5h
8x	C6h	C7h	C8h	C9h	CAh	D2h	D3h	D4h	D5h	D6h	D7h	D8h	D9h	DAh	E1h	E2h
9x	E3h	E4h	E5h	E6h	E7h	E8h	E9h	EAh	F1h	F2h	F3h	F4h	F5h	F6h	F7h	F8h
Ax	F9h	FAh														

The generated Huffman table with the AC and DC codes for the luminance symbols. After power-up, the Huffman table with index 0 is initialized with these values (all values in Hex):

**Table 17: Default Huffman Codes for Luminance**

	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xA		xF
0x	3AFFf	13FFF	17FFF	29FFF	3BFFF	4D7FF	6F1FF	7F8FF	9FD8F	FFF82	FFF83	~	13FFF
1x		3CFFF	4DFFF	6F3FF	8FB7F	AFEDF	FFF84	FFF85	FFF86	FFF87	FFF88	~	25FFF
2x		4E7FF	7F9FF	9FDFF	BFF4F	FFF89	FFF8A	FFF8B	FFF8C	FFF8D	FFF8E	~	27FFF
3x		5EBFF	8FBFF	BFF5F	FFF8F	FFF90	FFF91	FFF92	FFF93	FFF94	FFF95	~	29FFF
4x		5EFFF	9FE3F	FFF96	FFF97	FFF98	FFF99	FFF9A	FFF9B	FFF9C	FFF9D	~	2BFFF
5x		6F5FF	AFEFF	FFF9E	FFF9F	FFFA0	FFFA1	FFFA2	FFFA3	FFFA4	FFFA5	~	2DFFF
6x		6F7FF	BFF6F	FFFA6	FFFA7	FFFA8	FFFA9	FFFAA	FFFAB	FFFAC	FFFAD	~	3EFFF
7x		7FAFF	BFF7F	FFFAE	FFFAF	FFFB0	FFFB1	FFFB2	FFFB3	FFFB4	FFFB5	~	4F7FF
8x		8FC7F	EFF81	FFFB6	FFFB7	FFFB8	FFFB9	FFFBa	FFFBb	FFFBc	FFFBd	~	5FBFF
9x		8FCFF	FFFBE	FFFBF	FFFC0	FFFC1	FFFC2	FFFC3	FFFC4	FFFC5	FFFC6	~	6FDFF
Ax		8FD7F	FFFC7	FFFC8	FFFC9	FFFCa	FFFCb	FFFCc	FFFCd	FFFCe	FFFCf	~	7FEFF
Bx		9FE7F	FFFD0	FFFD1	FFFD2	FFFD3	FFFD4	FFFD5	FFFD6	FFFD7	FFFD8	~	8FF7F
Cx		9FEBF	FFFD9	FFFDa	FFFDb	FFFDc	FFFDd	FFFDE	FFFDf	FFFE0	FFFE1	~	
Dx		AFF1F	FFFE2	FFFE3	FFFE4	FFFE5	FFFE6	FFFE7	FFFE8	FFFE9	FFFEa	~	
Ex		FFFEb	FFFEc	FFFEd	FFFEe	FFFEf	FFFF0	FFFF1	FFFF2	FFFF3	FFFF4	~	
Fx	AFF3F	FFFF5	FFFF6	FFFF7	FFFF8	FFFF9	FFFAA	FFFBb	FFFCc	FFFDd	FFFEe	~	

### 7.2.2 Huffman Table for Chrominance

This table shows the number of codes for the chrominance DC table:

**Table 18: Chrominance Number of DC Codes**

Length	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Number	0h	3h	1h	1h	1h	1h	1h	1h	1h	1h	1h	0h	0h	0h	0h	0h

This table shows the assignment of generated code to the Size symbol for the chrominance DC table:

**Table 19: Chrominance DC Symbol to Code Assignment**

Code	0	1	2	3	4	5	6	7	8	9	10	11
Symbol	0h	1h	2h	3h	4h	5h	6h	7h	8h	9h	Ah	Bh

This table shows the number of codes for the chrominance AC table:

**Table 20: Chrominance Number of AC Codes**

Length	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Number	0h	2h	1h	2h	4h	4h	3h	4h	7h	5h	4h	4h	0h	1h	2h	77h

This table shows the assignment of generated code to the Run/Size symbol for the chrominance AC table:

**Table 21: Chrominance AC Symbol to Code Assignment**

	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xA	xB	xC	xD	xE	xF
0x	00h	01h	02h	03h	11h	04h	05h	21h	31h	06h	12h	41h	51h	07h	61h	71h
1x	13h	22h	32h	81h	08h	14h	42h	91h	A1h	B1h	C1h	09h	23h	33h	52h	F0h
2x	15h	62h	72h	D1h	0Ah	16h	24h	34h	E1h	25h	F1h	17h	18h	19h	1Ah	26h
3x	27h	28h	29h	2Ah	35h	36h	37h	38h	39h	3Ah	43h	44h	45h	46h	47h	48h
4x	49h	4Ah	53h	54h	55h	56h	57h	58h	59h	5Ah	63h	64h	65h	66h	67h	68h
5x	69h	6Ah	73h	74h	75h	76h	77h	78h	79h	7Ah	82h	83h	84h	85h	86h	87h
6x	88h	89h	8Ah	92h	93h	94h	95h	96h	97h	98h	99h	9Ah	A2h	A3h	A4h	A5h
7x	A6h	A7h	A8h	A9h	AAh	B2h	B3h	B4h	B5h	B6h	B7h	B8h	B9h	BAh	C2h	C3h
8x	C4h	C5h	C6h	C7h	C8h	C9h	CAh	D2h	D3h	D4h	D5h	D6h	D7h	D8h	D9h	DAh
9x	E2h	E3h	E4h	E5h	E6h	E7h	E8h	E9h	EAh	F2h	F3h	F4h	F5h	F6h	F7h	F8h
Ax	F9h	FAh														

The generated Huffman table with the AC and DC codes for the chrominance symbols. After power-up, the Huffman table with index 1 is initialized with these values (all values in Hex):

**Table 22: Default Huffman Codes for Chrominance**

	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xA	xF
0x	13FFF	17FFF	29FFF	3AFFF	4C7FF	4CFFF	5E3FF	6F1FF	8FA7F	9FDBF	BFF4F	~ 13FFF
1x		3BFFF	5E7FF	7F6FF	8FAFF	AFEDF	BFF5F	FFF88	FFF89	FFF8A	FFF8B	~ 17FFF
2x		4D7FF	7F7FF	9FDFF	BFF6F	EFF85	FFF8C	FFF8D	FFF8E	FFF8F	FFF90	~ 1BFFF
3x		4DFFF	7F8FF	9FE3F	BFF7F	FFF91	FFF92	FFF93	FFF94	FFF95	FFF96	~ 2DFFF
4x		5EBFF	8FB7F	FFF97	FFF98	FFF99	FFF9A	FFF9B	FFF9C	FFF9D	FFF9E	~ 3EFFF
5x		5EFFF	9FE7F	FFF9F	FFFA0	FFFA1	FFFA2	FFFA3	FFFA4	FFFA5	FFFA6	~ 4F7FF
6x		6F3FF	AFEFF	FFFA7	FFFA8	FFFA9	FFFAA	FFFAB	FFFAC	FFFAD	FFFAE	~ 5FBFF
7x		6F5FF	AFF1F	FFFAF	FFFB0	FFFB1	FFFB2	FFFB3	FFFB4	FFFB5	FFFB6	~ 6FDFF
8x		7F9FF	FFFB7	FFFB8	FFFB9	FFFBa	FFFBb	FFFBc	FFFBd	FFFBf	FFFBf	~ 7FEFF
9x		8FBFF	FFFC0	FFFC1	FFFC2	FFFC3	FFFC4	FFFC5	FFFC6	FFFC7	FFFC8	~ 8FF7F
Ax		8FC7F	FFFC9	FFFCa	FFFCb	FFFCc	FFFCd	FFFCe	FFFCf	FFFD0	FFFD1	~ 9FFBF
Bx		8FCFF	FFFD2	FFFD3	FFFD4	FFFD5	FFFD6	FFFD7	FFFD8	FFFD9	FFFDa	~ AFFDF
Cx		8FD7F	FFFDb	FFFDc	FFFDd	FFFDE	FFFDf	FFFE0	FFFE1	FFFE2	FFFE3	~
Dx		AFF3F	FFFE4	FFFE5	FFFE6	FFFE7	FFFE8	FFFE9	FFFEa	FFFEb	FFFEc	~
Ex		DFF83	FFFEd	FFFEe	FFFEf	FFFF0	FFFF1	FFFF2	FFFF3	FFFF4	FFFF5	~
Fx	9FEBF	EFF87	FFFF6	FFFF7	FFFF8	FFFF9	FFFFa	FFFFb	FFFFc	FFFFd	FFFFe	~

## 7.3 Default Huffman Tables for JE150 and JE160

### 7.3.1 Huffman Table for Luminance

This table shows the number of codes for the luminance DC table:

**Table 23: Luminance Number of DC Codes**

Length	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Number	0h	1h	5h	1h	1h	1h	1h	1h	1h	1h	1h	1h	1h	0h	0h	0h

This table shows the assignment of generated codes to the Size symbols for the luminance DC table:

**Table 24: Luminance DC Symbol to Code**

Code	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Symbol	0h	1h	2h	3h	4h	5h	6h	7h	8h	9h	Ah	Bh	Ch	Dh	Eh	Fh

This table shows the number of codes for the luminance AC table:

**Table 25: Luminance Number of AC Codes**

Length	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Number	0h	2h	1h	3h	2h	4h	6h	3h	0h	0h	1h	1h	0h	1h	1h	C9h

This table shows the assignment of generated codes to the Run/Size symbols for the luminance AC table:

**Table 26: Luminance AC Symbol to Code Assignment**

	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xA	xB	xC	xD	xE	xF
0x	01h	02h	03h	00h	04h	11h	05h	12h	21h	31h	41h	06h	13h	51h	61h	07h
1x	22h	71h	14h	32h	81h	91h	A1h	08h	23h	42h	B1h	C1h	15h	52h	D1h	F0h
2x	24h	33h	62h	72h	82h	09h	0Ah	16h	17h	18h	19h	1Ah	25h	26h	27h	28h
3x	29h	2Ah	34h	35h	36h	37h	38h	39h	3Ah	43h	44h	45h	46h	47h	48h	49h
4x	4Ah	53h	54h	55h	56h	57h	58h	59h	5Ah	63h	64h	65h	66h	67h	68h	69h
5x	6Ah	73h	74h	75h	76h	77h	78h	79h	7Ah	83h	84h	85h	86h	87h	88h	89h
6x	8Ah	92h	93h	94h	95h	96h	97h	98h	99h	9Ah	A2h	A3h	A4h	A5h	A6h	A7h
7x	A8h	A9h	AAh	B2h	B3h	B4h	B5h	B6h	B7h	B8h	B9h	BAh	C2h	C3h	C4h	C5h
8x	C6h	C7h	C8h	C9h	CAh	D2h	D3h	D4h	D5h	D6h	D7h	D8h	D9h	DAh	E1h	E2h
9x	E3h	E4h	E5h	E6h	E7h	E8h	E9h	EAh	F1h	F2h	F3h	F4h	F5h	F6h	F7h	F8h
Ax	F9h	FAh	0Bh	0Ch	0Dh	0Eh	1Bh	1Ch	1Dh	1Eh	2Bh	2Ch	2Dh	2Eh	3Bh	3Ch
Bx	3Dh	3Eh	4Bh	4Ch	4Dh	4Eh	5Bh	5Ch	5Dh	5Eh	6Bh	6Ch	6Dh	6Eh	7Bh	7Ch
Cx	7Dh	7Eh	8Bh	8Ch	8Dh	8Eh	9Bh	9Ch	9Dh	9Eh	ABh	ACh	ADh	A Eh	BBh	BCh
Dx	BDh	BEh	CBh	CCh	CDh	CEh	DBh	DCh	DDh	DEh	EBh	ECh	EDh	EEh	FBh	FCh
Ex	FDh	FEh														

The generated Huffman table with the AC and DC codes for the luminance symbols. After power-up, the Huffman table with index 0 is initialized with these values (all values in Hex, unused bits are set to one):

**Table 27: Default Huffman Codes for Luminance**

	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xA	xB	xC	xD	xE	xF
0x	3AFF	13FF	17FF	29FF	3BFF	4D7F	5EFFF	6F7FF	DFF33	FFF42	FFF43	FFFBF	FFFC0	FFFC1	FFFC2	13FFF
1x		3CFF	4DFF	6F1F	7FCFF	FFF39	FFF44	FFF45	FFF46	FFF47	FFF48	FFFC3	FFFC4	FFFC5	FFFC6	25FFF
2x		5E3F	6F9F	EFF35	FFF3D	FFF49	FFF4A	FFF4B	FFF4C	FFF4D	FFF4E	FFFC7	FFFC8	FFFC9	FFFCa	27FFF
3x		5E7F	7DF	FFF3E	FFF4F	FFF50	FFF51	FFF52	FFF53	FFF54	FFF55	FFFCB	FFFCc	FFFCd	FFFCe	29FFF
4x		5EBF	FFF36	FFF56	FFF57	FFF58	FFF59	FFF5A	FFF5B	FFF5C	FFF5D	FFFCF	FFFD0	FFFD1	FFFD2	2BFFF
5x		6F3F	FFF3A	FFF5E	FFF5F	FFF60	FFF61	FFF62	FFF63	FFF64	FFF65	FFFD3	FFFD4	FFFD5	FFFD6	2DFFF
6x		6F5F	FFF3F	FFF66	FFF67	FFF68	FFF69	FFF6A	FFF6B	FFF6C	FFF6D	FFFD7	FFFD8	FFFD9	FFFDa	3EFFF
7x		6FBF	FFF40	FFF6E	FFF6F	FFF70	FFF71	FFF72	FFF73	FFF74	FFF75	FFFDB	FFFDc	FFFDd	FFFDE	4F7FF
8x		7FEF	FFF41	FFF76	FFF77	FFF78	FFF79	FFF7A	FFF7B	FFF7C	FFF7D	FFFDf	FFFE0	FFFE1	FFFE2	5FBFF
9x		AFF1F	FFF7E	FFF7F	FFF80	FFF81	FFF82	FFF83	FFF84	FFF85	FFF86	FFFE3	FFFE4	FFFE5	FFFE6	6DFFF
Ax		BFF2F	FFF87	FFF88	FFF89	FFF8A	FFF8B	FFF8C	FFF8D	FFF8E	FFF8F	FFFE7	FFFE8	FFFE9	FFFEa	7FEFF
Bx		FFF37	FFF90	FFF91	FFF92	FFF93	FFF94	FFF95	FFF96	FFF97	FFF98	FFFEb	FFFEc	FFFEd	FFFEe	8FF7F
Cx		FFF38	FFF99	FFF9A	FFF9B	FFF9C	FFF9D	FFF9E	FFF9F	FFFA0	FFFA1	FFFEf	FFFF0	FFFF1	FFFF2	9FFBF
Dx		FFF3B	FFFA2	FFFA3	FFFA4	FFFA5	FFFA6	FFFA7	FFFA8	FFFA9	FFFAa	FFFF3	FFFF4	FFFF5	FFFF6	AFFDF
Ex		FFFAb	FFFAc	FFFAd	FFFAe	FFFAf	FFFB0	FFFB1	FFFB2	FFFB3	FFFB4	FFFF7	FFFF8	FFFF9	FFFFa	BFFEF
Fx	FFF3C	FFFB5	FFFB6	FFFB7	FFFB8	FFFB9	FFFBa	FFFBb	FFFBc	FFFBd	FFFBf	FFFFb	FFFFc	FFFFd	FFFFe	CFFF7

### 7.3.2 Huffman Table for Chrominance

This table shows the number of codes for the chrominance DC table:

**Table 28: Chrominance Number of DC Codes**

Length	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Number	0h	3h	1h	1h	1h	1h	1h	1h	1h	1h	1h	1h	1h	1h	1h	0h

This table shows the assignment of generated code to the Size symbol for the chrominance DC table:

**Table 29: Chrominance DC Symbol to Code Assignment**

Code	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Symbol	0h	1h	2h	3h	4h	5h	6h	7h	8h	9h	Ah	Bh	Ch	Dh	Eh	Fh

This table shows the number of codes for the chrominance AC table:

**Table 30: Chrominance Number of AC Codes**

Length	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Number	0h	2h	1h	1h	4h	5h	Bh	5h	0h	0h	1h	1h	1h	1h	2h	Bh

This table shows the assignment of generated code to the Run/Size symbol for the chrominance AC table:

**Table 31: Chrominance AC Symbol to Code Assignment**

	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xA	xB	xC	xD	xE	xF
0x	00h	01h	02h	03h	11h	04h	05h	21h	31h	06h	12h	41h	51h	07h	61h	71h
1x	13h	22h	32h	81h	08h	14h	42h	91h	A1h	B1h	C1h	09h	23h	33h	52h	F0h
2x	15h	62h	72h	D1h	0Ah	16h	24h	34h	E1h	25h	F1h	17h	18h	19h	1Ah	26h
3x	27h	28h	29h	2Ah	35h	36h	37h	38h	39h	3Ah	43h	44h	45h	46h	47h	48h
4x	49h	4Ah	53h	54h	55h	56h	57h	58h	59h	5Ah	63h	64h	65h	66h	67h	68h
5x	69h	6Ah	73h	74h	75h	76h	77h	78h	79h	7Ah	82h	83h	84h	85h	86h	87h
6x	88h	89h	8Ah	92h	93h	94h	95h	96h	97h	98h	99h	9Ah	A2h	A3h	A4h	A5h
7x	A6h	A7h	A8h	A9h	AAh	B2h	B3h	B4h	B5h	B6h	B7h	B8h	B9h	BAh	C2h	C3h
8x	C4h	C5h	C6h	C7h	C8h	C9h	CAh	D2h	D3h	D4h	D5h	D6h	D7h	D8h	D9h	DAh
9x	E2h	E3h	E4h	E5h	E6h	E7h	E8h	E9h	EAh	F2h	F3h	F4h	F5h	F6h	F7h	F8h
Ax	F9h	FAh	0Bh	0Ch	0Dh	0Eh	1Bh	1Ch	1Dh	1Eh	2Bh	2Ch	2Dh	2Eh	3Bh	3Ch
Bx	3Dh	3Eh	4Bh	4Ch	4Dh	4Eh	5Bh	5Ch	5Dh	5Eh	6Bh	6Ch	6Dh	6Eh	7Bh	7Ch
Cx	7Dh	7Eh	8Bh	8Ch	8Dh	8Eh	9Bh	9Ch	9Dh	9Eh	ABh	ACh	ADh	AEh	BBh	BCh
Dx	BDh	BEh	CBh	CCh	CDh	CEh	DBh	DCh	DDh	DEh	EBh	ECh	EDh	EEh	FBh	FCh
Ex	FDh	FEh														

The generated Huffman table with the AC and DC codes for the chrominance symbols. After power-up, the Huffman table with index 1 is initialized with these values (all values in Hex, unused bits are set to one):

**Table 32: Default Huffman Codes for Chrominance**

	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xA	xB	xC	xD	xE	xF
0x	13FFF	17FFF	29FFF	3AFFF	4BFFF	4C7FF	5D7FF	6E5FF	6F3FF	7FDFF	FFF41	FFFBF	FFFC0	FFFC1	FFFC2	13FFF
1x		4B7FF	5DBFF	6EBFF	6F5FF	DFF3B	FFF42	FFF48	FFF49	FFF4A	FFF4B	FFFC3	FFFC4	FFFC5	FFFC6	17FFF
2x		4CFFF	6EDFF	7FEFF	FFF43	FFF46	FFF4C	FFF4D	FFF4E	FFF4F	FFF50	FFFC7	FFFC8	FFFC9	FFFCa	1BFFF
3x		5D3FF	6EFFF	AFF1F	FFF44	FFF51	FFF52	FFF53	FFF54	FFF55	FFF56	FFFCB	FFFCc	FFFCd	FFFCe	2DFFF
4x		5DFFF	6F7FF	FFF57	FFF58	FFF59	FFF5A	FFF5B	FFF5C	FFF5D	FFF5E	FFFCF	FFFD0	FFFD1	FFFD2	3EFFF
5x		5E3FF	BFF2F	FFF5F	FFF60	FFF61	FFF62	FFF63	FFF64	FFF65	FFF66	FFFD3	FFFD4	FFFD5	FFFD6	4F7FF
6x		6E7FF	EFF3D	FFF67	FFF68	FFF69	FFF6A	FFF6B	FFF6C	FFF6D	FFF6E	FFFD7	FFFD8	FFFD9	FFFDa	5FBFF
7x		6E9FF	EFF3F	FFF6F	FFF70	FFF71	FFF72	FFF73	FFF74	FFF75	FFF76	FFFDB	FFFDc	FFFDd	FFFDE	6FDFF
8x		6F1FF	FFF77	FFF78	FFF79	FFF7A	FFF7B	FFF7C	FFF7D	FFF7E	FFF7F	FFFDf	FFFE0	FFFE1	FFFE2	7FEFF
9x		6F9FF	FFF80	FFF81	FFF82	FFF83	FFF84	FFF85	FFF86	FFF87	FFF88	FFFE3	FFFE4	FFFE5	FFFE6	8FF7F
Ax		7FAFF	FFF89	FFF8A	FFF8B	FFF8C	FFF8D	FFF8E	FFF8F	FFF90	FFF91	FFFE7	FFFE8	FFFE9	FFFEa	9FFBF
Bx		7FBFF	FFF92	FFF93	FFF94	FFF95	FFF96	FFF97	FFF98	FFF99	FFF9A	FFFEb	FFFEc	FFFEd	FFFEe	AFFDF
Cx		7FCFF	FFF9B	FFF9C	FFF9D	FFF9E	FFF9F	FFFA0	FFFA1	FFFA2	FFFA3	FFFEf	FFFF0	FFFF1	FFFF2	BFFEF
Dx		FFF40	FFFA4	FFFA5	FFFA6	FFFA7	FFFA8	FFFA9	FFFAA	FFFAb	FFFAc	FFFF3	FFFF4	FFFF5	FFFF6	CFFF7
Ex		FFF45	FFFAd	FFFAE	FFFAF	FFFB0	FFFB1	FFFB2	FFFB3	FFFB4	FFFB5	FFFF7	FFFF8	FFFF9	FFFFa	DFFFB
Fx	CFF37	FFF47	FFFB6	FFFB7	FFFB8	FFFB9	FFFBa	FFFBb	FFFBc	FFFBd	FFFBf	FFFFb	FFFFc	FFFFd	FFFFe	EFFFD

## **8. Literature and Links**

### **8.1 Documents from the Internet**

The JPEG Still Picture Compression Standard. Wallace GK (ed) (PDF document)

JPEG File Interchange Format. Hamilton E (ed) (PDF document)

### **8.2 Hard Book**

JPEG STILL IMAGE DATA COMPRESSION STANDARD. Pennebaker WB, Mitchell JL (eds) Kluwer Academic Publishers Boston, Dordrecht, London, 1993

### **8.3 Internet Links**

JPEG organization

<http://www.jpeg.org/>

NASA Vision Group Publications Menu

<http://vision.arc.nasa.gov/publications/publications.html#ImageCompression>

## 9. Revision Info

Rev.	Date	Changes
1.0	20. December 2002	Initial revision
2.0	25. July 2003	Extended for JE110, JE150 and JE160
3.0	09. February 2004	4.3 Needed Resource, table actualized 4.7.1 Pixel Input Interface, description for the signal "DCT_READY" extended 7. Huffman Tables, code words shifted to MSB