Penz
VHDL

JE300, JE310, JE350 and JE360

JPEG Baseline Encoder IP-Core

Users Manual

Rev 3.0

© by Penz VHDL
Frankenstr. 16
D-55299 Nackenheim
Germany
www.penz-vhdl.de

Table of Contents

List of Figures:

List of Tables:

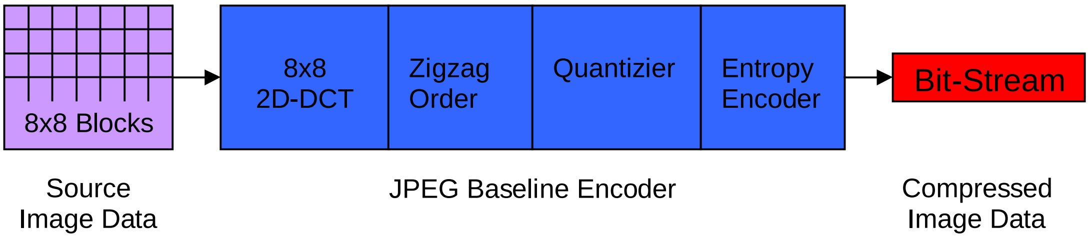
1. The JPEG Baseline Encoding Standard

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.

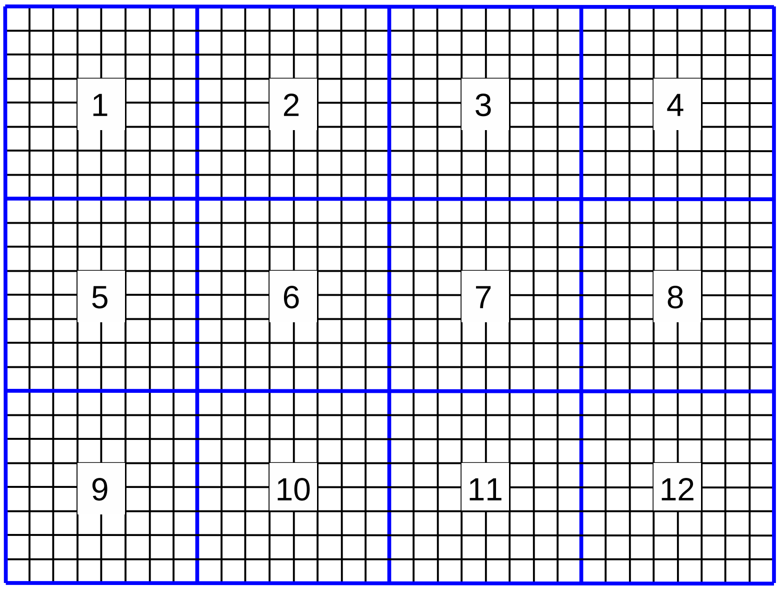
1.1 Level Shift

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

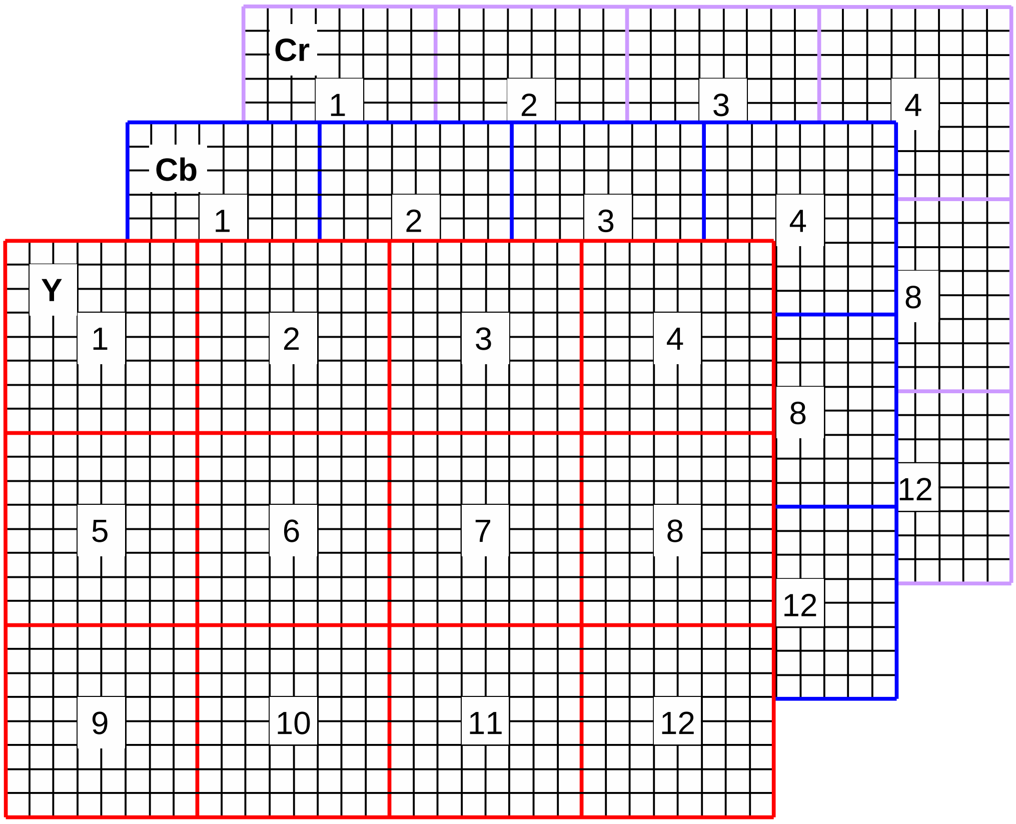


1.3 Components

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.

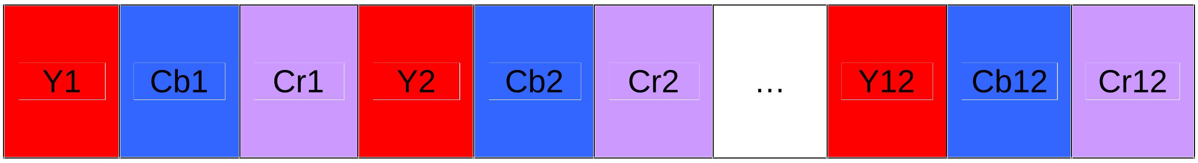
Here a color image with the components Y, Cb and Cr, no subsampling.

Figure 3: Color Image as Blocks and Components



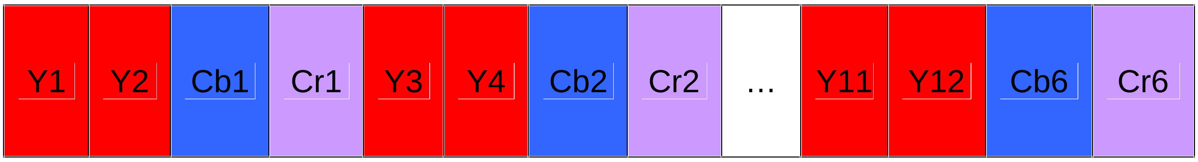
The blocks are processed in the order components and then the indexes.

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:

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

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

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

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.

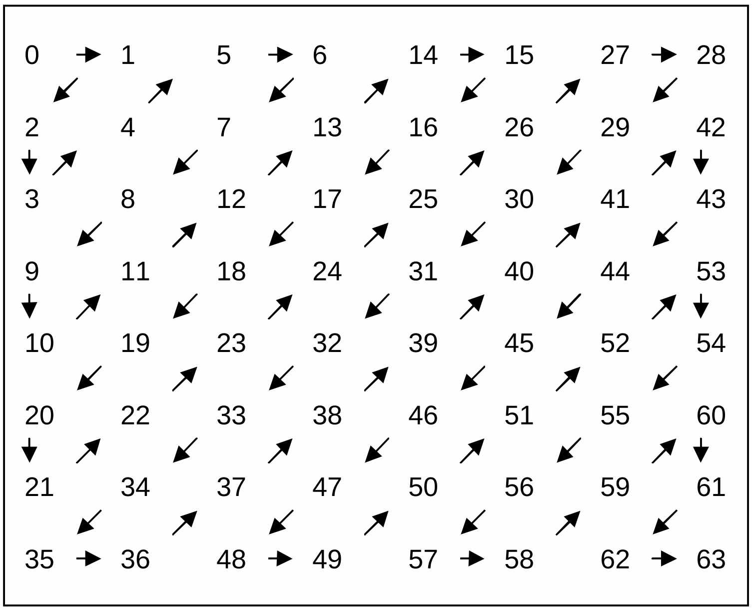
1.5 Quantization

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



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

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 **S**tart **O**f **I**mage 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 **D**efine **Q**uantization **T**able 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 **S**tart **O**f **F**rame 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 **D**efine **H**uffman **T**able 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 **S**tart **O**f **S**can 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 **E**nd **O**f **I**mage 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

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

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

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

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

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 bit / 31 bit) 16.5.

4. The JE300, JE310, JE350 and JE360 JPEG Encoder

4.1 Technical Features

- Optimized for Xilinx Spartan and Virtex FPGA
- Marker generation included
- JPEG file output
- Baseline Encoder
- Compliant with Baseline ISO/IEC 10918-1
- Block building RAM included, no external RAM needed
- Mono chrome or Color (YCbCr 4:2:2)
- Up to 4096 pixel per row
- Line by line pixel input
- Motion-JPEG capability
- 8-bit/pixel or 12-bit/pixel input (Core dependent)
- 2 Quantization tables
- 4 fixed Huffman tables (2 DC and 2 AC)
- Predefined luminance and chrominance tables
- Fully synchronous design
- Fully stall able design
- Simple CPU interface for Quantization table reprogramming
- Different clocks for encoder and CPU interface
- Single clock cycle per pixel encoding
- No pause cycles between blocks

4.2 Difference between JE300, JE310, JE350 and JE360

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	JE300	JE350
Spartan-III Virtex-II Virtex-IIP	JE310	JE360

4.3 Needed Resource

Table 4: Needed Resource for JE300

Family	Device	Pixel*	Flip Flops	4 Input Luts	Block RAM	TBufs	Clock IOBs	IOBs	Perfor-mance
Spartan-IIE	XC2S600E-6	4096	2254	4865	65	256	2	74	70 MHz
Spartan-IIE	XC2S600E-6	2048	2253	4980	33	224	2	74	70 MHz
Spartan-IIE	XC2S600E-6	1024	2252	4908	17	208	2	74	70 MHz
Spartan-IIE	XC2S600E-6	512	2251	4781	9	200	2	74	70 MHz
Virtex	XCV1000-4	1024	2252	4908	17	208	2	74	40 MHz
Virtex-E	XCV1600E-6	2048	2253	4980	33	224	2	74	70 MHz

Table 5: Needed Resource for JE310

Family	Device	Pixel*	Flip Flops	4 Input Luts	Block RAM	TBufs	Clock IOBs	IOBs	Multiplier Blocks	Perfor-mance
Spartan-III	XC3S1000-4	4096	1804	3507	17	-	2	74	16	85 MHz
Spartan-III	XC3S1000-4	2048	1800	3607	9	-	2	74	16	85 MHz
Spartan-III	XC3S1000-4	1024	1799	3548	5	-	2	74	16	85 MHz
Spartan-III	XC3S1000-4	512	1799	3403	3	-	2	74	16	85 MHz
Virtex-II	XC2V1000-4	2048	1800	3599	9	-	2	74	16	90 MHz
Virtex-IIP	XC2VP4-5	2048	1800	3599	9	-	2	74	16	120 MHz

Table 6: Needed Resource for JE350

Family	Device	Pixel*	Flip Flops	4 Input Luts	Block RAM	TBufs	Clock IOBs	IOBs	Perfor-mance
Spartan-IIE	XC2S600E-6	2048	2686	7650	49	272	2	78	55 MHz
Spartan-IIE	XC2S600E-6	1024	3007	7573	25	248	2	78	55 MHz
Spartan-IIE	XC2S600E-6	512	3004	7434	13	236	2	78	55 MHz
Spartan-IIE	XC2S600E-6	256	3018	7467	7	236	2	78	55 MHz
Virtex	XCV1000-4	512	3004	7434	13	236	2	78	45 MHz
Virtex-E	XCV1600E-6	2048	3020	7650	49	272	2	78	55 MHz

Table 7: Needed Resource for JE360

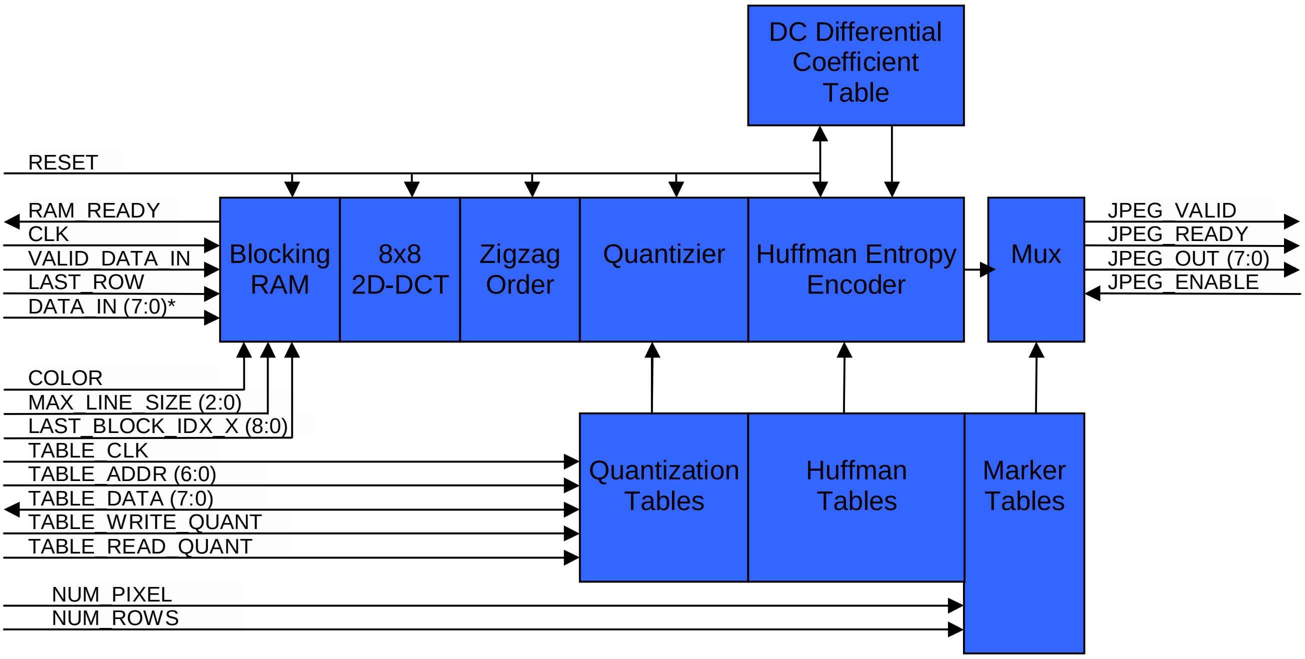
Family	Device	Pixel*	Flip Flops	4 Input Luts	Block RAM	TBufs	Clock IOBs	IOBs	Multiplier Blocks	Perfor-mance
Spartan-III	XC3S1600-4	2048	2329	4159	25	-	2	78	16	80 MHz
Spartan-III	XC3S1000-4	1024	2330	4219	13	-	2	78	16	80 MHz
Spartan-III	XC3S1000-4	512	2329	4148	7	-	2	78	16	80 MHz
Spartan-III	XC3S1000-4	256	2325	4008	4	-	2	78	16	80 MHz
Virtex-II	XC2V1000-4	2048	2329	4149	25	-	2	78	16	80 MHz
Virtex-IIP	XC2VP4-5	2048	2329	4149	25	-	2	78	16	120 MHz

*Maximal number of pixel in a line, when in mono chrome mode. When in color mode, divide the value by two.

4.4 Functional Description

The JE3xx is a stand alone 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 JE350 and JE360

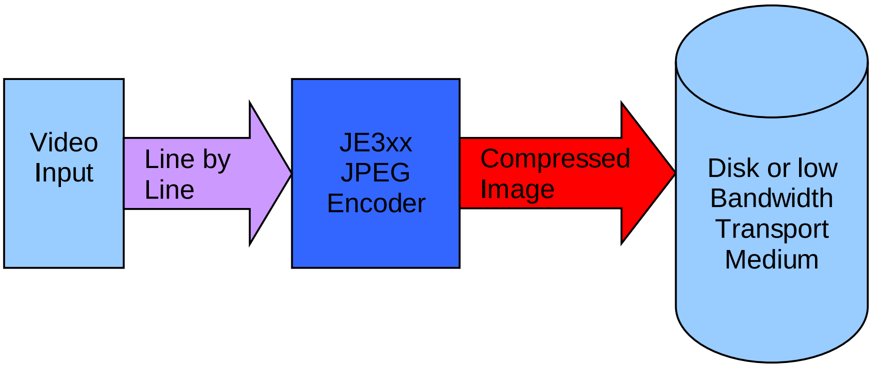
An additional interface, the “Table Programming Interface”, is used to change the Quantization tables. The “Table Programming Interface” has an own clock to minimize the glue logic to connect the core to a controller. Usually there is no need to change the tables, only if the compression rate is to change.

Each incoming line of pixels is stored in the internal RAM. When 8 lines are written, the pixels are read out in 8x8 blocks. These blocks 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 trough 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). The symbols are Huffman coded, using the code from the selected Huffman table. The resulting bit-stream of Huffman codes is divided into parts with 8 bit and embedded between all needed Markers. Finally, the JPEG file is stored in the output register.

4.5 Typical Application

In a typical application, first the video signal is digitized (if analog). The digitized pixels are written directly into the encoder. There is no need to reordering the pixel in 8x8 blocks, no external RAM is needed. The encoder outputs the bit-stream of the compressed image with all markers. The bit-stream can be transmitted over a system with limited bandwidth like a network or USB bus.

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 JE300 Core Entity

```vhdl
entity JE300 is Port (

-- Pixel Input
  CLK               : in std_logic;                       -- Main clock for encoder
  RESET             : in std_logic;                       -- Reset jpeg logic
  COLOR             : in std_logic;                       -- Color mode (0 = mono)
  VALID_DATA_IN     : in std_logic;                       -- Sample data is valid
  DATA_IN           : in std_logic_vector (7 downto 0);   -- Sample data
  LAST_ROW          : in std_logic;                       -- Last row in this image
  MAX_LINE_SIZE     : in std_logic_vector (2 downto 0);   -- 001:64, 010:128 ... 111:4096
  LAST_BLOCK_IDX_X  : in std_logic_vector (8 downto 0);   -- Number of used blocks - 1
  NUM_PIXEL         : in std_logic_vector (11 downto 0);  -- Number of pixel in a row
  NUM_ROWS          : in std_logic_vector (11 downto 0);  -- Number of row in the image
  RAM_READY         : out std_logic;                      -- Encoder is ready for new pixel

-- 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 (6 downto 0);   -- Table address
  TABLE_DATA        : inout std_logic_vector (7 downto 0); -- Table data
  TABLE_WRITE_QUANT : in std_logic;                       -- Write value in Quantization tab
  TABLE_READ_QUANT  : in std_logic                        -- Read Quantization table
);
end JE300;
```

4.6.2 JE310 Core Entity

```vhdl
entity JE310 is Port (

-- Pixel Input
  CLK               : in std_logic;                       -- Main clock for encoder
  RESET             : in std_logic;                       -- Reset jpeg logic
  COLOR             : in std_logic;                       -- Color mode (0 = mono)
  VALID_DATA_IN     : in std_logic;                       -- Sample data is valid
  DATA_IN           : in std_logic_vector (7 downto 0);   -- Sample data
  LAST_ROW          : in std_logic;                       -- Last row in this image
  MAX_LINE_SIZE     : in std_logic_vector (2 downto 0);   -- 001:64, 010:128 ... 111:4096
  LAST_BLOCK_IDX_X  : in std_logic_vector (8 downto 0);   -- Number of used blocks - 1
  NUM_PIXEL         : in std_logic_vector (11 downto 0);  -- Number of pixel in a row
  NUM_ROWS          : in std_logic_vector (11 downto 0);  -- Number of row in the image
  RAM_READY         : out std_logic;                      -- Encoder is ready for new pixel

-- 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 (6 downto 0);   -- Table address
  TABLE_DATA        : inout std_logic_vector (7 downto 0); -- Table data
  TABLE_WRITE_QUANT : in std_logic;                       -- Write value in Quantization tab
  TABLE_READ_QUANT  : in std_logic                        -- Read Quantization table
);
end JE310;
```

4.6.3 JE350 Core Entity

```
entity JE350 is Port (

-- Pixel Input
  CLK                 : in std_logic;                      -- Main clock for encoder
  RESET               : in std_logic;                      -- Reset jpeg logic
  COLOR               : in std_logic;                      -- Color mode (0 = mono)
  VALID_DATA_IN       : in std_logic;                      -- Sample data is valid
  DATA_IN             : in std_logic_vector (11 downto 0);   -- Sample data
  LAST_ROW            : in std_logic;                      -- Last row in this image
  MAX_LINE_SIZE       : in std_logic_vector (2 downto 0);    -- 001:64, 010:128 ... 111:4096
  LAST_BLOCK_IDX_X    : in std_logic_vector (8 downto 0);    -- Number of used blocks - 1
  NUM_PIXEL           : in std_logic_vector (11 downto 0);   -- Number of pixel in a row
  NUM_ROWS            : in std_logic_vector (11 downto 0);   -- Number of row in the image
  RAM_READY           : out std_logic;                     -- Encoder is ready for new pixel

-- 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 (6 downto 0);    -- Table address
  TABLE_DATA          : inout std_logic_vector (7 downto 0); -- Table data
  TABLE_WRITE_QUANT   : in std_logic;                      -- Write value in Quantization tab
  TABLE_READ_QUANT    : in std_logic                       -- Read Quantization table
);
end JE350;
```

4.6.4 JE360 Core Entity

```
entity JE360 is Port (

-- Pixel Input
  CLK                 : in std_logic;                      -- Main clock for encoder
  RESET               : in std_logic;                      -- Reset jpeg logic
  COLOR               : in std_logic;                      -- Color mode (0 = mono)
  VALID_DATA_IN       : in std_logic;                      -- Sample data is valid
  DATA_IN             : in std_logic_vector (11 downto 0);   -- Sample data
  LAST_ROW            : in std_logic;                      -- Last row in this image
  MAX_LINE_SIZE       : in std_logic_vector (2 downto 0);    -- 001:64, 010:128 ... 111:4096
  LAST_BLOCK_IDX_X    : in std_logic_vector (8 downto 0);    -- Number of used blocks - 1
  NUM_PIXEL           : in std_logic_vector (11 downto 0);   -- Number of pixel in a row
  NUM_ROWS            : in std_logic_vector (11 downto 0);   -- Number of row in the image
  RAM_READY           : out std_logic;                     -- Encoder is ready for new pixel

-- 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 (6 downto 0);    -- Table address
  TABLE_DATA          : inout std_logic_vector (7 downto 0); -- Table data
  TABLE_WRITE_QUANT   : in std_logic;                      -- Write value in Quantization tab
  TABLE_READ_QUANT    : in std_logic                       -- Read Quantization table
);
end JE360;
```

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
Pixel Input Interface		
CLK	In	Encoder clock
RESET	In	Reset encoder state machines
VALID_DATA_IN	In	Pixel is valid
DATA_IN (7 : 0)*	In	Pixel input
LAST_ROW	In	Last row in this image
COLOR	In	Mode 0 = mono chrome, 1 = color
MAX_LINE_SIZE (2:0)	In	Number of samples per row
LAST_BLOCK_IDX_X (8:0)	In	Index of the last 8 pixel-block in a row
NUM_PIXEL (11:0)	In	Number of pixel in a row
NUM_ROWS (11:0)	In	Number of rows in the image
RAM_READY	Out	RAM is ready for new pixel
Compressed Data Output Interface		
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
Tables Programming Interface		
TABLE_CLK	In	Clock for table reprogramming
TABLE_ADDR (6:0)	In	Table select and addressing
TABLE_DATA (7:0)	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

* Vector size (11:0) on JE350 and JE360

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
```

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

DATA_IN

```
DATA_IN            : in std_logic_vector (7 downto 0);    -- Sample data for JE300 and JE310
DATA_IN            : in std_logic_vector (11 downto 0);   -- Sample data for JE350 and JE360
```

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 "RAM_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.

RAM_READY

```
RAM_READY          : out std_logic;                       -- RAM 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 usually this signal will not be deasserted. If there are less than 80 pixels in the rows then this signal is deasserted, even if "JPEG_ENABLE" stays always asserted. This is so cause as first; the marker must be shifted out, before the compressed bit stream can be shifted out. The second reason for deasserting is, when the compression rate is lower as three on the JE300 and JE310 or lower as four on the JE350 and JE360.

LAST_ROW

```
LAST_ROW           : in std_logic;                        -- Last row in this image
```

This signal defines the number of lines in the image. Assert this signal while the last pixel of the last line is transferred. The number of lines must be multiple of 8. If the image height is not a multiple of 8, then repeat the last line until the right number is reached.

COLOR

```
COLOR              : in std_logic;                         -- Color mode (0 = mono)
```

This signal defines the kind of the image, a zero means a mono chrome image, a one a colored image with chrominance sub sampling (YCbCr 4:2:2). This signal must not change while a image is compressed. When only mono chrome images are compressed, assign a static zero to allow the mapper removing the unused logic.

MAX_LINE_SIZE

```
MAX_LINE_SIZE      : in std_logic_vector (2 downto 0);     -- Number of samples per row
```

This signal determinate the number of samples per row and determinate the number of block RAM, used for the pixel blocking. Assign a static value, to allow the mapper to remove the unused block RAM. An un-static value will result in 64 blocks of RAM. Allowed values are only there, which in the following table:

Table 9: Number of block RAM and pixel relation

Value of MAX_LINE_SIZE	Number* of Block RAM JE300	Number* of Block RAM JE310	Number* of Block RAM JE350	Number* of Block RAM JE360	Number of mono chrome pixel	Number of color pixel
0	--**	--**	--**	--**	--	--
1	1	--**	--**	--**	64	32
2	2	--**	3	--**	128	64
3	4	1	6	2	256	128
4	8	2	12	3	512	256
5	16	4	24	6	1024	512
6	32	8	48	12	2048	1024
7	64	16	96	24	4096	2048

* The core needs one additional Block RAM for the Quantization table.
** These values are not allowed for this core.

LAST_BLOCK_IDX_X

```
LAST_BLOCK_IDX_X   : in std_logic_vector (8 downto 0);     -- Number of used blocks - 1
```

This signal determinate the number of samples in a row. The value is right-shifted by 3 and decrement by 1. The number of samples must be a multiple of 8 in mono chrome mode and a multiple of 16 in color mode. If the image width is not a multiple of 8 (or 16), then repeat the last pixel until the desired number is reached. For internal reasons, the minimum size is 32 samples.

Example for calculation the right value:

768 mono chrome pixel: 768 / 8 - 1 = 95dez = 5Fh
768 color pixel: 2 * 768 / 8 - 1 = 191dez = BFh

NUM_PIXEL

```
NUM_PIXEL          : in std_logic_vector (11 downto 0);    -- Number of pixel in a row
```

This signal determinate the number of pixel in a row. These information is used, only for the marker generation and don’t affect the pixel Input interface. The value doesn’t have to be a multiple of 8.

NUM_ROWS

```
NUM_ROWS           : in std_logic_vector (11 downto 0);    -- Number of rows in the image
```

This signal determinate the number of rows in the image. These information is used, only for the marker generation and don’t affect the pixel Input interface. The value doesn’t have to be a multiple of 8.

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 and stays unchanged until “JPEG_ENABLE” is asserted. The data stream includes all the markers, that be needed to have a valid JPEG file.

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 (6 downto 0);     -- Table address
```

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

A5 ... A0 select one of the 64 elements from the table.
A6 select between the Luminance (‘0’) and Chrominance (’1’) table.

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

TABLE_DATA

```
  TABLE_DATA         : inout std_logic_vector (7 downto 0);  -- Table data
```

Bidirectional data bus to writing into the tables and read there out. 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. Because internal implementation reasons, there is not possible to change only some values. If at least on value is written (even if with the same value), all 64 values must be written. In color mode, even both tables.

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.

5. Using with Xilinx ISE

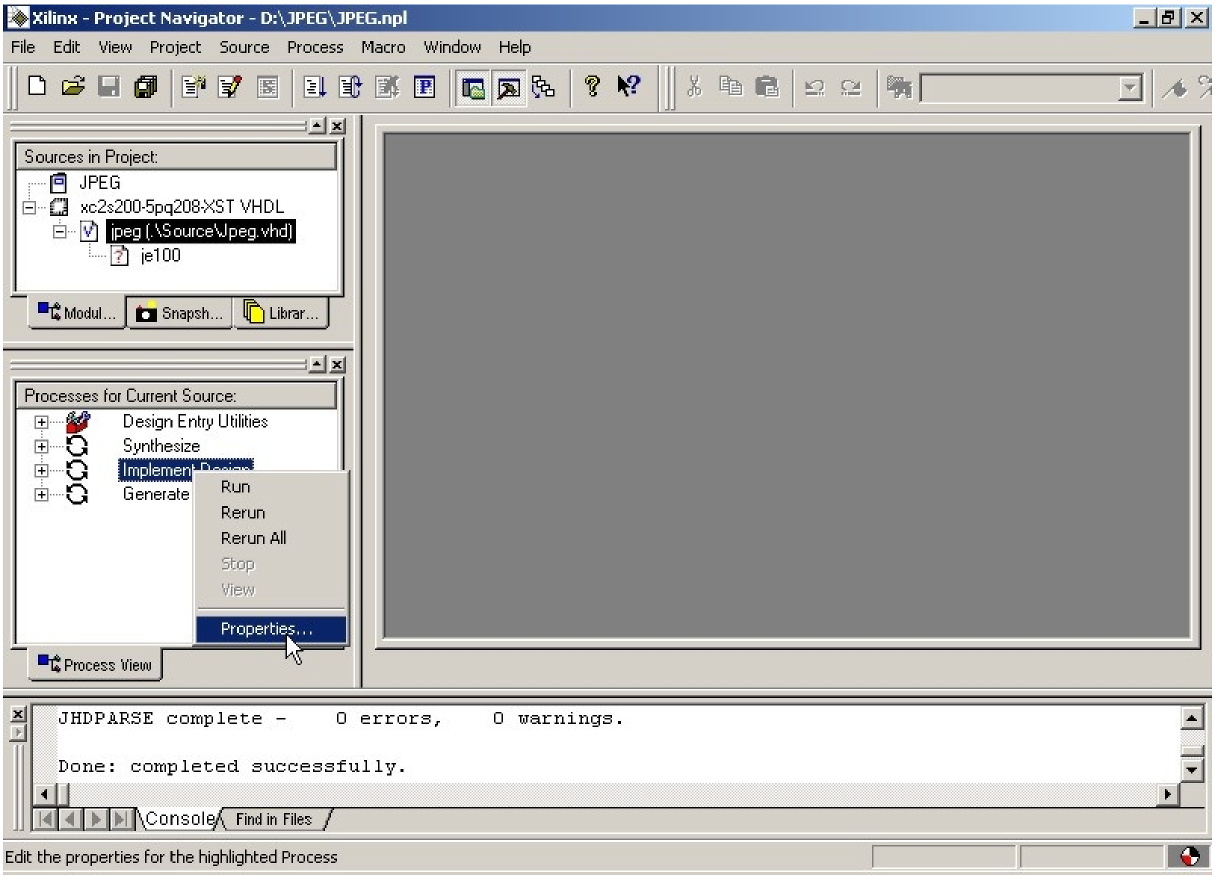
Including the JE3xx into the VHDL project is easy and need only three steps:

- set the “Macro Search Path” to the directory, that contain the NGC file
- copy the JE3xx Component declaration in the source file
- copy the JE3xx 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



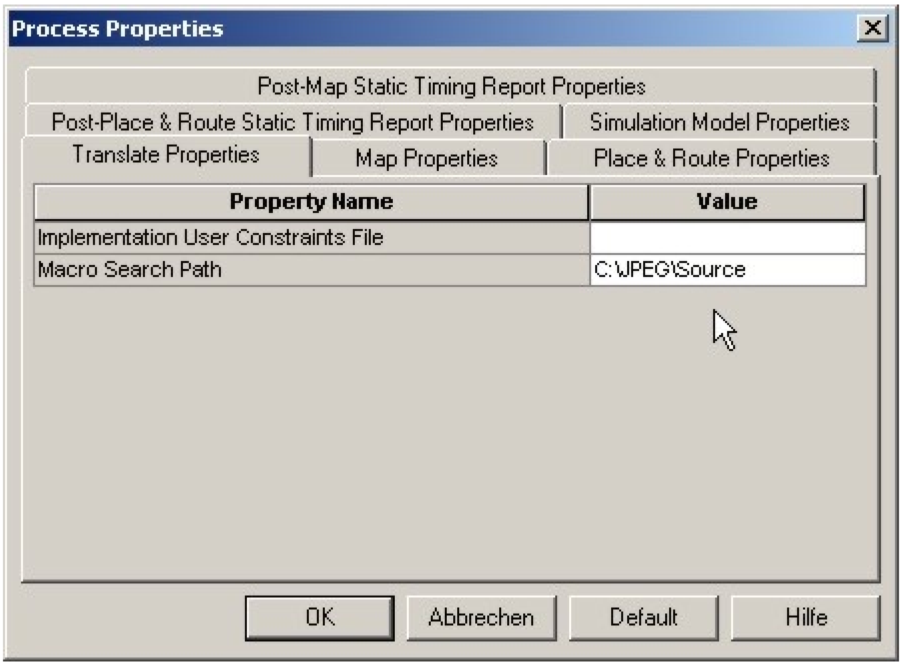
```
JHDPARSE complete -    0 errors,    0 warnings.

Done: completed successfully.
```

Edit the properties for the highlighted Process

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



Property Name	Value
Implementation User Constraints File	
Macro Search Path	C:\JPEG\Source

Set the “Macro Search Path” in the Process Properties to the directory where the NGC file (JE3xx.NGC) is inside.

The ISE shows the JE3xx module with a red question mark as absent. This can be ignored.

5.2 Component Declarations

Copy the JE3xx Component declaration into the architecture body:

5.2.1 JE300 Component Declaration

```vhdl
entity MyEntity is Port (

  .
  .
  .

);
end MyEntity;

architecture RTL of MyEntity is

  .
  .
  .

component JE300 is Port (

-- Pixel Input

  CLK               : in std_logic;                        -- Main clock for encoder
  RESET             : in std_logic;                        -- Reset jpeg logic
  COLOR             : in std_logic;                        -- Color mode (0 = mono)
  VALID_DATA_IN     : in std_logic;                        -- Sample data is valid
  DATA_IN           : in std_logic_vector (7 downto 0);    -- Sample data
  LAST_ROW          : in std_logic;                        -- Last row in this image
  MAX_LINE_SIZE     : in std_logic_vector (2 downto 0);    -- 001:64, 010:128 ... 111:4096
  LAST_BLOCK_IDX_X  : in std_logic_vector (8 downto 0);    -- Number of used blocks - 1
  NUM_PIXEL         : in std_logic_vector (11 downto 0);   -- Number of pixel in a row
  NUM_ROWS          : in std_logic_vector (11 downto 0);   -- Number of row in the image
  RAM_READY         : out std_logic;                       -- Encoder is ready for new pixel

-- 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 (6 downto 0);    -- Table address
  TABLE_DATA        : inout std_logic_vector (7 downto 0); -- Table data
  TABLE_WRITE_QUANT : in std_logic;                        -- Write value in Quantization tab
  TABLE_READ_QUANT  : in std_logic                         -- Read Quantization table
);
end component;

  .
  .
  .

begin

  .
  .
  .

end RTL;
```

5.2.2 JE310 Component Declaration

```
entity MyEntity is Port (

  .
  .
  .

);
end MyEntity;

architecture RTL of MyEntity is

  .
  .
  .

component JE310 is Port (

-- Pixel Input

  CLK                  : in std_logic;                       -- Main clock for encoder
  RESET                : in std_logic;                       -- Reset jpeg logic
  COLOR                : in std_logic;                       -- Color mode (0 = mono)
  VALID_DATA_IN        : in std_logic;                       -- Sample data is valid
  DATA_IN              : in std_logic_vector (7 downto 0);   -- Sample data
  LAST_ROW             : in std_logic;                       -- Last row in this image
  MAX_LINE_SIZE        : in std_logic_vector (2 downto 0);   -- 001:64, 010:128 ... 111:4096
  LAST_BLOCK_IDX_X     : in std_logic_vector (8 downto 0);   -- Number of used blocks - 1
  NUM_PIXEL            : in std_logic_vector (11 downto 0);  -- Number of pixel in a row
  NUM_ROWS             : in std_logic_vector (11 downto 0);  -- Number of row in the image
  RAM_READY            : out std_logic;                      -- Encoder is ready for new pixel

-- 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 (6 downto 0);   -- Table address
  TABLE_DATA           : inout std_logic_vector (7 downto 0); -- Table data
  TABLE_WRITE_QUANT    : in std_logic;                       -- Write value in Quantization tab
  TABLE_READ_QUANT     : in std_logic                        -- Read Quantization table
);
end component;

  .
  .
  .

begin

  .
  .
  .

end RTL;
```

5.2.3 JE350 Component Declaration

```vhdl
entity MyEntity is Port (

  .
  .
  .

);
end MyEntity;

architecture RTL of MyEntity is

  .
  .
  .

component JE350 is Port (

-- Pixel Input

  CLK                : in std_logic;                        -- Main clock for encoder
  RESET              : in std_logic;                        -- Reset jpeg logic
  COLOR              : in std_logic;                        -- Color mode (0 = mono)
  VALID_DATA_IN      : in std_logic;                        -- Sample data is valid
  DATA_IN            : in std_logic_vector (11 downto 0);   -- Sample data
  LAST_ROW           : in std_logic;                        -- Last row in this image
  MAX_LINE_SIZE      : in std_logic_vector (2 downto 0);    -- 001:64, 010:128 ... 111:4096
  LAST_BLOCK_IDX_X   : in std_logic_vector (8 downto 0);    -- Number of used blocks - 1
  NUM_PIXEL          : in std_logic_vector (11 downto 0);   -- Number of pixel in a row
  NUM_ROWS           : in std_logic_vector (11 downto 0);   -- Number of row in the image
  RAM_READY          : out std_logic;                       -- Encoder is ready for new pixel

-- 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 (6 downto 0);    -- Table address
  TABLE_DATA         : inout std_logic_vector (7 downto 0); -- Table data
  TABLE_WRITE_QUANT  : in std_logic;                        -- Write value in Quantization tab
  TABLE_READ_QUANT   : in std_logic                         -- Read Quantization table
);
end component;

  .
  .
  .

begin

  .
  .
  .

end RTL;
```

5.2.4 JE360 Component Declaration

```
entity MyEntity is Port (

  .
  .
  .

);
end MyEntity;

architecture RTL of MyEntity is

  .
  .
  .

component JE300 is Port (

-- Pixel Input

  CLK                : in std_logic;                        -- Main clock for encoder
  RESET              : in std_logic;                        -- Reset jpeg logic
  COLOR              : in std_logic;                        -- Color mode (0 = mono)
  VALID_DATA_IN      : in std_logic;                        -- Sample data is valid
  DATA_IN            : in std_logic_vector (11 downto 0);   -- Sample data
  LAST_ROW           : in std_logic;                        -- Last row in this image
  MAX_LINE_SIZE      : in std_logic_vector (2 downto 0);    -- 001:64, 010:128 ... 111:4096
  LAST_BLOCK_IDX_X   : in std_logic_vector (8 downto 0);    -- Number of used blocks - 1
  NUM_PIXEL          : in std_logic_vector (11 downto 0);   -- Number of pixel in a row
  NUM_ROWS           : in std_logic_vector (11 downto 0);   -- Number of row in the image
  RAM_READY          : out std_logic;                       -- Encoder is ready for new pixel

-- 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 (6 downto 0);    -- Table address
  TABLE_DATA         : inout std_logic_vector (7 downto 0); -- Table data
  TABLE_WRITE_QUANT  : in std_logic;                        -- Write value in Quantization tab
  TABLE_READ_QUANT   : in std_logic                         -- Read Quantization table
);
end component;

  .
  .
  .

begin

  .
  .
  .

end RTL;
```

5.3 Component Instantiations

Copy the JE3xx Component instantiation into the RTL and assign your signals to the entity signals.

5.3.1 JE300 Component Instantiation

```
entity MyEntity is Port (

  .
  .
  .

);
end MyEntity;

architecture RTL of MyEntity is

  .
  .
  .

begin

  .
  .
  .

UJE300: JE300 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_ROW           => ... ,                              -- Last row in this image
  COLOR              => ... ,                              -- Color mode (0 = mono)
  MAX_LINE_SIZE      => ... ,                              -- 001:64, 010:128 ... 111:4096
  LAST_BLOCK_IDX_X   => ... ,                              -- Number of used blocks - 1
  NUM_PIXEL          => ... ,                              -- Number of pixel in a row
  NUM_ROWS           => ... ,                              -- Number of row in the image
  RAM_READY          => ... ,                              -- Encoder is ready for new pixel

-- 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
);

  .
  .
  .

end RTL;
```

5.3.2 JE310 Component Instantiation

```
entity MyEntity is Port (

  .
  .
  .

);
end MyEntity;

architecture RTL of MyEntity is

  .
  .
  .

begin

  .
  .
  .

UJE310: JE310 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_ROW           => ... ,                            -- Last row in this image
  COLOR              => ... ,                            -- Color mode (0 = mono)
  MAX_LINE_SIZE      => ... ,                            -- 001:64, 010:128 ... 111:4096
  LAST_BLOCK_IDX_X   => ... ,                            -- Number of used blocks - 1
  NUM_PIXEL          => ... ,                            -- Number of pixel in a row
  NUM_ROWS           => ... ,                            -- Number of row in the image
  RAM_READY          => ... ,                            -- Encoder is ready for new pixel

-- 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
);

  .
  .
  .

end RTL;
```

5.3.3 JE350 Component Instantiation

```
entity MyEntity is Port (

  .
  .
  .

);
end MyEntity;

architecture RTL of MyEntity is

  .
  .
  .

begin

  .
  .
  .

UJE350: JE350 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_ROW            => ... ,                          -- Last row in this image
  COLOR               => ... ,                          -- Color mode (0 = mono)
  MAX_LINE_SIZE       => ... ,                          -- 001:64, 010:128 ... 111:4096
  LAST_BLOCK_IDX_X    => ... ,                          -- Number of used blocks - 1
  NUM_PIXEL           => ... ,                          -- Number of pixel in a row
  NUM_ROWS            => ... ,                          -- Number of row in the image
  RAM_READY           => ... ,                          -- Encoder is ready for new pixel

-- 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
);

  .
  .
  .

end RTL;
```

5.3.4 JE360 Component Instantiation

```
entity MyEntity is Port (

  .
  .
  .

);
end MyEntity;

architecture RTL of MyEntity is

  .
  .
  .

begin

  .
  .
  .

UJE360: JE360 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_ROW            => ... ,                           -- Last row in this image
  COLOR               => ... ,                           -- Color mode (0 = mono)
  MAX_LINE_SIZE       => ... ,                           -- 001:64, 010:128 ... 111:4096
  LAST_BLOCK_IDX_X    => ... ,                           -- Number of used blocks - 1
  NUM_PIXEL           => ... ,                           -- Number of pixel in a row
  NUM_ROWS            => ... ,                           -- Number of row in the image
  RAM_READY           => ... ,                           -- Encoder is ready for new pixel

-- 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
);

  .
  .
  .

end RTL;
```

6. Quantization Tables

The Quantization table contains the divisors for the coefficient quantization. There are two tables implemented, 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:

A6	A5 – A0
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 10: 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 JE3xx 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.

D7 – D0
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 11: 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 12: 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 $Q >= 50$

$XS_i = XD_i * S$

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.

7.1 Default Huffman Tables for JE300 and JE310

7.1.1 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														

7.1.2 Huffman Table for Chrominance

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

Table 17: 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 18: 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 19: 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 20: 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														

7.2 Default Huffman Tables for JE350 and JE360

7.2.1 Huffman Table for Luminance

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

Table 21: 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 22: 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 23: 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 24: 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	AEh	BBh	BCh
Dx	BDh	BEh	CBh	CCh	CDh	CEh	DBh	DCh	DDh	DEh	EBh	ECh	EDh	EEh	FBh	FCh
Ex	FDh	FEh														

7.2.2 Huffman Table for Chrominance

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

Table 25: 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 26: 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 27: 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	BFh

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

Table 28: 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														

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	08. September 2003	Initial revision
1.1	20. December 2003	Chapter 4.3 Needed Resource: Number of IOBs corrected, for JE300 and HE310. Tables added for JE350 and JE360. Chapter 4.7.1 Pixel Input Interface: Description for the signal “RAM_READY” extended.
3.0	18. February 2004	Chapter 4.3 Needed Resource, table actualized