AN1443
APPLICATION NOTE

ST100 VIDEO LIBRARY
Baseline JPEG Encoding and Decoding on the ST120DSP

By Maurizio Colombo

ABSTRACT

This application note shows the results of porting the JPEG application for a SW implementation on ST120DSP. JPEG standard is very generic and includes different techniques. The *Baseline method* is by far the most widely implemented JPEG method and has thus been chosen for this study. Baseline JPEG includes DCT, Quantization and Huffman encoding from the encoding side, IDCT, Inverse Quantization and Huffman decoding from the decoding side.

CONCLUSION

The number of clock cycles required to encode one 8x8 block of pixels is 1895, while its decoding requires 2467 cycles. It means that the encoding and the decoding of one QCIF image (176x144) takes respectively 1.12 MCycles and 1.465 MCycles. This analysis is based on *very good quality* encoding/ decoding, so this number of MCycles can be reduced accordingly to the quality and compression ratio required for the specific application.

1 - OVERVIEW ON JPEG ENCODING

The processing steps of a Baseline JPEG encoder and decoder are respectively shown in Figures 1 and 2. These are basically a subset of the MPEG functions.

Figure 1 : JPEG Encoder

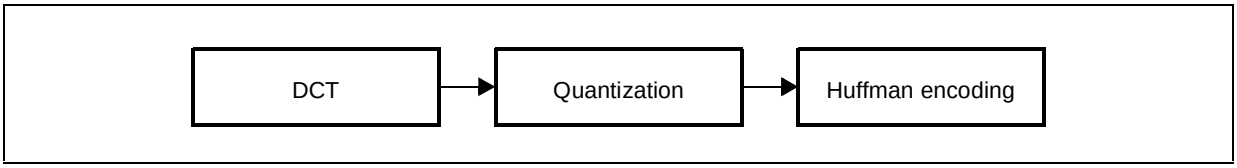
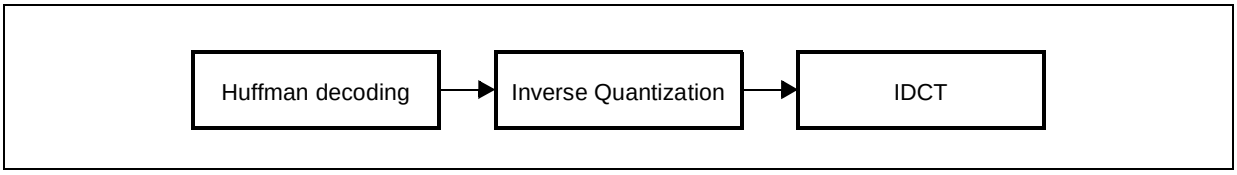


Figure 2 : JPEG Decoder



The *Discrete Cosine Transform* (DCT) is the same algorithm as the one defined for MPEG. This is followed by Quantization, where each coefficient is basically divided by a corresponding weight in a matrix. Then, coefficients are ordered in zigzag scan and Run-length coding is applied (as for MPEG, zero sequences are replaced with their length and the first non-zero coefficient). Finally, Huffman encoding removes the statistical redundancy and the result is packed into a bitstream. The decoding process is represented exactly by the same inverse sequential steps.

The quality of the compressed image is depending on the quantization matrix used. In this analysis a quantization matrix for *very good* quality has been chosen.

The source C code used for this implementation has been downloaded from the site ftp.uu.net/graphics/jpeg [1].

In the next paragraphs the performances of each one of the above functions are described.

2 - THE ENCODER

2.1 - Wang's DCT

The DCT algorithm is the same as the one developed for MPEG. Visual quality tests show that there are no visible differences between this algorithm and the one proposed in the source code (jfdctint.c).

Zigzag scan is directly performed in column DCT to allow run-length calculations into quantization (see next paragraph).

The results of the profiling are shown in Table 1.

Table 1 : Number of clock cycles and code size for one 8x8 block DCT

Function	Cycles	Code Size
DCT	314	2692

Regarding data memory usage, DCT needs 128 bytes for the block (however, to store intermediate results, between row and column processing, 128 bytes of stack are needed). For constants 72 bytes are needed.

2.2 - Quantization

Quantization consists in dividing each coefficient by the corresponding weight in the quantization table. In this step another operation is performed, to prepare the information for run-length coding.

Table 2 shows the performances and code size. Quantization uses 136 bytes for the block (8 bytes are for run-length information) and 256 bytes for the quantization matrices (one for luminance, one for chrominance).

Table 2 : Number of clock cycles for Quantization+run-length of one 8x8 block

Function	Cycles	Code Size
Quantization	177	664

2.3 - Huffman Encoding

It consists in coding the run/length couples into Huffman variable length codes. The results, obtained from a simulation on the test image provided with the sources are shown in Table 3. More intensive simulations, executed on real high quality photos[2] are shown in Table 4.

About memory usage, VLC needs four look-up tables of 1280 bytes each (5120 bytes total) and a buffer to store the output bitstream (in the implementation a size of 2 KBytes is used but it depends on the DMA policy used).

Table 3 : Number of cycles and code size for one 8x8 block encoding (test image)

Function	Cycles	Code Size
Huffman encoding	682	1452

Table 4 : Number of cycles for one 8x8 block encoding (simulations)

Image	Cycles
Cfive	886
Cucorn	1226
Icestorm	2031
Seagull	992
Palm	1331
Duneprint	1847
Glasses	1043
Flags	992
Average	1404

2.4 - The Complete Application

The blocks described in the previous paragraphs represent the core of the application. Another part that is worth to be considered is the routine writing headers at the beginning of the image. This part takes 24401 cycles and is executed only once. The rest of the program contains some additional parts which enable to read different image formats (in this case only ppm is considered, but this is not a constraint) to convert chroma formats (RGB to YUV and vice versa...), to write the bitstream into a physical file, to set up global configuration variables. All this has not been optimized for speed, but it has been compiled with the"Optimization for size" option in GP16 (16-bit instruction set) mode to guarantee code density. Depending on the embedded application targeted this can be customized: a MIPS-intensive application like a Printer one takes care only of performances, whereas a Wireless application targets low power consumption and small memory requirements without looking at speed (a typical application is encoding/ decoding of static images taken from the web on a 3G cellular phone).

The following Table 5 summarizes the program/data memory requirements for the whole application.

Table 5 : Memory requirements for the whole application

Section	Size
Startup	256
Code	31234
Libraries	32450
Read-only data	8880
Data	412
Zero init data	9100
Stack	3072
Heap	application-dependent

3 - THE DECODER

3.1 - Huffman Decoding

This function reads data from the input bitstream and decodes the DCT quantized coefficients (they are written in raster order). Table 7 represents the results obtained by a simulation on the test image provided with the sources. The results of intensive simulations made on real photos[2] are shown in Table 6.

Table 6 : Cycles for one 8x8 block (real photos simulations)

Image	Cycles
Cfive	1375
Cucorn	1902
Icestorm	3086
Seagull	438
Palm	2121
Duneprint	2806
Glasses	1619
Flags	1565
Average	1864

Table 7 : Cycles and code size for one 8x8 block (test image)

Function	Cycles	Code SIze
Huffman decoding	1073	2412

About data memory requirements, 4KB are needed for the internal buffer (It is customizable) and four look-up tables of 1564 bytes each. It means 10352 bytes. 128 bytes are needed to write the output block to be de quantized.

3.2 - Inverse Quantization

This consists simply in multiplying each coefficient by its corresponding weight in the quantization matrix. Each coefficient is represented with 12-bit precision, while the weights are represented by 8 bits. So, all multiplications are 16x16. The results are shown in Table 8.

As data memory128 bytes are needed for the block and 128 bytes for the two quantization tables.

Table 8 : Cycles for one 8x8 block inverse quantization

Function	Cycles	Code Size
Inverse Quant	71	148

3.3 - Wang's Inverse DCT

The same algorithm used for MPEG2 has been implemented here. This includes IDCT, clipping and add128. The input data are read in raster order, because zigzag is performed directly into Huffman decoding. The results are reported in Table 9. The required data memory is 128 bytes for the block and 30 bytes for the constants.

Table 9 : Clock cycles and code size for one 8x8 block IDCT

Function	Cycles	Code Size
IDCT	532	1108

3.4 - The Complete Application

As for the encoder, the complete application includes other sections which are tools around the application core, represented by IDCT, IQ and Huffman decoding. Again, these modules have been compiled with the "Optimization for size" (-OS) option in GP16 mode and thus do not represent the maximum performance achievable by the ST120 DSP. A summary of memory requirements for the whole application is reported in the following Table 10.

Table 10 : Memory requirements for the whole application

Section	Size
Startup	256
Code	31234
Libraries	31890
Read-only data	7552
Data	328
Zero init data	12236
Stack	3072
Heap	application-dependent

4 - CONCLUSIONS

In the following Table 11 is reported a summary of the results presented in this document. For each function are shown the number of clock cycles for one 8x8 block, the code size and the data memory usage.

Table 11 : Summary

Function	Cycles	Code size	Data Memory
DCT	314	2692	200
Quantization	177	664	392
Huffman encoding	1404	1452	7168
IDCT	532	1108	158
Inv quant	71	148	256
Huffman decoding	1864	2412	10352

REFERENCES

[1] Thomas G. Lane, *Independent JPEG Group's Software*, ftp.uu.net/graphics/jpeg

[2] Kodak Digital Images Offering, *www.kodak.com/digitalImaging/samples/*