A TINY JPEG DECOMPRESSOR

José Porfírio A. de Carvalho
Roberto Silva Cantanhêde
Prof. Dr. Ricardo P. Jacobi

Universidade de Brasília
Departamento de Ciência da Computação

ABSTRACT

This paper presents an implementation of the baseline, DCT-based JPEG algorithm for grayscale image decompression using a XESS-XS40 kit that has a 32k RAM and a Xilinx XC4010. The memory of the kit is used in run-time to store the status of each partial configuration and the intermediate results as well. VHDL is used with Xilinx Foundation software and some support code written in C for parallel port configuration and data communication.

1. INTRODUCTION

The original motivation for this work was to implement a JPEG decompressor that could be used in the Electronic Voting Machine 96 (EVM96) used by the Brazilian government. The EVM96 uses flash cards to store the system and user programs, the candidates data and respective pictures and has a LCD screen for output and a telephone keyboard-like for user input, its processor is a 80386SX that is quite slow for running the JPEG algorithm and put reconfigurable hardware in the expansion slots to implement the image and cypher routines in order to speed up the voting process and reuse the old equipment in future elections. Section 2 of this paper gives an overview of the JPEG format and the decompressing; Section 3 covers the DCT; Section 4 is about the implementation and the reconfiguration method and Section 5 gives some of our conclusions.

2. JPEG

The JPEG algorithm can be implemented in many ways varying the coding scheme, that can be Huffman coding or arithmetic coding; it can be progressive, hierarchical or baseline. The images may have three components (color image) or only one for grayscale images. With all those possibilities, the JFIF (JPEG File Interchange Format) file format was chosen because of its simplicity and because it states that the image will use the baseline DCT (Discrete Cosine Transform) and Huffman coding.

This implementation will focus only on the grayscale image decompressor implemented. The image has two Huffman tables, one for high frequency components (DC) and one for others (AC). Those components are spread in an 8x8 matrix of elements where the first is the DC component. Along with the Huffman table there is a quantization table that is used during the compression to maximize the number of zeros in the compressed matrix by rounding the coefficients towards zero. These tables are separated in blocks in the JPEG file and they are identified by specific bytes at the beginning of the block. There is also a control block that states the version of the file, the size, thumbnails, number of tables and other.

The decoding process consists basically of: parsing the file, create Huffman tables, create quantization tables, decode a 8x8 block, dequantize the block, dezigzag the block and apply the inverse DCT; then decode next 8x8 block until there are no more blocks. The dezigzag (figure 1) step is meant to reorder the matrix restoring the original configuration given by the forward DCT during compression; the purpose of zigzagging is increase the number of adjacent zeros. Though the reason says that the most frequent value should receive the smallest code and that unused codes should stay out of the tables, it is not implemented in some popular coders we used; rare tools offered the option to optimize the Huffman tables.

The grayscale decoding is somewhat easier than the coding because there is no need to worry about the size of the image not being a multiple of 8 and the respective padding, adjusting the quantizers or conversion between color schemes. In the implementation it is still possible to retrieve the size of the image, if desired.

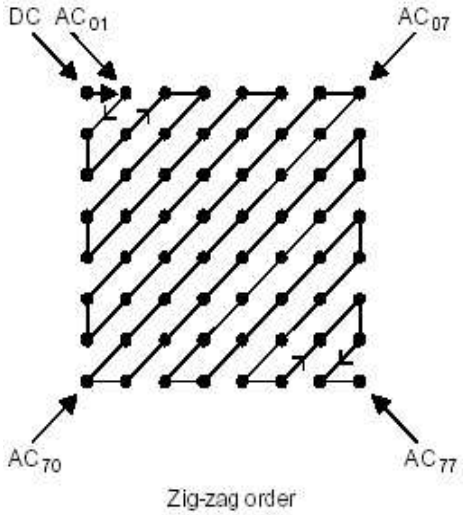

Figure 1 Zigzag order

3. iDCT

The Discrete Cosine transform is used to convert the image data from the space domain to the frequency domain. With the results of this transformation, it is possible to isolate the higher frequency components (image details) from the lower frequency components (image standard). The JPEG process reduces, based on the values of the quantizers, the higher frequency components to minimize the information necessary to represent a lower detailed image from the original. The transformation is executed in a bi dimensional form in blocks of 8x8 data values. In decompression, the inverse transform is used to re-convert the data to the space domain.

4. IMPLEMENTATION

We first coded the whole thing in C++ then rewrote it in behavioral VHDL. We made an attempt to implement the algorithm using the Foundation software but the result was just too big. Then we begun splitting the code until we had the following partition: a memory uploader, a file parser, a set up Huffman code, Huffman decoder, dequantization and dezigzagging, inverse DCT and a memory downloader.
All modules used the 7-segment display present on the board to report their status and error conditions. Though we could write a simple program to check if the part had accomplished its task we preferred to switch the configurations by hand so that debugging and the final version could be ready sooner.
The RAM is addressed using inverted addresses when writing control variables so that a big image will be decoded as much as possible before overwriting the control data. A memory lock could be provided, but to implement it, it required more area on the already-small-for-the-problem FPGA.
The first and last modules are the same. Its responsibility is to load the image file to the begin of the RAM and after the process is complete, to read back the decoded image. The address generation is external, a C program communicating via parallel port, starting at address zero for the downloading the encoded image and based on the size of the file for retrieving the decoded image.
The file parser checks the type and version of the file before decoding and signals if something is wrong. It also locates the quantization tables and the Huffman tables.
The Huffman setup tried to implement the codes in a way that it could use less memory but the actual implementation, with a reference table, a bit table for valid codes and the codes in specific bit-mapped memory areas, were not successful. It uses more RAM the the conventional approach (a structure with a value, a flag, a 0-branch pointer and a 1-branch pointer) but does less accesses to the memory for it gains with the locality principle and the uniform distribution of the codes.
The Huffman decoder scan the entropy coded segment of the JPEG file and writes the blocks according and flags the begin of the image.
The dequantization and dezigzagging scans from the begin of the image in 8x8 blocks multiplying each value for its quantizer, storing the result in a reserved memory area and writing it back in the correct dezigzaged order.
The inverse DCT is also applied in the 8x8 blocks using the same dequantization memory area.
The iDCT was divided in 2 matrix multiplication and 2 matrix transposes. The frequency data block is multiplied by a fixed coefficient block, component by component and rewritten to the memory, then the result is one more time multiplied by the coefficient block, but with reading the memory in a different access pattern to perform a transpose-like operation on the matrix, and in the end the final result block of 8x8 pixel image data is written to the memory performing another transpose-like operation. The coefficients were adjusted to a 6 bit precision by experiments on some images, until we obtained good visual results. The multiplier used is specific to this implementation and perform the product of a 6 bit coefficient by a 11 bit data value in 1 clock cycle of the process. The whole iDCT module fit in 218 CLBs of the Xilinx XC4010XL FPGA and takes a cost of 13314 clock cycles to perform a 2D iDCT in a block of 8x8 data values.

The reconfigurable approach used here is that we ran each configuration just once in the whole file and after each scan was finished the kit was reconfigured for the new task.

If we had more chip area, we could group the Huffman decoder, dequantizer and dezigzager, the inverse DCT and the uploader two by two; reconfigure the chip when the decoder is done (it really finishes faster), and when the dequantizer is done so that we could begin displaying the image before the DCT is done. A memory arbiter would have to be written. This mimics the pipeline we could do if we had an even bigger device.
With a higher clock and the same device area, we could implement a MJPEG decoder using the same configuration set for each image.

5. CONCLUSIONS

The approach of reconfigurable logic offers a good trade off between cost and speed, even without device support for partial reconfiguration. The case of the EVM needed more sophisticated equipment for in field tests and benchmarking, like an ISA or a PCI board interface with the FPGA.
The approach of counting the clock cycles was not applied to the whole process because the Huffman decoder and the Huffman setup phases were very input dependent and at that time we did not have the necessary skills for worst case analysis. If the image codes are optimized, the run time is about half the time for the less access made to the memory. The parsing takes time proportional to the file header that is 2 clock cycle per header byte, dequantization and dezigzaging takes 5 clock cycles per matrix entry.

Figure 2 – standard dct (left) and ours (right).

6. REFERENCES

[1] E. Hamilton, “JPEG File Interchange Format version 1.02”, http://www.w3.org/Graphics/JPEG/jfif.txt, California, pp 1-9, 1992
[2] ITU, “International Telegraph and Telephone Consultative committee; CCITT recommendation T.81, Information Technology – Digital Compression and Coding of Continuous-Tone Still Images – Requirements and Guidelines”,http://www.itu.int/rec/recommendation.asp?type=folders&lang=e&parent=T-REC-T.81, pp 1-186, 1992.
[3] XS40 DATASHEET, HTTP://WWW.XESS.COM/MANUALS/XS40-MANUAL-V1_1.PDF.