JPEG codec on Xilinx Microblaze processor

1. Introduction

This is an open source JPEG codec, including both encoder and decoder, for embedded systems. It can be fully synthesized and implemented on FPGA.

Different to a fully hardware implementation, this JPEG codec is designed based on Xilinx Microblaze processor with customized hardware accelerators. It is expected to achieve high flexibility, low complexity at little cost of size and performance. We aim to archive real time motion JPEG codec on a Xilinx Spartan X3S1000 equivalent FPGA.

You can open the project with Xilinx EDK8.1 or higher. The verification hardware platform I use is Xilinx XUP board with a Xilinx XC2V30P on it. It provides necessary peripherals such as CF card for image storage and video output. The board can be obtained at the cost of 300 euro if you are in a university. It's not difficult to port the design into other FPGA boards.

The code here includes two parts, a JPEG codec library and a test bench. The test bench is to read a BMP file (<64KB) from CF card, drive JPEG code library to compress it and write the JPG file back to CF card. You can also make your own design to play with camera and video output based on it.

The JPEG codec library also can be used as a library or IP core for video applications, for instance, MPEG codec. The IP cores can be integrated immediately. It is actually part of my master project and I try to write down in detail how I design and how to use it. Enjoy!

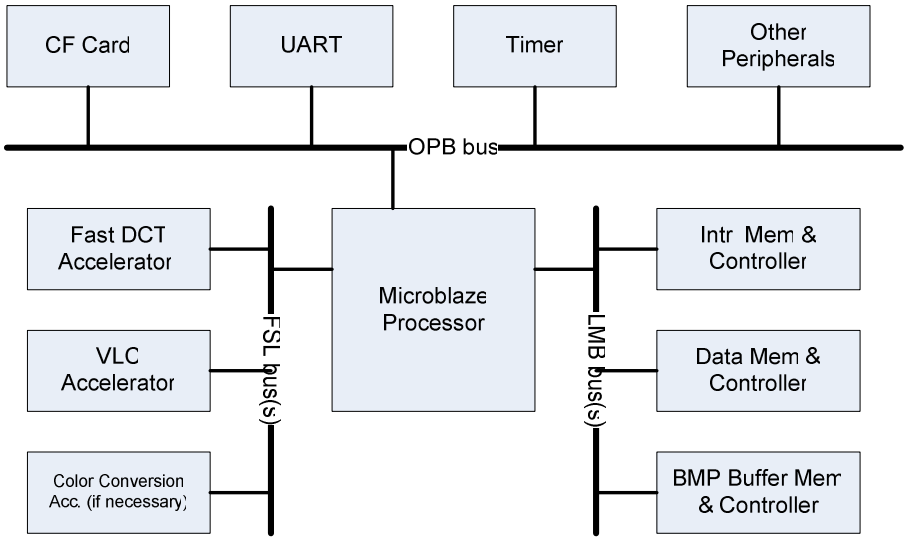
2. Features

1) Baseline JPEG encoder and decoder, compatible to

2) RGB to YUV conversion, 4:4:4 sampling

3. Architecture

3.1 Hardware Architecture



3.2 Data Flow

1) For whole BMP File from CF Card Controller -> OPB Bus -> Microblaze -> DLMB Bus -> BMP Buffer Memory.

The whole BMP file is read from CF card to BMP buffer memory. It is 64KB dedicated data memory. That's also the reason that the maximum BMP file size is 64KB in current implementation.

2) For every BMP macroblock from Buffer Memory -> DLMB Bus ->Microblaze->DLMB Bus -> Data Memory

Every BMP macroblock is then read and convert to YUV color domain and stored in global arrays located in Data Memory.

3) For every YUV macroblock from Data Memory ->DLMB Bus ->Microblaze->DLMB Bus->Data Memory

It is DCT processing. Every YUV domain block is converted from spatial domain to frequency domain.

4) For every frequency domain macroblock from Data Memory->DLMB Bus -> Microblaze->OPB Bus->CF Card or ->DLMB Bus->Data Memory

Every macroblock is then quantized, zigzag scanned and Huffman encoded. The result is written into CF card. Due to file buffer in file system, not every block is written into CF card immediately.

4. Porting Guide

To port the design into other FPGA board is not difficult. All of these design files is able to be reused for other FPGAs.

If you have other peripherals than CF card, for instance, camera, you need add your own peripheral and write your own code to deal with it. All of the code related to I/O is in bmp2jpg_mb.c and all other files can be left untouched.

5. Further improvement

1) Design hardware accelerator to improve performance and reduce memory consumption.
2) Use external memory instead of on-chip memory to store BMP file.
3) Support more peripherals, for instance, camera and video output.
4) Try multiprocessor implementation and compare the result.

6. For further information, please refer to

http://www.opencores.org/projects.cgi/web/mb-jpeg/overview.

Sunwei	sunwei388@gmail.com
Joris van Emden	joris.van.emden@gmail.com
Marcel Lauwerijssen	mlauwerijssen@morphore.com

Appendix A Release Notes

V0.11 2006/07/29

You can checkout source code with CVS tag Step2_2b. It's also possible to download bit file only from http://www.opencores.org/cvsweb.shtml/mb-jpeg/bitstreams/.

Features:

It includes a JPEG codec library and a testbench. It can read a BMP file (<64KB) named "image01.bmp" on CF card, compress it and write "image01.jpg" back to CF card.

Resource:

Number 4 input LUTs:	2,049 out of 27,392	7%
Number of PPC405s:	0 out of 2	0%
Number of Block RAMs:	64 out of 136	47%
Number of MULT18X18s:	3 out of 136	2%

Block RAMs are used for instruction memory (16 blocks, 32KB), data memory (16 blocks, 32KB) and BMP buffer memory (32blocks, 64KB).

Platform:

EDK8.1i2, ISE8.1i2 and Xilinx XUP2PRO board.

Appendix B Bug List

1. Non-8 multiplication image size is not supported.
2. BMP file size is limited to 64KB.