JPEG Decoder: Hardware/Software Co-Design

Yuhui Zheng
Department of Electrical and Computer Engineering,
Purdue University, West Lafayette, IN

Abstract – JPEG (Joint Photographic Experts Group), a most commonly used method for lossy image compression, trades off image quality for storage size. According to different demands, compression degree can be adjusted by user. In this report, various methods are explored to improve JPEG decoding speed without hurting its performance.

Keywords – JPEG, Hardware/Software Co-Design, System Architecture

1 – Introduction on JPEG Encoding & Decoding

The JPEG image compression starts with dividing a whole image into many 8-pixel×8-pixel blocks, where each pixel is represented by a (R, G, B) vector. In this step, an 8×8 block naturally generates three 8×8 square matrices, R-matrix, G-matrix, B-matrix, with every element of the matrices ranging from 0 to 255. Before encoding, matrices are linearly converted to YCbCr color space, where the range of each element is changed from [0,255] to [-128, 127]. Then, 2-dimensional DCT transformation and quantization are performed on each matrix. After this step, elements in lower right of the matrices are turned to 0, which means higher frequencies could be ignored. The last step is to encode each element with entropy encoding. JPEG decoding is realized simply by reversing above steps, as in Fig.1.

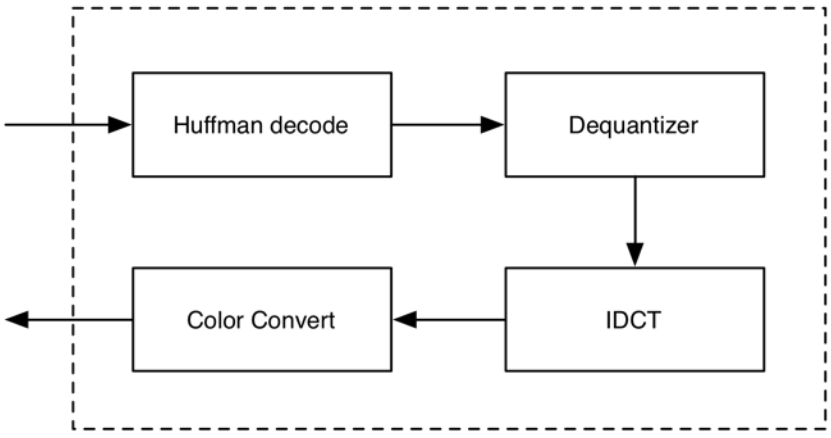

Fig.1 – JPEG decoding steps

2 – System Hardware Configuration

The JPEG decoder needs to read images from memory as input, perform decoding, and finally display decoded images as output. So, the hardware system of a JPEG decoder should, at least, consist of CPU, TRI_STATE_BRIDGE, SDRAM, PIXEL_BUFFER_DMA, and VGA_CONTROLLER. Besides, a PERFORMANCE_COUNTER is attached to the system to measure the time consumed within modules. The system hardware architecture in SOPC builder is as Fig.2.

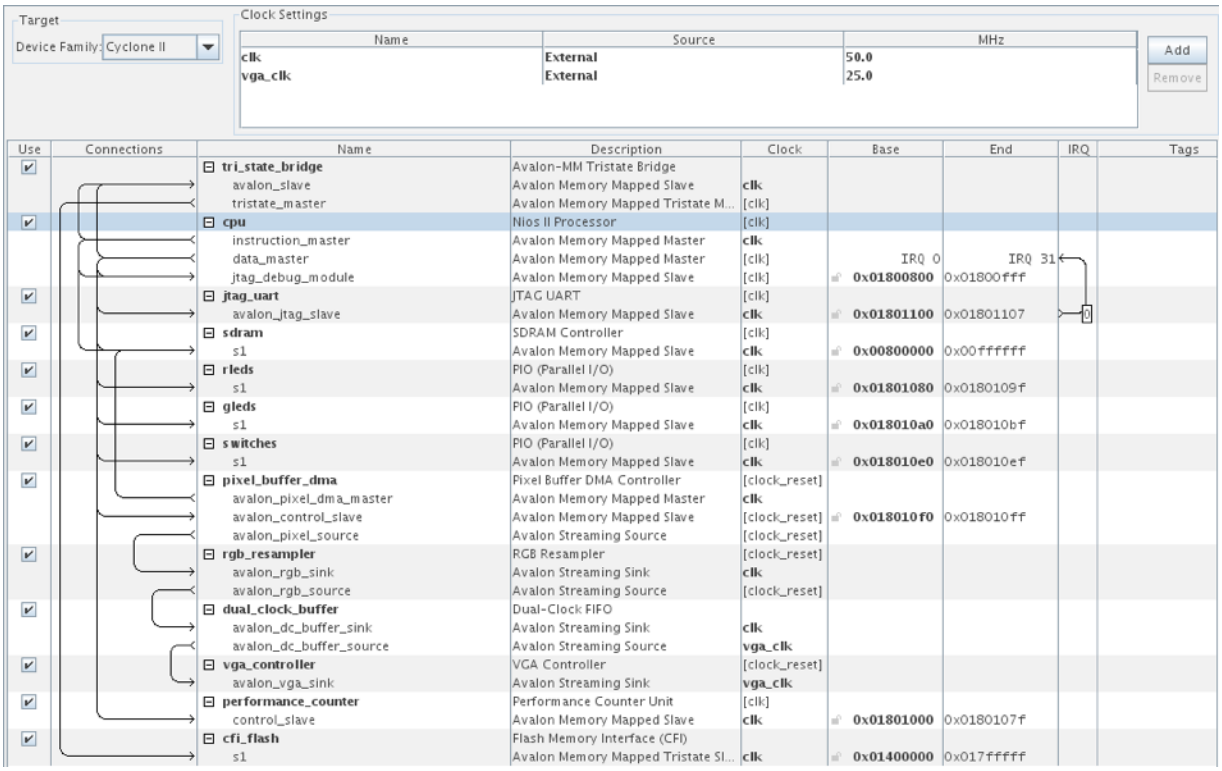

Fig.2 – JPEG decoder hardware architecture

Before optimizing, how many hardware resources left should be considered. As Fig.3, after synthesizing the baseline hardware system, there are still a plenty of resources left. Thus, for this project, the hardware resources should not be the principal limitation.

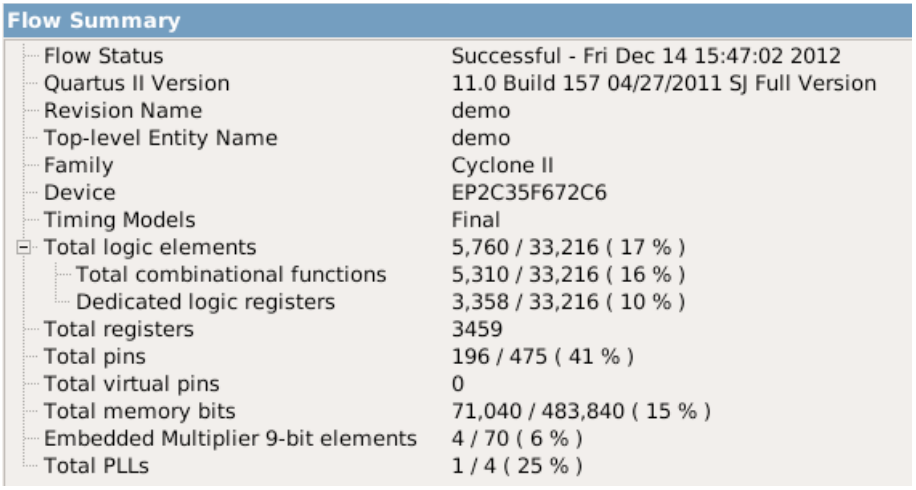

Fig.3 – Report on hardware synthesis

3 – Software performance on customized hardware

The software architecture of the given JPEG decoder is as below:

```
                                                         |-- Entropy decoding
                   |-- Coefficient controller --         |
                   |                                     |-- Dequantize, Inverse DCT
Main controller -- |
                   |
                   |                                     |-- Upsampling
                   |-- Postprocessing controller --      |       |-- Colorspace conversion
                                                         |-- Color quantization
                                                         |-- Color precision reduction
```

Thus, the software system performance measurement mainly focuses on: Huffman Decoder, Quantizer, IDCT, Color Conversion, and Up Sampling. Configure the instruction cache and data cache as below. Let performance counter measure time elapsed on each module.

```
// Configuration
$I              =4KBytes
$D              =2KBytes
$D_blocksize    =32Bytes
Decode in:      software
Input File:     6.6KBytes
IDCT:           FAST
```

--Performance Counter Report--
Total Time: 1.77202E+06 seconds (88600895 clock-cycles)

Section	%	Time (sec)	Time (clocks)	Occurrences
reading and decomp.	71.3	1263293.20000	63164660	256
writing to display	16.2	287066.28000	14353314	1
h2v2	11	194189.10000	9709455	256
ycc_rgb_conver	34	601983.92000	30099196	256
IFAST	14.5	257632.02000	12881601	1536

The result shows that Huffman Decoder and Quantizer take very little time(under 10%, and not shown in data above). Thus the optimization could be mainly focused on Up Sampling, Color Conversion, and IDCT.

Within Up Sampling, Color Conversion and IDCT, the Color Conversion takes the most time, yet of least importance. To be concrete, color conversion is a linear function convert YCC color to RGB color. Without it, we could still see the picture, but in different color. Yet it calls a subroutine repeatedly, which significantly hurts the overall performance. Thus, the Color Conversion module is given the top priority to be accelerated.

On the contrary, though IDCT is the key part of JPEG Decoding, there is little gain in optimizing it. Thus, one would hardly want to touch this module. Up Sampling may not be needed in some images, and the performance gain is not that much.

4 – Optimize for Overall System Performance

Since JPEG Decoding is intensively memory-needed program, most time is consumed in fetching and storing data to the memory system. In this thought, optimization could be done in these ways:

- Improve data cache.
- Reduce fetching and storing by reorganizing software.
- Reduce fetching and storing by implementing constant data in to hardware.
- Increase fetching and storing speed by prefetching.

As chapter 3 analyzed, Method 2 and 3 could be used to optimize Color Conversion and Up Sampling module, while Method 1 could be used to improve the whole system. Result for Method 1 is shown in Part B, Method 2 in Part C, Method 3 in Part A. As time is limited, I didn't go further for Method 4. In addition, Part D shows the system performance on 3 different images.

Part A: Hardware Implementation

As one can see in jdcolor.c and jdsample.c file, software inefficiency is brought by looking up constant data table in each calculation. The performance could be optimized when putting the data table into hard-wired circuit. There are two ways to achieve this goal: Hardware Accelerator and Custom Instruction Set. Since Hardware Accelerator may introduce extra overhead in communication on system bus, implementation is done in Custom Instruction Set, which implements it as a new Function Unit in CPU. The logic element used is shown in Fig.4. The performance gain is in Table 1.

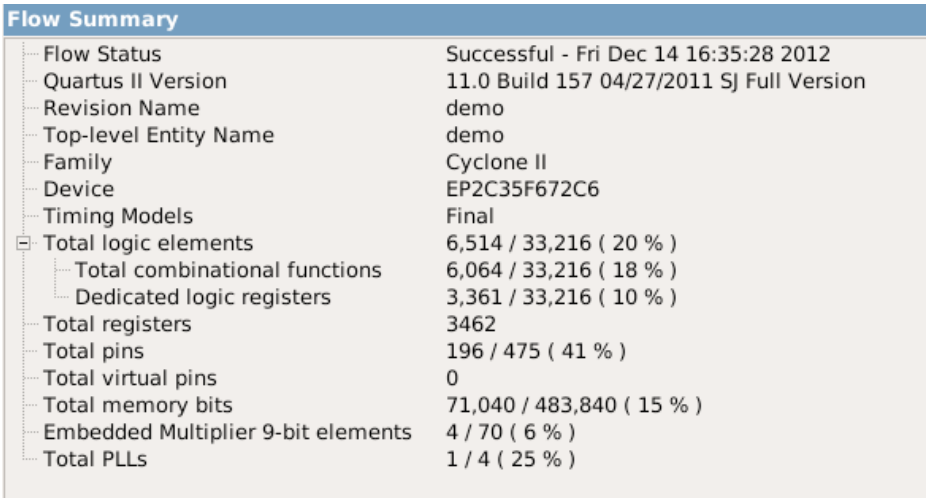

Fig.4 – Hardware consumed for Custom Instruction Set implementation

	SW	HW	Speed up
image used	6.6KBytes		
total (*10^6)	1.77272	1.34689	
reading & decomp.	71.3%	61.8%	
writing to display	16.2%	21.7%	
up sampling	10.9%	11.5%	5.432%
color convert	33.9%	12.7%	20.571%
IDCT (fast)	14.5%	21.7%	

Table 1 – Performance gain on Custom Instruction Set

Part B: D Cache Size & D Cache Block Size

Generally speaking, a bigger cache size is preferred, which could contain more data, and increase hit rate. Yet, with limited cache size, block size should not be too big. Because bigger block size may reduce cache entries, which could result higher probability on conflicts. Also a bigger cache block may need longer time to write back, whenever there is a conflict. Test results are as Table 2.

I Cache	4KBytes					
D Cache	2KBytes			4KBytes		
D Cache Block Size	32B	16B	4B	32B	16B	4B
Test #	#1	#2	#3	#4	#5	#6

Test #	Total (10^6S)	Reading & Decomp.	Writing to Display	Up Sampling	Color Convert	IDCT (FAST)
#1	1.77202	71.3%	16.2%	11%	34%	14.5%
#2	1.57608	68.9%	17.5%	11.7%	30.3%	15.1%
#3	2.04742	68.3%	17.6%	16.6%	26.2%	13.6%
#4	1.27755	63.1%	20.8%	13.1%	19.8%	18%
#5	1.22968	61.9%	21.4%	13.5%	19.2%	17.7%
#6	1.74729	63.5%	20.5%	18.9%	18.1%	14.3%

Table 2 – Performance variation on different cache configuration

Part C: Loop Unrolling

It seems that loop unrolling could remove branch overhead, thus performance could be improved. Yet in our case, loop unrolling introduces extra ending test and alignment overhead. Performance for loop unrolling on h2v2_fancy_sample() function is as Table 3.

Unrolling	factor=0	factor=2	factor=4
total time	1.77202	1.72244	1.82396
h2v2	11%	9%	17.9%

Table 3 – Performance variation on loop unrolling

Part D: Different Images

Different images may use different modules of decoding, for example Up Sampling is only used in some images. Also, the time needed for a same module in different images could be also different. Table 4 is the performance test result for three different images.

5 – Summary

Putting all of above together, Custom Instruction Set and Cache Reconfiguration are the most efficient ways to improve overall system performance.

title	colours.jpg		sea.jpg		lena.jpg	
	SW	**HW**	**SW**	**HW**	**SW**	**HW**
cache parameter	#1					
size	197.8 KBytes		46.4 KBytes		89.7 KBytes	
total(10^6)	10.0176	8.1473	4.14342	3.07322	9.30887	7.35239
up sampling	0%	0%	8.52%	9.87%	0%	0%
color convert	15.7%	7.28%	34.2%	11.3%	27.2%	11.9%
IDCT (fast)	26.4%	29.8%	13.3%	24.4%	23.5%	29.6%

Table 4 – Performance variation on different images