IP Core Design
Homework 1
Baseline JPEG Software Encoder

Instructor: Prof. Chein-Wei Jen
Announcement: 2001.10.29

Design a baseline JPEG software encoder and then optimize it for ARM7TDMI processor. You can either write your own design or modify an existing reference code such as those listed in [1]. Be aware of the differences between the PC environment and the final target platform (i.e., ARM development boards) because this JPEG software encoder will be ported to ARM development boards in homework 2. Also, the data structures and the partition of functional calls should be carefully defined because portions of this JPEG encoder will be modeled as hardware components in homework 3 and mapped to FPGA in homework 4.

The benefits of such baseline JPEG software encoder include

- Be familiar with writing code (driver) for ARM processor
- Be familiar with using ARM software tools
- Verify algorithms used for hardware IPs
- Generate test pattern and/or files for hardware IPs

Try to improve the performance and memory requirement of your JPEG encoder without sacrificing the image quality. Memory requirement includes the program itself and the temporal memory for data processing. The following approaches may possibly be helpful in such improvement.

- Select or modify the algorithms or the code segments used in JPEG to fit to ARM's architecture. By taking constraints of the ARM core hardware resources into consideration, some algorithms may be more suitable for ARM core than others. An example of such consideration can be found in [2].
- Create SIMD operations. Though current ARM architecture has no specific instructions to support single-instruction, multiple-data (SIMD) operation, certain SIMD operations can be synthesized using a sequence of normal ARM instructions [3].
- Use ARM/Thumb mode for different code segments.

[1] JPEG image compression FAQ, part 2/2, http://www.faqs.org/faqs/jpeg-faq/
[2] Tadashi Sakamoto and Tomohiro Hase, “Software JPEG for a 32-bit MCU with dual issue,” IEEE Transactions on Consumer Electronics, Vol. 44 Issue: 4, Nov. 1998, pp. 1334 -1341.
[3] Alan Lewis and Paul Carpenter, “Optimizing digital video codecs in ARM cores,” EE Times, Sep. 20, 2001.

Deliverable

Your deliverable has to include:
1. Report that describes your idea and result.
2. Source code of your JPEG encoder.
3. All setting and information required for regenerating the result shown in your report.

Please specify the memory requirement, profiling and statistics (Section 5.5.8 Debugger Internals System View, ADS Debuggers Guide v1.1). Separate the result of statistics for file I/O routines and JPEG kernel. State your approaches, key ideas and results clearly and formally, and void redundant description. Your report can be written in Chinese or English. However, make sure your report is readable. A manual report won't degrade your score, unless it is scrabbled.

If your JPEG encoder is modified from an existing reference code, please acknowledge in the last section of your documentation that you've used the reference code.

Important Date

Due : 5:00 p.m. Nov. 12, 2001

For more information

- The contents of this document: Kun-Bin Lee
- ARM development tools: contact the TA with the number = your team number %5

No.		Email	Ext.
0	Kun-Bin Lee	kblee@yankees.ee.nctu.edu.tw	54225
1	Yuan-Chung Lee	yzlee@yankees.ee.nctu.edu.tw	54225
2	Jih Yiing Lin	jinlin@yankees.ee.nctu.edu.tw	54243
3	Nelson Yen-Chung Chang	ycchang@yankees.ee.nctu.edu.tw	54243
4	Tzung-Shian Yang	tsyang@yankees.ee.nctu.edu.tw	54243

JPEG encoder specification

Basic requirement

- baseline JPEG
- The input is a BMP file and the output is a standard JFIF file, which starts with the four bytes (hex) FF D8 FF E0, followed by two variable bytes (often hex 00 10), followed by 'JFIF' (i.e., FF D8 FF E0 00 10 4A 46 49 46).
- 24 bits per pixel for color images and 8 bits/pixel for grayscale image
- Maximum image size is 2048 × 1536 (e.g., Nikon Coolpix E995)
- Huffman coding for the final coding stage
- Quality: using the default huffman and quantization tables shown in the later section.

In addition to the basic requirement listed above, you may add a variety of features to your JPEG encoder and implement them into your hardware in the next homework.

Default huffman and quantization tables

The source of the following tables is Independent JPEG Group's JPEG software release 6b.

```
luminance 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
};

chrominance quantization table {
      17,  18,  24,  47,  99,  99,  99,  99,
      18,  21,  26,  66,  99,  99,  99,  99,
      24,  26,  56,  99,  99,  99,  99,  99,
      47,  66,  99,  99,  99,  99,  99,  99,
      99,  99,  99,  99,  99,  99,  99,  99,
      99,  99,  99,  99,  99,  99,  99,  99,
      99,  99,  99,  99,  99,  99,  99,  99,
      99,  99,  99,  99,  99,  99,  99,  99
};

//-------------------------------------------------------------
// standard Huffman tables
// (cf. JPEG standard section K.3)
//-------------------------------------------------------------
   bits_dc_luminance[17] =
      { 0, 0, 1, 5, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0 };
   val_dc_luminance[] =
      { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 };

   bits_dc_chrominance[17] =
      { 0, 0, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0 };
   val_dc_chrominance[] =
      { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 };

   bits_ac_luminance[17] =
      { 0, 0, 2, 1, 3, 3, 2, 4, 3, 5, 5, 4, 4, 0, 0, 1, 0x7d };
   val_ac_luminance[] =
      { 0x01, 0x02, 0x03, 0x00, 0x04, 0x11, 0x05, 0x12,
        0x21, 0x31, 0x41, 0x06, 0x13, 0x51, 0x61, 0x07,
        0x22, 0x71, 0x14, 0x32, 0x81, 0x91, 0xa1, 0x08,
        0x23, 0x42, 0xb1, 0xc1, 0x15, 0x52, 0xd1, 0xf0,
        0x24, 0x33, 0x62, 0x72, 0x82, 0x09, 0x0a, 0x16,
        0x17, 0x18, 0x19, 0x1a, 0x25, 0x26, 0x27, 0x28,
```

```
        0x29, 0x2a, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39,
        0x3a, 0x43, 0x44, 0x45, 0x46, 0x47, 0x48, 0x49,
        0x4a, 0x53, 0x54, 0x55, 0x56, 0x57, 0x58, 0x59,
        0x5a, 0x63, 0x64, 0x65, 0x66, 0x67, 0x68, 0x69,
        0x6a, 0x73, 0x74, 0x75, 0x76, 0x77, 0x78, 0x79,
        0x7a, 0x83, 0x84, 0x85, 0x86, 0x87, 0x88, 0x89,
        0x8a, 0x92, 0x93, 0x94, 0x95, 0x96, 0x97, 0x98,
        0x99, 0x9a, 0xa2, 0xa3, 0xa4, 0xa5, 0xa6, 0xa7,
        0xa8, 0xa9, 0xaa, 0xb2, 0xb3, 0xb4, 0xb5, 0xb6,
        0xb7, 0xb8, 0xb9, 0xba, 0xc2, 0xc3, 0xc4, 0xc5,
        0xc6, 0xc7, 0xc8, 0xc9, 0xca, 0xd2, 0xd3, 0xd4,
        0xd5, 0xd6, 0xd7, 0xd8, 0xd9, 0xda, 0xe1, 0xe2,
        0xe3, 0xe4, 0xe5, 0xe6, 0xe7, 0xe8, 0xe9, 0xea,
        0xf1, 0xf2, 0xf3, 0xf4, 0xf5, 0xf6, 0xf7, 0xf8,
        0xf9, 0xfa };

    bits_ac_chrominance[17] =
       { 0, 0, 2, 1, 2, 4, 4, 3, 4, 7, 5, 4, 4, 0, 1, 2, 0x77 };
    val_ac_chrominance[] =
       { 0x00, 0x01, 0x02, 0x03, 0x11, 0x04, 0x05, 0x21,
         0x31, 0x06, 0x12, 0x41, 0x51, 0x07, 0x61, 0x71,
         0x13, 0x22, 0x32, 0x81, 0x08, 0x14, 0x42, 0x91,
         0xa1, 0xb1, 0xc1, 0x09, 0x23, 0x33, 0x52, 0xf0,
         0x15, 0x62, 0x72, 0xd1, 0x0a, 0x16, 0x24, 0x34,
         0xe1, 0x25, 0xf1, 0x17, 0x18, 0x19, 0x1a, 0x26,
         0x27, 0x28, 0x29, 0x2a, 0x35, 0x36, 0x37, 0x38,
         0x39, 0x3a, 0x43, 0x44, 0x45, 0x46, 0x47, 0x48,
         0x49, 0x4a, 0x53, 0x54, 0x55, 0x56, 0x57, 0x58,
         0x59, 0x5a, 0x63, 0x64, 0x65, 0x66, 0x67, 0x68,
         0x69, 0x6a, 0x73, 0x74, 0x75, 0x76, 0x77, 0x78,
         0x79, 0x7a, 0x82, 0x83, 0x84, 0x85, 0x86, 0x87,
         0x88, 0x89, 0x8a, 0x92, 0x93, 0x94, 0x95, 0x96,
         0x97, 0x98, 0x99, 0x9a, 0xa2, 0xa3, 0xa4, 0xa5,
         0xa6, 0xa7, 0xa8, 0xa9, 0xaa, 0xb2, 0xb3, 0xb4,
         0xb5, 0xb6, 0xb7, 0xb8, 0xb9, 0xba, 0xc2, 0xc3,
         0xc4, 0xc5, 0xc6, 0xc7, 0xc8, 0xc9, 0xca, 0xd2,
         0xd3, 0xd4, 0xd5, 0xd6, 0xd7, 0xd8, 0xd9, 0xda,
         0xe2, 0xe3, 0xe4, 0xe5, 0xe6, 0xe7, 0xe8, 0xe9,
         0xea, 0xf2, 0xf3, 0xf4, 0xf5, 0xf6, 0xf7, 0xf8,
         0xf9, 0xfa };
```