JPEG Compression/Decompression using SystemC

EE8205: Embedded Computer Systems
http://www.ecb.torontomu.ca/~courses/ee8205/

Dr. Gul N. Khan
http://www.ecb.torontomu.ca/~gnkhan
Electrical and Computer Engineering
Toronto Metropolitan University

Overview

- Introduction to JPEG Coding and Decoding
- Hardware-Software Partitioning
- FDCT and IDCT HW module for 8 x 8 Block
- JPEG Implementation

Introductory Articles on JPEG Compression and SystemC tutorial documents available at the course webpage. Digital Image Processing by Gonzolez and Wood Chapter 6

JPEG-based Encoding

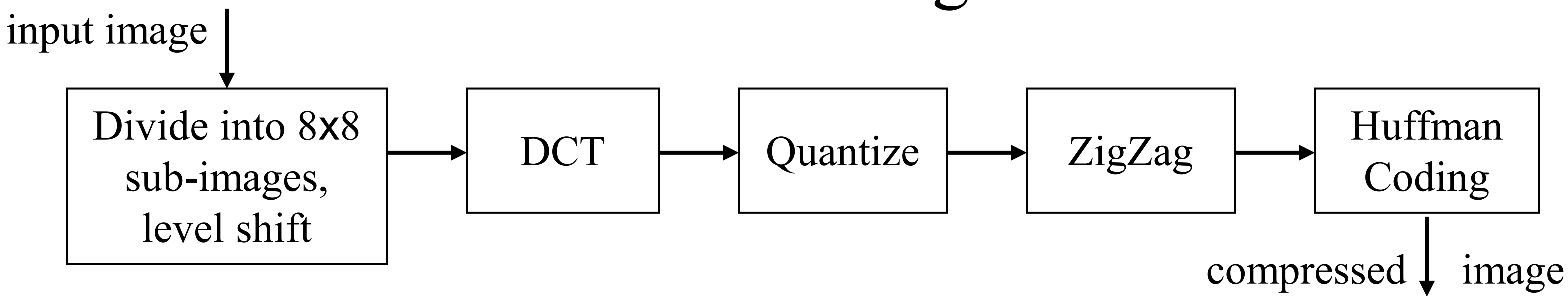
Four Stages of JPEG Compression

- Preprocessing and dividing an image into 8 x 8 blocks

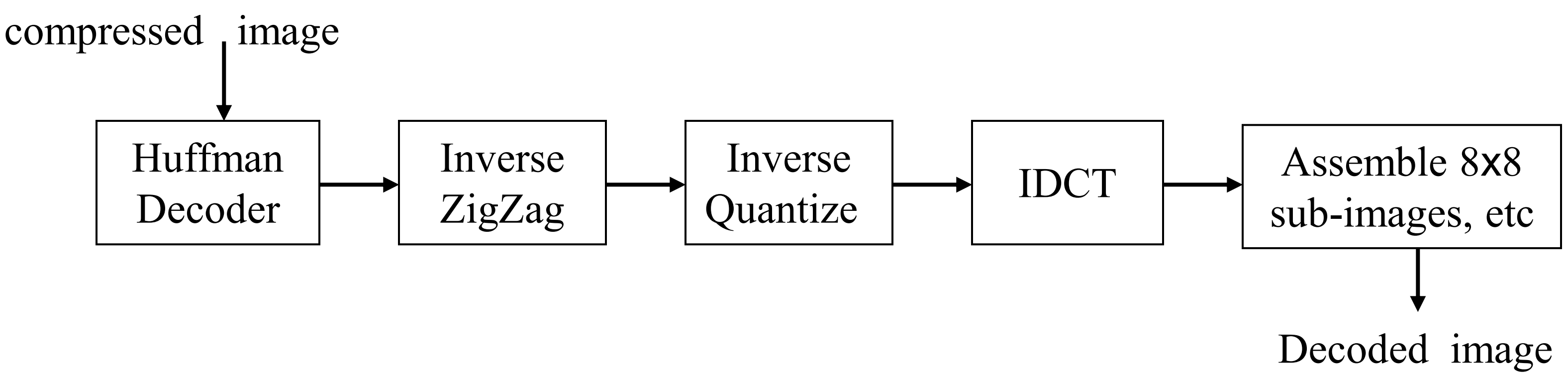
  Level-shift, for 8-bit gray scale images, subtract 128 from each pixel i.e. pixel[i] = pixel[i] – 128 ;
- DCT, Discrete Cosine Transform of 8 x 8 image blocks.
- Quantization
- ZigZag
- Entropy Encoding either of:
  - Huffman coding
  - Variable Length Coding

JPEG Encoding and Decoding

Encoding



Decoding



DCT: Discrete Cosine Transform

Mathematical definitions of 8 x 8 DCT and 8 x 8 IDCT respectively.

$$F(u,v) = \frac{1}{4} C(u)\, C(v) \left[\sum_{x=0}^{7} \sum_{y=0}^{7} f(x,y) * cos((2x+1)u\pi)/16 * cos((2y+1)v\pi/16 \right]$$

$$f(x,y) = \frac{1}{4} \left[\sum_{u=0}^{7} \sum_{v=0}^{7} C(u)\, C(v) F(u,v) * cos((2x+1)u\pi)/16 * cos((2y+1)v\pi/16 \right]$$

where $C(u), C(v) = 1/\sqrt{2}$ for $u,v = 0$

$C(u), C(v) = 1$ otherwise

$F(u,v)$ is the Discrete Cosine Transform of 8 x 8 block

$f(x,y)$ is the Inverse Discrete Cosine Transform

Why DCT instead of DFT

DCT is similar to DFT with many advantages

- DCT coefficients are purely real
- Near-optimal for energy compaction
- DCT computation is efficient due to faster algorithms
- Hardware solutions available that do not need multipliers

DCT is extensively used in image compression standards including, JPEG, MPEG-1, MPEG-2, MPEG-4, etc.

52	55	61	66	70	61	64	73
63	59	66	90	109	85	69	72
62	59	68	113	144	104	66	73
63	58	71	122	154	106	70	69
67	61	68	104	126	88	68	70
79	65	60	70	77	68	58	75
85	71	64	59	55	58	65	83
87	79	69	68	65	65	78	94

DCT $\Rightarrow$

-415	-29	-62	25	55	-20	-1	3
7	-21	-62	9	11	-7	-6	6
-46	8	77	-25	-30	10	7	-5
-50	13	35	-15	-9	6	0	3
11	-8	-13	-2	-1	1	-4	1
-10	1	3	-3	-1	0	2	-1
-4	-1	2	-1	0	-3	1	-2
-1	-1	-1	-2	-3	-1	0	-1

Quantization

The 8x8 block of DCT transformed values is divided by a quantization value for each block entry.

$$F_{quantized}(u,v) = F(u,v) / Quantization\_Table(x,y)$$

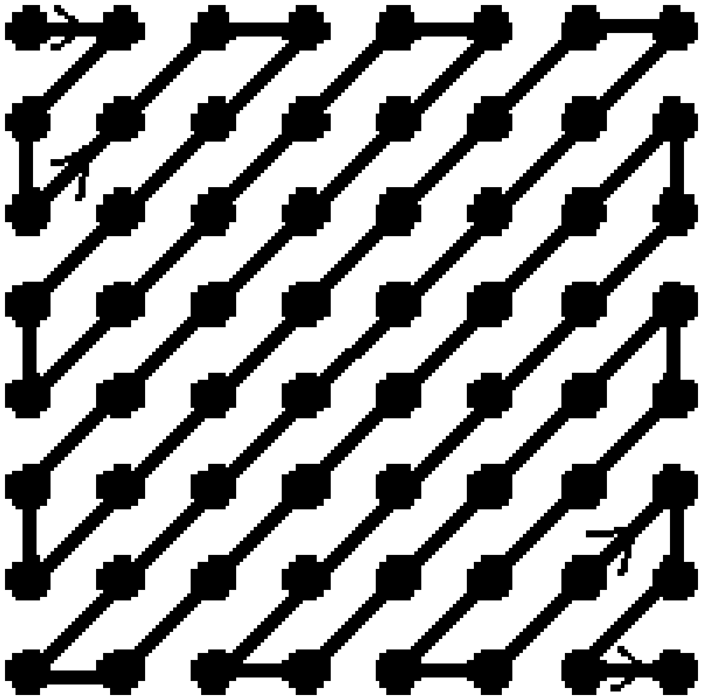
Quantization table :

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

Zig-Zag

It takes the quantized 8x8 block and orders it in a 'Zig-Zag' sequence, resulting in a 1-D array of 64 entries,

- This process place low-frequency coefficients (larger values) before the high-frequency ones (nearly zero).
- One can ignore any continuous zeros at the end of block
- Insert a (EOB) at the end of each 8x8 block encoding.



Quantization and ZigZag

$$
\begin{pmatrix}
-415 & -29 & -62 & 25 & 55 & -20 & -1 & 3 \\
7 & -21 & -62 & 9 & 11 & -7 & -6 & 6 \\
-46 & 8 & 77 & -25 & -30 & 10 & 7 & -5 \\
-50 & 13 & 35 & -15 & -9 & 6 & 0 & 3 \\
11 & -8 & -13 & -2 & -1 & 1 & -4 & 1 \\
-10 & 1 & 3 & -3 & -1 & 0 & 2 & -1 \\
-4 & -1 & 2 & -1 & 0 & -3 & 1 & -2 \\
-1 & -1 & -1 & -2 & -3 & -1 & 0 & -1
\end{pmatrix}
\div
\begin{pmatrix}
16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\
12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\
14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\
14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\
18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\
24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\
49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\
72 & 92 & 95 & 98 & 112 & 100 & 103 & 99
\end{pmatrix}
$$

$$
\begin{pmatrix}
-26 & -3 & -6 & 2 & 2 & 0 & 0 & 0 \\
1 & -2 & -4 & 0 & 0 & 0 & 0 & 0 \\
-3 & 1 & 5 & -1 & -1 & 0 & 0 & 0 \\
-4 & 1 & 2 & -1 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}
$$

Zig-Zag

[-26 -3 1 -3 -2 -6 2 – 4 1 -4 1 1 5 0 2
0 0 -1 2 0 0 0 0 0 -1 -1 EOB]

FDCT SC_Module

```
struct fdct : sc_module {
  sc_out<double> out64[8][8]; // the dc transformed 8x8 block
  sc_in<double> fcosine[8][8]; // cosine table input
  sc_in<FILE *> sc_input; // input file pointer port
  sc_in<bool> clk; // clock signal
  char input_data[8][8]; // the data read from the input file
  void read_data( void ); // read the 8x8 block
  void calculate_dct( void ); // perform dc transform
  // define fdct as a constructor
  SC_CTOR( fdct ) {
  // read_data method sensitive to +ve & calculate_dct sensitive to
  // -ve clock edge, entire read and dct will take one clock cycle
       SC_METHOD( read_data ); // define read_data as a method
       dont_initialize();
       sensitive << clk.pos;
       SC_METHOD( calculate_dct );
       dont_initialize();
       sensitive << clk.neg;
  }
};
```

DCT Module

```
#include "fdct.h"
void fdct :: calculate_dct( void ) {
unsigned char  u, v, x, y;
double         temp;
   for (u = 0; u < 8; u++) // do forward discrete cosine transform
     for (v = 0; v < 8; v++) {  temp = 0.0;
        for (x = 0; x < 8; x++)
        for (y = 0; y < 8; y++)
          temp += input_data[x][y] * fcosine[x][u].read() *
               fcosine[y][v].read();
        if ((u == 0) && (v == 0)) temp /= 8.0;
        else if (((u == 0) && (v != 0)) || ((u != 0) && (v == 0)))
          temp /= (4.0*sqrt(2.0)); else temp /= 4.0;
        out64[u][v].write(temp);
}  }
void fdct :: read_data( void ) { // read the 8*8 block
   fread(input_data, 1, 64, sc_input.read());
   // shift from range [0, 2^8 - 1] to [2^(8-1), 2^(8-1) - 1]
   for (unsigned char uv = 0; uv < 64; uv++)
        input_data[uv/8][uv%8] -= (char) (pow(2,8-1));
}
```

DCT Module Structures

```
#define PI 3.1415926535897932384626433832795 // the value of PI
unsigned char quant[8][8] =            // 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}};
unsigned char zigzag_tbl[64]={         // zigzag table
       0,1,5,6,14,15,27,28,
       2,4,7,13,16,26,29,42,
       3,8,12,17,25,30,41,43,
       9,11,18,24,31,40,44,53,
       10,19,23,32,39,45,52,54,
       20,22,33,38,46,51,55,60,
       21,34,37,47,50,56,59,61,
       35,36,48,49,57,58,62,63};
signed char MARKER = 127; // end of block marker
```

Functions: Read File Header

```
#define rnd(x) (((x)>=0)?((signed char)((signed char)((x)+1.5)-
   1)):((signed char)((signed char)((x)-1.5)+1)))
#define rnd2(x) (((x)>=0)?((short int)((short int)((x)+1.5)-
   1)):((short int)((short int)((x)-1.5)+1)))

// read the header of the bitmap and write it to the output file

void write_read_header(FILE *in, FILE *out) {
   unsigned char temp[60]; // temporary array of 60 characters,
          // which is enough for the bitmap header: 54 bytes
   printf("\nInput Header read and written to the output file");
   fread(temp, 1, 54, in); // read 54 bytes and store them in temp
   fwrite(temp, 1, 54, out); // write 54 bytes to the output file
   printf("......Done\n");
   printf("Image is a %d bit Image. Press Enter to Continue\n>",
   temp[28]);
   getchar();
}
```

Functions: Cosine-table

```
// make the cosine table
void make_cosine_tbl(double cosine[8][8]);
void make_cosine_tbl(double cosine[8][8]) {
  printf("Creating the cosine table to be used in FDCT and
  IDCT");
  // calculate the cosine table as defined in the formula
  for (unsigned char i = 0; i < 8; i++)
      for (unsigned char j = 0; j < 8; j++)
            cosine[i][j] = cos((((2*i)+1)*j*PI)/16);
  printf("......Done\n");
}
```

Functions: ZigZag

```
// zigzag the quantized input data

void zigzag_quant(double data[8][8], FILE *output) {
  signed char to_write[8][8];
    // this is the rounded values, to be written to the file
  char last_non_zero_value = 0; // index to last non-zero in a block
   // zigzag data array & copy it to to_write, round the values
   // and find out the index to the last non-zero value in a block
  for (unsigned char i = 0; i < 64; i++) {
   to_write[zigzag_tbl[i]/8][zigzag_tbl[i]%8] =
       rnd(data[i/8][i%8] /   quant[i/8][i%8]);
       if (to_write[i/8][i%8] != 0) last_non_zero_value = i;
  }
  // write all values in the block including the last non-zero value
  for (unsigned char i = 0; i <= last_non_zero_value; i++)
      fwrite(&to_write[i/8][i%8], sizeof(signed char), 1, output);
        // write the end of block marker
  fwrite(&MARKER, sizeof(signed char), 1, output);
}
```

Functions: Main

```
#include "systemc.h"
#include "functions.h"
#include "fdct.h"
#include "idct.h"
#define NS *1e-9 // constant for clock signal is in nanoseconds
int sc_main(int argc, char *argv[]) {
   char choice;
   sc_signal<FILE *> sc_input; // input file pointer signal
   sc_signal<FILE *> sc_output; // output file pointer signal
   sc_signal<double> dct_data[8][8]; // signal to the dc transformed
   sc_signal<double> cosine_tbl[8][8]; // signal for cos-table values
   sc_signal<bool> clk1, clk2; // clock signal for FDCT and IDCT
   FILE *input, *output; // input and output file pointers
   double cosine[8][8]; // cosine table
   double data[8][8]; // data read from signals to be zigzagged
   if (argc == 4) {
      if (!(input = fopen(argv[1], "rb")))
           // some error occurred while trying to open the input file
      printf("\nSystemC JPEG-LAB:\nCannot Open File '%s'\n",argv[1]),
                                                              exit(1);
```

Functions - Main

```
write_read_header(input, output);
            // write the header read from the input file
make_cosine_tbl(cosine); // make the cosine table

// copy cosine and quantization tables onto corresponding signals
for (unsigned char i = 0; i < 8; i++)
     for (unsigned char j = 0; j < 8; j++)
            cosine_tbl[i][j].write(cosine[i][j]);

fdct FDCT("fdct"); // call the forward discrete transform module
// bind the ports
for (unsigned char i = 0; i < 8; i++)
     for (unsigned char j = 0; j < 8; j++) {
        FDCT.out64[i][j](dct_data[i][j]);
        FDCT.fcosine[i][j](cosine_tbl[i][j]);
     }
FDCT.clk(clk1);
FDCT.sc_input(sc_input);
```

Functions: Main

cont.

```
// we must use two different clocks. That will make sure that when
// we want to compress, we only compress and don't decompress it
  sc_start(SC_ZERO_TIME);     // initialize the clock
  if ((choice == 'c') || (choice == 'C')) { // for compression
       while (!(feof(input))) { // create the FDCT clock signal
         clk1.write(1);          // convert the clock to high
         sc_start(10, SC_NS);  // cycle high for 10 nanoseconds
         clk1.write(0);         // start the clock as low
         sc_start(10, SC_NS); //cycle low for 10 nanoseconds
              // read all the signals into the data variable
              // to use these values in a software block
           for (unsigned char i = 0; i < 8; i++)
             for (unsigned char j = 0; j < 8; j++)
                     data[i][j] = dct_data[i][j].read();
         zigzag_quant(data, output);
              // zigzag and quantize the read data
       }
   }
}
```