EEL 4783: Hardware/Software Co-design with FPGAs

Lecture 4: Digital Camera: Software Implementation*

Prof. Mingjie Lin

UCF
Stands For Opportunity

* Some slides based on ISU CPrE 588

Digital Camera Introduction

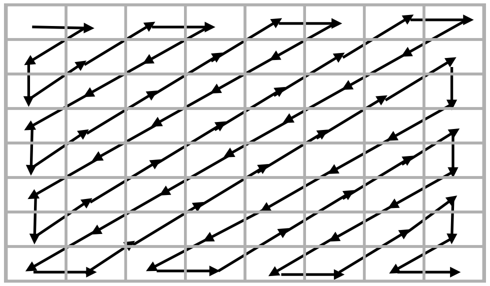
- Captures images
- Stores images in digital format
  - No film
  - Multiple images stored in camera
    - Number depends on amount of memory and bits used per image
- Downloads images to PC
- Only recently possible
  - Systems-on-a-chip
    - Multiple processors and memories on one IC
  - High-capacity flash memory
- Very simple description used for example
  - Many more features with real digital camera

Compression

- Store more images
- Transmit image to PC in less time
- JPEG (Joint Photographic Experts Group)
  - Popular standard format for representing digital images in a compressed form
  - Mode used in this chapter provides high compression ratios using DCT (discrete cosine transform)
  - Image data divided into blocks of 8 x 8 pixels
  - 3 steps performed on each block
    - DCT
    - Quantization
    - Huffman encoding

Huffman Encoding Step

- Serialize 8 x 8 block of pixels
  - Values are converted into single list using zigzag pattern



- Perform Huffman encoding
  - More frequently occurring pixels assigned short binary code
  - Longer binary codes left for less frequently occurring pixels
- Each pixel in serial list converted to Huffman encoded values
  - Much shorter list, thus compression

Huffman Decoding

- In 1951, David Huffman and his MIT information theory classmates given the choice of a term paper or a final exam
- Huffman hit upon the idea of using a frequency-sorted binary tree and quickly proved this method the most efficient.
- In doing so, the student outdid his professor, who had worked with information theory inventor Claude Shannon to develop a similar code.
- Huffman built the tree from the bottom up instead of from the top down

A simple example

- Suppose we have a message consisting of 5 symbols, e.g. [►♣♣♠☻►♣☼►☻]
- How can we code this message using 0/1 so the coded message will have minimum length (for transmission or saving!)

►	000
♣	001
☻	010
♠	011
☼	100

- 5 symbols → at least 3 bits
- For a simple encoding,

length of code is 10*3=30 bits

A simple example – cont.

- Intuition: Those symbols that are more frequent should have smaller codes, yet since their length is not the same, there must be a way of distinguishing each code
- For Huffman code,

length of encoded message

will be ►♣♣♠☻►♣☼►☻

=3*2 +3*2+2*2+3+3=24bits

Symbol	Freq.	Code
►	3	00
♣	3	01
☻	2	10
♠	1	110
☼	1	111

Frequency	Number	Huffman code
45	1000	00
20	100	01
10	10	100
5	5	1010
1	1	1011

JPEG encoding compresses data in five ways

- Because DC coefficients do not change significantly between adjacent blocks, they are encoded as differences. (Diff = DCi - DCi-1) This coding technique is known as Differential Pulse Code Modulation (DPCM).

- Quantized AC coefficients usually contain a run of consecutive zeroes. For this reason AC codes specify the run-length (number of consecutive zeroes preceding a non-zero coefficient) in addition to the amplitude of the coefficient.

JPEG encoding compresses data in five ways (cont.)

- An end-of-block (EOB) code compresses data by indicating that the data in the rest of the scan are zeroes.
- Variable-length Huffman codes are selected such that shorter codes are used for frequently occurring run-length/coefficient sizes and longer codes are used for less-frequently occurring run-length/coeffi- cient sizes.
- There is a unique Huffman code for each combination of run-length and coefficient size. There are separate tables for AC and DC Huffman codes because they exhibit different characteristics

JPEG-Lite

- The input has been modified to 4x4 blocks as opposed to the 8x8 blocks used in the JPEG standard in order to reduce the layout effort of hardware elements
- A simplified Huffman table will be used by the Encoder and Decoder that contains 10-bit Huffman codes and allows a maximum run-length of 3. The JPEG-baseline standard contains 16-bit Huffman codes and supports a maximum run-length of 15.
- Only a single AC and a single DC Huffman table will be used. In the JPEG baseline standard, two AC and DC tables were supported.
- 1-bit Huffman codes are not allowed in JPEG-lite. • Only grayscale images will be decoded in JPEG-lite.

What is in a JPEG bitstream?

- There are two sets of Huffman codes
  - DC codes and AC codes
- DC Huffman codes are used to represent the first coefficient in the 4x4 block
- AC Huffman codes are used to represent the remaining coefficients

- Coefficients are the transformed values of the pixels in the 4x4 block
- Only the non-zero coefficients are explicitly passed in the bitstream which improves the compression ratio.

Definitions: a 4x4 block encoded on a bitstream.

15	0	2	1
1	1	0	0
0	4	0	0
0	0	5	0

DC Term

AC Terms

Definitions:

Coefficient: Pixel value after having been transformed by the JPEG algorithm

DC term: The first coefficient in the upper-left corner.

AC terms: The remaining coefficients in the 4x4 block.

Coeff_size: The number of binary bits needed to represent the coefficient. (0-10 bits)

Run-length: The number of zeros preceding a non-zero coefficient. (Range of 0-3 zeroes allowed)

EOB: End-of-block. If the remaining coefficients are all zero, a special Huffman code indicates that the end of the 4x4 block has been reached.

Example: a 4x4 block encoded on a bitstream.

15	0	2	1
1	1	0	0
0	4	0	0
0	0	5	0

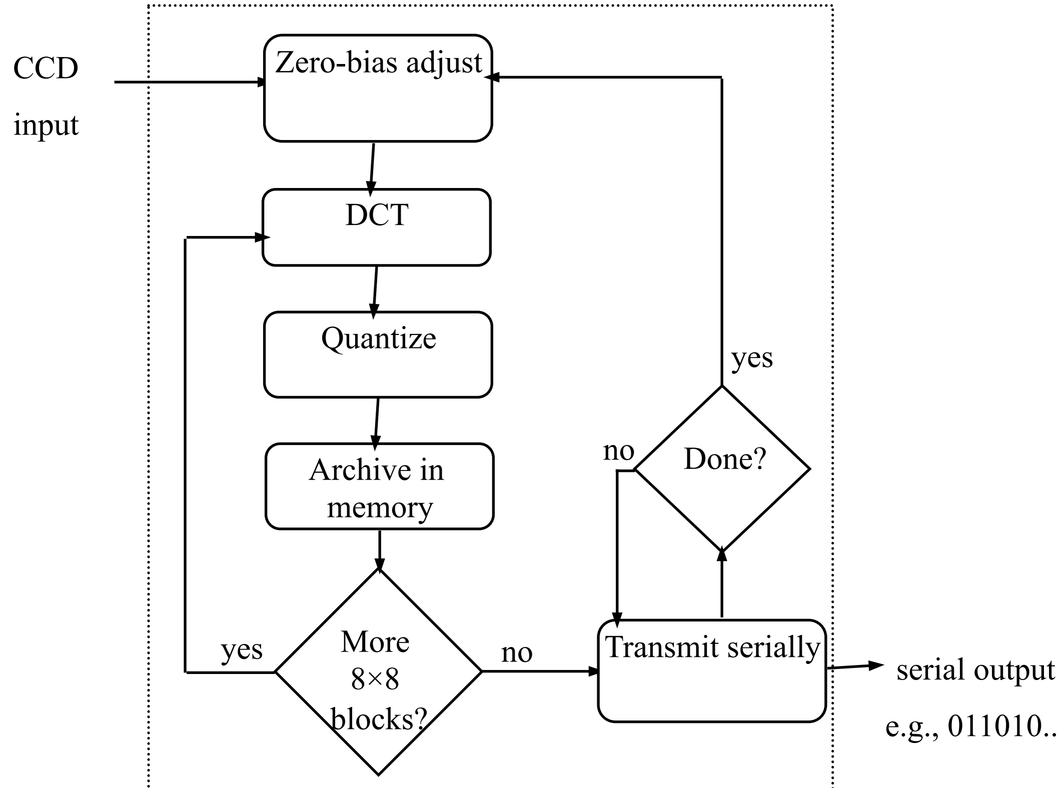
DC Coefficient of previous block=12

Run-length	Coeff_size	Huffman Code	Coefficient
0	2	011	11 (3=15-12)
1	2	11011	10 (2)
0	1	00	1 (1)
0	1	00	1 (1)
0	1	00	1 (1)
3	3	111110101	100 (4)
3	0	111001	none
0	3	100	101 (5)
0	0	1010	none - EOB

Bitstream = 01111110111000100100111111010110011100110010110 10

Informal Functional Specification

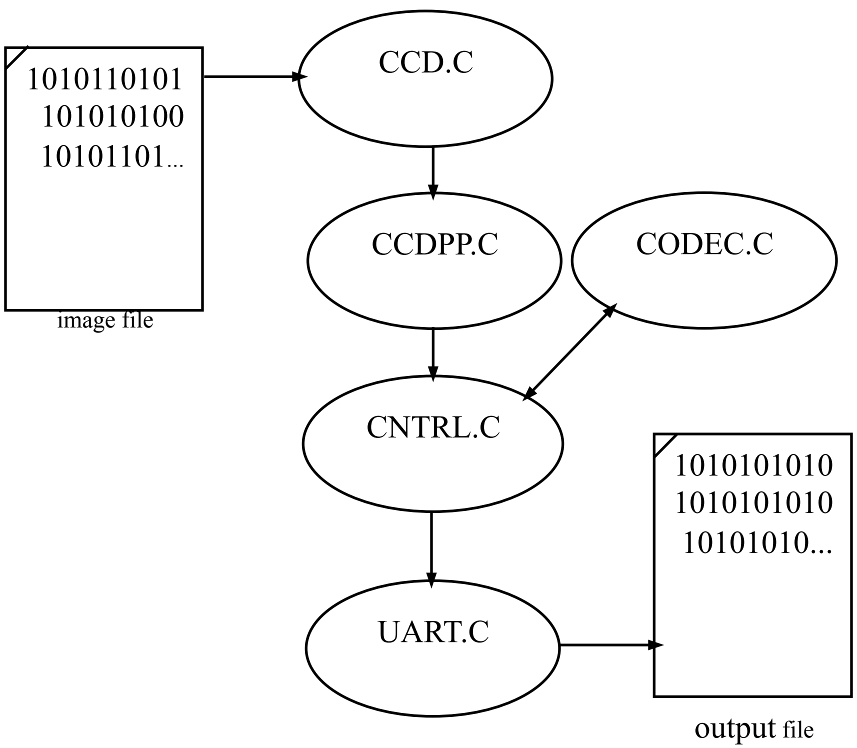
- Flowchart breaks functionality down into simpler functions
- Each function's details could then be described in English
- Low quality image has resolution of 64 x 64
- Mapping functions to a particular processor type not done at this stage



Refined Functional Specification

- Refine informal specification into one that can actually be executed
- Can use C/C++ code to describe each function
  - Called system-level model, prototype, or simply model
  - Also is first implementation
- Can provide insight into operations of system
  - Profiling can find computationally intensive functions
- Can obtain sample output used to verify correctness of final implementation

Executable model of digital camera



CCD Module

- Simulates real CCD
- *CcdInitialize* is passed name of image file
- *CcdCapture* reads “image” from file
- *CcdPopPixel* outputs pixels one at a time

```
#include <stdio.h>
#define SZ_ROW       64
#define SZ_COL       (64 + 2)
static FILE *imageFileHandle;
static char buffer[SZ_ROW][SZ_COL];
static unsigned rowIndex, colIndex;
```

```
char CcdPopPixel(void) {
    char pixel;
    pixel = buffer[rowIndex][colIndex];
    if( ++colIndex == SZ_COL ) {
        colIndex = 0;
        if( ++rowIndex == SZ_ROW ) {
            colIndex = -1;
            rowIndex = -1;
        }
    }
    return pixel;
}
```

```
void CcdInitialize(const char *imageFileName) {
    imageFileHandle = fopen(imageFileName, "r");
    rowIndex = -1;
    colIndex = -1;
}
```

```
void CcdCapture(void) {
    int pixel;
    rewind(imageFileHandle);
    for(rowIndex=0; rowIndex<SZ_ROW; rowIndex++) {
        for(colIndex=0; colIndex<SZ_COL; colIndex++) {
            if( fscanf(imageFileHandle, "%i", &pixel) == 1 ) {
                buffer[rowIndex][colIndex] = (char)pixel;
            }
        }
    }
    rowIndex = 0;
    colIndex = 0;
}
```

CCDPP Module

- Performs zero-bias adjustment
- *CcdppCapture* uses *CcdCapture* and *CcdPopPixel* to obtain image
- Performs zero-bias adjustment after each row read in

```
void CcdppCapture(void) {
    char bias;
    CcdCapture();
    for(rowIndex=0; rowIndex<SZ_ROW; rowIndex++) {
        for(colIndex=0; colIndex<SZ_COL; colIndex++) {
            buffer[rowIndex][colIndex] = CcdPopPixel();
        }
        bias = (CcdPopPixel() + CcdPopPixel()) / 2;
        for(colIndex=0; colIndex<SZ_COL; colIndex++) {
            buffer[rowIndex][colIndex] -= bias;
        }
    }
    rowIndex = 0;
    colIndex = 0;
}
```

```
#define SZ_ROW      64
#define SZ_COL      64
static char buffer[SZ_ROW][SZ_COL];
static unsigned rowIndex, colIndex;
```

```
void CcdppInitialize() {
    rowIndex = -1;
    colIndex = -1;
}
```

```
char CcdppPopPixel(void) {
    char pixel;
    pixel = buffer[rowIndex][colIndex];
    if( ++colIndex == SZ_COL ) {
        colIndex = 0;
        if( ++rowIndex == SZ_ROW ) {
            colIndex = -1;
            rowIndex = -1;
        }
    }
    return pixel;
}
```

UART Module

- Actually a half UART
  - Only transmits, does not receive
- *UartInitialize* is passed name of file to output to
- *UartSend* transmits (writes to output file) bytes at a time

```
#include <stdio.h>
static FILE *outputFileHandle;
void UartInitialize(const char *outputFileName) {
    outputFileHandle = fopen(outputFileName, "w");
}
void UartSend(char d) {
    fprintf(outputFileHandle, "%i\n", (int)d);
}
```

CODEC Module

- Models FDCT encoding
- *ibuffer* holds original 8 x 8 block
- *obuffer* holds encoded 8 x 8 block
- *CodecPushPixel* called 64 times to fill *ibuffer* with original block
- *CodecDoFdct* called once to transform 8 x 8 block
  - Explained in next slide
- *CodecPopPixel* called 64 times to retrieve encoded block from *obuffer*

```
static short ibuffer[8][8], obuffer[8][8], idx;

void CodecInitialize(void) { idx = 0; }
```

```
void CodecPushPixel(short p) {
    if( idx == 64 ) idx = 0;
    ibuffer[idx / 8][idx % 8] = p; idx++;
}
```

```
void CodecDoFdct(void) {
    int x, y;
    for(x=0; x<8; x++) {
        for(y=0; y<8; y++)
            obuffer[x][y] = FDCT(x, y, ibuffer);
    }
    idx = 0;
}
```

```
short CodecPopPixel(void) {
    short p;
    if( idx == 64 ) idx = 0;
    p = obuffer[idx / 8][idx % 8]; idx++;
    return p;
}
```

CODEC Module (cont.)

- Implementing FDCT formula
  C(h) = if (h == 0) then 1/sqrt(2) else 1.0
  F(u,v) = ¼ x C(u) x C(v) $\Sigma_{x=0..7}$ $\Sigma_{y=0..7}$ D_{xy} x cos(π(2x + 1)u/16) x cos(π(2y + 1)v/16)
- Only 64 possible inputs to *COS*, so table can be used to save performance time
  - Floating-point values multiplied by 32,678 and rounded to nearest integer
  - 32,678 chosen in order to store each value in 2 bytes of memory
  - Fixed-point representation explained more later
- *FDCT* unrolls inner loop of summation, implements outer summation as two consecutive for loops

```
static short ONE_OVER_SQRT_TWO = 23170;
static double COS(int xy, int uv) {
    return COS_TABLE[xy][uv] / 32768.0;
}
static double C(int h) {
    return h ? 1.0 : ONE_OVER_SQRT_TWO / 32768.0;
}
```

```
static const short COS_TABLE[8][8] = {
    { 32768,  32138,  30273,  27245,  23170,  18204,  12539,   6392 },
    { 32768,  27245,  12539,  -6392, -23170, -32138, -30273, -18204 },
    { 32768,  18204, -12539, -32138, -23170,   6392,  30273,  27245 },
    { 32768,   6392, -30273, -18204,  23170,  27245, -12539, -32138 },
    { 32768,  -6392, -30273,  18204,  23170, -27245, -12539,  32138 },
    { 32768, -18204, -12539,  32138, -23170,  -6392,  30273, -27245 },
    { 32768, -27245,  12539,   6392, -23170,  32138, -30273,  18204 },
    { 32768, -32138,  30273, -27245,  23170, -18204,  12539,  -6392 }
};
```

```
static int FDCT(int u, int v, short img[8][8]) {
    double s[8], r = 0; int x;
    for(x=0; x<8; x++) {
     s[x] = img[x][0] * COS(0, v) + img[x][1] * COS(1, v) +
            img[x][2] * COS(2, v) + img[x][3] * COS(3, v) +
            img[x][4] * COS(4, v) + img[x][5] * COS(5, v) +
            img[x][6] * COS(6, v) + img[x][7] * COS(7, v);
    }
    for(x=0; x<8; x++) r += s[x] * COS(x, u);
    return (short)(r * .25 * C(u) * C(v));
}
```

CNTRL (Controller) Module

- Heart of the system
- *CntrlInitialize* for consistency with other modules only
- *CntrlCaptureImage* uses CCDPP module to input image and place in buffer
- *CntrlCompressImage* breaks the 64 x 64 buffer into 8 x 8 blocks and performs FDCT on each block using the CODEC module
  - Also performs quantization on each block
- *CntrlSendImage* transmits encoded image serially using UART module

```
void CntrlCaptureImage(void) {
    CcdppCapture();
    for(i=0; i<SZ_ROW; i++)
        for(j=0; j<SZ_COL; j++)
            buffer[i][j] = CcdppPopPixel();
}
```

```
#define SZ_ROW          64
#define SZ_COL          64
#define NUM_ROW_BLOCKS  (SZ_ROW / 8)
#define NUM_COL_BLOCKS  (SZ_COL / 8)
static short buffer[SZ_ROW][SZ_COL], i, j, k, l, temp;
void CntrlInitialize(void) {}
```

```
void CntrlSendImage(void) {
    for(i=0; i<SZ_ROW; i++)
        for(j=0; j<SZ_COL; j++) {
            temp = buffer[i][j];
            UartSend(((char*)&temp)[0]);     /* send upper byte */
            UartSend(((char*)&temp)[1]);     /* send lower byte */
        }
    }
}
```

```
void CntrlCompressImage(void) {
    for(i=0; i<NUM_ROW_BLOCKS; i++)
        for(j=0; j<NUM_COL_BLOCKS; j++) {
            for(k=0; k<8; k++)
                for(l=0; l<8; l++)
                    CodecPushPixel(
                        (char)buffer[i * 8 + k][j * 8 + l]);
            CodecDoFdct();/* part 1 - FDCT */
            for(k=0; k<8; k++)
                for(l=0; l<8; l++) {
                    buffer[i * 8 + k][j * 8 + l] = CodecPopPixel();
                    /* part 2 - quantization */
                    buffer[i*8+k][j*8+l] >>= 6;
                }
        }
}
```

Putting it All Together

- *Main* initializes all modules, then uses CNTRL module to capture, compress, and transmit one image
- This system-level model can be used for extensive experimentation
  - Bugs much easier to correct here rather than in later models

```
int main(int argc, char *argv[]) {
    char *uartOutputFileName = argc > 1 ? argv[1] : "uart_out.txt";
    char *imageFileName = argc > 2 ? argv[2] : "image.txt";
    /* initialize the modules */
    UartInitialize(uartOutputFileName);
    CcdInitialize(imageFileName);
    CcdppInitialize();
    CodecInitialize();
    CntrlInitialize();
    /* simulate functionality */
    CntrlCaptureImage();
    CntrlCompressImage();
    CntrlSendImage();
}
```

Final issues

- Come by my office hours (right after class)

- Any questions or concerns?