

## Administrative Stuff

---

- Homework 3 due Now
- Homework 4 posted on WWW
  - Due Thursday 10/24



## Digital Camera Example

## Outline

---

- Introduction to a simple digital camera
- Designer's perspective
- Requirements specification
- Design
  - Four implementations



3

*CS122A: Embedded System Design, Fall 02*

## Introduction

---

- Putting it all together
  - General-purpose processor
  - Single-purpose processor
    - Custom
    - Standard
  - Memory
  - Interfacing
- General-purpose vs. single-purpose processors
- Functionality partitioning among processor types



4

*CS122A: Embedded System Design, Fall 02*

## Introduction to a simple digital camera

---

- 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
- “Optional” features for real digital camera
  - Image sizing/zooming, image deletion, special effects,...etc



## Two Key Tasks

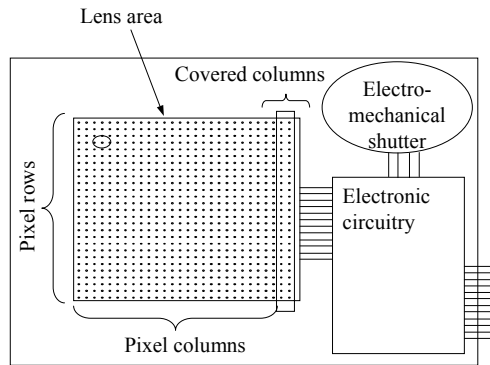
---

- Processing images and storing in memory
  - When shutter pressed:
    - Image captured
    - Converted to digital form by charge-coupled device (CCD)
    - Compressed and archived in internal memory
- Uploading images to PC
  - Software commands camera to transmit images serially



## Charge-coupled device (CCD)

- Special sensor that captures an image
- Light-sensitive silicon solid-state device
  - Composed of many cells

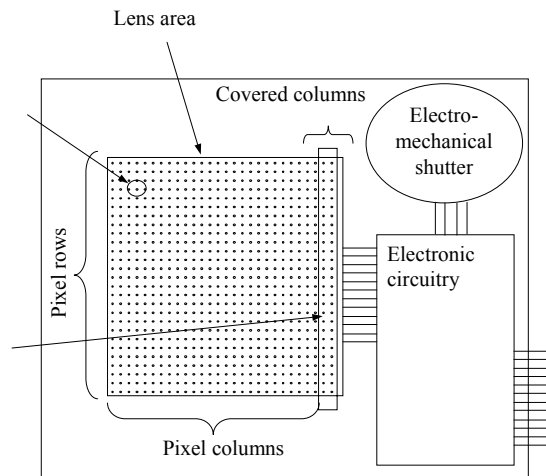


7

CS122A: Embedded System Design, Fall 02

## Charge-coupled device (CCD)

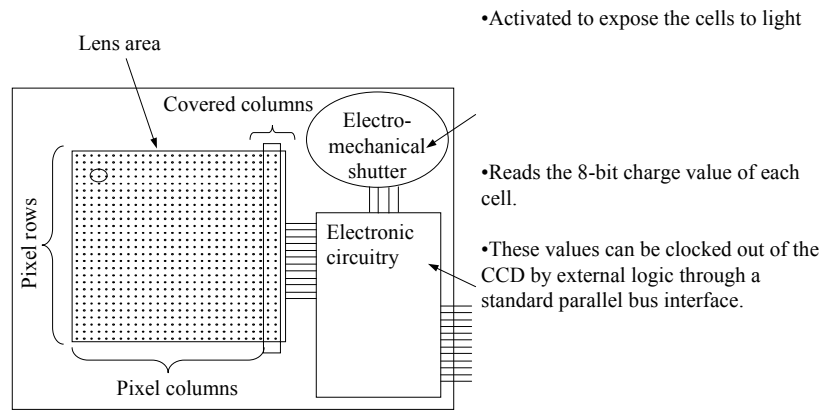
- When exposed to light, each cell becomes electrically charged.
- The charge can then be converted to a 8-bit value where 0 represents no exposure while 255 represents very intense exposure.
- covered with a black strip of paint.
- used for zero-bias adjustments for that row.



8

CS122A: Embedded System Design, Fall 02

## Charge-coupled device (CCD)



CS122A: Embedded System Design, Fall 02

9

## Zero-bias error

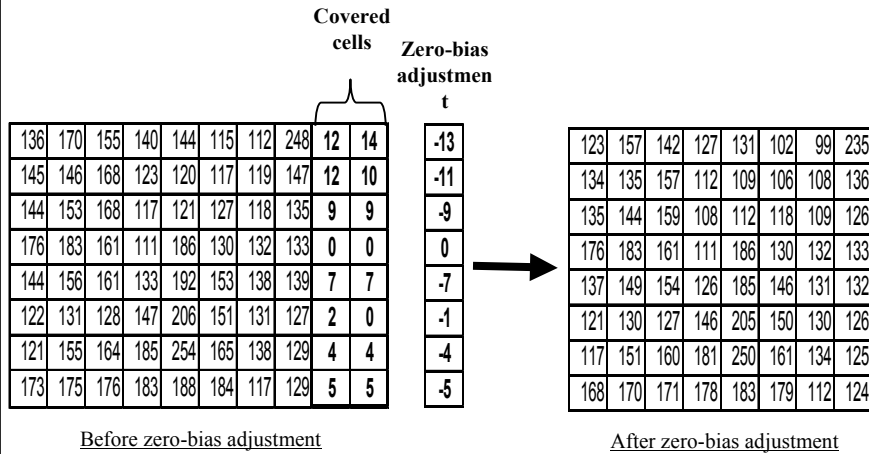
- Manufacturing errors
  - Cause cells to measure above or below actual light intensity
- Different cells in a row
  - Same error
- Different cells in a column
  - Different error
- Some columns blocked by black paint to detect zero-bias error
  - Reading of other than 0 in blocked cells is zero-bias error
  - Each row is corrected by
    - subtracting the average error found in blocked cells for that row



CS122A: Embedded System Design, Fall 02

10

## Zero-bias error



## Compression

- Store more images
- Transmit image to PC in less time
- JPEG (Joint Photographic Experts Group)
  - Popular standard for representing digital images in a compressed form
  - Provides for a number of different modes of operation
  - Mode used here provides compression using Discrete Cosine Transform
  - Image data divided into blocks of 8 x 8 pixels
  - 3 steps performed on each block
    - DCT
    - Quantization
    - Huffman encoding



## DCT step

- Transforms original 8 x 8 block into cosine-frequency domain
  - Upper-left corner values represent more of the essence of the image
  - Lower-right corner values represent finer details
    - Can reduce precision of these values and retain reasonable image quality
- FDCT (Forward DCT) formula
  - $C(h) =$ 
    - if  $(h == 0)$  then  $1/\sqrt{2}$  else 1.0
    - Auxiliary function used in main function  $F(u,v)$
  - $F(u,v) = 1/4 * C(u) * C(v) * \sum_{x=0..7} \sum_{y=0..7} D_{xy} * \cos(\pi(2x+1)u/16) * \cos(\pi(2y+1)v/16)$ 
    - Gives encoded pixel at row  $u$ , column  $v$
    - $D_{xy}$  is original pixel value at row  $x$ , column  $y$
- IDCT (Inverse DCT)
  - Reverses process to obtain original block (not needed for this design)



## DCT Step

123	157	142	127	131	102	99	235
134	135	157	112	109	106	108	136
135	144	159	108	112	118	109	126
176	183	161	111	186	130	132	133
137	149	154	126	185	146	131	132
121	130	127	146	205	150	130	126
117	151	160	181	250	161	134	125
168	170	171	178	183	179	112	124



1150	39	-43	-10	26	-83	11	41
-81	-3	115	-73	-6	-2	22	-5
14	-11	1	-42	26	-3	17	-38
2	-61	-13	-12	36	-23	-18	5
44	13	37	-4	10	-21	7	-8
36	-11	-9	-4	20	-28	-21	14
-19	-7	21	-6	3	3	12	-21
-5	-13	-11	-17	-4	-1	7	-4



## Quantization step

- Achieve high compression by reducing image quality
  - Reduce bit precision of encoded data
    - Fewer bits needed for encoding
    - One way is to divide all values by a factor of 2
  - Dequantization would reverse process for decompression

1150	39	-43	-10	26	-83	11	41
-81	-3	115	-73	-6	-2	22	-5
14	-11	1	-42	26	-3	17	-38
2	-61	-13	-12	36	-23	-18	5
44	13	37	-4	10	-21	7	-8
36	-11	-9	-4	20	-28	-21	14
-19	-7	21	-6	3	3	12	-21
-5	-13	-11	-17	-4	-1	7	-4

Divide each cell's value by 8



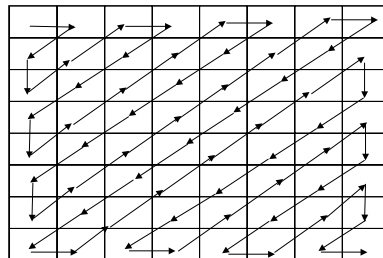
144	5	-5	-1	3	-10	1	5
-10	0	14	-9	-1	0	3	-1
2	-1	0	-5	3	0	2	-5
0	-8	-2	-2	5	-3	-2	1
6	2	5	-1	1	-3	1	-1
5	-1	-1	-1	3	-4	-3	2
-2	-1	3	-1	0	0	2	-3
-1	-2	-1	-2	-1	0	1	-1

After being decoded using DCT

After quantization

## 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 encoding example

- Pixel frequencies
  - Pixel value -1 occurs 15 times
  - Pixel value 14 occurs 1 time

144	5	-5	-1	3	-10	1	5
-10	0	14	-9	-1	0	3	-1
2	-1	0	-5	3	0	2	-5
0	-8	-2	-2	5	-3	-2	1
6	2	5	-1	1	-3	1	-1
5	-1	-1	-1	3	-4	-3	2
-2	-1	3	-1	0	0	2	-3
-1	-2	-1	-2	-1	0	1	-1



Pixel frequencies

-1	15x
0	8x
-2	6x
1	5x
2	5x
3	5x
5	5x
-3	4x
-5	3x
-10	2x
144	1x
-9	1x
-8	1x
-4	1x
6	1x
14	1x

17

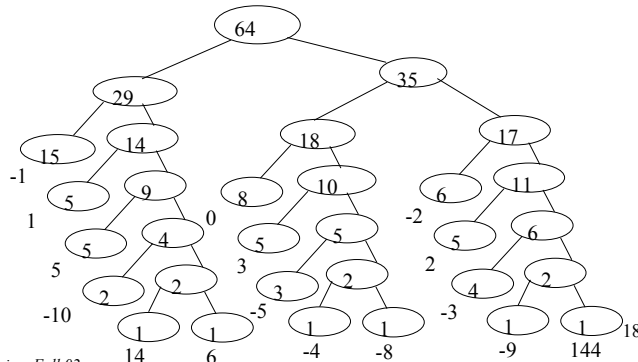
## Huffman encoding example

- Build Huffman (binary) tree from bottom up
  - Create one leaf node for each pixel value
  - Assign frequency as node's value
  - Create an internal node by joining any two minimum-sum nodes
  - This sum is internal nodes value

Pixel frequencies

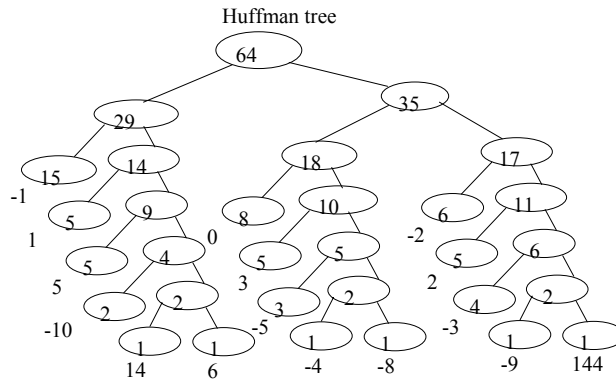
-1	15x
0	8x
-2	6x
1	5x
2	5x
3	5x
5	5x
-3	4x
-5	3x
-10	2x
144	1x
-9	1x
-8	1x
-4	1x
6	1x
14	1x

Huffman tree



## Huffman encoding example

- Traverse tree from root to leaf to obtain binary code
  - Append 0 for left traversal, 1 for right traversal
- Huffman encoding is reversible
  - No code is a prefix of another code



Huffman codes

-1	00
0	100
-2	110
1	010
2	1110
3	1010
5	0110
-3	11110
-5	10110
-10	01110
144	111111
-9	111110
-8	101111
-4	101110
6	011111
14	011110

19

CS122A: Embedded System Design, Fall 02

## Transmitting...

Huffman codes

144	5	-5	-1	3	-10	1	5
-10	0	14	-9	-1	0	3	-1
2	-1	0	-5	3	0	2	-5
0	-8	-2	-2	5	-3	-2	1
6	2	5	-1	1	-3	1	-1
5	-1	-1	-1	3	-4	-3	2
-2	-1	3	-1	0	0	2	-3
-1	-2	-1	-2	-1	0	1	-1

-1	00
0	100
-2	110
1	010
2	1110
3	1010
5	0110
-3	11110
-5	10110
-10	01110
144	111111
-9	111110
-8	101111
-4	101110
6	011111
14	011110

- 111111 0110 01110 1110 100 10100 00 011110 00 100 011111  
.....

144

20

CS122A: Embedded System Design, Fall 02

## Outline

---

- Introduction to a simple digital camera
- Designer's perspective
- Requirements specification
- Design
  - Four implementations



21

*CS122A: Embedded System Design, Fall 02*

## Requirements Specification

---

- System's requirements – what system should do
  - Nonfunctional requirements
    - Constraints on design metrics (e.g., “should use 0.001 watt or less”)
  - Functional requirements
    - System's behavior (e.g., “output X should be input Y times 2”)
  - Initial specification may be very general and come from marketing dept.
    - captures and stores at least 50 low-res (64x64) images and uploads to PC,
    - costs around \$100 with single medium-size IC costing less than \$25,
    - has long as possible battery life,
    - has expected sales volume of 200,000 if market entry < 6 months,
    - 100,000 if between 6 and 12 months,
    - insignificant sales beyond 12 months



22

*CS122A: Embedded System Design, Fall 02*

## Nonfunctional requirements

- Design metrics of importance based on initial specification
  - Performance: time required to process image
  - Size: max number of logic gates (2-input NAND gate) in IC
  - Power: measure of avg. electrical energy consumed while processing
  - Energy: battery lifetime (power x time)
- Constrained metrics
  - Values must be “better” than certain threshold
- Optimization metrics
  - Improved as much as possible to improve product
- Metric can be both constrained and optimization

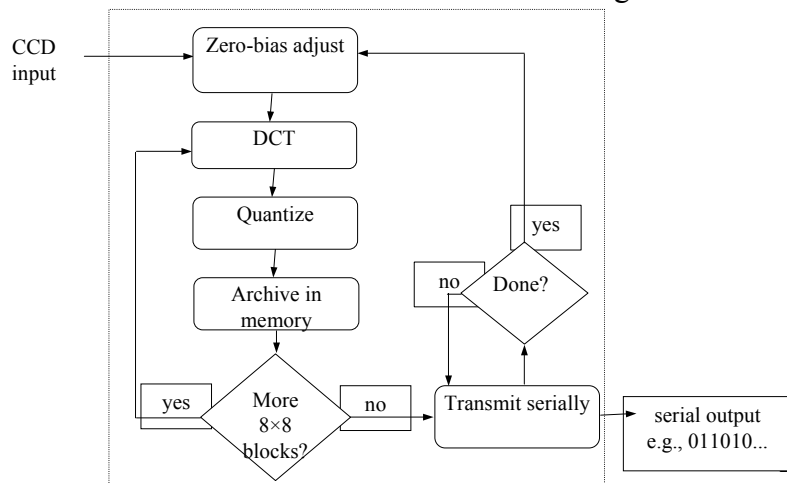


23

CS122A: Embedded System Design, Fall 02

## Informal functional specification

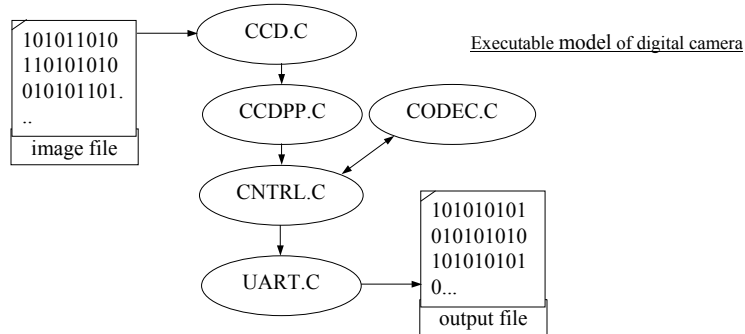
- Flowchart breaks functionality into simpler functions
- Each function’s details could be described in English



CS122A: Embedded System Design, Fall 02

## Refined functional specification

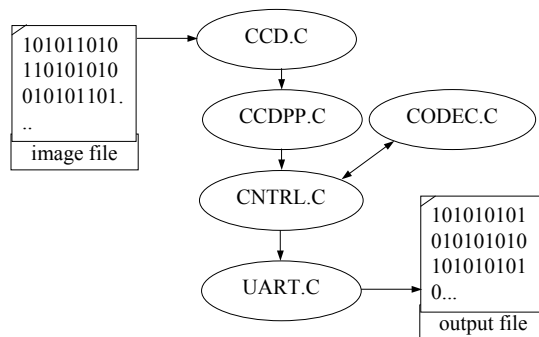
- Refine informal specification into executable specification
  - Can use C/C++ code to describe each function
    - May also be the first implementation
  - Can provide insight into operations of system
    - Profiling can find computationally intensive functions
  - Can obtain sample output used to verify correctness of design



25

## CCD module

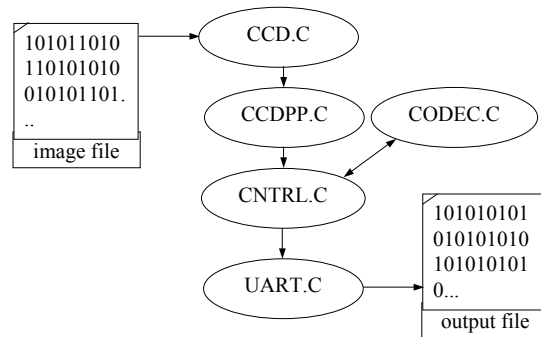
- Simulates real CCD
  - Start with an image file
  - Reads “image” from file
  - Outputs pixels one at a time



26

## CCDPP (CCD PreProcessing) module

- Performs zero-bias adjustment after each row read in

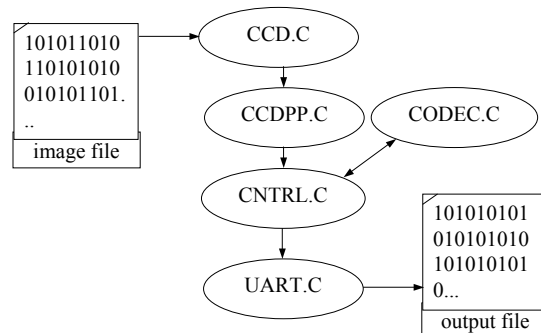


27

CS122A: Embedded System Design, Fall 02

## 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

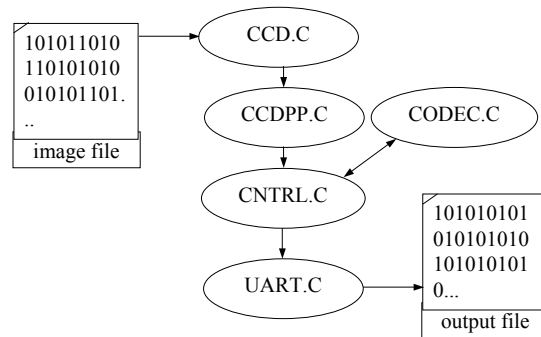


28

CS122A: Embedded System Design, Fall 02

## CODEC module

- Models FDCT encoding
- transform 8 x 8 block



29

CS122A: Embedded System Design, Fall 02

## CODEC (cont.)

- Implementing FDCT formula
  - $C(h) = \text{if } (h == 0) \text{ then } 1/\sqrt{2} \text{ else } 1.0$
  - $F(u,v) = \frac{1}{4} \times C(u) \times C(v) \sum_{x=0..7} \sum_{y=0..7} D_{xy} \times \cos(\pi(2x+1)u/16) \times \cos(\pi(2y+1)v/16)$

```

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;
}
  
```

30

CS122A: Embedded System Design, Fall 02

## CODEC (cont.)

- $C(h) = \text{if } (h == 0) \text{ then } 1/\sqrt{2} \text{ else } 1.0$   
 $F(u,v) = \frac{1}{4} X C(u) X C(v) \sum_{x=0..7} \sum_{y=0..7} D_{xy} X \cos(\pi(2x+1)u/16) X \cos(\pi(2y+1)v/16)$
- Only 64 possible inputs to *COS*, so use table to save performance time
  - Floating-point values multiplied by 32,768 and rounded to nearest integer
  - 32,768 chosen in order to store each value in 2 bytes of memory

```
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 }
};
```

31

CS122A: Embedded System Design, Fall 02

## CODEC (cont.)

- Implementing FDCT formula  
 $C(h) = \text{if } (h == 0) \text{ then } 1/\sqrt{2} \text{ else } 1.0$   
 $F(u,v) = \frac{1}{4} X C(u) X C(v) \sum_{x=0..7} \sum_{y=0..7} D_{xy} X \cos(\pi(2x+1)u/16) X \cos(\pi(2y+1)v/16)$
- *FDCT*
  - unrolls inner loop of summation
  - implements outer summation as two consecutive for loops

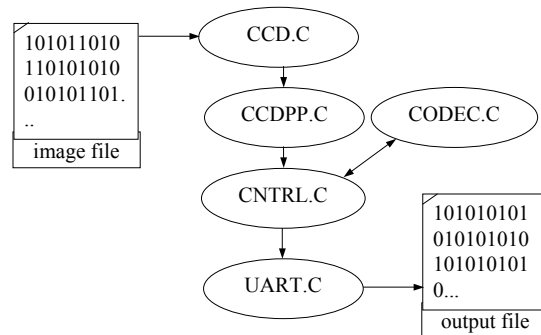
```
static int FDCT(int u, int v, short img[8][8]) {
  double s[8], r = 0; int x;
  for(x=0; x<8; x++) {
    s[x] = img[x][0] * COS(0, v) + img[x][1] * COS(1, v) +
           img[x][2] * COS(2, v) + img[x][3] * COS(3, v) +
           img[x][4] * COS(4, v) + img[x][5] * COS(5, v) +
           img[x][6]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

32

CS122A: Embedded System Design, Fall 02

## CNTRL (controller) module

- Uses CCDPP to input image & place in buffer
- Break the 64 x 64 buffer into 8 x 8 blocks and performs FDCT on each block using the CODEC module
- Transmits encoded image serially using UART module



33

CS122A: Embedded System Design, Fall 02

## Putting it all together

- *Main*
  - initializes all modules,
  - uses CNTRL module to capture, compress, and transmit one image
- This system-level model can be used for 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();
}
  
```

34

CS122A: Embedded System Design, Fall 02

## Outline

---

- Introduction to a simple digital camera
- Designer's perspective
- Requirements specification
- Design
  - Four implementations



35

*CS122A: Embedded System Design, Fall 02*

## Design

---

- Determine system's architecture
- Map functionality to that architecture
- Implementation
  - A particular architecture and mapping
  - Solution space
    - Set of all implementations
- Starting point for the digital camera
  - Low-end general-purpose processor connected to flash memory
    - All functionality mapped to software running on processor
    - Usually satisfies power, size, and time-to-market constraints
  - If timing constraint not satisfied then later implementations could
    - either use single-purpose processors for time-critical function
    - or rewrite functional specification



36

*CS122A: Embedded System Design, Fall 02*

## Implementation 1: Microcontroller alone

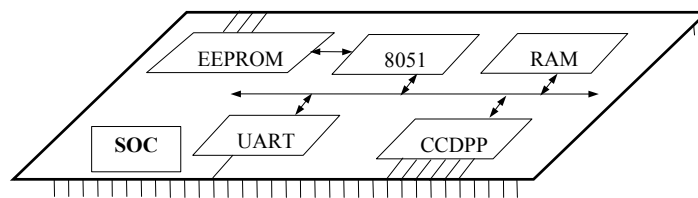
- Low-end processor could be Intel 8051 microcontroller
- Total IC cost including NRE about \$5
- Well below 200 mW power
- Time-to-market about 3 months
- However, one image per second not possible
  - Assume 12 MHz, 12 cycles per instruction
    - Executes one million instructions per second
  - CcdppCapture has nested loops resulting in 4096 (64 x 64) iterations
    - ~100 assembly instructions each iteration
    - 409,000 (4096 x 100) instructions per image
    - Half of budget for reading image alone
  - Would be over budget after adding compute-intensive tasks
    - DCT and Huffman encoding



CS122A: Embedded System Design, Fall 02

37

## Implementation 2: Microcontroller and CCDPP



- CCDPP on custom single-purpose processor
  - Improves performance – less microcontroller cycles
  - Increases NRE cost and time-to-market
  - Easy to implement
    - Simple datapath
    - Few states in controller
- UART also implement as single-purpose processor
- EEPROM for program and RAM for data added as well



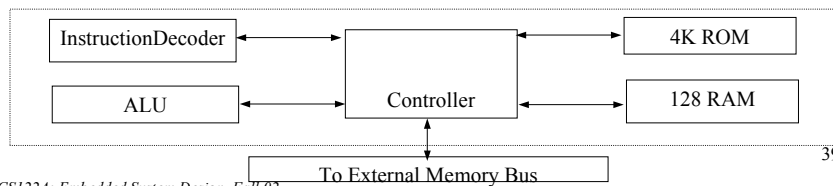
CS122A: Embedded System Design, Fall 02

38

## Microcontroller

- Synthesizable version of Intel 8051 available
  - Written in VHDL
  - Captured at register transfer level (RTL)
- Fetches instruction from ROM
- Decodes using Instruction Decoder
- ALU executes arithmetic operations
  - Source and destination registers reside in RAM
- Special data movement instructions used to load/store externally
- Program generates VHDL for ROM from output of C compiler

Block diagram of Intel 8051 processor core



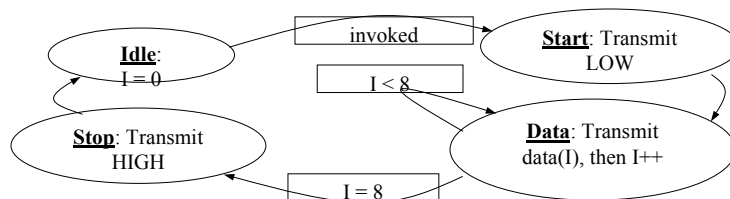
CS122A: Embedded System Design, Fall 02

39

## UART

- UART in idle mode until invoked
  - invoked when 8051 executes store with UART's enable register as target
- Memory-mapped communication between 8051 and all SSPs
- Start state transmits 0
  - indicating start of byte transmission then transitions to Data state
- Data state sends 8 bits serially then transitions to Stop state
- Stop state transmits 1
  - indicating transmission done then transitions back to idle mode

FSMD description of UART



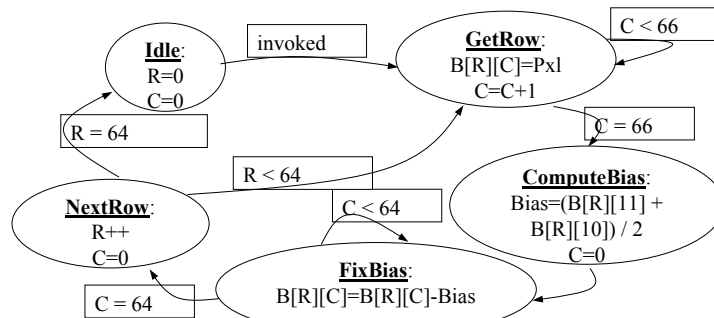
CS122A: Embedded System Design, Fall 02

40

## CCDPP

- Hardware implementation of zero-bias operations
- Interacts with external CCD chip
  - Combining CCD with ordinary logic not feasible
- Internal buffer, B, memory-mapped to 8051
- Variables R, C are buffer's row, column indices

FSMD description of CCDPP



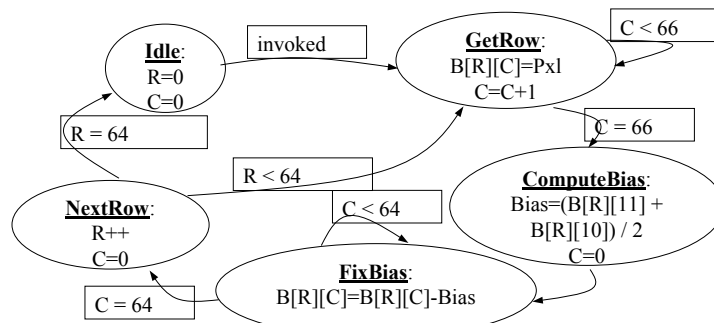
41

CS122A: Embedded System Design, Fall 02

## CCDPP

- GetRow state reads in one row from CCD to B
  - 66 bytes: 64 pixels + 2 blacked-out pixels
- ComputeBias computes bias for that row and stores in Bias
- FixBias state iterates over same row subtracting Bias
- NextRow transitions to GetRow for repeat of process

FSMD description of CCDPP

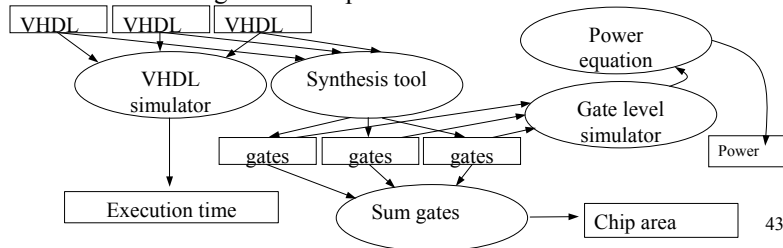


42

CS122A: Embedded System Design, Fall 02

## Analysis

- Entire SOC tested on VHDL simulator
  - Interprets VHDL descriptions and functionally simulates execution
  - Tests for correct functionality
  - Measures clock cycles to process one image (performance)
- Gate-level description obtained through synthesis
  - Synthesis tool like compiler for SPPs
  - Simulate gate-level models to obtain data for power analysis
    - Number of times gates switch from 1 to 0 or 0 to 1
  - Count number of gates for chip area



CS122A: Embedded System Design, Fall 02

## Implementation 2: Microcontroller and CCDPP

- Analysis of implementation 2
  - Total execution time for processing one image:
    - 9.1 seconds
  - Power consumption:
    - 0.033 watt
  - Energy consumption:
    - 0.30 joule (9.1 s x 0.033 watt)
  - Total chip area:
    - 98,000 gates

44

CS122A: Embedded System Design, Fall 02

## Implementation 3: Microcontroller and CCDPP/Fixed-Point DCT

- 9.1 seconds still doesn't meet performance constraint
  - Need 1 second
- DCT operation prime candidate for improvement
  - Microprocessor spends most cycles here
    - Shown by execution of implementation 2
  - Could design custom hardware like we did for CCDPP
    - More complex so more design effort
  - Could speed up DCT functionality by modifying behavior



## DCT floating-point cost

- Floating-point cost
  - DCT uses ~260 floating-point operations per pixel transformation
  - 4096 (64 x 64) pixels per image
  - 1 million floating-point operations per image
  - No floating-point support with Intel 8051
    - Compiler must emulate
    - Generates procedures for each floating-point operation (mult, add)
    - Each procedure uses tens of integer operations
  - Thus, > 10 million integer operations per image
  - Procedures increase code size
- Fixed-point arithmetic can improve on this



## Fixed-point arithmetic

- Integer used to represent a real number
  - Constant number of integer's bits represents fractional portion of real
    - More bits, more accurate the representation
  - Remaining bits represent portion of real number before decimal point
- Translating a real constant to a fixed-point representation
  - Multiply real value by  $2^{\#}$  (# of bits used for fractional part)
  - Round to nearest integer
  - E.g., represent 3.14 as 8-bit integer with 4 bits for fraction
    - $2^4 = 16$
    - $3.14 \times 16 = 50.24 \approx 50 = 00110010$
    - 16 ( $2^4$ ) possible values for fraction, each represents 0.0625 (1/16)
    - Last 4 bits (0010) = 2
    - $2 \times 0.0625 = 0.125$
    - $3(0011) + 0.125 = 3.125 \approx 3.14$
    - (more bits for fraction would increase accuracy)



CS122A: Embedded System Design, Fall 02

47

## Fixed-point arithmetic operations

- Addition
  - Simply add integer representations
  - E.g.,  $3.14 + 2.71 = 5.85$ 
    - $3.14 \rightarrow 50 = 00110010$
    - $2.71 \rightarrow 43 = 00101011$
    - $50 + 43 = 93 = 01011101$
    - $5(0101) + 13(1101) \times 0.0625 = 5.8125 \approx 5.85$
- Multiply
  - Multiply integer representations
  - Shift result right by # of bits in fractional part
  - E.g.,  $3.14 * 2.71 = 8.5094$ 
    - $50 * 43 = 2150 = 100001100110$
    - $\gg 4 = 10000110$
    - $8(1000) + 6(0110) \times 0.0625 = 8.375 \approx 8.5094$



CS122A: Embedded System Design, Fall 02

48

## Fixed-point implementation of CODEC

- COS\_TABLE gives 8-bit fixed-point cosine values

```
static const char code COS_TABLE[8][8] = {
    { 64, 62, 59, 53, 45, 35, 24, 12 },
    { 64, 53, 24, -12, -45, -62, -59, -35 },
    { 64, 35, -24, -62, -45, 12, 59, 53 },
    { 64, 12, -59, -35, 45, 53, -24, -62 },
    { 64, -12, -59, 35, 45, -53, -24, 62 },
    { 64, -35, -24, 62, -45, -12, 59, -53 },
    { 64, -53, 24, 12, -45, 62, -59, 35 },
    { 64, -62, 59, -53, 45, -35, 24, -12 }
};
```

```
static const char ONE_OVER_SQRT_TWO = 5;
static short xdata inBuffer[8][8], outBuffer[8][8], idx;
void CodecInitialize(void) { idx = 0; }
```



49

CS122A: Embedded System Design, Fall 02

## Fixed-point implementation of CODEC

- 6 bits used for fractional portion
- Result of multiplications shifted right by 6

```
void CodecPushPixel(short p) {
    if( idx == 64 ) idx = 0;
    inBuffer[idx / 8][idx % 8] = p >> 6; idx++;
}
```

```
void CodecDoFdct(void) {
    unsigned short x, y;
    for(x=0; x<8; x++)
        for(y=0; y<8; y++)
            outBuffer[x][y] = F(x, y, inBuffer);
    idx = 0;
}
```



50

CS122A: Embedded System Design, Fall 02

## Implementation 3: Microcontroller and CCDPP/Fixed-Point DCT

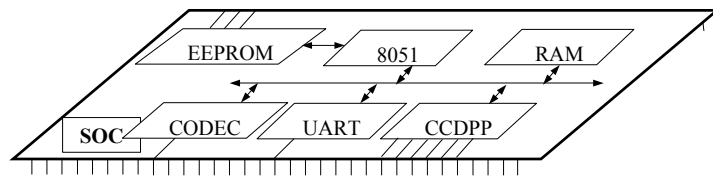
- Analysis of implementation 3
  - Use same analysis techniques as implementation 2
  - Total execution time for processing one image:
    - 1.5 seconds
  - Power consumption:
    - 0.033 watt (same as 2)
  - Energy consumption:
    - 0.050 joule (1.5 s x 0.033 watt)
    - Battery life 6x longer!!
  - Total chip area:
    - 90,000 gates
    - 8,000 less gates (less memory needed for code)



CS122A: Embedded System Design, Fall 02

51

## Implementation 4: Microcontroller and CCDPP/DCT



- Performance close but not good enough
- Must resort to implementing CODEC in hardware
  - Single-purpose processor to perform DCT on 8 x 8 block



CS122A: Embedded System Design, Fall 02

52

## CODEC design

- 4 memory mapped registers
  - C\_DATAI\_REG/C\_DATAO\_REG used to push/pop 8 x 8 block into and out of CODEC
  - C\_CMND\_REG used to command CODEC
    - Writing 1 to this register invokes CODEC
  - C\_STAT\_REG indicates CODEC done, ready for next block
    - Polled in software
- Direct translation of C code to VHDL for actual hardware implementation
  - Fixed-point version used



53

*CS122A: Embedded System Design, Fall 02*

## Implementation 4: Microcontroller and CCDPP/DCT

- Analysis of implementation 4
  - Total execution time for processing one image:
    - 0.099 seconds (well under 1 sec)
  - Power consumption:
    - 0.040 watt
    - Increase over 2 and 3 because SOC has another processor
  - Energy consumption:
    - 0.00040 joule (0.099 s x 0.040 watt)
    - Battery life 12x longer than previous implementation!!
  - Total chip area:
    - 128,000 gates
    - Significant increase over previous implementations



54

*CS122A: Embedded System Design, Fall 02*

## Summary of implementations

- Implementation 3
  - Close in performance
  - Cheaper
  - Less time to build
- Implementation 4
  - Great performance and energy consumption
  - More expensive and may miss time-to-market window
    - If DCT designed ourselves then increased NRE cost and time-to-market
    - If existing DCT purchased then increased IC cost
- Which is better?

	Implementation 2	Implementation 3	Implementation 4
Performance (second)	9.1	1.5	0.099
Power (watt)	0.033	0.033	0.040
Size (gate)	98,000	90,000	128,000
Energy (joule)	0.30	0.050	0.0040

55

CS122A: Embedded System Design, Fall 02

## Homework 1

- 90-100    1
- 80-89    12
- 70-79    7
- 60-69    5
- 0-59     4



- You have ONE week from the day it is handed back for regrade...

56

CS122A: Embedded System Design, Fall 02

## Quiz 1

---

- 90-100 2
- 80-89 9
- 70-79 9
- 60-69 4
- 0-60 5



- 5% of your grade
- Midterm counts 25%...

