

Embedded Systems Design: A Unified Hardware/Software Introduction

Chapter 7 Digital Camera Example

Outline

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

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
- Very simple description used for example
 - Many more features with real digital camera
 - Variable size images, image deletion, digital stretching, zooming in and out, etc.

Designer's perspective

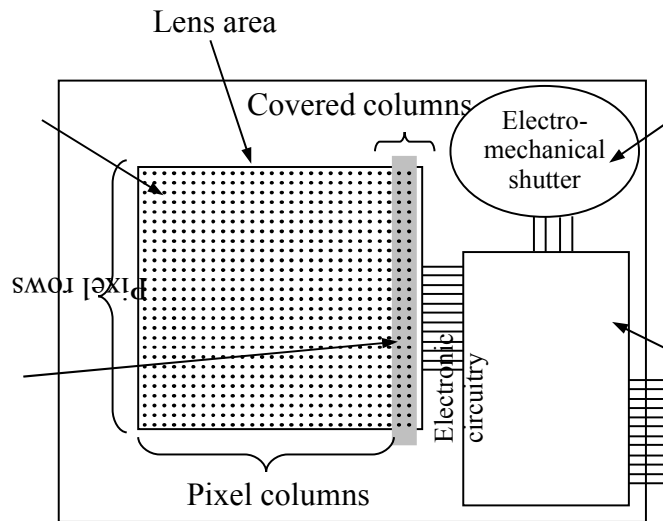
- Two key tasks
 - Processing images and storing in memory
 - When shutter pressed:
 - Image captured
 - Converted to digital form by charge-coupled device (CCD)
 - Compressed and archived in internal memory
 - Uploading images to PC
 - Digital camera attached to PC
 - Special software commands camera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

Charge-coupled device (CCD)

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

When exposed to light, each cell becomes electrically charged. This charge can then be converted to a 8-bit value where 0 represents no exposure while 255 represents very intense exposure of that cell to light.

Some of the columns are covered with a black strip of paint. The light-intensity of these pixels is used for zero-bias adjustments of all the cells.

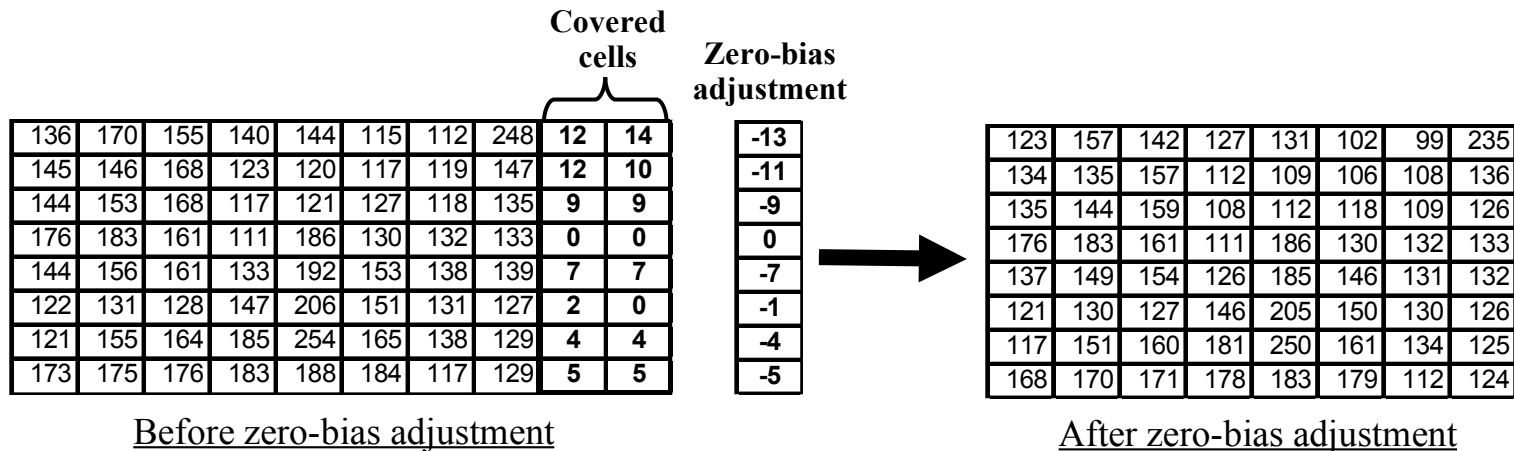


The electromechanical shutter is activated to expose the cells to light for a brief moment.

The electronic circuitry, when commanded, discharges the cells, activates the electromechanical shutter, and then reads the 8-bit charge value of each cell. These values can be clocked out of the CCD by external logic through a standard parallel bus interface.

Zero-bias error

- Manufacturing errors cause cells to measure slightly above or below actual light intensity
- Error typically same across columns, but different across rows
- Some of right most 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



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
 - Provides for a number of different modes of operation
 - 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

DCT step

- Transforms original 8 x 8 block into a 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
 - $C(h) = \text{if } (h == 0) \text{ then } 1/\sqrt{2} \text{ else } 1.0$
 - Auxiliary function used in main function $F(u,v)$
 - $F(u,v) = \frac{1}{4} \times C(u) \times C(v) \sum_{x=0..7} \sum_{y=0..7} D_{xy} \times \cos(\pi(2u + 1)x/16) \times \cos(\pi(2y + 1)y/16)$
 - 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)

Quantization step

- Achieve high compression ratio 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
 - Simple right shifts can do this
 - 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

After being decoded using DCT

Divide each cell's
value by 8

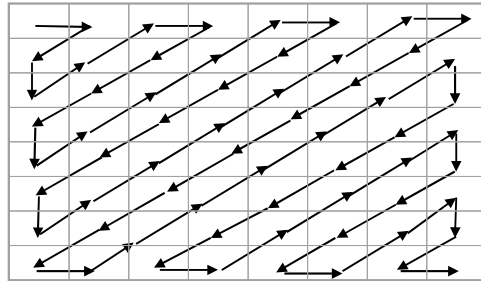


144	5	-5	-1	3	-10	1	5
-10	0	14	-9	-1	0	3	-1
2	-1	0	-5	3	0	2	-5
0	-8	-2	-2	5	-3	-2	1
6	2	5	-1	1	-3	1	-1
5	-1	-1	-1	3	-4	-3	2
-2	-1	3	-1	0	0	2	-3
-1	-2	-1	-2	-1	0	1	-1

After 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

Archive step

- Record starting address and image size
 - Can use linked list
- One possible way to archive images
 - If max number of images archived is N:
 - Set aside memory for N addresses and N image-size variables
 - Keep a counter for location of next available address
 - Initialize addresses and image-size variables to 0
 - Set global memory address to $N \times 4$
 - Assuming addresses, image-size variables occupy $N \times 4$ bytes
 - First image archived starting at address $N \times 4$
 - Global memory address updated to $N \times 4 + (\text{compressed image size})$
- Memory requirement based on N, image size, and average compression ratio

Uploading to PC

- When connected to PC and upload command received
 - Read images from memory
 - Transmit serially using UART
 - While transmitting
 - Reset pointers, image-size variables and global memory pointer accordingly

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.
 - E.g., short document detailing market need for a low-end digital camera that:
 - captures and stores at least 50 low-res 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

Nonfunctional requirements

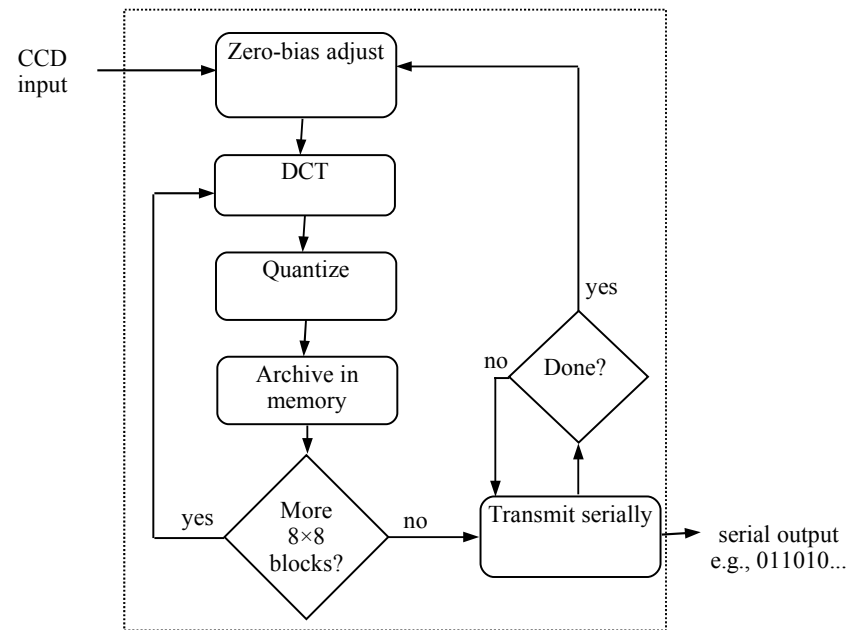
- Design metrics of importance based on initial specification
 - **Performance**: time required to process image
 - **Size**: number of elementary logic gates (2-input NAND gate) in IC
 - **Power**: measure of avg. electrical energy consumed while processing
 - **Energy**: battery lifetime (power x time)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

Nonfunctional requirements (cont.)

- Performance
 - Must process image fast enough to be useful
 - 1 sec reasonable constraint
 - Slower would be annoying
 - Faster not necessary for low-end of market
 - Therefore, constrained metric
- Size
 - Must use IC that fits in reasonably sized camera
 - Constrained and optimization metric
 - Constraint may be 200,000 gates, but smaller would be cheaper
- Power
 - Must operate below certain temperature (cooling fan not possible)
 - Therefore, constrained metric
- Energy
 - Reducing power or time reduces energy
 - Optimized metric: want battery to last as long as possible

Informal functional specification

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



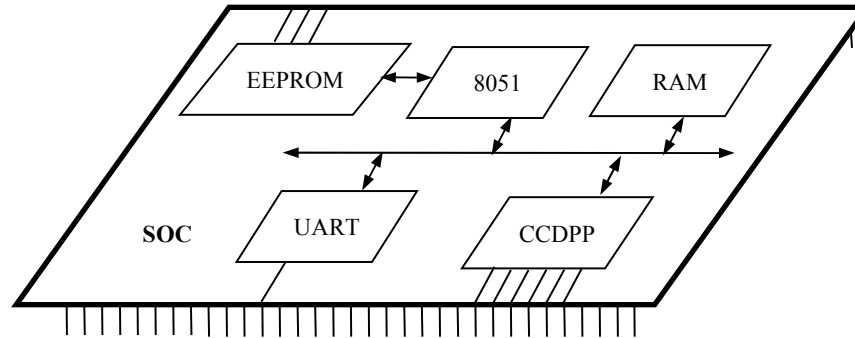
Design

- Determine system's architecture
 - Processors
 - Any combination of single-purpose (custom or standard) or general-purpose processors
 - Memories, buses
- Map functionality to that architecture
 - Multiple functions on one processor
 - One function on one or more processors
- Implementation
 - A particular architecture and mapping
 - Solution space is set of all implementations
- Starting point
 - 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:
 - use single-purpose processors for time-critical functions
 - rewrite functional specification

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
 - 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 DCT and Huffman encoding

Implementation 2: Microcontroller and CCDPP

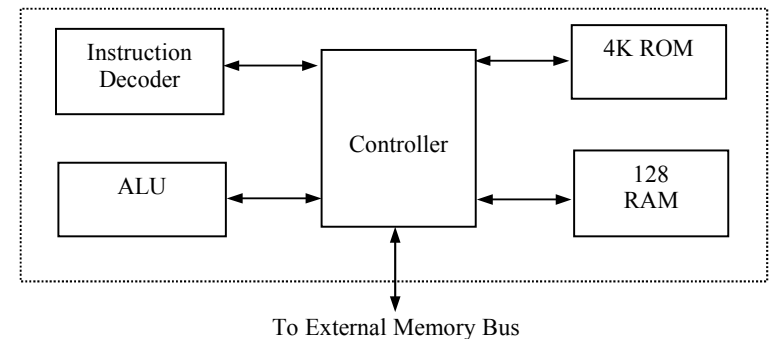


- CCDPP function implemented on custom single-purpose processor
 - Improves performance – less microcontroller cycles
 - Increases NRE cost and time-to-market
 - Easy to implement
 - Simple datapath
 - Few states in controller
- Simple UART easy to implement as single-purpose processor also
- EEPROM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

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 and store externally
- Special program generates VHDL description of ROM from output of C compiler/linker

Block diagram of Intel 8051 processor core



Memory Mapped I/O

- The 8051 has an 8 bit data bus, and a 16 bit address bus. Gives us $2^{16} = 65,536$ addresses.
- Map RAM to address zero and up. ie. If we read/write to an address in this range, the RAM responds.
- Map addresses at 65,535 and down for other devices (UART and CCDP).
- Devices might have 1 or more registers that can be read and/or written. Each register is assigned a unique memory address.

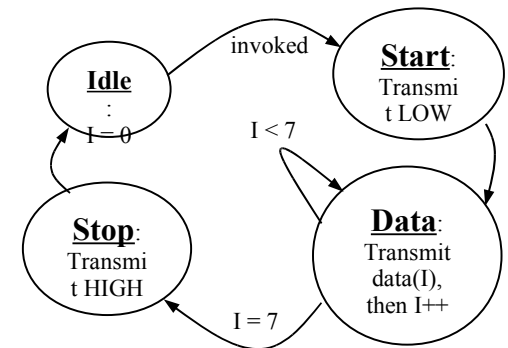
Memory Mapped I/O Cont.

- The device watches address bus, and when it sees an address for one of its registers, it responds.
- For example, if it's a write operation, then the register that corresponds to the address will latch the contents of the data bus.
- If it's a read operation, then it will drive the data bus.
- Creates an easy way for CPU to communicate with other devices via external bus.
- From program, just write 8 bits to the assigned address.

UART

- UART in idle mode until invoked
 - UART invoked when 8051 executes store instruction with UART's enable register as target address
 - Memory-mapped communication between 8051 and all single-purpose processors
 - Lower 8-bits of memory address for RAM
 - Upper 8-bits of memory address for memory-mapped I/O devices
- 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

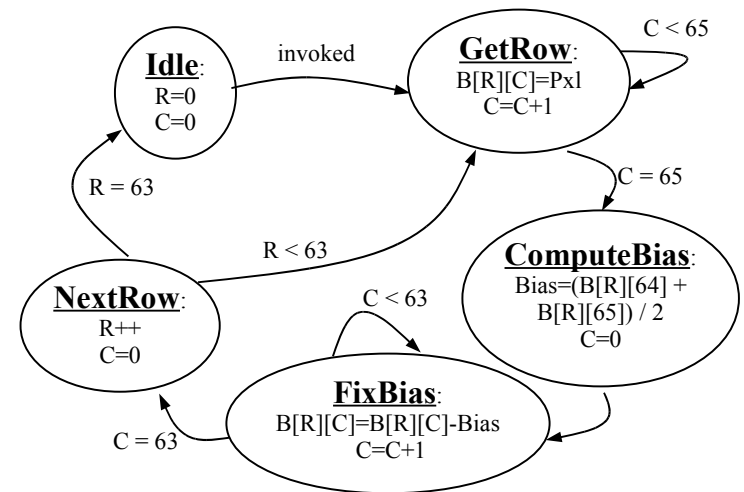
FSMD description of UART



CCDPP

- Hardware implementation of zero-bias operations
- Interacts with external CCD chip
 - CCD chip resides external to our SOC mainly because combining CCD with ordinary logic not feasible
- Internal buffer, B , memory-mapped to 8051
- Variables R , C are buffer's row, column indices
- GetRow state reads in one row from CCD to B
 - 66 bytes: 64 pixels + 2 blacked-out pixels
- ComputeBias state computes bias for that row and stores in variable $Bias$
- FixBias state iterates over same row subtracting $Bias$ from each element
- NextRow transitions to GetRow for repeat of process on next row or to Idle state when all 64 rows completed

FSMD description of CCDPP



Software

- System-level model provides majority of code
 - Module hierarchy, procedure names, and main program unchanged
- Code for UART and CCDPP modules must be redesigned
 - Simply replace with memory assignments
 - *xdata* used to load/store variables over external memory bus
 - *_at_* specifies memory address to store these variables
 - Byte sent to *U_TX_REG* by processor will invoke UART
 - *U_STAT_REG* used by UART to indicate its ready for next byte
 - UART may be much slower than processor
 - Similar modification for CCDPP code
- All other modules untouched

Original code from system-level model

```
#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);
}
```



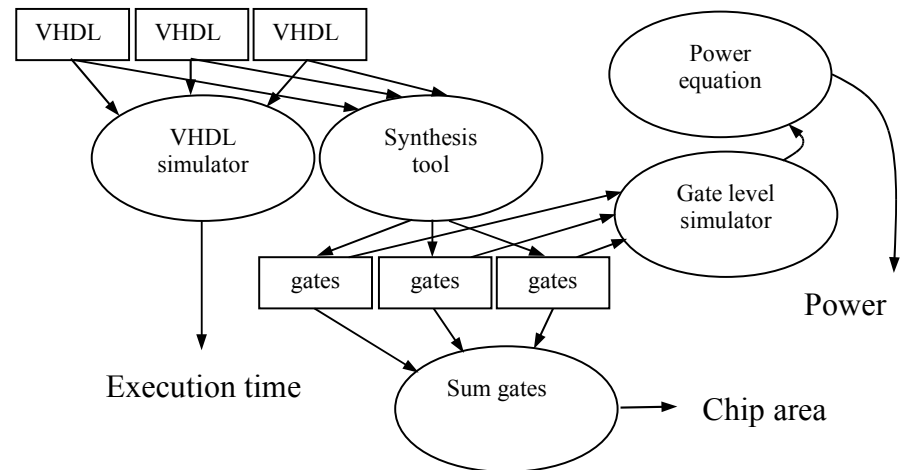
Rewritten UART module

```
static unsigned char xdata U_TX_REG_at_ 65535;
static unsigned char xdata U_STAT_REG_at_ 65534;
void UARTInitialize(void) {}
void UARTSend(unsigned char d) {
    while( U_STAT_REG == 1 ) {
        /* busy wait */
    }
    U_TX_REG = d;
}
```

Analysis

- Entire SOC tested on VHDL simulator
 - Interprets VHDL descriptions and functionally simulates execution of system
 - Recall program code translated to VHDL description of ROM
 - 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

Obtaining design metrics of interest



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

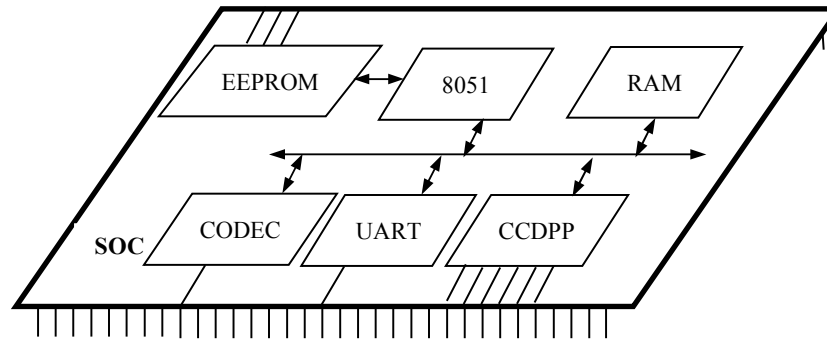
Implementation 3: Microcontroller and CCDPP/Fixed-Point DCT

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

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)

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

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

Summary of implementations

	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

- 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?

Summary

- Digital camera example
 - Specifications in English and executable language
 - Design metrics: performance, power and area
- Several implementations
 - Microcontroller: too slow
 - Microcontroller and coprocessor: better, but still too slow
 - Fixed-point arithmetic: almost fast enough
 - Additional coprocessor for compression: fast enough, but expensive and hard to design
 - Tradeoffs between hw/sw – the main lesson of this book!