EECS 222A: System-on-Chip Description and Modeling Lecture 5

Rainer Dömer

doemer@uci.edu

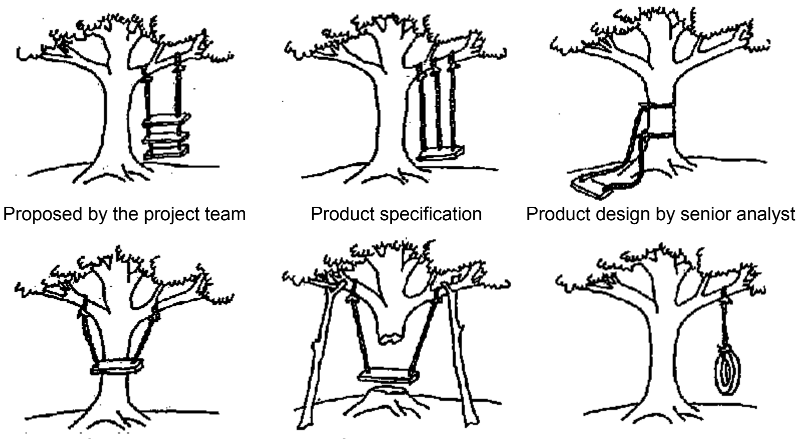
The Henry Samueli School of Engineering
Electrical Engineering and Computer Science
University of California, Irvine

Lecture 4: Overview

- System-on-Chip Specification
  - Essential issues
  - Top-down SoC design flow
  - Specification Model
  - Specification Modeling Guidelines
- Current Research
  - Computer-Aided Recoding
- Project Discussion
  - Digital Camera Example
  - JPEG Application
  - Assignment 2
  - Assignment 3

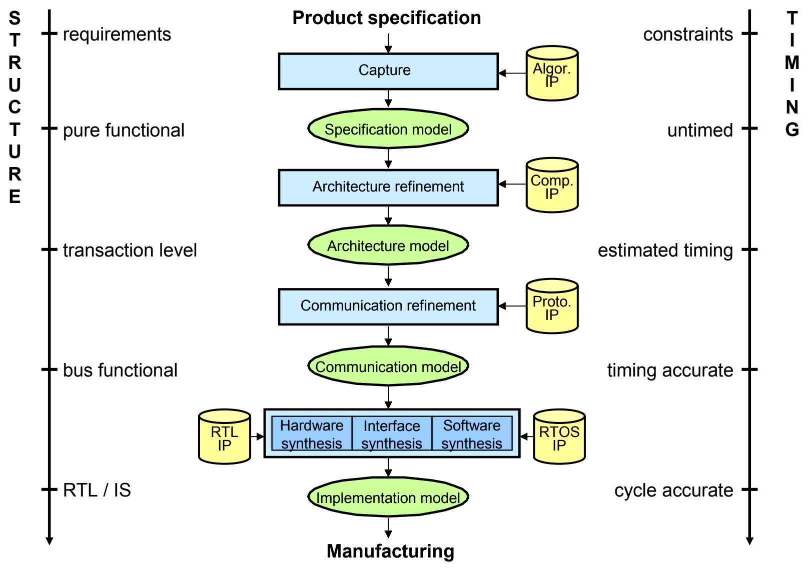
Essential Issues in Specification

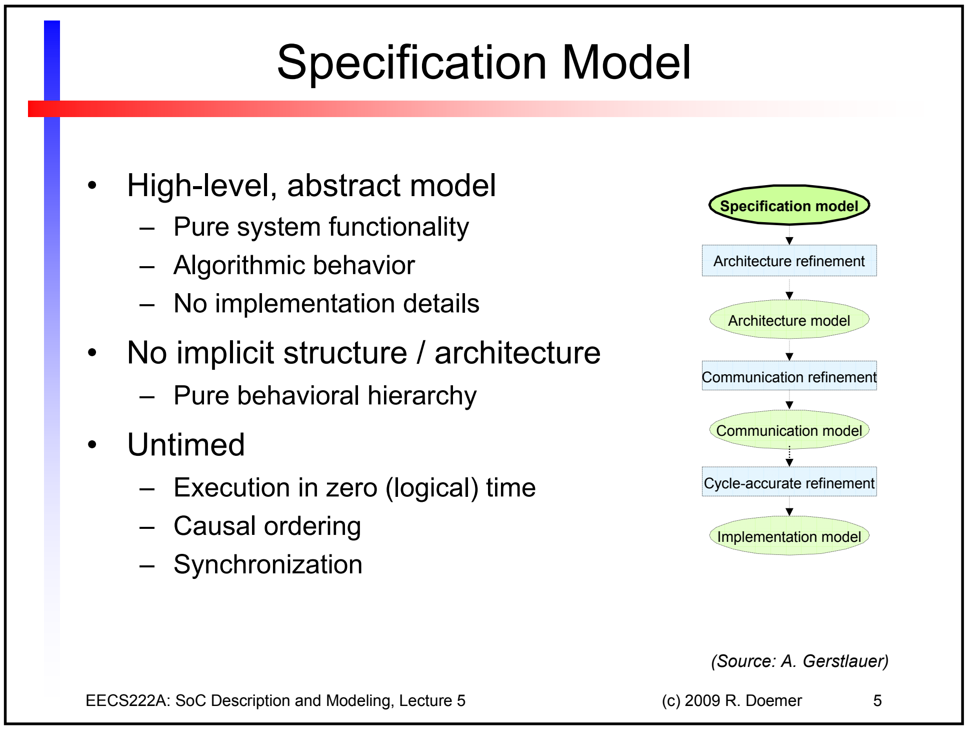
- An Example ...


Proposed by the project team
Product specification
Product design by senior analyst
Product after implementation
Product after acceptance by user
What the user wanted

Source: unknown author

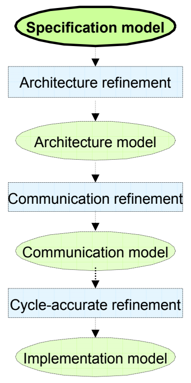
Top-Down SoC Design Flow





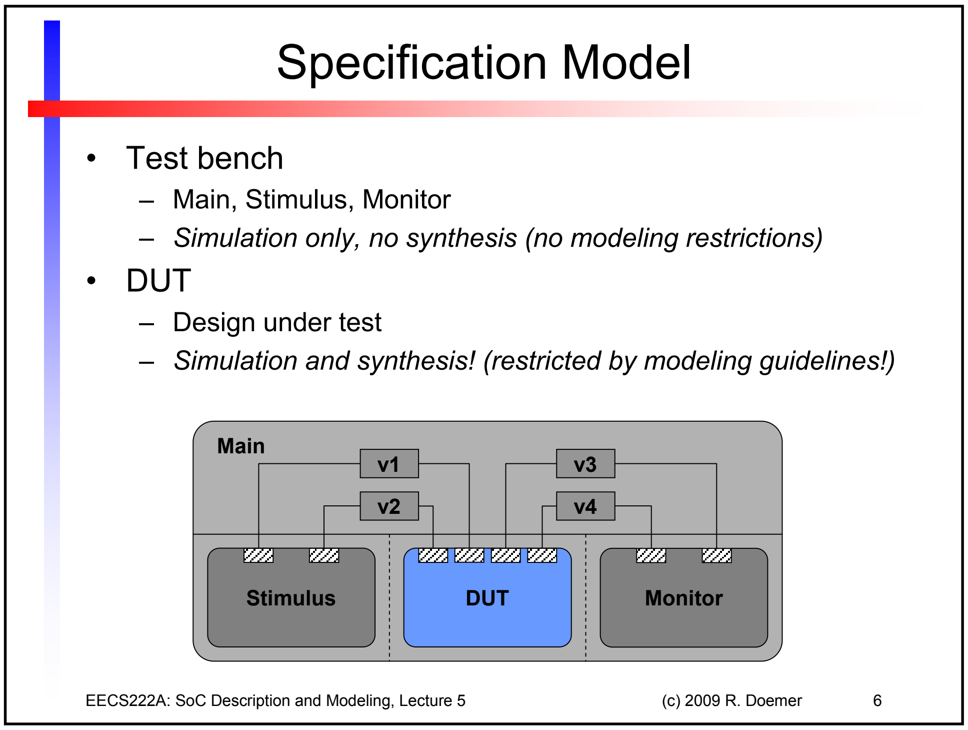
Specification Model

- High-level, abstract model
  - Pure system functionality
  - Algorithmic behavior
  - No implementation details
- No implicit structure / architecture
  - Pure behavioral hierarchy
- Untimed
  - Execution in zero (logical) time
  - Causal ordering
  - Synchronization



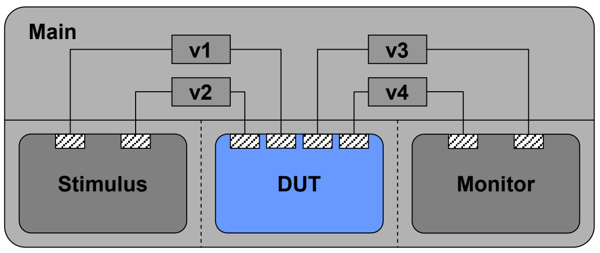
(Source: A. Gerstlauer)

EECS222A: SoC Description and Modeling, Lecture 5 (c) 2009 R. Doemer 5

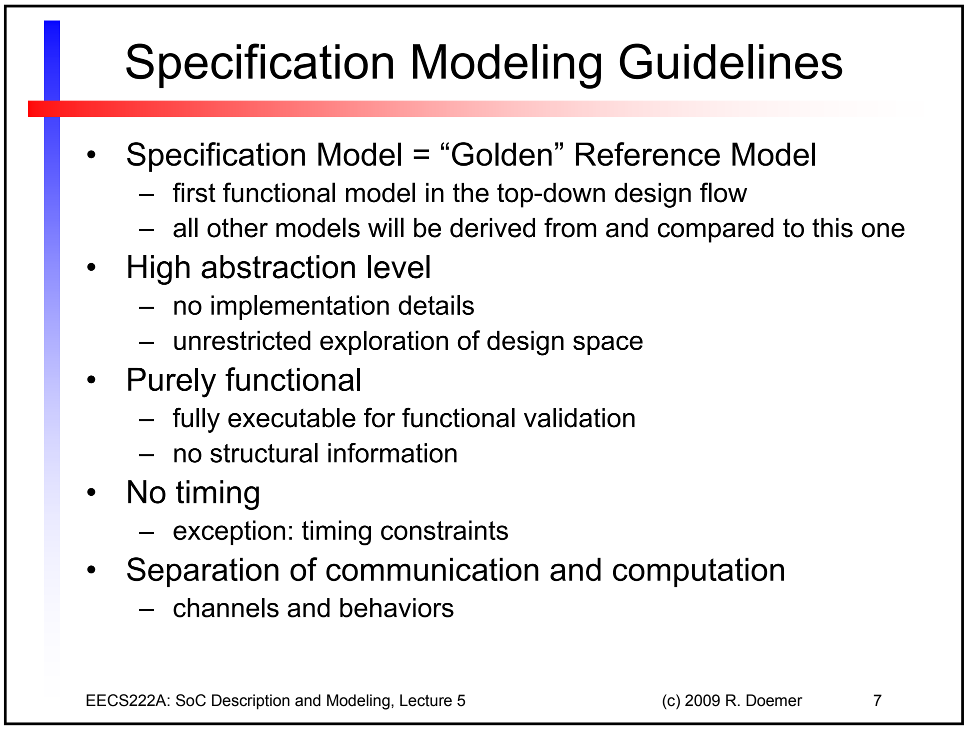


Specification Model

- Test bench
  - Main, Stimulus, Monitor
  - *Simulation only, no synthesis (no modeling restrictions)*
- DUT
  - Design under test
  - *Simulation and synthesis! (restricted by modeling guidelines!)*



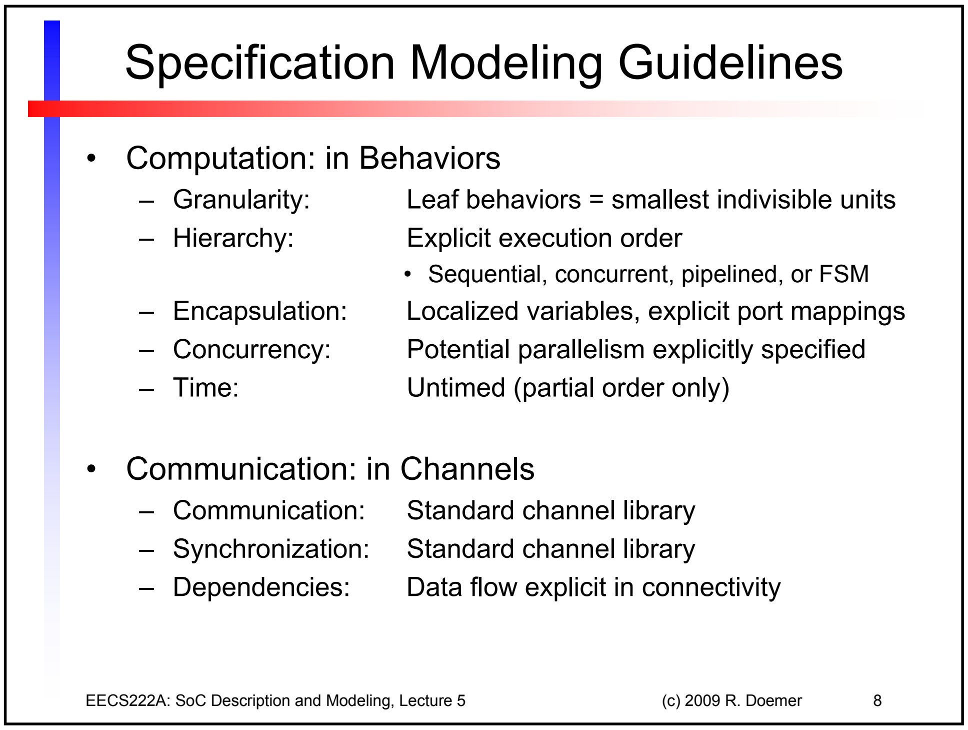
EECS222A: SoC Description and Modeling, Lecture 5 (c) 2009 R. Doemer 6



Specification Modeling Guidelines

- Specification Model = “Golden” Reference Model
  - first functional model in the top-down design flow
  - all other models will be derived from and compared to this one
- High abstraction level
  - no implementation details
  - unrestricted exploration of design space
- Purely functional
  - fully executable for functional validation
  - no structural information
- No timing
  - exception: timing constraints
- Separation of communication and computation
  - channels and behaviors

EECS222A: SoC Description and Modeling, Lecture 5 (c) 2009 R. Doemer 7



Specification Modeling Guidelines

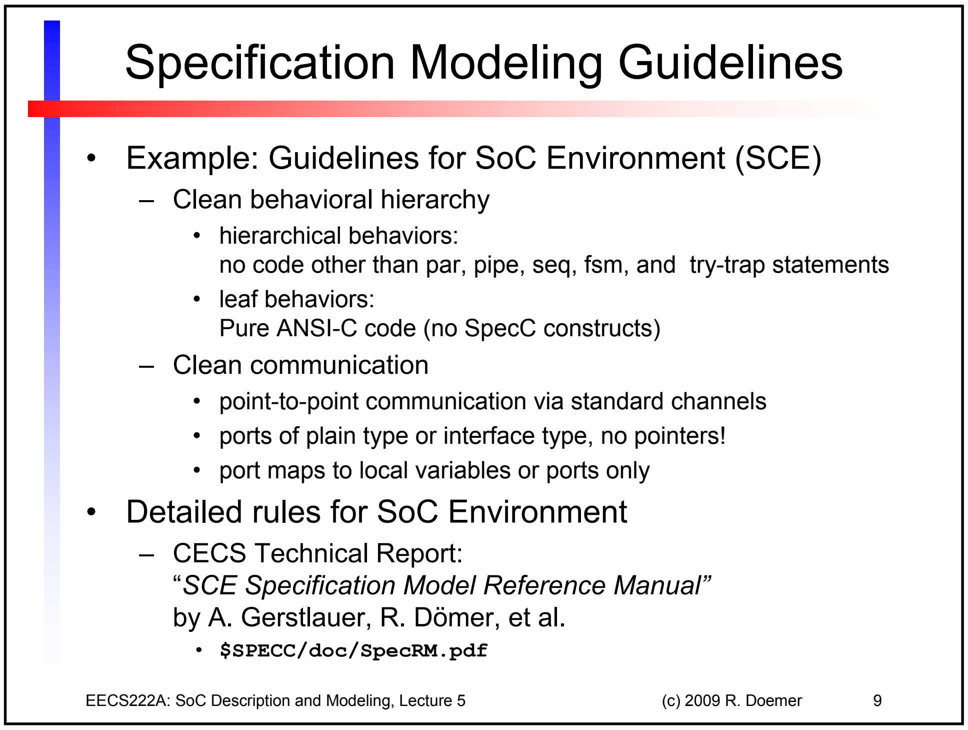
- Computation: in Behaviors

– Granularity:	Leaf behaviors = smallest indivisible units
– Hierarchy:	Explicit execution order
	• Sequential, concurrent, pipelined, or FSM
– Encapsulation:	Localized variables, explicit port mappings
– Concurrency:	Potential parallelism explicitly specified
– Time:	Untimed (partial order only)

- Communication: in Channels

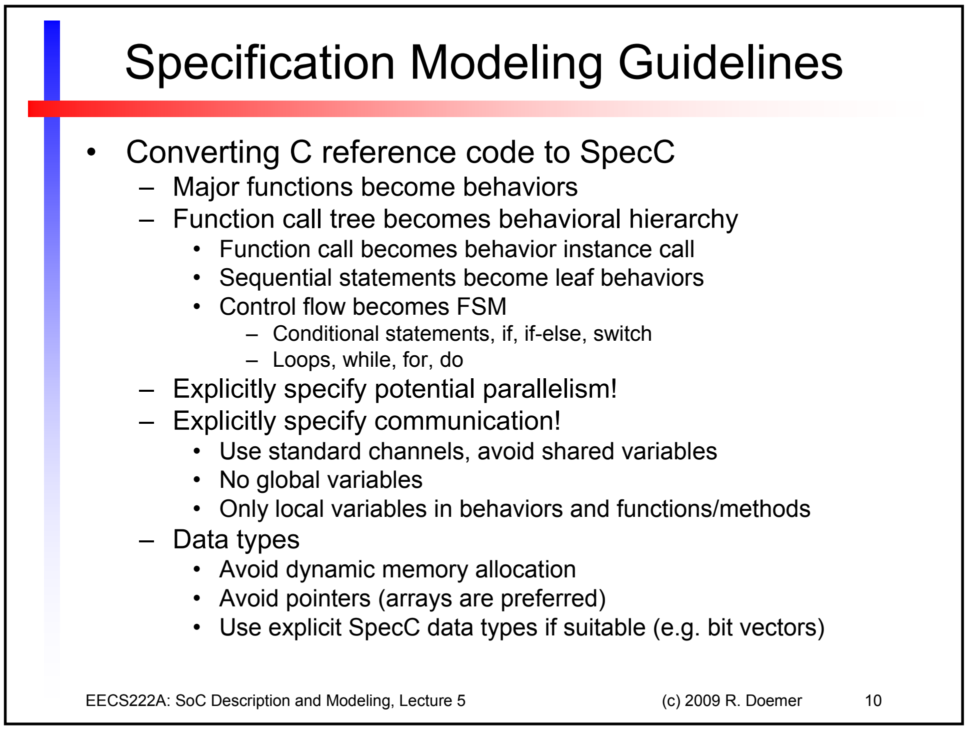
– Communication:	Standard channel library
– Synchronization:	Standard channel library
– Dependencies:	Data flow explicit in connectivity

EECS222A: SoC Description and Modeling, Lecture 5 (c) 2009 R. Doemer 8



Specification Modeling Guidelines

- Example: Guidelines for SoC Environment (SCE)
  - Clean behavioral hierarchy
    - hierarchical behaviors:
      no code other than par, pipe, seq, fsm, and try-trap statements
    - leaf behaviors:
      Pure ANSI-C code (no SpecC constructs)
  - Clean communication
    - point-to-point communication via standard channels
    - ports of plain type or interface type, no pointers!
    - port maps to local variables or ports only
- Detailed rules for SoC Environment
  - CECS Technical Report:
    "*SCE Specification Model Reference Manual*"
    by A. Gerstlauer, R. Dömer, et al.
    - `$SPECC/doc/SpecRM.pdf`

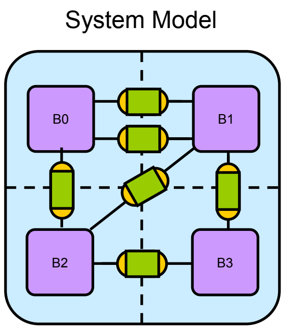


Specification Modeling Guidelines

- Converting C reference code to SpecC
  - Major functions become behaviors
  - Function call tree becomes behavioral hierarchy
    - Function call becomes behavior instance call
    - Sequential statements become leaf behaviors
    - Control flow becomes FSM
      - Conditional statements, if, if-else, switch
      - Loops, while, for, do
  - Explicitly specify potential parallelism!
  - Explicitly specify communication!
    - Use standard channels, avoid shared variables
    - No global variables
    - Only local variables in behaviors and functions/methods
  - Data types
    - Avoid dynamic memory allocation
    - Avoid pointers (arrays are preferred)
    - Use explicit SpecC data types if suitable (e.g. bit vectors)

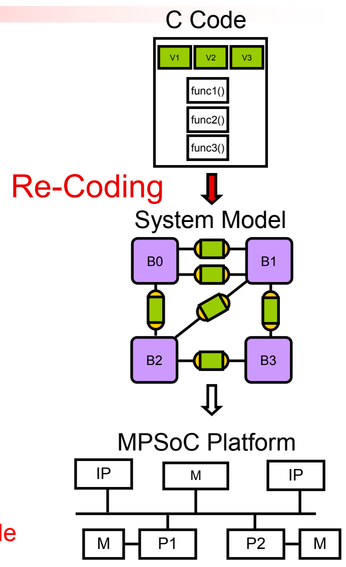
Current Research

- Specification Model Generation
  - It is desirable to *automatically generate* a Specification Model!
- Key Concepts needed for System Modeling
  - Explicit Structure
    - Block diagram structure
    - Connectivity through ports
  - Explicit Hierarchy
    - System composed of components
  - Explicit Concurrency
    - Potential for parallel execution
    - Potential for pipelined execution
  - Explicit Communication and Computation
    - Channels and Interfaces
    - Behaviors / Modules



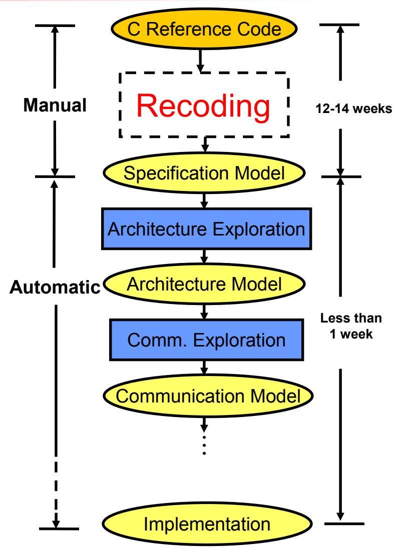
Current Research

- Existing System Design Flow
  - Input: System model
  - Output: MPSoC platform
- Actual Starting Point
  - C reference code
  - Flat, unstructured, sequential
  - Insufficient for system exploration
- Need: System Model
  - System-Level Description Language (SLDL)
  - Well-structured
    - Explicit computation, explicit communication
    - Potential parallelism explicitly exposed
  - Analyzable, synthesizable, verifiable
- Research: Automatic *Re-Coding*
  - How to get from flat and sequential C code to a flexible and parallel system model?



Recoding: Motivation

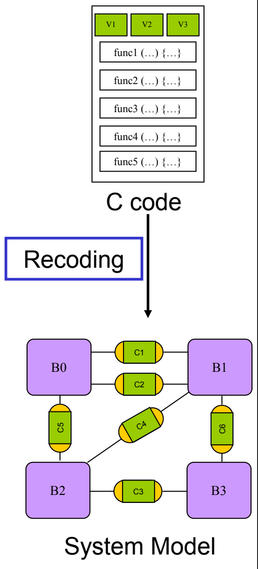
- Extend of Automation
  - Refinement-based design flow
  - Automatic
    - Specification model down to implementation
    - Example: SCE (mostly automatic)
    - MP3 decoder: less than 1 week
  - Manual
    - C reference code to SpecC specification model
    - Source code transformations
    - MP3 decoder: 12-14 weeks!
- Automation Gap
  - 90% of overall design time is spent on re-coding!
- Research: Automatic Recoding



Source: *System Design: A Practical Guide with SpecC*

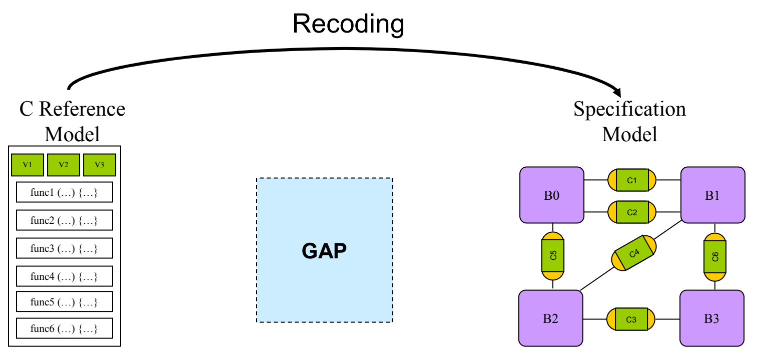
Recoding: Problem Definition

- How to get from flat, sequential C code to a flexible, parallel system model?
- Recoding
  - Create structural hierarchy
  - Partition code and data
    - Expose concurrency (parallelize/pipeline)
  - Expose communication
  - Eliminate pointers
  - Make the code compliant to the design tools, …
- Current Research
  - *Computer-Aided* Recoding
    - Automated source code transformations



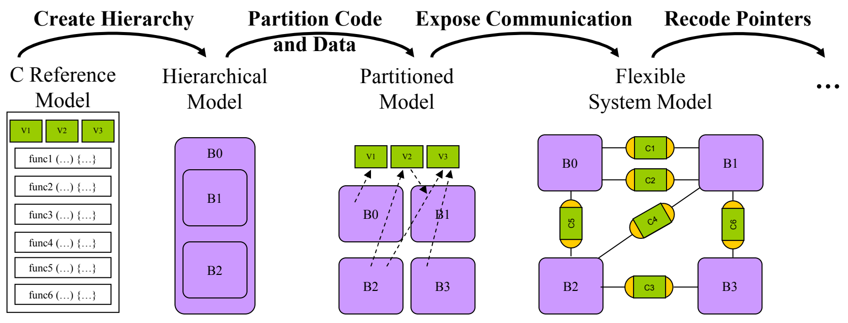
Recoding: Overcoming the Specification Gap

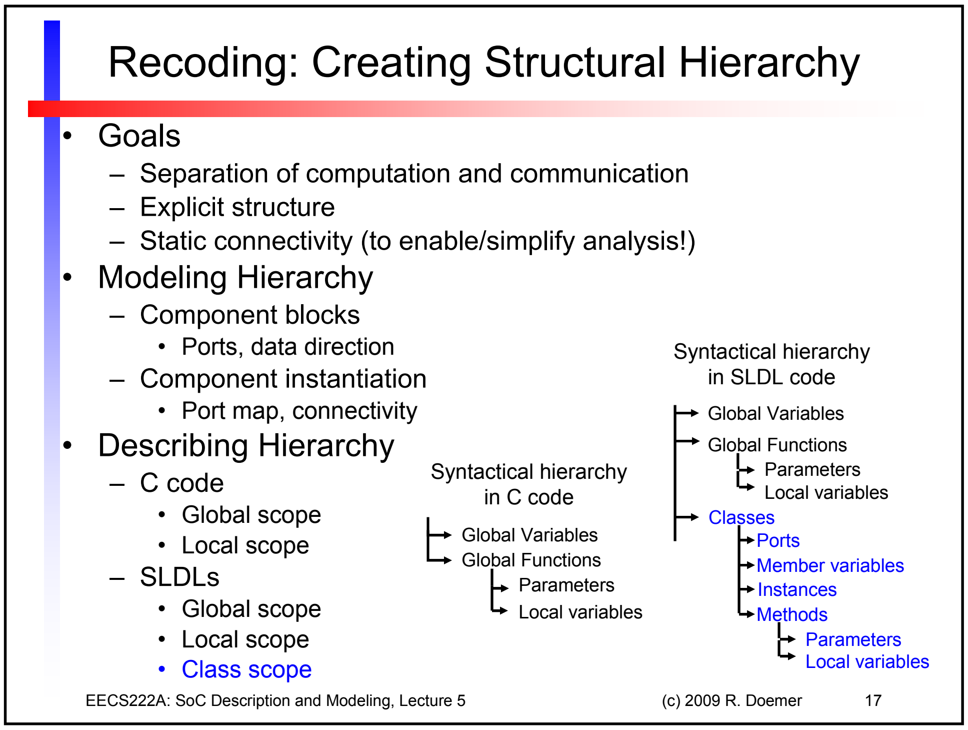
- Source-to-Source Transformations



Recoding: Overcoming the Specification Gap

- Step-wise Source-to-Source Transformations
  - Creating structural hierarchy [ASPDAC'08]
  - Code and data partitioning [DAC'07]
  - Creating explicit communication [ASPDAC'07]
  - Recode pointers [ISSS/CODES'07]





Recoding: Creating Structural Hierarchy

- Goals
  - Separation of computation and communication
  - Explicit structure
  - Static connectivity (to enable/simplify analysis!)
- Modeling Hierarchy
  - Component blocks
    - Ports, data direction
  - Component instantiation
    - Port map, connectivity
- Describing Hierarchy
  - C code
    - Global scope
    - Local scope
  - SLDLs
    - Global scope
    - Local scope
    - Class scope

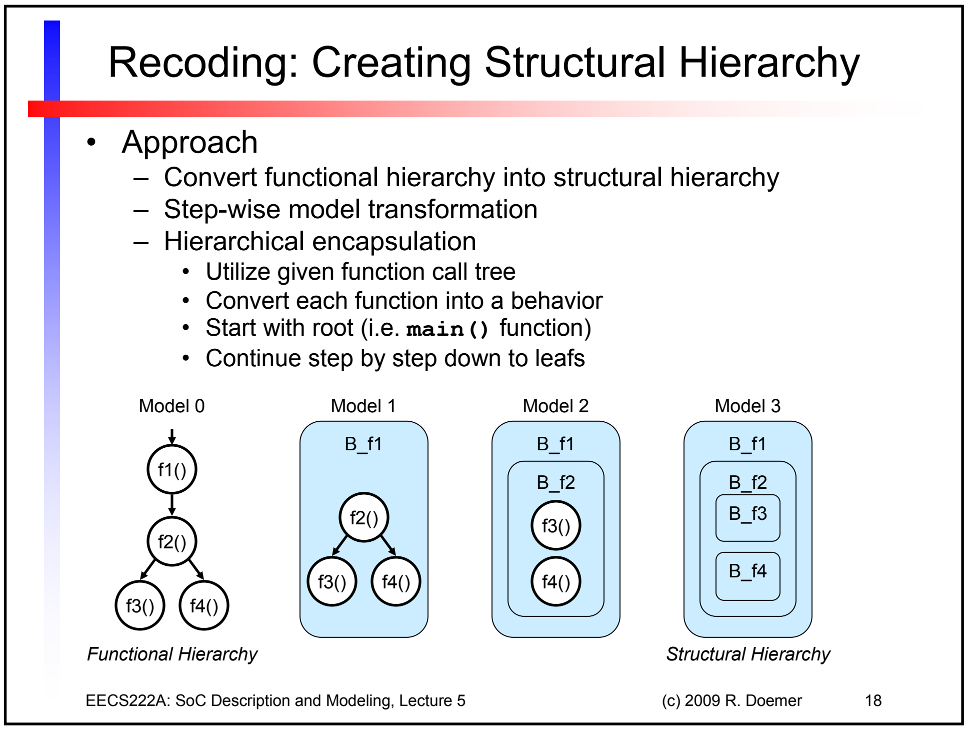
Syntactical hierarchy in C code

- Global Variables
- Global Functions
  - Parameters
  - Local variables

Syntactical hierarchy in SLDL code

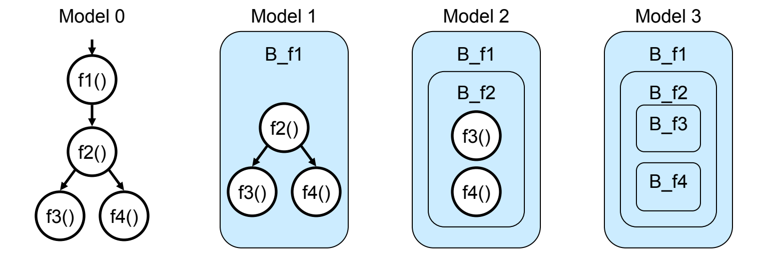
- Global Variables
- Global Functions
  - Parameters
  - Local variables
- Classes
  - Ports
  - Member variables
  - Instances
  - Methods
    - Parameters
    - Local variables

EECS222A: SoC Description and Modeling, Lecture 5 (c) 2009 R. Doemer 17



Recoding: Creating Structural Hierarchy

- Approach
  - Convert functional hierarchy into structural hierarchy
  - Step-wise model transformation
  - Hierarchical encapsulation
    - Utilize given function call tree
    - Convert each function into a behavior
    - Start with root (i.e. `main()` function)
    - Continue step by step down to leafs

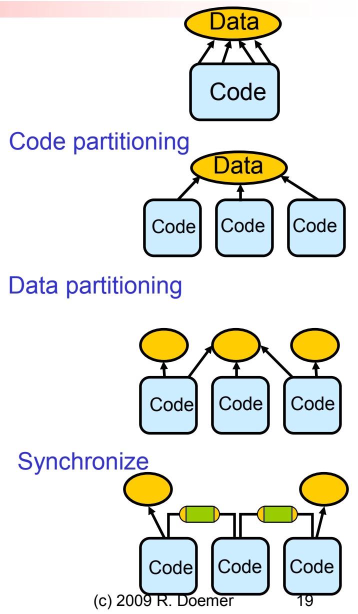


Functional Hierarchy *Structural Hierarchy*

EECS222A: SoC Description and Modeling, Lecture 5 (c) 2009 R. Doemer 18

Recoding: Exposing Potential Parallelism

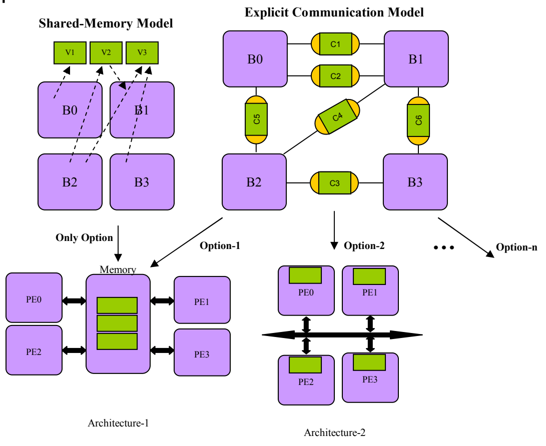
- Desirable model features
  - Enable parallel execution
  - Allow mapping to different PEs
- Recoding tasks
  - Partition code
  - Partition data
  - Synchronize dependents
- Recoding transformations
  1. Loop splitting
  2. Cumulative Access Type analysis
  3. Partitioning of vector dependents
  4. Synchronizing dependent variables
  - ➢ [DAC'07, TCAD'08]



Recoding: Exposing Communication

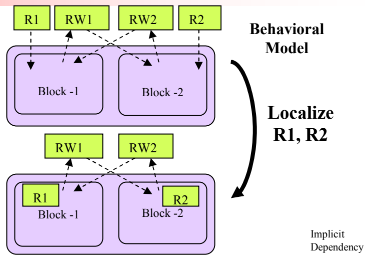
- Why create explicit communication?

- Quality of Communication Exploration
  - Number of explorations
  - Extent of automation
  - Time
- Shared-Memory Model
  - Global variables limit the number of possible automatic explorations
- Explicit Communication Model
  - Enables automatic exploration of more design alternatives



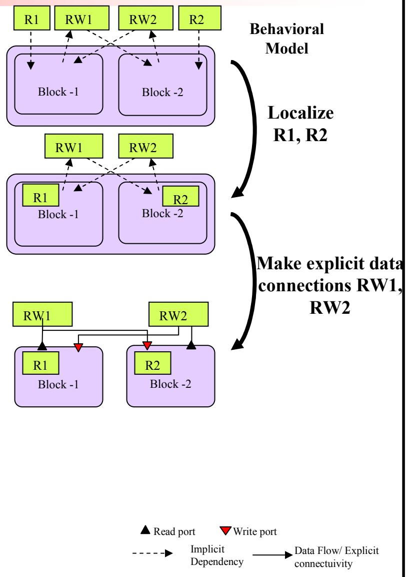
Recoding: Exposing Communication (1)

- Localize global variables to partitions
  - To enable multiple explorations
- Procedure
  - Find the global variable
  - Determine the functions and behaviors accessing it
  - If only one behavior is accessing it, migrate the variable into this behavior



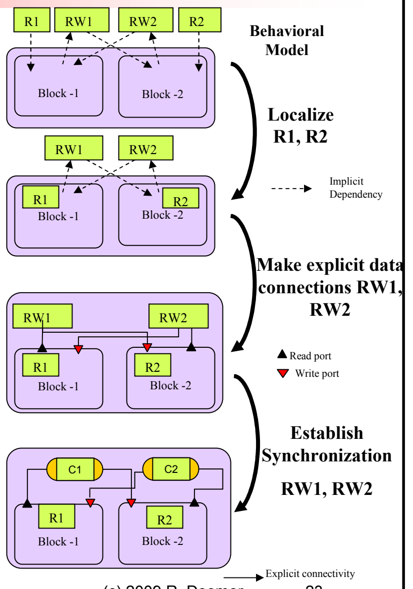
Recoding: Exposing Communication (2)

- Localize global variables to common parent and provide explicit access
  - Simplifies subsequent analysis of models
- Procedure
  - Find the global variable
  - Determine the functions and behaviors accessing it
  - If multiple behaviors are accessing it, find the lowest common parent
  - Migrate the variable to the parent
  - Provide access to the variable by recursively inserting ports in behaviors



Recoding: Exposing Communication (3)

- Use message passing channels instead of variables
  - Defines synchronization scheme
  - Guides exploration tools
- Procedure
  - Create a typed synchronization channel
  - Replace the ports corresponding to the original variable with the channel interface type
  - Modify each access to the variable to call the appropriate interface function of the channel
    - read() / receive()
    - write() / send()



EECS222A: SoC Description and Modeling, Lecture 5 (c) 2009 R. Doemer 23

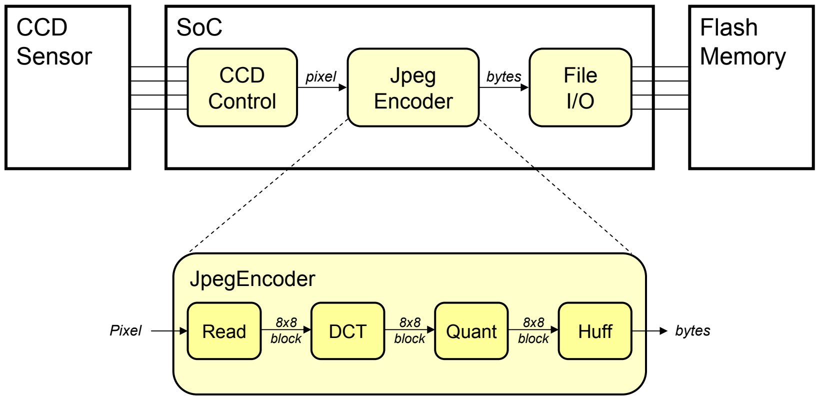
Recoding: References

- [ASPDAC'07] P. Chandraiah, J. Peng, R. Dömer, *"Creating Explicit Communication in SoC Models Using Interactive Re-Coding"*, Proceedings of the Asia and South Pacific Design Automation Conference 2007, Yokohama, Japan, January 2007.
- [IESS'07] P. Chandraiah, R. Dömer, *"An Interactive Model Re-Coder for Efficient SoC Specification"*, Proceedings of the International Embedded Systems Symposium, "Embedded System Design: Topics, Techniques and Trends" (ed. A. Rettberg, M. Zanella, R. Dömer, A. Gerstlauer, F. Rammig), Springer, Irvine, California, May 2007.
- [DAC'07] P. Chandraiah, R. Dömer, *"Designer-Controlled Generation of Parallel and Flexible Heterogeneous MPSoC Specification"*, Proceedings of the Design Automation Conference 2007, San Diego, California, June 2007.
- [ISSS+CODES'07] P. Chandraiah, R. Dömer, *"Pointer Re-coding for Creating Definitive MPSoC Models"*, Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis, Salzburg, Austria, September 2007.
- [ASPDAC'08] P. Chandraiah, R. Dömer, *"Automatic Re-coding of Reference Code into Structured and Analyzable SoC Models"*, Proceedings of the Asia and South Pacific Design Automation Conference 2008, Seoul, Korea, January 2008.
- [TCAD'08] P. Chandraiah, R. Dömer, *"Code and Data Structure Partitioning for Parallel and Flexible MPSoC Specification Using Designer-Controlled Re-Coding"*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems vol. 27, no. 6, pp. 1078-1090, June 2008.
- [DATE'09] R. Leupers, A. Vajda, M. Bekooij, S. Ha, R. Dömer, A. Nohl, *"Programming MPSoC Platforms: Road Works Ahead!"*, Proceedings of Design Automation and Test in Europe, Nice, France, April 2009.

EECS222A: SoC Description and Modeling, Lecture 5 (c) 2009 R. Doemer 24

Project Discussion

- Digital Camera Example
  - Component Model

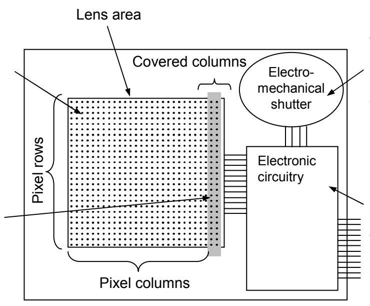


Project Discussion

- Digital Camera Example
  - 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.

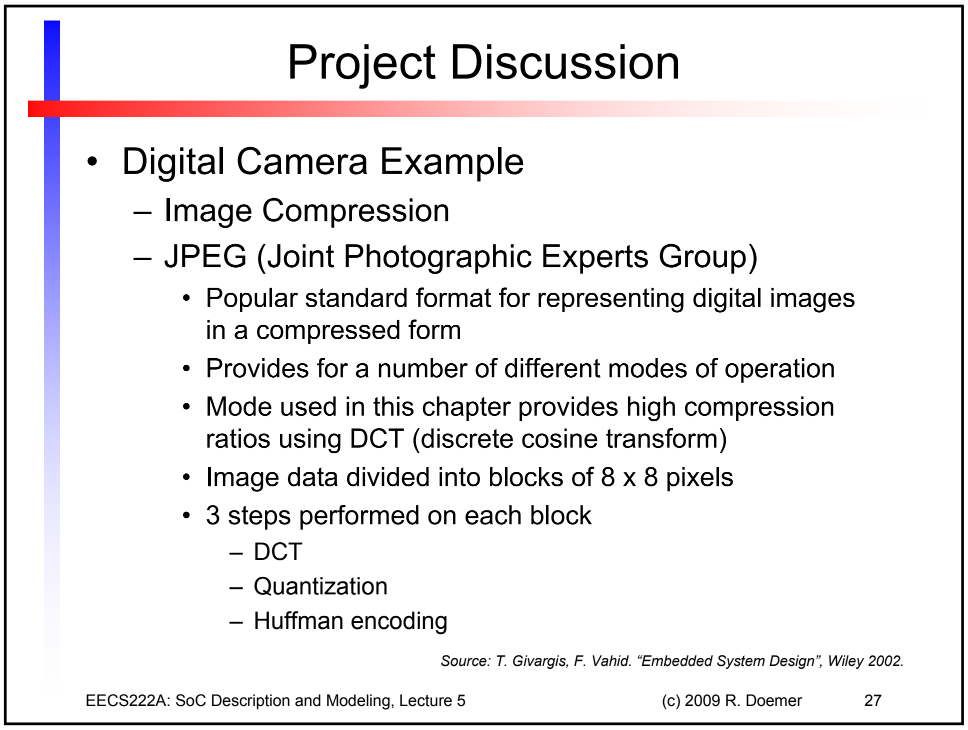
Some of the columns are covered with a black strip of paint. The light-intensity of these pixels is used for zero-bias adjustments of all the cells.



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.

Source: T. Givargis, F. Vahid. "Embedded System Design", Wiley 2002.

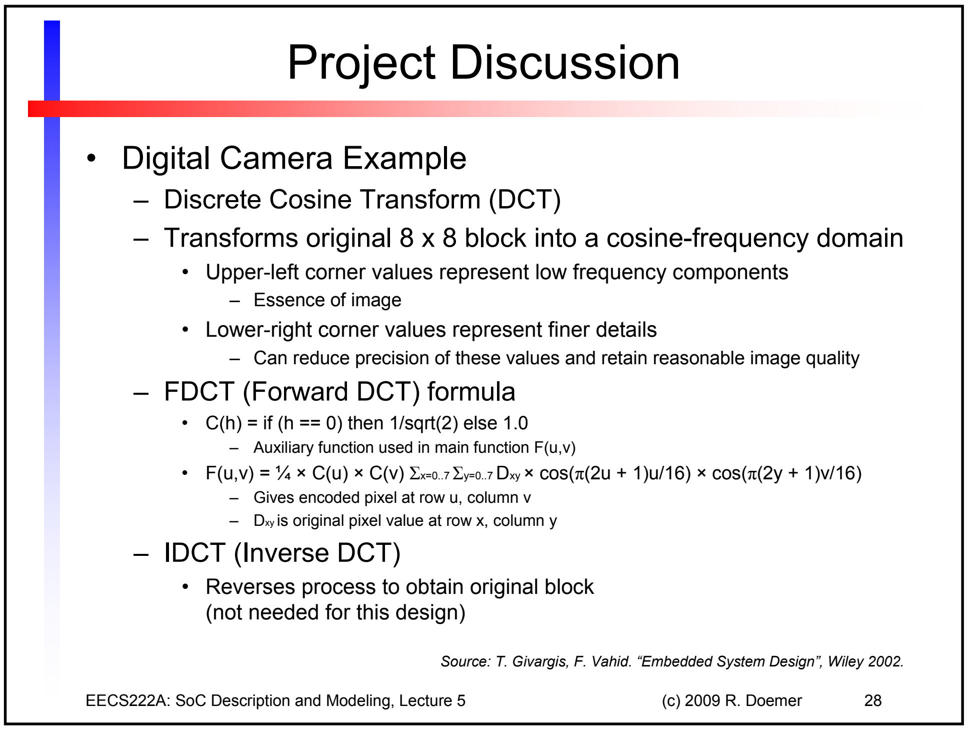


Project Discussion

- Digital Camera Example
  - Image Compression
  - 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

Source: T. Givargis, F. Vahid. "Embedded System Design", Wiley 2002.

EECS222A: SoC Description and Modeling, Lecture 5 (c) 2009 R. Doemer 27



Project Discussion

- Digital Camera Example
  - Discrete Cosine Transform (DCT)
  - Transforms original 8 x 8 block into a cosine-frequency domain
    - Upper-left corner values represent low frequency components
      - Essence of 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)
    - $F(u,v) = \frac{1}{4} \times C(u) \times C(v) \Sigma_{x=0..7} \Sigma_{y=0..7} D_{xy} \times \cos(\pi(2u + 1)u/16) \times \cos(\pi(2y + 1)v/16)$
      - 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)

Source: T. Givargis, F. Vahid. "Embedded System Design", Wiley 2002.

EECS222A: SoC Description and Modeling, Lecture 5 (c) 2009 R. Doemer 28

Project Discussion

- Digital Camera Example
  - Quantization
  - 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 DCT

Divide each cell's value by 8 $\longrightarrow$

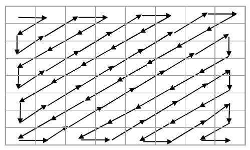
144	5	-5	-1	3	-10	1	5
-10	0	14	-9	-1	0	3	-1
2	-1	0	-5	3	0	2	-5
0	-8	-2	-2	5	-3	-2	1
6	2	5	-1	1	-3	1	-1
5	-1	-1	-1	3	-4	-3	2
-2	-1	3	-1	0	0	2	-3
-1	-2	-1	-2	-1	0	1	-1

After quantization

Source: T. Givargis, F. Vahid. "Embedded System Design", Wiley 2002.

Project Discussion

- Digital Camera Example
  - Huffman Encoding
  - 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

Source: T. Givargis, F. Vahid. "Embedded System Design", Wiley 2002.

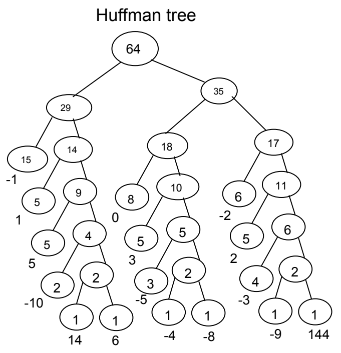
Project Discussion

- Digital Camera Example
  - Huffman Encoding (2)

- Pixel frequencies on left
  - Pixel value –1 occurs 15 times
  - Pixel value 14 occurs 1 time
- Build Huffman tree from bottom up
  - Create one leaf node for each pixel and assign frequency as node's value
  - Create an internal node by joining any two nodes whose sum is a minimal value
    - This sum is internal nodes value
  - Repeat until complete binary tree
- Traverse tree from root to leaf to obtain binary code for leaf's pixel value
  - Append 0 for left traversal, 1 for right traversal
- Huffman encoding is reversible
  - No code is a prefix of another code

Pixel frequency

-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 codes

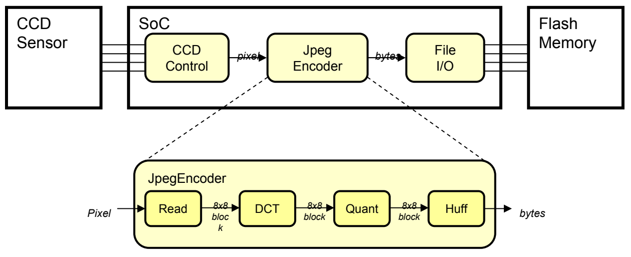
-1	00
0	100
-2	110
1	010
2	1110
3	1010
5	0110
-3	11110
-5	10110
-10	01110
144	111111
-9	111110
-8	101111
-4	101110
6	011111
14	011110

Source: T. Givargis, F. Vahid. "Embedded System Design", Wiley 2002.

EECS222A: SoC Description and Modeling, Lecture 5 (c) 2009 R. Doemer 31

Project Discussion

- Digital Camera Example
  - Component Model

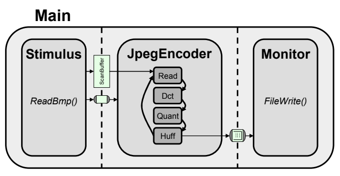


  - Homework Assigment 2
    - Become familiar with JPEG Encoder Application
      - Study reference code: **`/home/doemer/EECS222A_F09/jpegencoder.tar.gz`**
      - Draw block diagram of files, functions, and key communication variables
      - Simplify code for a 116×96 pixel CCD (eliminate **`malloc`** calls!)

EECS222A: SoC Description and Modeling, Lecture 5 (c) 2009 R. Doemer 32

Homework Assignment 3

- Task
  - Convert JPEG Encoder Application to a proper SpecC Specification Model



- Deliverables
  - Email to **`doemer@uci.edu`** with subject "EECS222A Assignment 3"
    - Brief status description (in body of your email)
    - Source code **`jpegencoder.tar.gz`** (attachment)
- Due
  - Next week: October 30, 2009, 12pm (noon)

Homework Assignment 3

- Hint:
  - Use the **`sir_tree`** tool to validate your hierarchy
  - The final model should look like this:

```
doemer@epsilon.eecs.uci.edu:3 > sir_tree -blt digicam.sir
B i o   behavior Main
B i o   |------ JpegEncoder jpeg
B i s   |       |------ Dct dct
B i l   |       |       |------ Bound bound
B i l   |       |       |------ ChenDct chendct
B i l   |       |       \------ Preshift preshift
B i s   |       |------ Huff huff
B i l   |       |       |------ Huffencode huffencode
B i l   |       |       \------ Zigzag zigzag
B i l   |       |------ Quantize quantize
B i l   |       \------ ReadBlock readblock
B i l   |------ Monitor monitor
B i l   |------ Stimulus stimulus
C i l   |------ c_queue data
C i l   \------ c_handshake start
```