EECS 222: Embedded System Modeling Lecture 11

Rainer Dömer

doemer@uci.edu

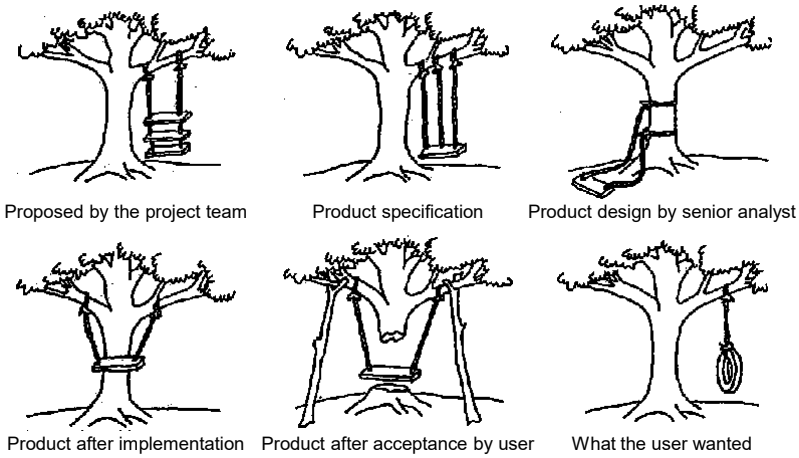
The Henry Samueli School of Engineering
Electrical Engineering and Computer Science
University of California, Irvine

Lecture 11: Overview

- Embedded System Specification
  - Essential issues
  - Top-down SoC design flow
  - Specification Model
  - Specification Modeling Guidelines
- Recent Research
  - Computer-Aided Model Recoding
- Homework Assignment 5
  - Structural Model of the Canny Edge Detector

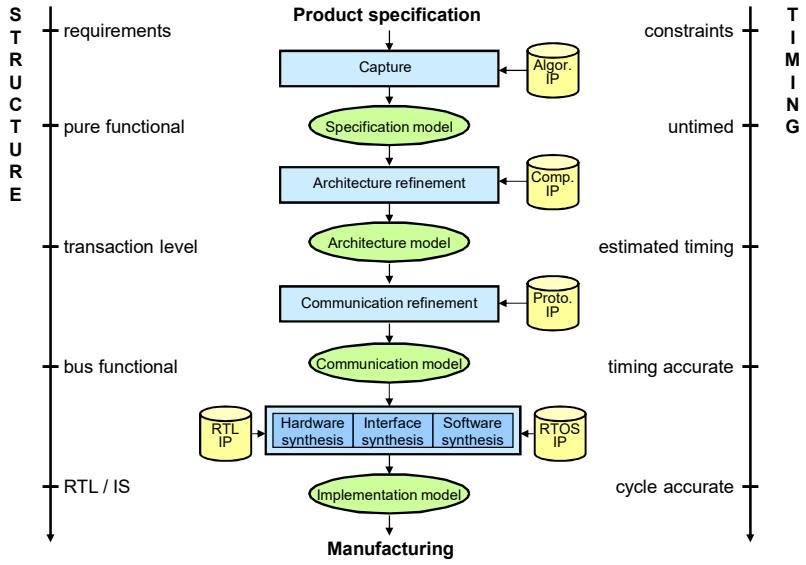
Essential Issues in Model Specification

- An Example ...



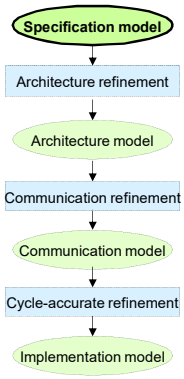
Source: unknown author

Top-Down Embedded Design Flow



Specification Model

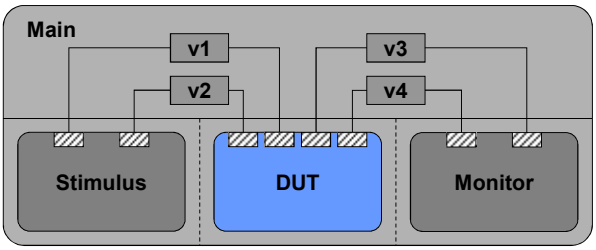
- High-level, abstract model
  - Pure system functionality
  - Algorithmic behavior
  - No implementation details
- No implicit structure / architecture
  - Pure behavioral hierarchy
- Untimed
  - Execution in zero (logical) time
  - Causal ordering
  - Synchronization



(Source: A. Gerstlauer)

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!)*



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

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

Specification Modeling Guidelines

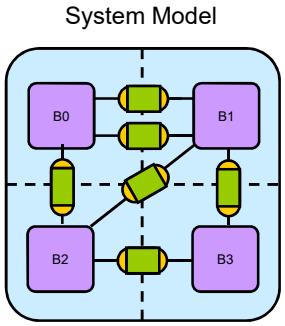
- Example: Guidelines for SoC Environment (SCE)
  - Clean behavioral hierarchy
    - hierarchical behaviors:
      no code other than par, pipe, seq, fsm, and try-trap statements
    - leaf behaviors:
      Pure ANSI-C code (no SpecC constructs)
  - Clean communication
    - point-to-point communication 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.
    - **`/opt/sce-20100908/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-while`**
  - 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 data types if suitable (e.g. bit vectors)

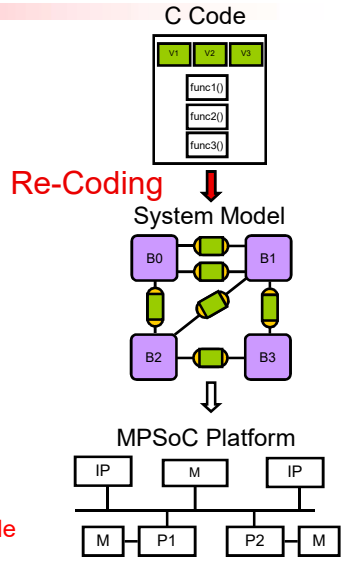
Recent Research: Model Recoding

- 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



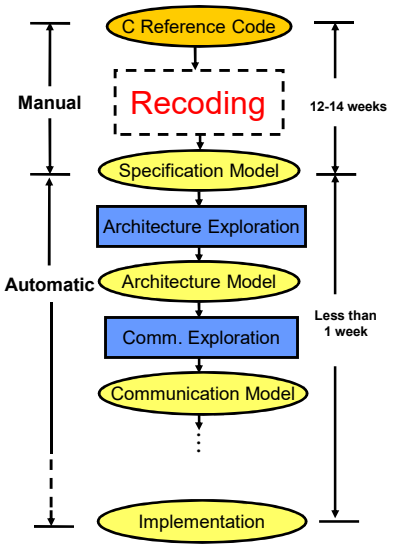
Recent Research: Model Recoding

- 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

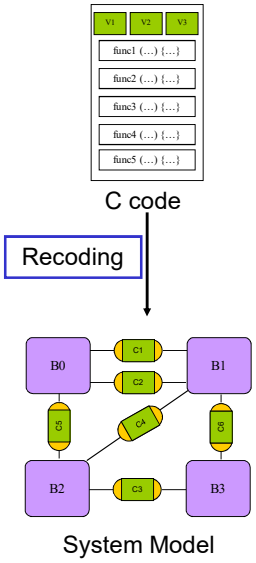
- Extend of Automation
  - Refinement-based design flow
  - Automatic
    - Specification model down to implementation
    - Example: SCE (mostly automatic)
    - MP3 decoder: less than 1 week
  - Manual
    - C reference code to SpecC specification model
    - Source code transformations
    - MP3 decoder: 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*

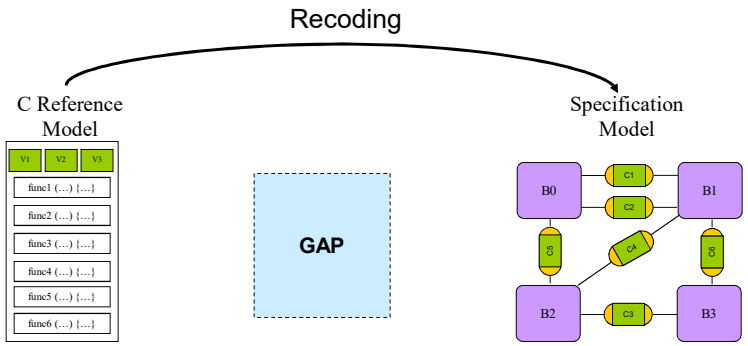
Recoding: Problem Definition

- 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



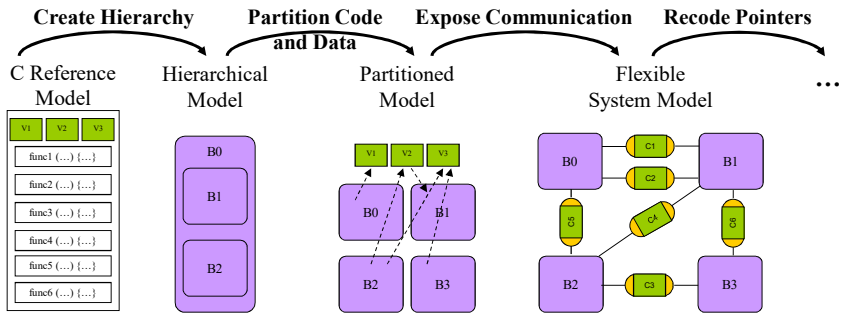
Recoding: Overcoming the Specification Gap

- 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

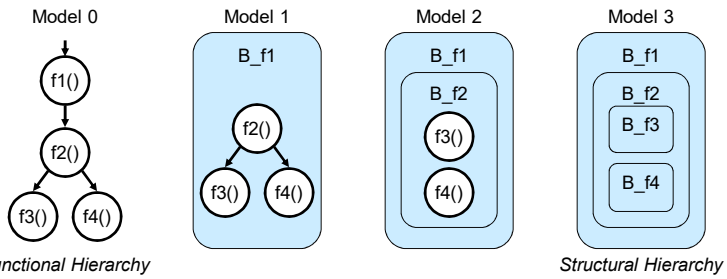
- Global Variables
- Global Functions
  - Parameters
  - Local variables

Syntactical hierarchy in SLDL code

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

Recoding: Creating Structural Hierarchy

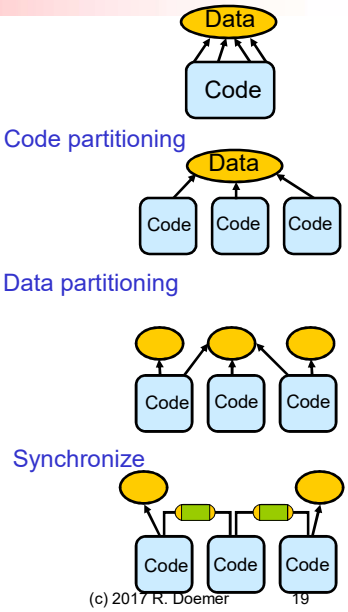
- Approach
  - Convert functional hierarchy into structural hierarchy
  - Step-wise model transformation
  - Hierarchical encapsulation
    - Utilize given function call tree
    - Convert each function into a behavior
    - Start with root (i.e. `main()` function)
    - Continue step by step down to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



Functional Hierarchy *Structural Hierarchy*

Recoding: Exposing Potential Parallelism

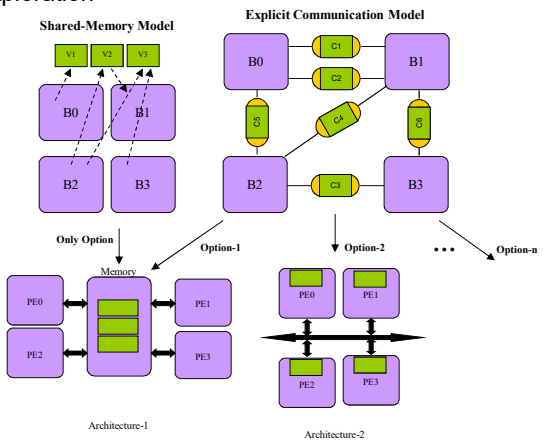
- Desirable model features
  - Enable parallel execution
  - Allow mapping to different PEs
- Recoding tasks
  - Partition code
  - Partition data
  - Synchronize dependents
- Recoding transformations
  1. Loop splitting
  2. Cumulative Access Type analysis
  3. Partitioning of vector dependents
  4.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



Recoding: Exposing Communication

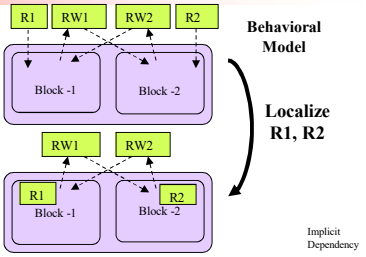
- Why create explicit communication?

- 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



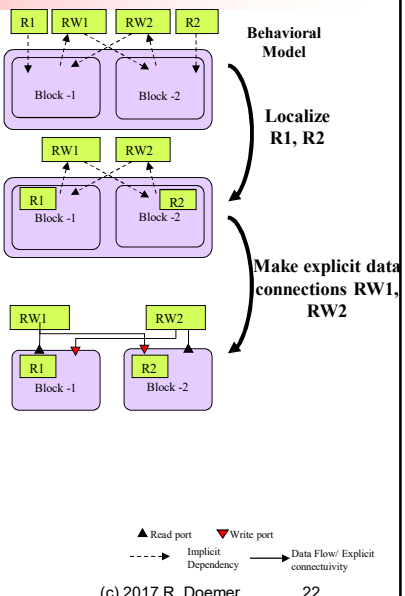
Recoding: Exposing Communication (1)

- 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



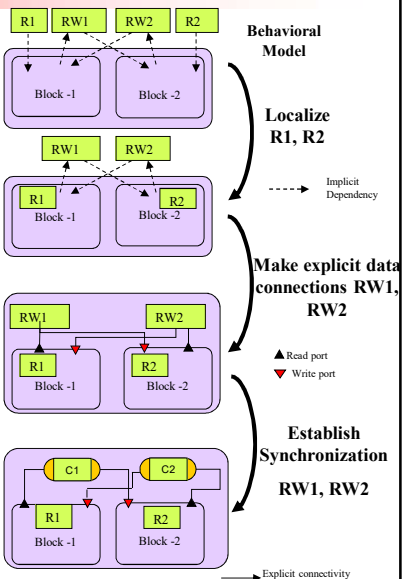
Recoding: Exposing Communication (2)

- 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()



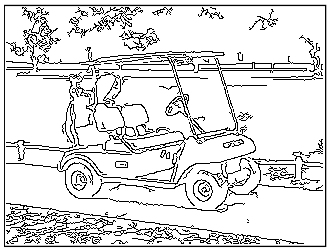
Model Recoding: References

- [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.

EECS 222 Project

- Application Example: Canny Edge Detector
  - Embedded system model for image processing: Automatic Edge Detection in a Digital Camera

golfcart.pgm



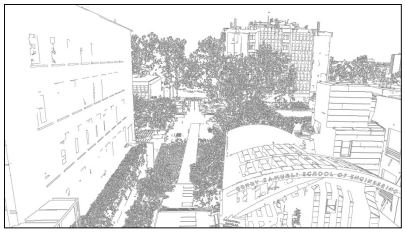
golfcart.pgm_s_0.60_l_0.30_h_0.80.pgm

  - Application Source and Documentation:
    - http://marathon.csee.usf.edu/edge/edge_detection.html
    - http://en.wikipedia.org/wiki/Canny_edge_detector

EECS 222 Project

- Application Example: Canny Edge Detector
  - Embedded system model for image processing: Automatic Edge Detection in a Digital Video Camera

EngPlaza001.bmp



EngPlaza001_edges.pgm

  - Video taken by a drone hovering over UCI Engineering Plaza
    - Available on the server: **`~eecs222/public/video/`**
    - High resolution, 2704 by 1520 pixes
    - Video length 9 seconds, using 20 extracted frames for test bench model

Homework Assignment 5

- Task: Structural Model of the Canny Edge Detector
  - Convert the application to process a stream of video frames
  - Add test bench structure to the SLDL model from Assignment 4
  - Choose either SpecC or SystemC for simulation
- Steps
  1. Create test bench structure: Stimulus, Platform, Monitor
  2. Create platform model: DataIn, DUT, DataOut
  3. Localize functions and add loops for stream processing
- Deliverables
  - **`Canny.sc`** or **`Canny.cpp`** (choose one!)
  - **`Canny.txt`**
- Due
  - By next week: May 15, 2017, 12pm (noon!)