

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 5: Overview

- System-on-Chip Specification
 - Essential issues
 - Top-down SoC design flow
 - Specification Model
 - Specification Modeling Guidelines
- Ongoing Research
 - Computer-Aided Recoding
- Project Discussion
 - Assignment 2
 - Assignment 3
 - Next steps

Essential Issues in Specification

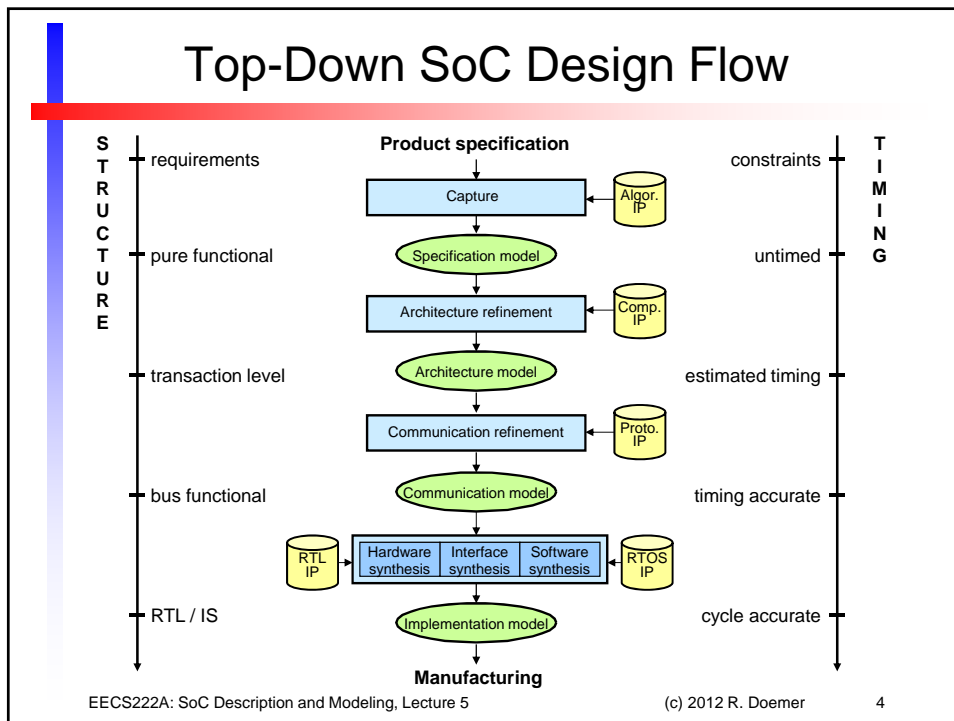
- An Example ...

The six drawings show the following stages:

- Proposed by the project team: A tree with a simple wooden swing.
- Product specification: A tree with a more complex, ornate swing.
- Product design by senior analyst: A tree with a swing that has a seat and a chain, but the seat is attached to the trunk.
- Product after implementation: A tree with a simple wooden swing, identical to the first drawing.
- Product after acceptance by user: A tree with a swing that has a seat and a chain, but the seat is attached to the trunk, identical to the third drawing.
- What the user wanted: A tree with a simple wooden swing, identical to the first drawing.

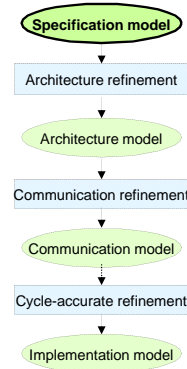
Source: unknown author

EECS222A: SoC Description and Modeling, Lecture 5 (c) 2012 R. Doemer 3



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)

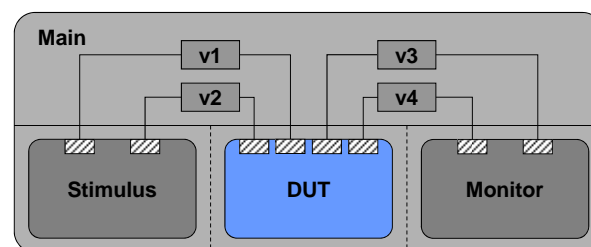
EECS222A: SoC Description and Modeling, Lecture 5

(c) 2012 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) 2012 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) 2012 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) 2012 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.
 - `/opt/sce-20100908/doc/SpecRM.pdf`

EECS222A: SoC Description and Modeling, Lecture 5

(c) 2012 R. Doemer

9

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)

EECS222A: SoC Description and Modeling, Lecture 5

(c) 2012 R. Doemer

10

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

The diagram shows a 'System Model' with four blocks labeled B0, B1, B2, and B3. B0 and B1 are at the top, B2 and B3 are at the bottom. They are interconnected by various channels and interfaces, including a central diagonal channel connecting B0 to B3 and B1 to B2.

EECS222A: SoC Description and Modeling, Lecture 5 (c) 2012 R. Doemer 11

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?

The flowchart illustrates the process of automatic re-coding. It starts with 'C Code' represented as a flat, sequential block with three functions (func1, func2, func3). A red arrow labeled 'Re-Coding' points to a 'System Model' diagram, which is a structured block diagram with four blocks (B0, B1, B2, B3) and their interconnections. A second arrow points from the System Model to an 'MPSoC Platform' diagram, which shows a hardware-like structure with IP blocks, memory (M), and processors (P1, P2).

EECS222A: SoC Description and Modeling, Lecture 5 (c) 2012 R. Doemer 12

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*
(c) 2012 R. Doemer 13

EECS222A: SoC Description and Modeling, Lecture 5

Recoding: Problem Definition

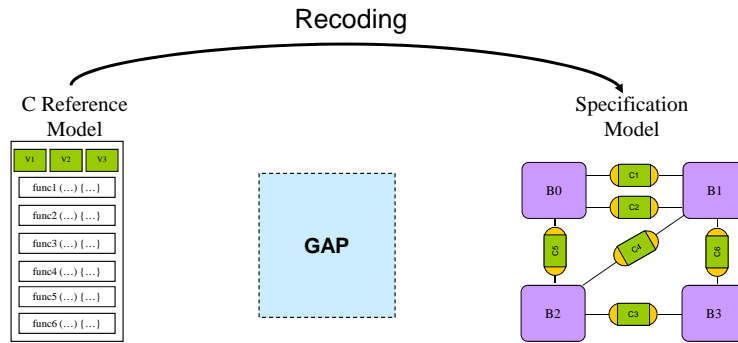
- How to get from flat, sequential C code to a flexible, parallel system model?
- Recoding
 - Create structural hierarchy
 - Partition code and data
 - Expose concurrency (parallelize/pipeline)
 - Expose communication
 - Eliminate pointers
 - Make the code compliant to the design tools, ...
- Current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

(c) 2012 R. Doemer 14

EECS222A: SoC Description and Modeling, Lecture 5

Recoding: Overcoming the Specification Gap

- Source-to-Source Transformations



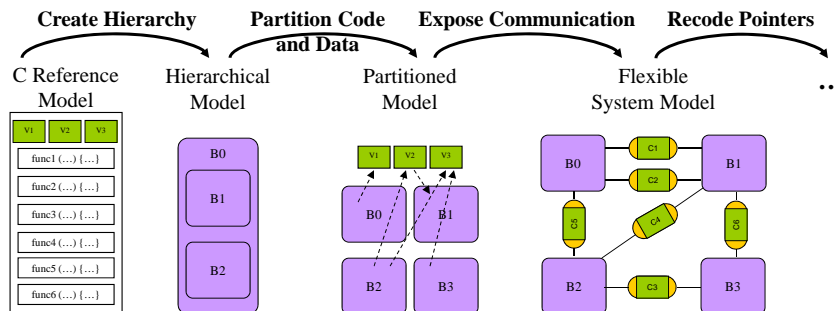
EECS222A: SoC Description and Modeling, Lecture 5

(c) 2012 R. Doemer

15

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]



EECS222A: SoC Description and Modeling, Lecture 5

(c) 2012 R. Doemer

16

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

Syntactical hierarchy in SLDL code

EECS222A: SoC Description and Modeling, Lecture 5
(c) 2012 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 leaves

Model 0

Functional Hierarchy

Model 1

Model 2

Model 3

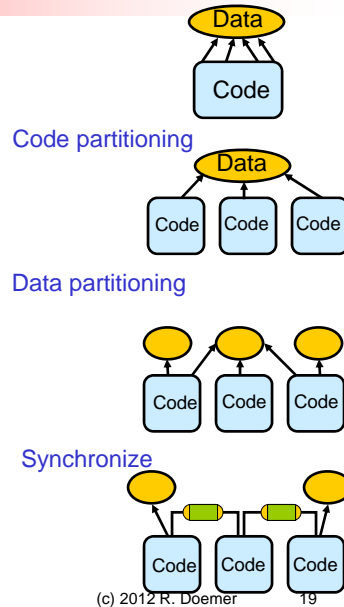
Structural Hierarchy

EECS222A: SoC Description and Modeling, Lecture 5
(c) 2012 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]

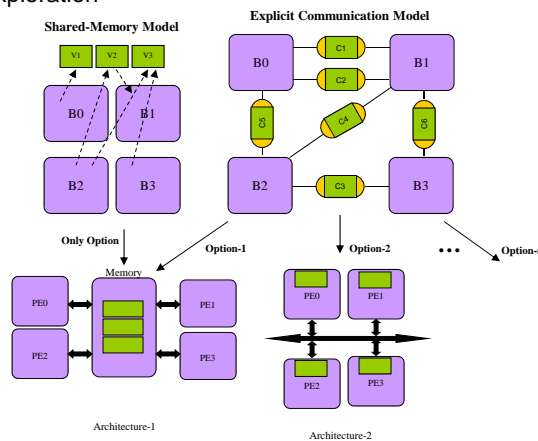


EECS222A: SoC Description and Modeling, Lecture 5

(c) 2012 R. Doemer 19

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

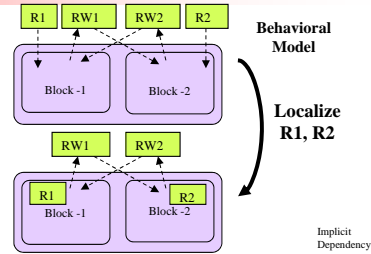


EECS222A: SoC Description and Modeling, Lecture 5

(c) 2012 R. Doemer 20

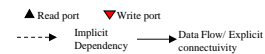
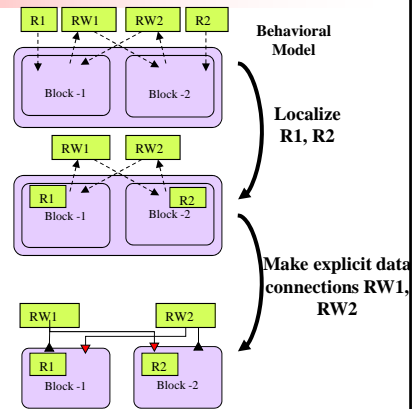
Recoding: Exposing Communication (1)

- 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



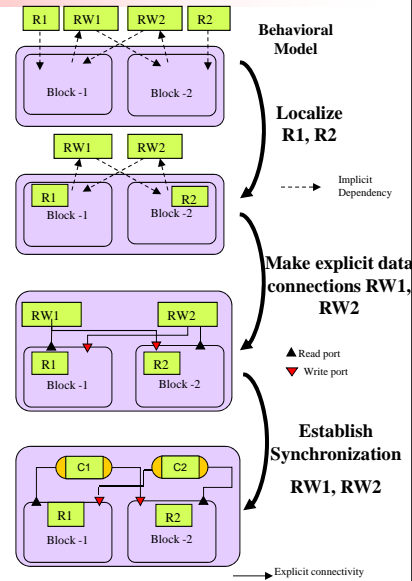
Recoding: Exposing Communication (2)

- 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) 2012 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) 2012 R. Doemer

24

Homework Assignment 2

- Task: Introduction to Canny Edge Detector
 - Setup: use latest `scc` to edit, compile, and simulate
 - `source /opt/sce/bin/setup.csh`
 - Application Example
 - Canny Edge Detector: SoC for edge detection in digital camera
 - Tools
 - Image tools: command-line tools (pbm, pgm, pnm), eog, gimp
 - Editor tools: SpecC-enhanced Eclipse
 - Steps
 - Download and compile with `gcc`
 - Study the application
 - Convert and compile with `scc` (timed!)
- Deliverables
 - Source file: `Canny.sc`
 - Description: `Canny.txt`
- Due
 - By next week: April 27, 2012, 12pm (noon!)

EECS222A: SoC Description and Modeling, Lecture 5

(c) 2012 R. Doemer

25

Homework Assignment 3

- Task: Structural Hierarchy for Canny Edge Detector
 - Setup: use latest `scc` to edit, compile, and simulate
 - `source /opt/sce/bin/setup.csh`
 - Provided Files
 - `canny_a3_start.sc`, Makefile
 - `golfcart.pgm`, `ref_golfcart.pgm_s_0.60_l_0.30_h_0.80.pgm`
 - Eclipse Support
 - Outline View: Source code structure
 - Behavior Hierarchy: SpecC structural hierarchy
- Deliverables
 - Source file: `canny.sc`
 - Description: `canny.txt`
- Due
 - By next week: May 4, 2012, 12pm (noon!)

EECS222A: SoC Description and Modeling, Lecture 5

(c) 2012 R. Doemer

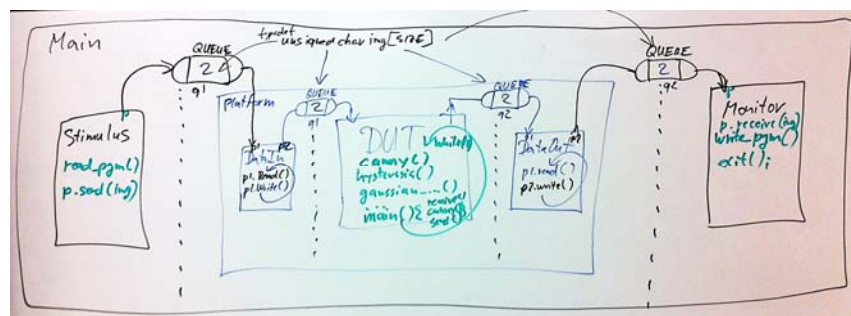
26

Homework Assignment 3

- Structural Hierarchy for Canny Edge Detector
 - Test bench Structure
 - B i o behavior Main
 - B i l |----- Monitor monitor
 - B i c |----- Platform platform
 - B i l | |----- DUT canny
 - B i l | |----- DataIn din
 - B i l | |----- DataOut dout
 - C i l | |----- c_img_queue q1
 - C i l | \----- c_img_queue q2
 - B i l |----- Stimulus stimulus
 - C i l |----- c_img_queue q1
 - C i l | \----- c_img_queue q2

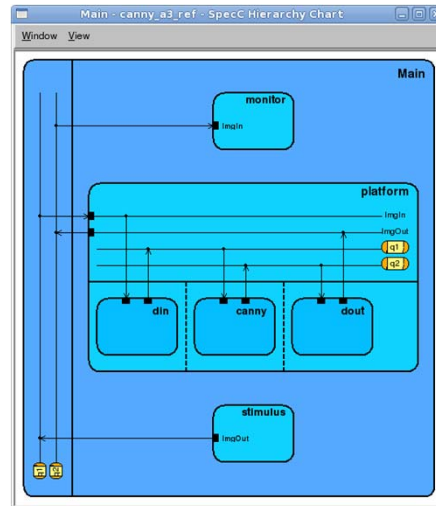
Homework Assignment 3

- Structural Hierarchy for Canny Edge Detector
 - Test bench Structure



Homework Assignment 3

- Chart of canny_a3_ref.sc



EECS222A: SoC Description and Modeling, Lecture 5

(c) 2012 R. Doemer

29

Homework Assignment 3

- Eclipse screen shot of canny_a3_ref.sc

```

typedef unsigned char img[SIZE]; /* define our communication data types */

DEFINE_I_TYPED_SENDER(img, img) // creates interface i_block_sender
DEFINE_I_TYPED_RECEIVER(img, img) // creates interface i_block_receiver
DEFINE_I_TYPED_TRANCEIVER(img, img) // creates interface i_block_tranceiver
DEFINE_C_TYPED_QUEUE(img, img) // creates channel c_block_queue

behavior Monitor[i_img_receiver imgIn]
{
    unsigned char edge[SIZE];

    /* *****
    * Function: write_pgm_image
    * Purpose: This function writes an image in PGM format. The file is either
    * written to the file specified by outfilename or to standard output if
    * outfilename = NULL. A comment can be written to the header if coment != N
    * *****
    int write_pgm_image(char *outfilename, unsigned char *image, int rows,
  
```

EECS222A: SoC Description and Modeling, Lecture 5

(c) 2012 R. Doemer

30

Project Discussion

- Next Steps
 - Create additional hierarchy inside DUT
 - Behavioral Composition
 - Parallel execution desirable
 - Sequential execution as needed
 - Structural Composition
 - Standard channels, or
 - Variables shared through port maps
 - Canny Edge Detector: `canny()`
 - `gaussian_smooth()`
 - » `make_gaussian_kernel()`
 - `derrivative_x_y()`
 - `magnitude_x_y()`
 - `non_max_supp()`
 - `apply_hysteresis()`
 - » `follow_edges()`