ECPS 203
Embedded Systems Modeling and Design
Lecture 10

Rainer Dömer

doemer@uci.edu

Center for Embedded and Cyber-physical Systems
University of California, Irvine

Lecture 10: Overview

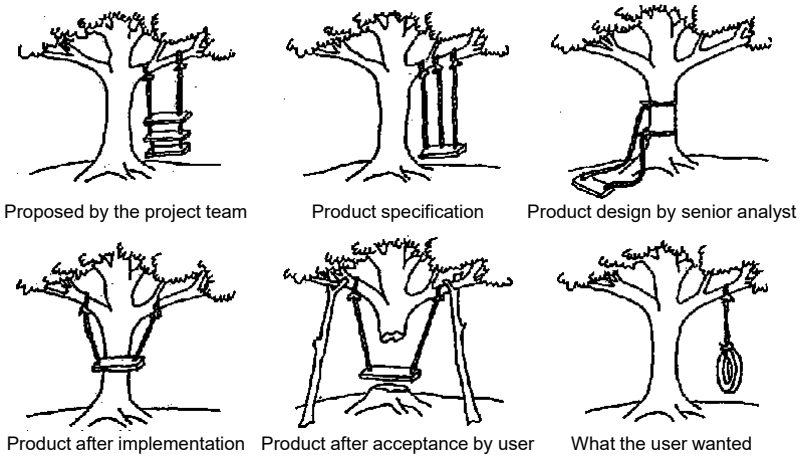
- Course Administration
  - Midterm course evaluation
- Embedded System Specification
  - Essential issues
  - Top-down design flow
  - Specification model
  - Specification modeling guidelines
- Project Discussion
  - Status and next steps
  - Assignment 5
  - Test bench model of the Canny Edge Detector
    - Model development on the whiteboard

Course Administration

- Midterm Course Evaluation
  - This week!
  - Monday, Oct. 30, 8am – Friday, Nov. 3, 11pm
  - Online via EEE Evaluation application
- Feedback from students to instructors
  - Completely voluntary
  - Completely anonymous
  - Very valuable
    - Help to improve this class!
- Mandatory Final Course Evaluation
  - expected for week 10 (TBA)

Essential Issues in Model Specification

- An Example ...

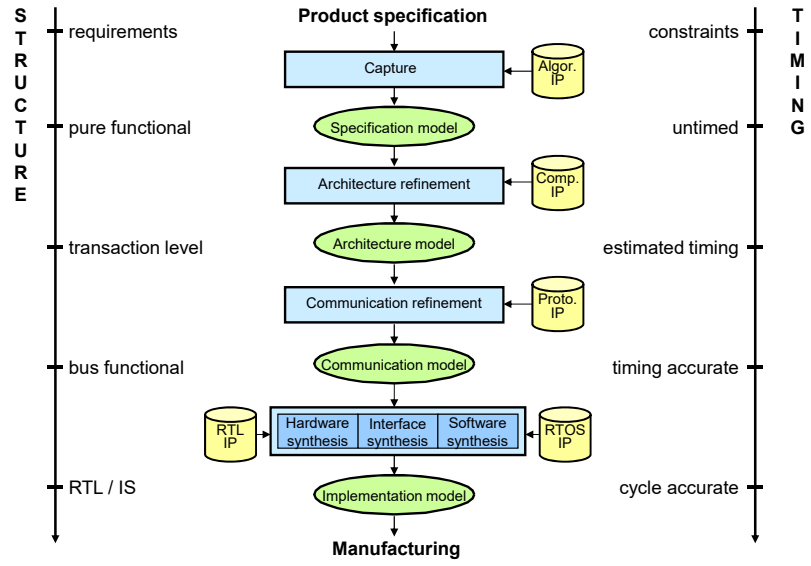


Proposed by the project team | Product specification | Product design by senior analyst

Product after implementation | Product after acceptance by user | What the user wanted

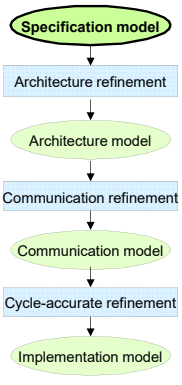
Source: unknown author

Top-Down Design Flow



Specification Model

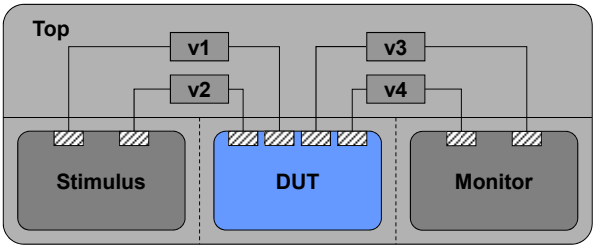
- High-level, abstract model
  - Pure system functionality
  - Algorithmic behavior
  - No implementation details
- No implicit structure, no architecture
  - Pure behavioral hierarchy in DUT
- Untimed
  - Execution in zero logical time
  - Causal ordering and synchronization
- Test bench
  - Stimulus, DUT, monitor



(Adapted from A. Gerstlauer)

Specification Model

- Test bench
  - Top, Stimulus, Monitor
  - *Simulation only, no synthesis (no modeling restrictions)*
- DUT
  - Design under test (aka. unit 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 (aside from test bench)
- No timing
  - Exception: timing constraints
- Separation of computation and communication
  - Modules and channels

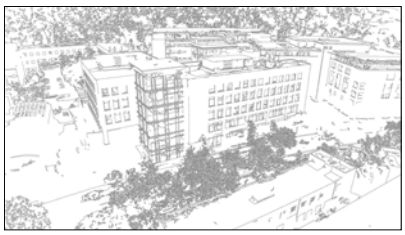
Specification Modeling Guidelines

- Computation: in Modules
  - Granularity: Leaf modules = smallest indivisible units
  - Hierarchy: Explicit execution order
    - Sequential, pipelined, or parallel
  - Encapsulation: Localized variables, explicit port mappings
  - Concurrency: Potential parallelism explicitly specified
  - Time: Untimed (partial order and synchronization)
- Communication: in Channels
  - Communication: Standard channels
  - Synchronization: Standard channels
  - Dependencies: Data flow explicit in connectivity

ECPS 203 Project

- Application Example: Canny Edge Detector
  - Embedded system model for image processing:
    Automatic edge detection in a digital video camera

Engineering012.png



Engineering012_edges.pgm

  - Video taken by a drone flying over UCI Engineering Plaza
    - Available on the server: **`~ecps203/public/DroneFootage/`**
    - High resolution, 2704 by 1520 pixes
    - Representative sample, using 30 extracted frames for test bench model

Project Assignment 4

- Task: From Single Image to Video Stream Processing
  - Prepare a sequence of image frames from the video
  - Convert the Canny application the selected video frames
- Steps
  1. Extract 30 of video frames suitable for use in a test bench
  2. Convert the color frames to grey-scale images in PGM format
  3. Recode your Canny C++ model to process the video frames
     - To run Canny application successfully, increase stack size
- Deliverables
  - Source code and text file: **`Canny.cpp`**, **`Canny.txt`**
- Due
  - Wednesday, November 1, 2017, 6pm

Project Assignment 5

- Task: Test Bench for the Canny Edge Detector
  - Convert C++ model to SystemC model
  - Add a test bench structure around the C++ model
  - Wrap DUT into a platform model with dedicated I/O units
- Steps
  1. Create test bench structure: Stimulus, Platform, Monitor
  2. Create platform model: DataIn, DUT, DataOut
  3. Localize functions and use **`sc_fifo`** channels for communication
     - Pay attention to thread stack sizes
- Deliverables
  - SystemC source code and text file: **`Canny.cpp`**, **`Canny.txt`**
- Due
  - Wednesday, November 8, 2017, 6pm

Project Assignment 5

- Task: Test Bench for the Canny Edge Detector
  - Expected instance tree

```
Top top
|------ Monitor monitor
|------ Platform platform
|       |------ DUT canny
|       |------ DataIn din
|       |------ DataOut dout
|       |------ sc_fifo<IMAGE> q1
|       \------ sc_fifo<IMAGE> q2
|------ Stimulus stimulus
|------ sc_fifo<IMAGE> q1
\------ sc_fifo<IMAGE> q2
```


Project Assignment 5

- Task: Structural Model of the Canny Edge Detector
  - Discussion on whiteboard: Chart of top-level structure