EECS 222: Embedded System Modeling Lecture 14

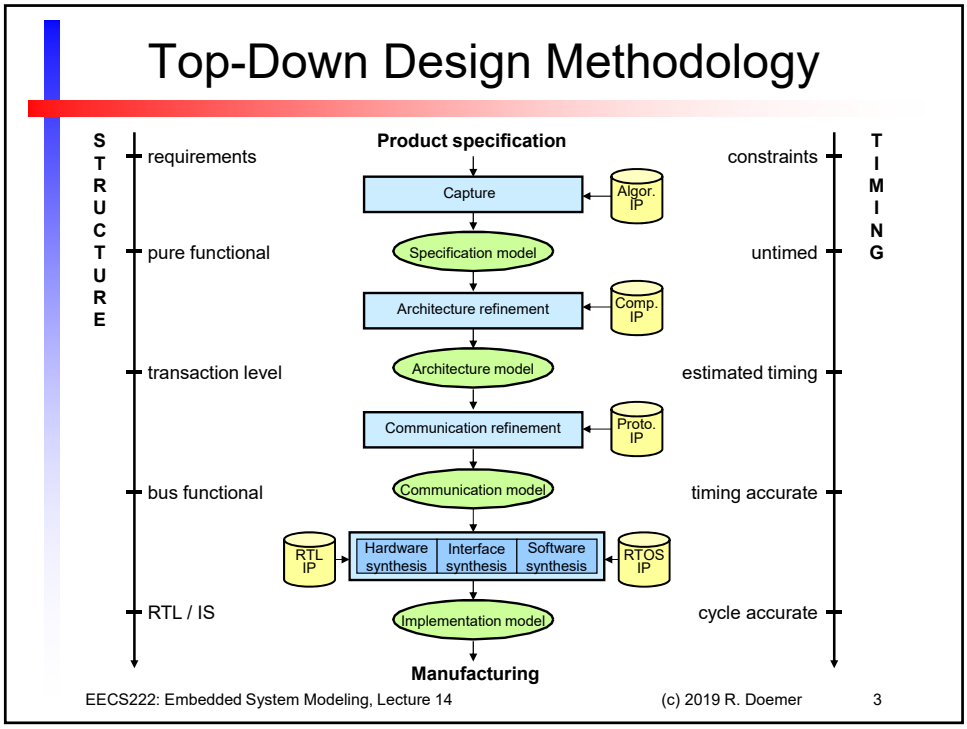
Rainer Dömer

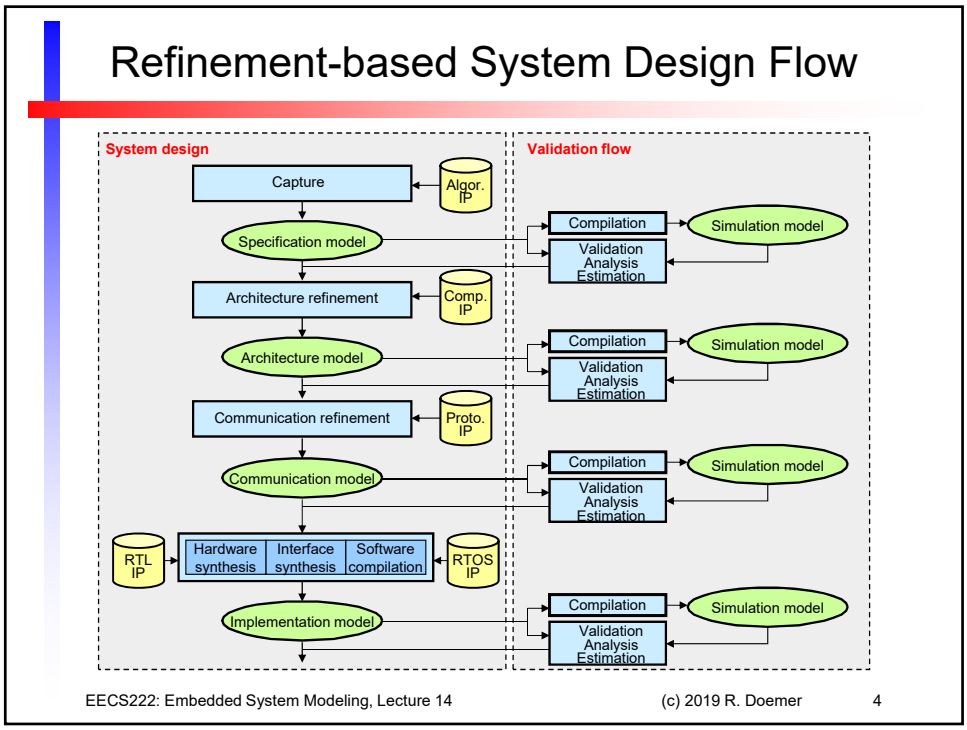
doemer@uci.edu

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

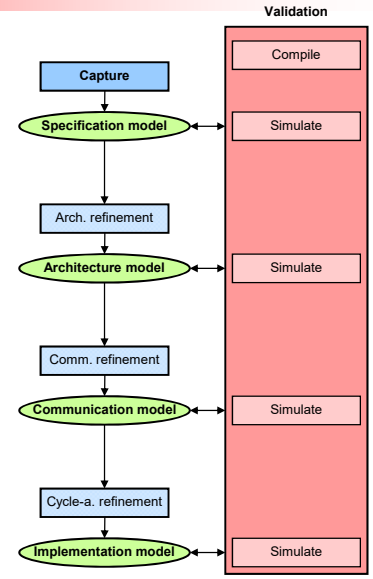
Lecture 14: Overview

- Embedded System Design Flow
  - Top-down design methodology
  - Refinement-based design flow
    - Specify
    - Explore
    - Refine
- System-on-Chip Environment (SCE)
  - Profiling and performance estimation
  - Design Example: GSM Vocoder
  - ➢ Interactive demonstration
- Homework Assignment 7
  - Performance estimation of the Canny Edge Detector


Top-Down Design Methodology
STRUCTURE
TIMING
requirements
pure functional
transaction level
bus functional
RTL / IS
constraints
untimed
estimated timing
timing accurate
cycle accurate
Product specification
Capture
Algor. IP
Specification model
Architecture refinement
Comp. IP
Architecture model
Communication refinement
Proto. IP
Communication model
RTL IP
Hardware synthesis
Interface synthesis
Software synthesis
RTOS IP
Implementation model
Manufacturing
EECS222: Embedded System Modeling, Lecture 14
(c) 2019 R. Doemer
3

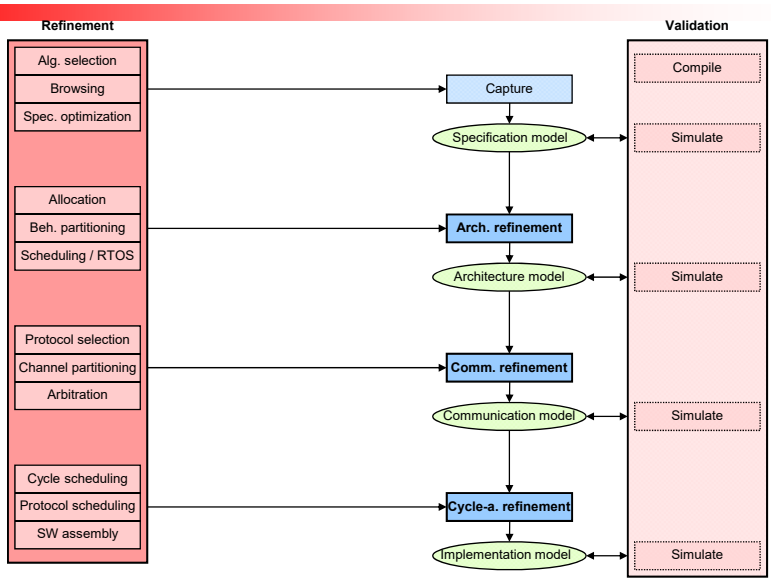

Refinement-based System Design Flow
System design
Validation flow
Capture
Algor. IP
Specification model
Architecture refinement
Comp. IP
Architecture model
Communication refinement
Proto. IP
Communication model
RTL IP
Hardware synthesis
Interface synthesis
Software compilation
RTOS IP
Implementation model
Compilation
Validation Analysis Estimation
Simulation model
EECS222: Embedded System Modeling, Lecture 14
(c) 2019 R. Doemer
4

SCE Modeling Engine



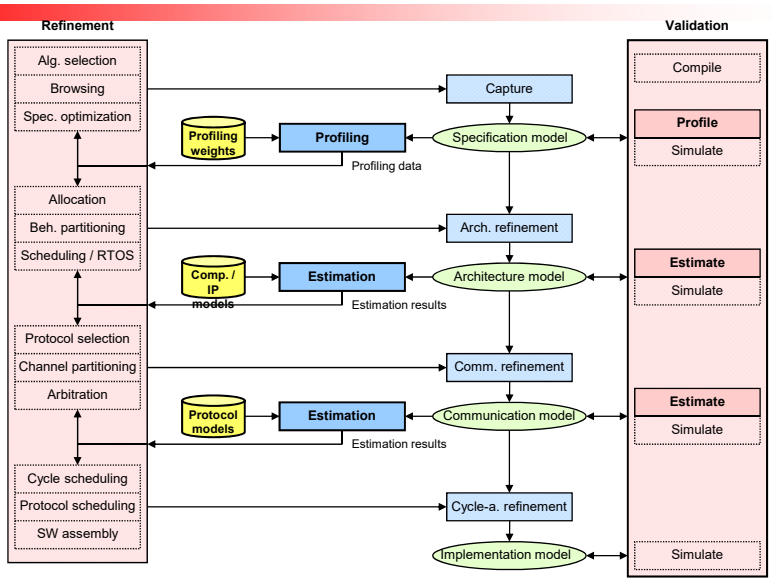
EECS222: Embedded System Modeling, Lecture 14 Copyright © 2003 CECS 5

SCE Refinement Engine

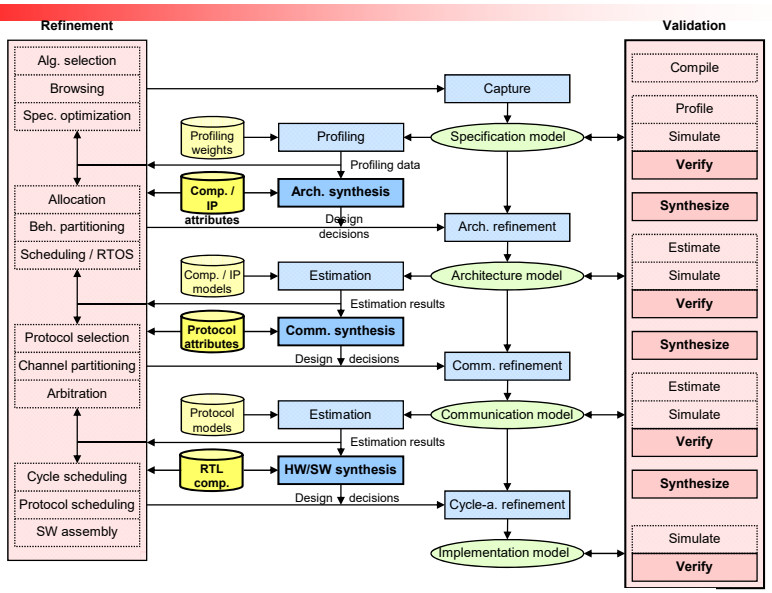


EECS222: Embedded System Modeling, Lecture 14 Copyright © 2003 CECS 6

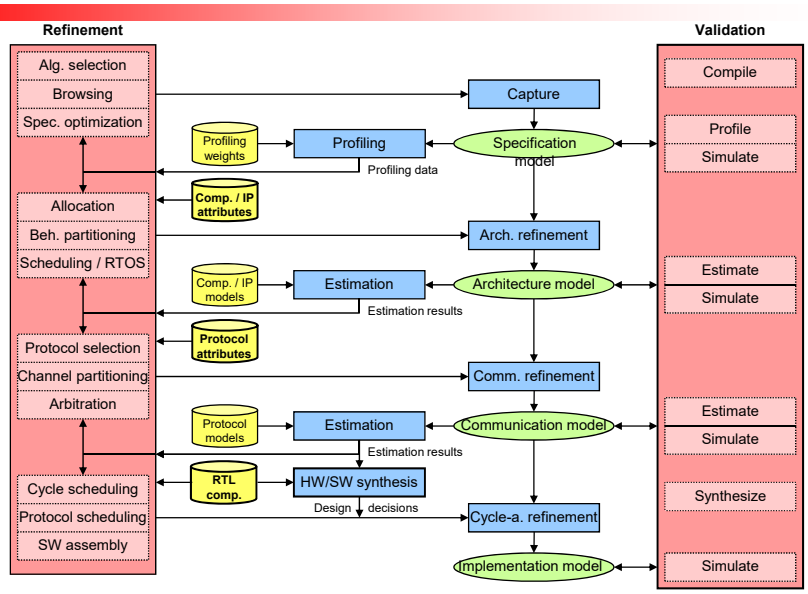
SCE Exploration Engine



SCE Synthesis Engine



System-on-Chip Environment (SCE)



SCE Demonstration

- Application Example: GSM Vocoder
  - Exploration of Specification Model
    - Simulation
    - Profiling
    - Performance estimation

  - Interactive demonstration

SCE Demonstration

- Application Example: GSM Vocoder
  - Enhanced full-rate voice codec
  - GSM standard for mobile telephony (GSM 06.10)
  - Lossy voice encoding/decoding
    - Incoming speech samples @ 104 kbit/s
    - Encoded bit stream @ 12.2 kbit/s
    - Frames of 4 x 40 = 160 samples (4 x 5ms = 20ms of speech)
  - Real-time constraint:
    - max. 20ms per speech frame
      (max. total of 3.26s for sample speech file)
  - SpecC specification model
    - 29 hierarchical behaviors (9 par, 10 seq, 10 fsm)
    - 73 leaf behaviors
    - 9139 formatted lines of SpecC code
      (~13000 lines of original C code, including comments)

Project Assignment 7

- Task: Performance Estimation of the Canny Edge Detector
  - Profiling to estimate relative computational complexity
  - Instrumentation to measure absolute timing as reference
- Steps
  1. Profile the application, identify performance bottle-necks
     - SpecC: Use SCE profiling tools
     - SystemC: Use GNU profiling tools
  2. Instrument the application, measure timing on reference platform
- Deliverables
  - **`Canny.sc`** or **`Canny.cpp`** (choose one!)
  - **`Canny.txt`** (with numerical values for obtained results)
- Due
  - Next week: February 27, 2019, 6pm

Project Assignment 7

- Step 1: Profile the application components
  - ➢ Performance Estimation of the Canny Edge Detector
  - – SpecC model profiling: Use SCE profiler
    - ➢ **`/opt/sce/bin/sce`**
    - Create a new project, import SpecC source code
    - Compile and simulate in SCE (with instrumentation)
    - Run the profiler, analyze tables and charts
  - – SystemC model profiling: Use GNU profiler
    - ➢ **`g++ -pg, gprof`**
    - Compile the SystemC source code with option **`-pg`**
    - Run the simulation once (with instrumentation, **`gmon.out`**)
    - Run the profiler: **`gprof Canny`**
    - Validate the reported call tree
    - Analyze the "flat profile" for the DUT components

Project Assignment 7

- Step 1: Profile the application components, obtain relative computational complexity
  - – Expected complexity comparison (in **`Canny.txt`**):

```
Gaussian_Smooth                    ...%
|------ Receive_Image      ...%
|------ Gaussian_Kernel    ...%
|------ BlurX              ...%
\------ BlurY              ...%
Derivative_X_Y                     ...%
Magnitude_X_Y                      ...%
Non_Max_Supp                       ...%
Apply_Hysteresis                   ...%
                                   100%
```

Project Assignment 7

- Step 2: Instrument the application components
  - ➢ Performance Measurement of the Canny Edge Detector
  - ➢ Since we do not have a prototyping platform available, we use the department server as reference
  - – Instrument your model source code:

```
#include <time.h>
clock_t Tstart, Tstop;
double T1 = 0.0;
...
Tstart = clock();
f();
Tstop = clock();
T1 = (double)(Tstop-Tstart)/CLOCKS_PER_SEC;
```

  - – Use global variables for this temporary instrumentation

EECS222: Embedded System Modeling, Lecture 14 (c) 2019 R. Doemer 15

Project Assignment 7

- Step 2: Instrument the application components, obtain absolute timing on reference platform
  - – Expected complexity comparison (also in **`Canny.txt`**):

```
Gaussian_Smooth                    ...sec ...%
|------ Receive_Image      ...sec ...%
|------ Gaussian_Kernel    ...sec ...%
|------ BlurX              ...sec ...%
\------ BlurY              ...sec ...%
Derivative_X_Y                     ...sec ...%
Magnitude_X_Y                      ...sec ...%
Non_Max_Supp                       ...sec ...%
Apply_Hysteresis                   ...sec ...%
                                          100%
```

EECS222: Embedded System Modeling, Lecture 14 (c) 2019 R. Doemer 16