

EECS 222C: System-on-Chip Software Synthesis Lecture 4

Rainer Dömer

doemer@uci.edu

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

Lecture 4: Overview

- Assignment 2
- System-on-Chip Design with SpecC
 - Top-down Design Methodology
 - Refinement-based System Design Flow
 - System-on-Chip Environment (SCE)
 - Interactive Demonstration
- Assignment 3

Assignment 2

1. Practice SpecC Tools

- Setup
 - `source /opt/sce-20080601/bin/setup.csh`
- Examine simple examples
 - `mkdir simple_tests`
 - `cd simple_tests`
 - `cp $SPECC/examples/simple/* .`
 - `ls`
 - `vi HelloWorld.sc`
- Practice the compiler
 - `man scc`
 - `scc HelloWorld -sc2out -vv -ww`
- Practice the simulator
 - `./HelloWorld`
- Practice the tools
 - `man sir_tree`
 - `scc Adder -sc2sir -o Adder.sir`
 - `sir_tree -bt Adder.sir FA`

EECS222C: SoC Software Synthesis, Lecture 4

(c) 2008 R. Doemer

3

Assignment 2

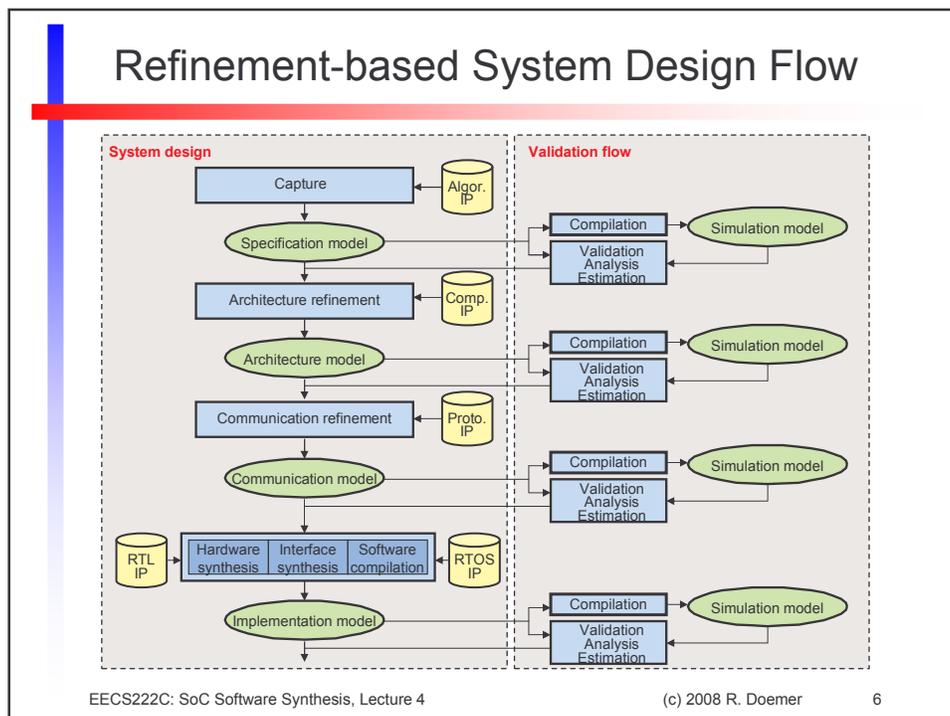
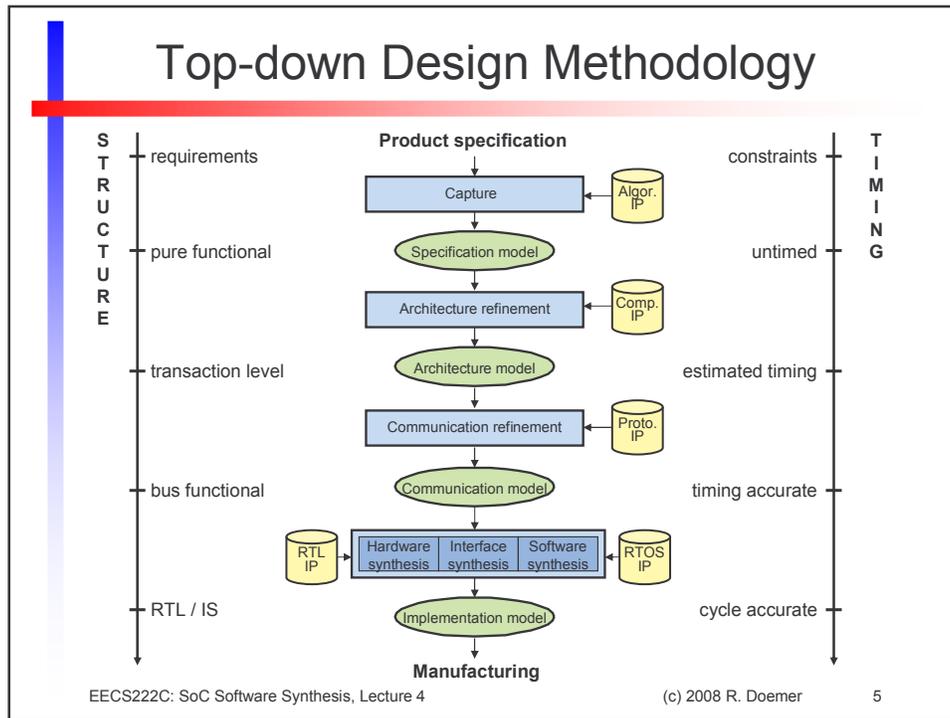
2. Convert JPEG Encoder application into SpecC Model

- Version 0
 - Compile JPEG Encoder with SpecC compiler
 - `scc jpegencoder.sc -vv -ww`
- Version 1
 - Introduce test bench
 - Stimulus behavior (`ReadBmp`)
 - Design-under-Test behavior (`JPEGencoder`)
 - » Seq. child behaviors (`DCT1`, `DCT2`, `Quantize`, `Zigzag`, `Huffman`)
 - » Communication through variables mapped to ports
 - Monitor behavior (`DiffGolden`)
- Version 1.1
 - Add timing to test bench
 - Print encoding time for each block (in Stimulus and/or Monitor)
- Version 2.0
 - Create a parallel model
 - Change DUT execution to `'par { }'`
 - Change communication to typed `double_handshake` channels
- Version 2.1
 - Create a pipelined model
 - Change communication to typed `queue` channels

EECS222C: SoC Software Synthesis, Lecture 4

(c) 2008 R. Doemer

4



Refinement-based System Design Flow

- Step 1: Architecture Refinement
 - Allocation of Processing Elements (PE)
 - Type and number of processors
 - Type and number of custom hardware blocks
 - Type and number of system memories
 - Mapping to PEs
 - Map each behavior to a PE
 - Map each channel to a PE
 - Map each variable to a PE
 - Result:
System architecture of concurrent PEs
with abstract communication in channels

EECS222C: SoC Software Synthesis, Lecture 4

(c) 2008 R. Doemer

7

Refinement-based System Design Flow

- Step 2: Scheduling Refinement
 - For each PE, serialize the execution of behaviors to a single thread of control
 - Option (a): Static scheduling
 - For each set of concurrent behaviors, determine fixed order of execution
 - Option (b): Dynamic scheduling by RTOS
 - Choose scheduling policy, i.e. Round-robin or priority-based
 - For each set of concurrent behaviors, determine scheduling priority
 - Result:
System model with abstract RTOS scheduler inserted in each PE

EECS222C: SoC Software Synthesis, Lecture 4

(c) 2008 R. Doemer

8

Refinement-based System Design Flow

- Step 3: Communication Refinement
 - Allocation of system busses
 - Type and number of system busses
 - Type of bus protocol for each bus (if applicable)
 - Number of transducers (if applicable)
 - System connectivity
 - Mapping of channels to busses
 - Map each communication channel to a system bus (or multiple busses, if applicable)
 - Result:
 - Bus-functional model of the system

Refinement-based System Design Flow

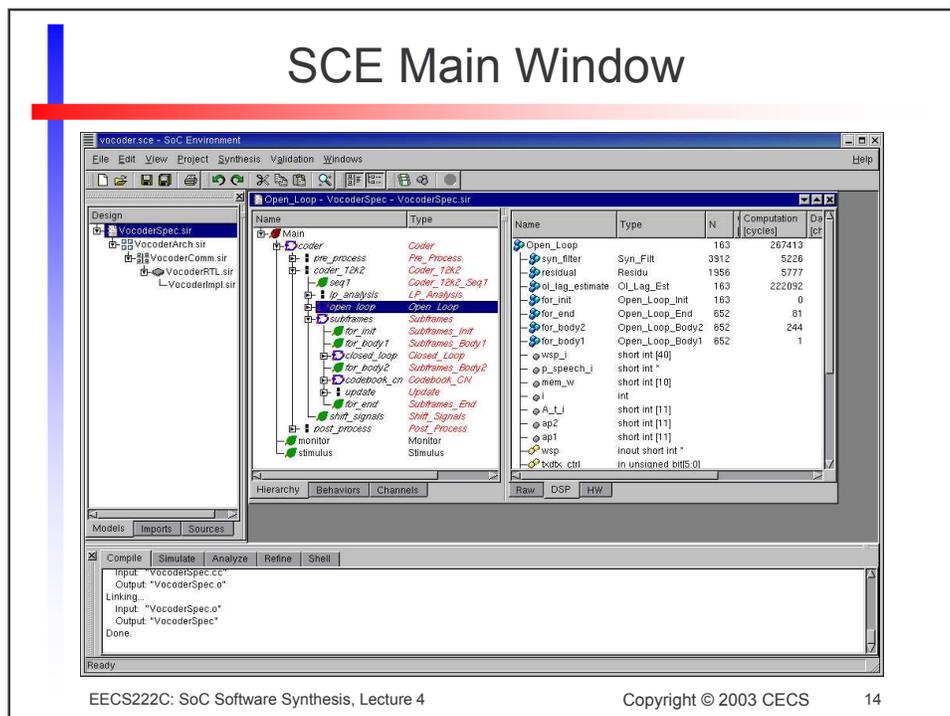
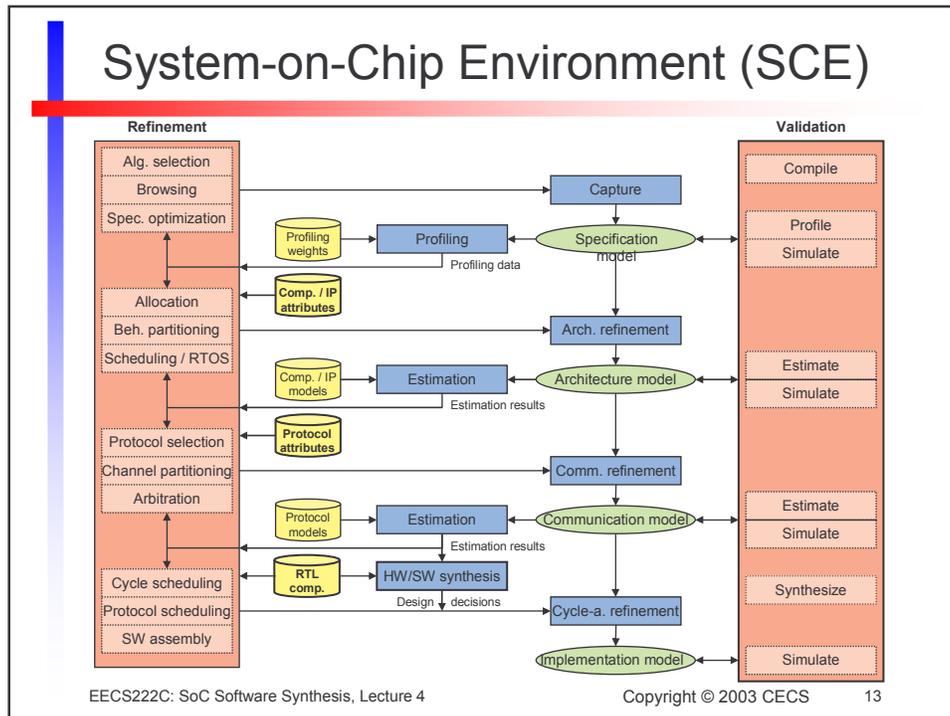
- Step 4: Hardware Refinement (for HW PE)
 - Allocation of Register Transfer Level (RTL) components
 - Type and number of functional units (e.g. adder, multiplier, ALU)
 - Type and number of storage units (e.g. registers, register files)
 - Type and number of interconnecting busses (drivers, multiplexers)
 - Scheduling
 - Basic blocks assigned to super-states
 - Individual operations assigned to states (clock cycles)
 - Binding
 - Bind functional operations to functional units
 - Bind variables to storage units
 - Bind assignments/transfers to busses
 - Result:
 - Clock-cycle accurate model of each HW PE
 - Output: Synthesizable Verilog description

Refinement-based System Design Flow

- Step 5: Software Refinement (for SW PE)
 - C code generation
 - For selected target processor
 - RTOS targeting
 - For selected target RTOS
 - Compilation to Instruction Set Architecture
 - for Instruction Set Simulation (ISS)
 - Assembly
 - Result:
Clock-cycle accurate model of each SW PE
 - Output: downloadable binary image

System-on-Chip Environment (SCE)

- Integrated Development Environment (IDE) with support of:
 - Graphical frontend (*sce*, *scchart*)
 - SLDL-aware editor (*sced*)
 - Compiler and simulator (*scc*)
 - Profiling and analysis (*scprof*)
 - Architecture refinement (*scar*)
 - RTOS refinement (*scos*)
 - Communication refinement (*sccr*)
 - RTL refinement (*scrtl*)
 - Software refinement (*sc2c*)
 - Scripting interface (*scsh*)
 - Tools and utilities ...



SCE Source Editor

The screenshot shows the SCE Source Editor interface. On the left, a project hierarchy tree is visible under 'VocoderSpec.sir', including 'VocoderArch.sir', 'VocoderComm.sir', 'VocoderRTL.sir', and 'VocoderImpl.sir'. The main editor window displays the source code for 'behavior Coder_13-2_Seq1'. The code includes comments and logic for initializing pointers to speech vectors and setting up DSP and HW components.

```

behavior Coder_13-2_Seq1
{
  in Word16 speech_proc[L_FRAME],
  Word16 old_speech[L_TOTAL],
  Word16 *speech,
  out Word16 *p_window,
  Word16 old_wsp[L_FRAME + PIT_MAK],
  out Word16 *wsp,
  Word16 old_exc[L_FRAME + PIT_MAK + L_INTERPOL],
  out Word16 *exc,
  out Flag reset,
  out DTxctrl txdtx_ctrl,
  in Flag reset_flag
}

implements Ireset
{
void init(void)
{
  /* Initialize pointers to speech vector. */

  speech = old_speech + L_TOTAL - L_FRAME; /* New speech */
  p_window = old_speech + L_TOTAL - L_WINDOW; /* For LFC window */

  /* Initialize pointers */
  wsp = old_wsp + PIT_MAK;
  exc = old_exc + PIT_MAK + L_INTERPOL;

  /* vectors to zero */
  Set_zero (old_speech, L_TOTAL);
  Set_zero (old_exc, PIT_MAK + L_INTERPOL);
  Set_zero (old_wsp, PIT_MAK);

  txdtx_ctrl = TX_SP_FLAG | TX_VAD_FLAG;
  ptxch = 1;
}
}
    
```

At the bottom of the screenshot, the text 'EECS222C: SoC Software Synthesis, Lecture 4' and 'Copyright © 2003 CECS' is visible, along with the slide number '15'.

SCE Hierarchy Displays

The screenshot displays two views of the SCE Hierarchy. The left window, titled 'Open_Loop - VocoderSpec - SpecC Hierarchy Chart', shows a flowchart with blocks for 'for_init', 'for_body1', 'for_body2' (containing 'weight_n1' and 'weight_n2'), 'residual', and 'sps_filler'. The right window, titled 'Coder - VocoderComm - SpecC Hierarchy Chart', shows a more complex block diagram with a 'Coder' block at the top, multiple signal lines, and 'DSP' and 'HW' blocks at the bottom.

At the bottom of the screenshot, the text 'EECS222C: SoC Software Synthesis, Lecture 4' and 'Copyright © 2003 CECS' is visible, along with the slide number '16'.

SCE Compiler and Simulator

EECS222C: SoC Software Synthesis, Lecture 4

Copyright © 2003 CECS

17

SCE Profiling and Analysis

Type	N	Computation [cycles]
code	652	8617
upd_sh	652	1250
Ex_Syn_Upd_Sh	652	7367

EECS222C: SoC Software Synthesis, Lecture 4

Copyright © 2003 CECS

18

Interactive Demonstration

- Design 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 $4 \times 40 = 160$ samples ($4 \times 5\text{ms} = 20\text{ms}$ 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)

Assignment 3

1. Become familiar with the System-on-Chip Environment (SCE)
 - Setup
 - Note that we will use the 2004 version of SCE for the tutorial:
 - `source /opt/sce-20041007/bin/setup.csh`
 - `rm -rf ~/.sce`
 - `mkdir demo`
 - `cd demo`
 - `setup_demo`
 - Open the SCE Tutorial document
 - `acroread SCE_Tutorial/sce-tutorial.pdf &`
 - To protect the environment and save some trees, please *do not print* the tutorial document! It contains 250 pages and you will likely read it only once...;-)
 - Follow the SCE Tutorial instructions
 - `sce &`
 - ...
 - Cleanup
 - When done (or to start over), clean up your demo directory
 - `cd ..`
 - `rm -rf demo`

Assignment 3

2. Simulate your JPEG Encoder model in SCE
 - Setup
 - Note that we will use the 2008 version of SCE for the JPEG Encoder:
 - `source /opt/sce-20080601/bin/setup.csh`
 - `rm -rf ~/.sce`
 - `cd jpegencoder`
 - `sce`
 - Create a new project in SCE
 - **Project->New**
 - **Project->Settings**
 - Set verbosity level to 3 and warning level to 2
 - Adjust any other options the compiler may need to compile your model
 - **Project->SaveAs "jpegencoder.sce"**
 - Load your design model into SCE
 - **File->Import "jpegencoder.sc"**
 - **Project->AddDesign**
 - Right-click on `jpegencoder.sir` in the project window, and **Rename** the model to `JPEGencSpec`
 - Compile and simulate your model in SCE
 - **Validation->Compile**
 - **Validation->Simulate**

EECS222C: SoC Software Synthesis, Lecture 4

(c) 2008 R. Doemer

21

Assignment 3

3. Analyze your JPEG Encoder model in SCE
 - Setup
 - ...continued from step 2 (previous page)
 - View the structural hierarchy chart
 - Select the **main** behavior in the behavior browser
 - Right-click ->**Chart**
 - Double-click the chart to add a level of hierarchy
 - **View->Connectivity**
 - ...
 - **Window->Print...** to file `"jpegencoder.ps"`
 - Deliverables
 - SpecC source file
 - `"jpegencoder.sc"`
 - Hierarchy chart
 - `"jpegencoder.ps"`
 - Due
 - by Friday, Oct 24, 2008, at noon
 - by email to `doemer@uci.edu` with subject "EECS222C HW3"

EECS222C: SoC Software Synthesis, Lecture 4

(c) 2008 R. Doemer

22