EECS 222C: System-on-Chip Software Synthesis Lecture 10

Rainer Dömer

doemer@uci.edu

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

Lecture 10: Overview

- Course Administration
  - Final exam
  - Final course evaluation
- Project Discussion
  - JPEG Encoder Design Assignments
  - Final report
- Review of Embedded Software Synthesis
  - SLDL Modeling, down to…
  - Instruction Set Simulation
- Outlook
  - Next course on SoC Design

Course Administration

- Final Exam
  - Date and time
    - Friday, December 12, 2008, 2-4pm
  - Location
    - Room ET201
  - Format
    - Delivery of Final Technical Report
      - Option 1: Hardcopy
      - Option 2: PDF by email to **doemer@uci.edu**
    - Hard deadline
      - December 12, 2008, 4pm

Course Administration

- Final Course Evaluation
  - 8th through 10th week
  - Nov. 17, 2008 through Dec. 7, 2008, 11:45pm
  - Closes end of this week (Sunday night)
  - Online via EEE Evaluation application
- Evaluation of Course and Instructor
  - Voluntary
  - Anonymous
  - Very valuable
- Help to improve this class!
  - Please spend 5 minutes!

Project Discussion: Assignment 1

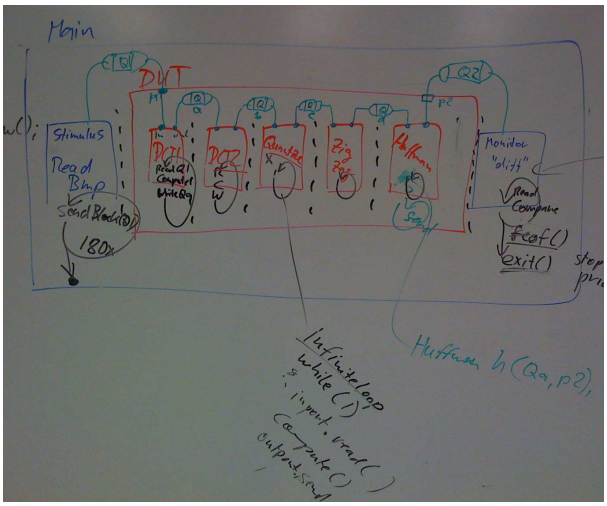
- Login on Server via SSH
  - **`epsilon.eecs.uci.edu`**
  - Account infos have been emailed
- Install JPEG Encoder example
  - **`mkdir eecs222c`**
  - **`cd eecs222c`**
  - **`gtar xvzf /home/doemer/EECS222C_F08/jpegencoder.tar.gz`**
  - **`cd jpegencoder`**
  - **`Make`**
- Become familiar with the application and its structure
  - Browse and read the source files
  - Combine all code into one single ANSI-C file
    - Keep the functional hierarchy, we need it!
  - Draw a block diagram of the functions and their communication

Project Discussion: 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

Project Discussion

- Targeted Specification Model



Project Discussion: Assignment 4

2. Complete 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

Project Discussion: Assignment 4

3. Simulate your JPEG Encoder model "V2.1" 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`

No warnings!
Successful!

Project Discussion: Assignment 4

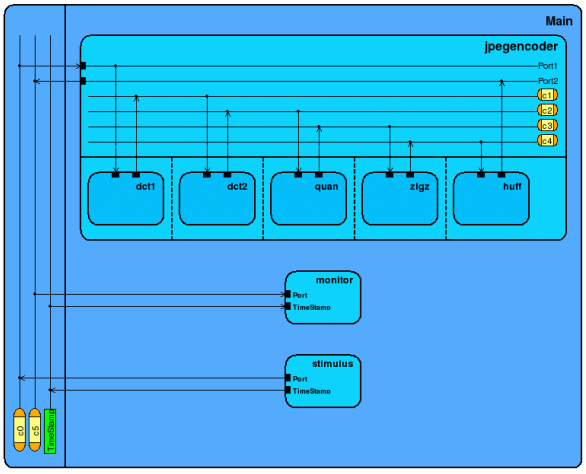
4. Analyze your JPEG Encoder model in SCE
   - Setup
     - …continued from step 2 (previous page)
   - View the structural hierachy chart
     - Select the `Main` behavior in the behavior browser
     - Right-click `->Chart`
     - Double-click the chart to add a level of hierarchy
     - `View->Connectivity`
     - `...`
     - `Window->Print…` to file `"jpegencoder.ps"`
- Deliverables
  - SpecC source file
    - `"jpegencoder.sc"` **One single/complete file!**
  - Hierarchy chart
    - `"jpegencoder.ps"` **One chart with connectivity!**
- Due
  - by Friday, Oct 31, 2008, at noon
  - by email to `doemer@uci.edu` with subject "EECS222C HW4"

Project Discussion

- Excellent results from Assignment 4!
  - 90% of submissions achieved scores of 95% or better (although SpecC was completely new for most students)
- Continue design flow with a "perfect" model
  - Improved version of "best" submission
    - Re-formatted code to create "clean" SpecC source
      - `scc jpegencoder -sc2sc -i best_student_model.sc -o jpegencoder.sc -vv -www -sl -sn -psi -pui`
    - Zero warnings
    - Clean hierarchy
      - `scc jpegencoder -sc2sir`
      - `sir_tree -blt jpegencoder.sir`
    - No global variables, no global functions
      - `sir_list -BCI +VF -lt jpegencoder.sir`
    - Proper communication from Huffman to Monitor
    - Detailed timing for each encoded block
    - Moved writing of **"test.jpg"** file into Monitor

Project Discussion

- "Perfect" Specification Model



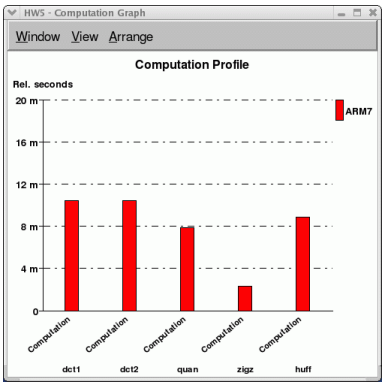
Project Discussion: Assignment 5

1. Examine the "perfect" JPEG Encoder source code
   - **`/home/doemer/EECS222C_F08/jpegencoder.sc`**
2. Examine the "perfect" JPEG Encoder model in SCE
   - Setup
     - Same as before (use SCE version 20080601)
   - Browse the structural hierachy
   - View the hierarchy chart
   - Validate the model (compile and simulate)
   - Profile, analyze, estimate the model
     - For a single ARM_7TDMI CPU
     - For complexity of "Computation"

- Deliverables
  - Bar graph of Computation Profile
    - **`"ARM7TDMI.ps"`**
- Due
  - by Friday, Nov 7, 2008, at noon
  - by email to **`doemer@uci.edu`** with subject "EECS222C HW5"

Project Discussion

- Design Space Exploration
  - Estimation results
    - For ARM_7TDMI CPU at 100 MHz
    - For encoding of 180 blocks
      - ChenDCT1: 10.41ms
      - ChenDCT2: 10.41ms
      - Quantize: 7.84ms
      - Zigzag: 2.32ms
      - Huffman: 8.88ms



Project Discussion

- Design Space Exploration
  - Timing back-annotation
    - Automatic approach
      - Use SCE Architecture Refinement
      - Generate Architecture Model
        - » enable “Insert avg. delays”
      - Estimated times are automatically inserted
        - » at granularity of leaf behaviors only!
        - » Problem: no per-block timing during simulation!
      - Example:

Better:
Per-Block Timing!

```
behavior ChenDCT1(...)
{
  void main(void)
    { int v1, v2, …;

      waitfor 10411200000ull;
      while(1)
      { waitfor 10411200000ull / 180;
        Port1.receive( &in_block);
        if (iter % 2 == 1)
        { ...
```


Project Discussion: Assignment 6

- Design Space Exploration
  1. Timing back-annotation
     - Manual insertion of estimated computation delays
       - Start from “perfect” specification model
         - » **`jpegencoder.sc`**
       - Add timing statements (**`waitfor`** after **`port.receive()`**)
         - » ChenDCT1: 10411200ns / 180
         - » ChenDCT2: 10411200ns / 180
         - » Quantize: 7839030ns / 180
         - » Zigzag: 2316600ns / 180
         - » Huffman: 8882810ns / 180
       - Save as timed model
         - » **`JpegTimed.sc`**
     - When executed, the resulting model should end at time 10574ms:
       - **`10574: Monitor exits simulation.`**

Project Discussion: Assignment 6

- Design Space Exploration
  2. Architecture Exploration
     - Explore various system architectures
       - Use only ARM_7TDMI processors
       - Use only 100MHz core clock frequency
       - Use only 50MHz AMBA AHB bus
       - Vary between 1 and 5 CPUs
       - Vary the mapping of blocks in the DUT to CPUs
     - Note:
       Do not let the architecture refinement tool insert additional timing!
     - Example:
       - Use 3 CPUs, ARM1, ARM2, and ARM3
       - Map DCT1 to ARM1
       - Map DCT2 to ARM2
       - Map Quantize, Zigzag, and Huffman to ARM3
     - Note:
       Without scheduling, any architecture model will end at time 10574ms

Project Discussion: Assignment 6

- Design Space Exploration
  3. Scheduling Exploration
     - Explore various scheduling strategies for each selected CPU
     - Choose from
       - Static scheduling
         - » with varying execution order
       - Round-Robin scheduling
       - Priority-based scheduling
         - » with varying priorities
     - Example:
       - 3 ARM CPUs with mapping as above
       - ARM1 statically scheduled
       - ARM2 statically scheduled
       - ARM3 scheduled with Round-Robin
     - When executed, the example model should end at time 19154ms

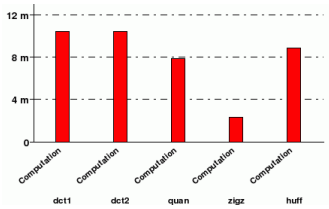
Project Discussion: Assignment 6

- Design Space Exploration
  4. Deliverable
     - Text file "`JPEG_Exploration.txt`" with table:
       - "Best" mapping and scheduling for architecture with 1 CPU
       - "Best" mapping and scheduling for architecture with 2 CPUs
       - "Best" mapping and scheduling for architecture with 3 CPUs
       - "Best" mapping and scheduling for architecture with 4 CPUs
       - "Best" mapping and scheduling for architecture with 5 CPUs
     - For each "best" architecture above, note the overall execution time in the table.

- Due
  - by Friday, Nov 14, 2008, at noon
  - by email to **`doemer@uci.edu`** with subject "EECS222C HW6"

Project Discussion

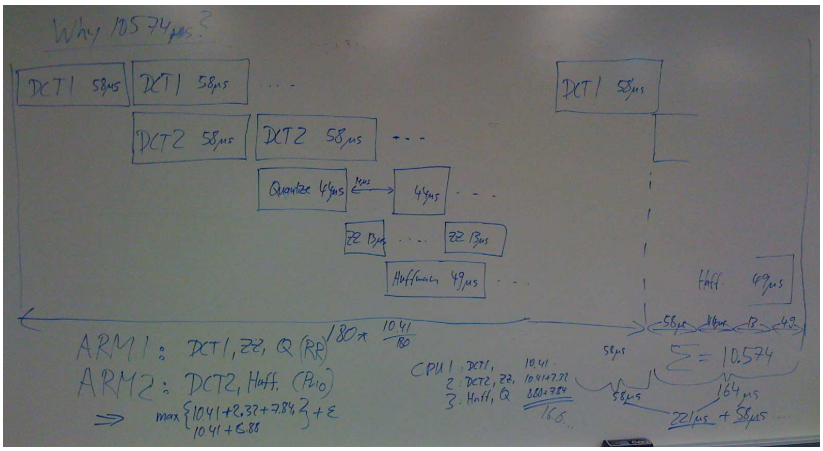
- Design Space Exploration
  - Estimation results
    - For ARM_7TDMI CPU at 100 MHz
    - For encoding of 180 blocks
      - ChenDCT1: 10.41ms (10411200ns / 180 ≈ 58us)
      - ChenDCT2: 10.41ms (10411200ns / 180 ≈ 58us)
      - Quantize: 7.84ms (7839030ns / 180 ≈ 44us)
      - Zigzag: 2.32ms (2316600ns / 180 ≈ 13us)
      - Huffman: 8.88ms (8882810ns / 180 ≈ 49us)
        - » Sum: 39.86ms (per block ≈ 221us)



  - Exploration results (in-class discussion)
    - Trade-offs between Cost and Speed
    - Limitations

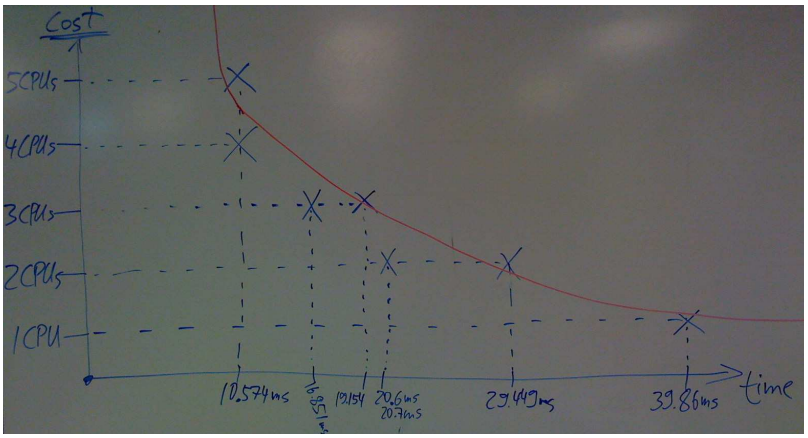
Project Discussion

- Detailed Timing Analysis



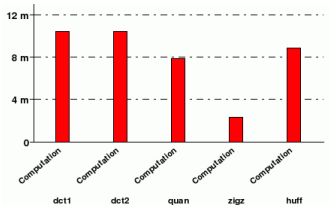
Project Discussion

- Design Space Exploration: Cost/Speed Trade-off



Project Discussion

- Design Space Exploration
  - Estimation results
    - For ARM_7TDMI CPU at 100 MHz
    - For encoding of 180 blocks
      - ChenDCT1: 10.41ms (10411200ns / 180 ≈ 58us)
      - ChenDCT2: 10.41ms (10411200ns / 180 ≈ 58us)
      - Quantize: 7.84ms (7839030ns / 180 ≈ 44us)
      - Zigzag: 2.32ms (2316600ns / 180 ≈ 13us)
      - Huffman: 8.88ms (8882810ns / 180 ≈ 49us)
        - » Sum: 39.86ms (per block ≈ 221us)
  - Reality-Check!
    - 40ms for encoding the test JPEG image…
    - …is that fast or is it slow???

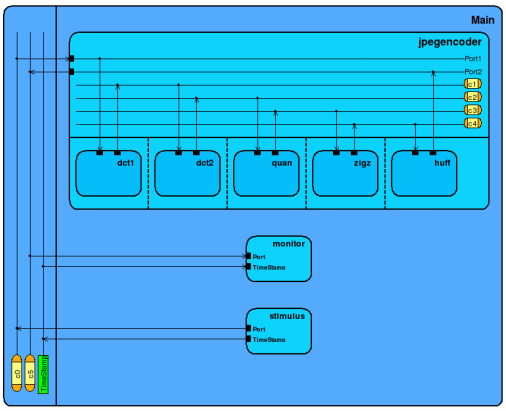


Project Discussion

- Design Space Exploration
  - Estimation results
    - For ARM_7TDMI CPU at 100 MHz
    - For encoding of 180 blocks
      - Sum: 39.86ms (per block ≈ 221us)
  - Reality-Check
    - about 40ms for encoding a 116x96 pixel image in B&W
      - 116x96 pixel, that is only 0.011136 mega-pixels!
      - Need about a factor 1000 to scale up to 11.1 mega pixels!
      - Need another factor of 3 to support color!
    - For a high-resolution (11 mega-pixel) photo: about 120sec!!
    - We need to speed up by improving this architecture!
      - Let's add special-purpose hardware accelerators!

Project Discussion

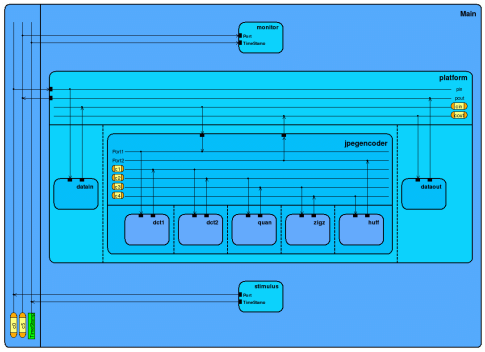
- Timed and fixed “perfect” Model



➢ Does not support Bus-Functional Model (BFM) for CPU

Project Discussion

- Platform Model



– Communication in **`jpegencoder`** can be refined to actual CPU bus
  - I/O units **`datain`** and **`dataout`** convert between
    – Abstract test bench communication (typed double-handshake)
    – BFM communication via CPU bus

Project Discussion

- Current SCE Limitations
  - Instruction Set Simulator
    - Only available for ARM_7TDMI
    - Max. 1 system-wide instance
  - RTOS
    - Only available port for ARM_7TDMI is micro-OS II
    - Requires priority-based scheduling
      - with different priorities for each task
  - Code generator
    - CPU-internal channels limited to
      - Type-less **`c_handshake`**
      - Type-less **`c_double_handshake`**
    - CPU-external channels
      - Type-less **`c_handshake`**
      - Type-less and typed **`c_double_handshake`**
      - Type-less and typed **`c_queue`**

Project Discussion: Assignment 8

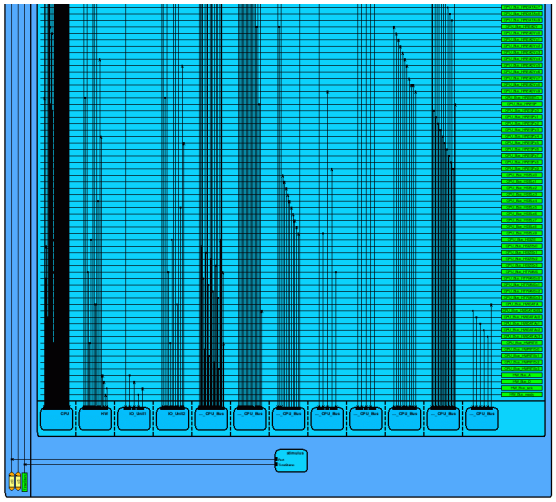
- Software Synthesis and Instruction Set Simulation
  1. Similar to the demo given in Lecture 8, refine the JPEG Encoder example down to a pin- and cycle-accurate Instruction Set Model
     - For details, see
       - **`/home/doemer/EECS222C_F08/HW8.txt`**
     - Platform Model is available here
       - **`/home/doemer/EECS222C_F08/JpegPlatform.sc`**
- Deliverables
  - Hierarchy Chart of ISS Model
    - Print out from SCE Chart window, "PlatformISS.pdf"
    - Manually drawn version (as PDF, or on paper)
  - Log of Instruction Set Simulation
    - "PlatformISS.log"
- Due
  - by Friday, Dec 5, 2008, at noon

Project Discussion

- Creating the Instruction Set Simulation Model
  - System Architecture
    - 4 processing elements (PEs)
      - main ARM CPU (for quantize, zigzag, and huffman)
        - » Priority-based scheduling using microC-OS2 RTOS
      - a hardware accellerator for the DCTs
      - 2 I/O units for data in- and output
    - connected by
      - AMBA-AHB bus (the built-in CPU bus)
      - custom double-handshake bus
  - Problems:
    - C code generator inserts unwanted TaskDelay()
      - See the work-around discussed on the course noteboard
    - Bus-functional models “get stuck” after encoding for block 177
      - Suspected deadlock in RTOS scheduler…
      - TBD…
    - Encoding is much too slow
      - (~142 milliseconds for 177 blocks!) …TBD…

Project Discussion

- Hierarchy Chart of Bus Functional Models
  - PlatformComm
  - PlatformCommC
  - PlatformISS



Project Discussion

- Final Technical Report
  - Title
    - Embedded Software Synthesis for a JPEG Encoder
  - Contents
    - Describe the software synthesis approach
    - Outline the major steps in the synthesis flow
    - Use the JPEG Encoder application as case study
      - Use results of the assignments!
    - Conclude with a summary of the lessons learned
  - Length
    - about 10 pages

Project Discussion

- Final Technical Report: Suggested Outline
  - Title page
    - Project title, author, abstract
  - Introduction
    - Embedded SW design flow using SCE
  - Case Study on a JPEG Encoder
    - Specification model
    - Validation and estimation
    - Architecture exploration
    - Bus functional model
    - C code generation and cross-compilation
    - Instruction Set Model
  - Conclusion
    - Summary of results, lessons learned
  - References

Review

- Embedded Software Synthesis
  - SLDL Modeling
    - Create a system specification model in SpecC
  - Estimation and Exploration
    - Design space exploration to find suitable target platform
  - RTOS selection and targeting
    - Task creation, scheduling, and communication
  - Code Generation
    - automatically generate ANSI-C code for cross-compilation
  - Instruction Set Simulation (ISS)
    - Simulate the execution on the target platform
      - Cycle-accurate
      - Pin-accurate

Review

- Embedded Software Generation from SLDL
  - Software Design Flow in SCE
    - Designer-controlled, automatic generation of targeted ANSI-C code from SpecC SLDL
  - Paper presented at ASPDAC 2004
    - Haobo Yu, Rainer Doemer, Daniel Gajski
    - *"Embedded Software Generation from System Level Design Languages"*

Outlook

- Next course on System-on-Chip Design
  - EECS 222B
  - *"System-on-Chip Design and Exploration"*
  - Winter Quarter '09
  - Instructor: Daniel D. Gajski