EECS 222A
System-on-Chip Description and Modeling
Spring 2012

Assignment 5

Posted: May 18, 2012
Due: May 25, 2012 at 12pm (noon)

Topic: Refinement of Application Example

1. Setup:

We will use the same setup as for Assignment 4 and again use the latest SCE version:

```
source /opt/sce/bin/setup.csh
```

In order to use **`turnin`** to submit your deliverables, create a new directory named **`hw5`** (next to your **`hw4`** directory) and work there:

```
mkdir hw/hw5
cd hw/hw5
```

2. Application Example

With this assignment, we aim to complete the project of designing a system-level model for the *Canny Edge Detector* algorithm. This assignment starts one step after the point where Assignment 4 ended. For your reference, we have provided a solution file.

```
cp ~eecs222/EECS222A_S12/canny_a4_ref.sc .
```

As discussed in Lecture 7, we have further adjusted the application specification and added a timing report into the test bench. Again, some additional other adjustments have been applied also. So, for this Assignment 5, we have again prepared a source file to start from, namely **`canny_a5_start.sc`**.

To set up, use the following as starting point for this assignment:

```
cp ~eecs222/EECS222A_S12/canny_a5_start.sc canny.sc
cp ~eecs222/EECS222A_S12/Makefile .
cp ~eecs222/EECS222A_S12/golfcart.pgm .
cp ~eecs222/EECS222A_S12/ref_golfcart.pgm
       _s_0.60_l_0.30_h_0.80.pgm .
```

3. Instructions

Note that this assignment is much more open-ended than the previous ones! While there are some deliverables, it will be up to you how far you want to push it. The goal is to collect enough information so that you can wrap up this project in a technical report in the end.

The purpose of this assignment is to refine the specification model of our Canny application towards a high-level System-on-Chip implementation. Specifically, we will use the System-on-Chip Environment (SCE) for the refinement.

Step 1, Architecture Refinement: Import the specification model into SCE and compile and run it. Allocate PEs, namely an ARM7_TDMI as main processor, two HW_Virtual as IOunits (for DataIn and DataOut), and several HW_Standard units for the HW accelerated blocks (e.g. the parallel blocks in Gaussian Smooth). Next, use the Evaluate function of the SCE profiler to estimate the execution time.

Once you have a satisfactory allocation and mapping, run the Architecture Refinement tool to generate a platform model. Compile and simulate to ensure functional correctness.

Step 2, Scheduling Refinement: In the generated architecture model, schedule the PEs as you feel appropriate. Use the Scheduling Refinement tool to generate a corresponding scheduled model. Compile and simulate to ensure functional correctness. Check whether or not the model meets your timing constraints.

Step 3, Network Refinement: Allocate an AMBA_AHB bus as main bus of the processor (should already be pre-selected) and one (or more) simple DblHndShkBus for communication between HW units directly. Connect the busses to the PEs so that you have sufficient connectivity for all channels. Remember, the processor is always a master on its bus. HW units, on the other hand, can play the role of masters or slaves.

Use Network Refinement to generate a network model. Compile and simulate to ensure functional correctness.

Step 4, Communication Refinement: For each allocated system bus, assign appropriate link parameters. In many cases, SCE can assign suitable addresses automatically when you right-click into the Link Parameters window and select Autofill All Addresses.

Use Communication Refinement to generate a Transaction Level Model (TLM) or Pin-Accurate Model (PAM), or both. Compile and simulate to ensure functional correctness. Check whether or not the model meets your timing constraints.

Step 5, Instruction Level Model (optional): Using SCE, generate C code for the ARM7 processor. Cross-compile the generated code for the target processor (in your shell, go into the generated CPU subdirectory and type ‘make’). Compile

and simulate the generated ISS model to ensure functional correctness. Note the cycle-accurate timing reported by the simulator. While this might differ significantly from the estimated timing reported in earlier models, this is the actual accurate timing of the System-on-Chip that you have implemented. Congratulations!

Note that the above instructions have not been tested, your mileage may vary! Please make use of the class message board to discuss any issues you encounter during this adventure. Good luck!

4. Submission:

For this assignment, submit the following deliverables:

```
canny.tree
canny.txt
```

The file **`canny.tree`** should contain the output of the **`sir_tree`** tool with **`-blt`** options for the latest model that you obtained after the refinement.

The text file should briefly describe your refinement efforts. Please note the design decisions you took at each step, and what (if any) problems you encountered. Finally, note the type of model you have reached (e.g. TLM) and what its execution time was during simulation.

To submit the deliverables, change into the parent directory of your **`hw5`** directory and enter **`turnin`**. As in the previous assignments, the **`turnin`** command will locate the files listed above and allow you to submit them.

Remember that you can use the **`turnin`** tool to submit at any time before the deadline, *but not after!* Since you can submit as many times as you want (newer submissions will overwrite older ones), it is highly recommended to submit early and even incomplete work, in order to avoid missing the deadline.

Late submissions cannot be considered!

--
Rainer Doemer (EH3217, x4-9007, doemer@uci.edu)