EECS 222: Embedded System Modeling
Winter 2020

Assignment 6 (shortened)

Posted: February 11, 2020
Due: February 19, 2020 at 6pm

Topic: Hierarchical DUT model of the Canny Edge Decoder

1. Setup:

This assignment continues the modeling of our application example, the Canny Edge Detector, as a proper system-level specification model which we can then use to design our SoC target implementation. Now we will refine the previous model with a suitable structural hierarchy inside the design-under-test (DUT) block.

Again, we will use the same setup as for the previous assignments. Start by creating a new working directory, so that you can properly submit your deliverables in the end.

```
mkdir hw6
cd hw6
```

For functional validation, create again a symbolic link to the video frames, as follows:

```
ln –s ~eecs222/public/video video
```

As in the previous assignments, you have again the choice of using either SpecC or SystemC for your modeling and estimation. Both SLDLs are equally suitable for this assignment.

As starting point, you can use your own SLDL model which you have created in the previous Assignment 5. Alternatively, you may start from the provided solution for Assignment 5 which you can copy as follows:

```
cp ~eecs222/public/cannyA5_ref.sc canny.sc
cp ~eecs222/public/cannyA5_ref.cpp canny.cpp
```

You may also want to reuse the simple **`Makefile`** from the previous assignment:

```
cp ~eecs222/public/MakefileA5SpecC ./
cp ~eecs222/public/MakefileA5SystemC ./
```

Again, depending on whether you choose SpecC or SystemC for your modeling, rename the corresponding file into the actual **`Makefile`** to be used by **`make`**.

2. Refining the model with structural hierarchy in the DUT

Step 1: Create an additional level of hierarchy in the DUT

The original canny function consists of a sequence of function calls to five functions, namely **`gaussian_smooth`**, **`derivative_x_y`**, **`magnitude_x_y`**, **`non_max_supp`**, and **`apply_hysteresis`**. In the previous model, these are all local methods in the DUT. In contrast, we will now wrap those into separate blocks (child behaviors or modules, respectively) by themselves.

The expected instance tree of the **`Platform`** block should then look like this:

```
Platform platform
|------ DataIn din
|------ DUT canny
|       |------ Gaussian_Smooth gaussian_smooth
|       |------ Derivative_X_Y derivative_x_y
|       |------ Magnitude_X_Y magnitude_x_y
|       |------ Non_Max_Supp non_max_supp
|       \------ Apply_Hysteresis apply_hysteresis
\------ DataOut dout
```

If you are using SpecC, then the **`Canny`** behavior should be a sequential composition of its children. For communication, the child behaviors should be connected by ports directly mapped to connecting variables (which will be of type **`IMAGE`** or similar). Be sure to use only port directions **`in`** or **`out`**, not **`inout`** (**`inout`** ports would lead to problems later in the design process).

If you are using SystemC, then the **`Canny`** module should be a concurrent composition of its children (where each child will have its own thread). For communication, the child modules should be connected by ports mapped to connecting channels (which will be of **`sc_fifo<IMAGE>`** or similar type, and should have a buffer size of 1 element). Be sure to use suitable ports with directions **`sc_fifo_in`** or **`sc_fifo_out`**. Also, since some intermediate images in the Canny algorithm are generated by one function and then used by multiple others, you may need to duplicate some channel instances.

After this level of hierarchy has been added, you should compile and simulate your model to ensure functional correctness.

Step 2: Visualize the structural hierarchy of your model

For both SpecC and SystemC, we have tools available that can analyze and visualize the hierarchical structure and connectivity of the model. These tools can generate a simple ASCII-chart of the instance tree or a graphical image of the model structure and port mapping.

For SpecC, use the **`sir_tree`** and **`scchart`** tools provided by the System-on-Chip Environment (SCE), as follows:

```
source /opt/sce/bin/setup.csh
scc canny -sc2sir -vv
sir_tree –blt canny.sir
scchart canny.sir
```

For SystemC, use the **`tree`** and **`visual`** tools provided by the Recoding Infrastructure for SystemC (RISC), as follows:

```
source /opt/pkg/risc_v0.5.0/bin/setup.csh
tree canny.cpp
visual canny.cpp
```

As deliverable for this assignment, submit the generated hierarchy tree of your model as a text file. For SpecC, use **`sir_tree`**, as follows:

```
sir_tree –blt canny.sir > canny.tree
```

For SystemC, use **`tree`**, as follows:

```
tree canny.cpp > canny.tree
```

Note that you inspect the **`canny.tree`** file with your regular text editor.

3. Submission:

For this assignment, submit the following deliverables:

`canny.sc` *or* **`canny.cpp`**
`canny.tree`

To submit these files, change into the parent directory of your **hw6** directory and run the **`~eecs222/bin/turnin.sh`** script. As before, note that the submission script will ask for both the SystemC and SpecC models, but you need to submit only the one that you have chosen for your modeling.

Again, be sure to submit on time. Late submissions will not be considered!

To double-check that your submitted files have been received, you can run the **`~eecs222/bin/listfiles.py`** script.

For any technical questions, please use the course message board.

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