EECS 222: Embedded System Modeling
Spring 2017

Assignment 5

Posted: May 8, 2017
Due: May 15, 2017 at 12pm (noon)

Topic: Structural test bench model of the Canny Edge Decoder

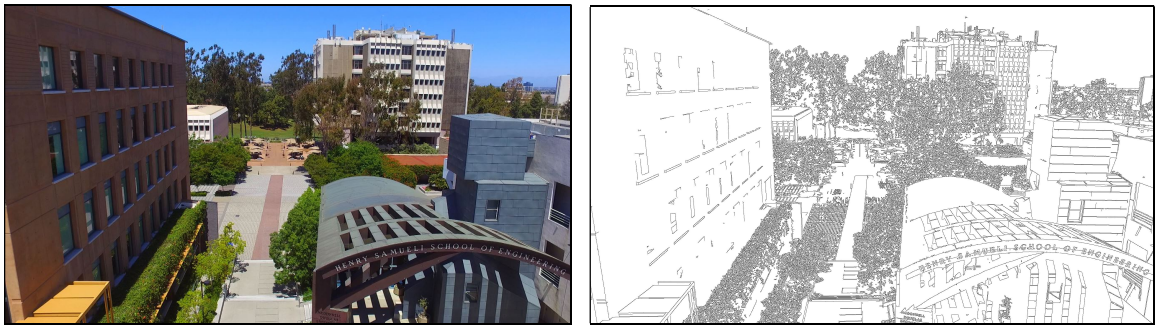
1. Setup:

This assignment is the next step in modeling 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. In this assignment, we will create a model with a suitable top-level structural hierarchy including a test bench. We will also convert the application from single image processing to real-time video handling. More specifically, we will process a sequence of images extracted from a stream of video frames.

We will use the same Linux account and the same remote servers as for the previous assignments. Start by creating a new working directory, so that you can properly submit your deliverables in the end.

```
mkdir hw5
cd hw5
```

Instead of the previous golf cart image, we will from now on use a video captured by a drone hovering over the Engingeering Plaza at UCI. Again, we will convert the original image into an edge image using the Canny algorithm.



The video is available in a shared directory on the server. To save disk space, do not copy the data into your account, but create a symbolic link to it, as follows:

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

As in the previous assignment, you have the choice of using either SpecC or SystemC for your modeling. Both languages are equally capable of describing the intended top-level structural hierarchy in this assignment. Also, both simulation environments are equally able to simulate your model in order to validate its functional correctness.

As starting point, you can use your own SLDL model which you have created in the previous Assignment 4. Alternatively, you may start from the provided solution for Assignment 4 which you can copy as follows:

```
cp ~eecs222/public/CannyA4_ref.sc Canny.sc
cp ~eecs222/public/CannyA4_ref.cpp Canny.cpp
```

For your convenience, we also provide a simple **`Makefile`** for use in this assignment which you can copy as follows:

```
cp ~eecs222/public/MakefileA4SpecC ./
cp ~eecs222/public/MakefileA4SystemC ./
```

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

A simple call to **`make`** will then compile your model into an executable, and a call to **`make test`** will simulate the model and compare the generated edge images against the reference images provided in the **`video`** directory.

2. Creating a test bench model with top-level structural hierarchy

Step 1: Convert the application to process a stream of video frames

Instead of the previous golf cart image (input file "**`golfcart.pgm`**" and output file "**`golfcart.pgm_s_0.60_l_0.30_h_0.80.pgm`**"), we will process a stream of video frames from now on.

Adjust the model source code so that it processes 20 images in a loop. The input images are named "**`video/EngPlaza001.pgm`**" and so on, with increasing numbering. After processing the image, your model should output the generated edge image as "**`EngPlaza001_edges.pgm`**", and so on.

With these new file names in place, you should be able to simulate and check your model with a simple **`make test`** command.

Step 2: Add a test bench and platform structure to your SLDL model

The purpose of this assignment is to introduce a proper test bench and overall structural hierarchy into our application model. In particular, we will introduce the top-level behavior **`Main`** (SpecC) or top-level module **`Top`** (SystemC). This will consist of three blocks, namely **`Stimulus`**, **`Platform`**, and **`Monitor`**.

The **`Platform`** behavior/module, in turn, should contain a dedicated input unit **`DataIn`**, an output unit **`DataOut`**, and the actual design under test **`DUT`**.

For communication, we will introduce queue-type channels from the respective standard channel library.

For SpecC modeling, we will use typed-queue channels (of size 2) to send and receive the image data between the behaviors. For reference, please see the simple example of a typed-queue channel in **`~eecs222/public/queue.sc`** which we have discussed at the end of Lecture 5. As data type for the queue channels, please define the following:

```
typedef unsigned char img[SIZE]; // image data type
```

On the other hand, for SystemC modeling, use the standard first-in-first-out channel **`sc_fifo<IMAGE>`** where **`IMAGE`** is the type of the data you need to communicate. Since **`IMAGE`** is an array and C++ does not provide an operator for array assignment, however, we need to wrap the array into a proper class with overloaded operators. To simplify this technicality, copy the **`class IMAGE`** from this provided file:

```
~eecs222/public/Image.cpp
```

For the above described top-level structural hierarchy, a total of four channel instances will be needed, two at the test bench level (**`Main`** behavior or **`Top`** module), and two within the **`Platform`** behavior.

Overall, your model should be structured as the following instance tree shows:

```
Main / Top
|------ Monitor monitor
|------ Platform platform
|       |------ DUT canny
|       |------ DataIn din
|       |------ DataOut dout
|       |------ c_img_queue q1
|       \------ c_img_queue q2
|------ Stimulus stimulus
|------ c_img_queue q1
\------ c_img_queue q2
```

Specifically, the **Main** behavior or **Top** module should instantiate **Stimulus**, **Platform** and **Monitor** in parallel. The **Stimulus** block should read the input image from the file system and pass it into the **Platform** via the first queue channel. Correspondingly, the **Monitor** should receive via the second channel the generated edge image from the **Platform** and write it out into the output file.

In the **Platform**, the **DataIn** block should, in an endless loop, receive an input image and pass it unmodified to the **DUT**. Similar, the **DataOut** block should, also in an endless loop, receive an input image from the **DUT** and pass it on. These two instances will be needed later during model refinement. They will allow our test bench to remain unmodified even when later in the design flow the communication to the DUT is implemented via detailed bus protocols.

Finally, the **DUT** block should contain the entire Canny algorithm source code. Its main thread will receive an image via the input port, call the **canny()** function to process it, and then send out the edge image via the output port. Since our target SoC will never stop working (unless its power is turned off), this processing will run in an endless loop, similar as the infinite loops in the **DataIn** and **DataOut** blocks.

Throughout your model recoding, ensure that it still compiles, simulates, and generates the correct output images. You are done with this assignment when the hierarchy described above has been created and your code compiles fine without errors or warnings.

In the end, your final model should not contain any global functions (except for **sc_main** in SystemC), neither any global variables, nor any wait-for-time statements. For communication, only standard library channels should be used (no plain events or user-defined channels).

3. Submission:

For this assignment, submit the following deliverables:

Canny.sc *or* **Canny.cpp**
Canny.txt

Again, the text file should briefly mention whether or not your efforts were successful and what (if any) problems you encountered. Please be brief!

To submit these files, change into the parent directory of your **hw5** 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.

Finally, remember that you can use the turnin-script to submit your work 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 hard deadline.

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