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

Assignment 3

Posted: April 27, 2012
Due: May 4, 2012 at 12pm (noon)

Topic: Structural Hierarchy for Application Example

1. Setup:

We will use the same setup as for Assignment 2 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 **hw3** (next to your **hw2** directory) and work in there:

```
mkdir hw/hw3
cd hw/hw3
```

2. Application Example

We will continue with the project of designing a system-level model for the *Canny Edge Detector* algorithm. This assignment basically starts where Assignment 2 ended. For your reference, we have provided a solution file.

```
cp ~eecs222/EECS222A_S12/canny_a2_ref.sc .
```

For this Assignment 3, we have prepared another file, **canny_a3_start.sc**, where some more clean-up has been done and a few other adjustments have been applied. So, use the following as starting point for this assignment:

```
cp ~eecs222/EECS222A_S12/canny_a3_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 .
```

Note the **Makefile** and the reference image. With these, you can compile, run, and test your code quickly (type **make** in your shell or simply use Eclipse).

3. Tools

Please refer to the previous assignment regarding helpful Linux tools for this project. Again, you may use any text editor of your choice and use the SpecC compiler via the command line interface. Alternatively, we offer our extended version of *Eclipse*, an open source IDE, which includes specific support for SpecC projects.

3.1 Eclipse Update:

In addition to *(a) SpecC syntax highlighting* and *(b) Automatic compiling on save*, supported features now include an Outline View and a Behavior Hierarchy display which provide you with overview and quick navigation of your code.

(c) Outline View: This is open by default. You can find it in the window at the right side of your editor. You can quickly navigate the code (jump to items) by clicking the listed items in Outline.

(d) Behavior Hierachy: This is not open initially. To open it, select from the menu **`Window -> Show View -> Other`**, find category **`SpecC`**, and select **`Behavior Hierachy`**. Once the Behavior Hierarchy is shown, you need to re-save (re-compile) your file to refresh the view. This will update the hierarchy display if your code compiles successfully.

In this assignment, both features should proof to be quite useful.

4. Instructions

The purpose of this assignment is to introduce a proper test bench and overall structural hierarchy into our application model.

Please time yourself for this assignment. At the end, we would like to know how many minutes this took for you. Thanks!

In particular, we will introduce the top-level behavior **`Main`** consisting of **`Stimulus`**, **`Platform`**, and **`Monitor`** behaviors. The **`Platform`** behavior, in turn, should contain an input unit **`DataIn`**, an output unit **`DataOut`**, and the actual design under test **`DUT`**.

For communication, we will introduce proper channels. Specifically, we will use queue channels (of size 2) to send and receive the image data between the behaviors. For the above structural hierarchy, four channels will be needed, two at the test bench level (**`Main`** behavior), and two within the **`Platform`** behavior.

As data type for the channels, please define the following:

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

Overall, your model should be structured as the following **sir_tree** log shows:

```
sir_tree -blt canny.sir
B i o   behavior Main
B i l   |------ Monitor monitor
B i c   |------ Platform platform
B i l   |       |------ DUT canny
B i l   |       |------ DataIn din
B i l   |       |------ DataOut dout
C i l   |       |------ c_img_queue q1
C i l   |       \------ c_img_queue q2
B i l   |------ Stimulus stimulus
C i l   |------ c_img_queue q1
C i l   \------ c_img_queue q2
```

The **Main** behavior should instantiate and run the **Stimulus**, **Platform**, and **Monitor** in parallel. In addition (optional), it may handle command line parameters (e.g. the image file name) and pass them into the **Stimulus** and/or **Monitor**.

The **Stimulus** behavior should read the input image from the file system and pass it into the **Platform** via a queue channel. Correspondingly, the **Monitor** should receive the edge image from the **Platform** and write it out into the output file.

In the **Platform**, the **DataIn** behavior should, in an endless loop, receive an input image and pass it unmodified to the **DUT**. Similar, the **DataOut** behavior should, also in an endless loop, receive an input image from the **DUT** and pass it on. These two behaviors 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** behavior should contain all the Canny algorithm code. Its main method should receive an image, call **canny()** to process it, and send out the edge image. Since our target chip will never stop working (unless its power is turned off), this processing should run in an endless loop, similar as the **DataIn** and **DataOut** behaviors.

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

3. Submission:

For this assignment, submit the following deliverables:

`canny.sc`
`canny.txt`

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

To submit the deliverables, change into the parent directory of your **`hw3`** 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!

In addition to these deliverables, we would like to ask you to complete a short survey on your experience with the extended **`eclipse`**. Please check the course message board for this survey.

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