Lab Assignment 3

Topic: Structural test bench model of the Canny Edge Decoder

Assigned:	**10/04/2019**
Due:	**10/11/2019 noon**

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.

1 Setup

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 lab3
cd lab3
```

As starting point, you can use your own SLDL model which you have created in the previous lab 2. Copy your code, input and reference image, as well as the Makefile.

Upgrade your Makefile. A simple call to `make` should compile your model into an executable, and a call to `make` test should simulate the model and compare the generated edge image against the reference image from your initial C model.

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

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. This will consist of three blocks, namely *Stimulus*, *Platform*, and *Monitor*. The *Platform* behavior, in turn, should contain a dedicated input unit *DataIn*, an output unit *DataOut*, and the actual design under test *DUT*.

For communication, 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 on Canvas.

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

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

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, 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 Refining the model with structural 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. While in the previous model, these are all local methods in the DUT, we will now wrap them 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
|------ c_img_queue q1
\------ c_img_queue q2
```

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 IMAGE or similar type). Be sure to use only port directions in or out, not inout (inout ports would lead to problems later in the design process).

Complete this step by validating that your refined model still compiles, simulates, and generates the correct output images. Ensure also that your code still compiles cleanly without any errors or warnings.

4 Write-up Instructions and Lab Assignment

1. Create a brief lab report. Include a diagram (and description) of the structure created.
2. Briefly describe whether or not your efforts were successful and what (if any) problems you encountered. We will take this input into account when grading your submission. Please be brief!

Please submit your writeup as a PDF and your modified source code (2). Your code should compile on the COE Linux systems. Validate the functional correctness of your code by comparing the compressed image against the reference code. Provide a Makefile to compile your modified code. Submit your PDF and the file tar.gz archive through Canvas. Please follow the naming convention with this format:

```
<myneu_id>_<hw|lab>_<Number>.<file_suffix>
```