Embedded System Design and Modeling

EE382V, Fall 2009

Lab #3

Specification

Due: October 22, 2009 in class (3:30pm)

Instructions:

- Please submit your solutions via Blackboard. Submissions should include a single PDF with the lab report and a single Zip or Tar archive with the source files.
- You are allowed to work in teams of up to three people and you are free to switch partners between labs and the project. Please submit one solution per team.

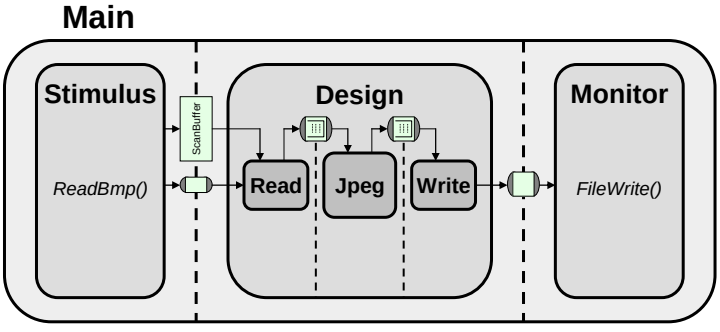
Part 1: Digital Camera Specification Model

The purpose of this lab is to finalize the SpecC model of the digital camera example into a clean specification model that conforms to the structure, rules and guidelines discussed in class. We start from the SpecC code developed as a result of Lab #2:

```
/home/projects/courses/fall_09/ee382v-17220/jpegencoder2.tar.gz
```

1. Insert timing checks into the testbench. Update *Stimulus* in connection with *Monitor* to print the total delay required for encoding of each single picture. Compile and simulate the model to check timing info is printed correctly (delays should be zero at this point).

2. Develop an accurate model of the actual I/O structure for the digital camera. Move the *ReadBlock* behavior outside of the *JpegEncoder*, move the waiting for the start signal *ReadBlock* and modify *ReadBlock* to independently loop over all blocks in a picture and send them over its outgoing queue after the start signal has been received. Introduce a *WriteBlock* behavior (*write.sc*) that continuously reads bytes from a queue and forwards them into an outgoing double-handshake channel. Introduce an additional level of hierarchy as a *Design* behavior (*design.sc*) that sits between *Monitor* and *Stimulus* and is a parallel composition of *ReadBlock*, *JpegEncoder* and *WriteBlock* instances communicating via *c_queue* channels:



3. Convert the *JpegEncoder* into a parallelized KPN-style model. Remove the *ReadBlock* instance and change the top-level *JpegEncoder* execution into a single `par` statement in which the three remaining child behaviors communicate via *c_typed_queue* channels. An example and tutorial for use of typed queues can be found at:

```
$SPECC/examples/sync/c_bit64_queue.sc
$SPECC/examples/sync/typed_queue.sc
```

Modify *Dct*, *Quantize* and *Huff* to work on continuous streams of input and output data over *c_int64_queue* channels. Change the sequential sub-composition inside *Dct* and *Huff* behaviors into an `fsm` that runs child behaviors sequentially in an endless loop. Introduce an additional level of hierarchy in *quantize.sc* as a behavior *Quant* that runs *Quantize* in an endlessly looping `fsm`. Replace the top-level *Quantize* instance in *JpegEncoder* with *Quant*.

4. Compile and simulate the whole design to validate its correctness.

Part 2: System-On-Chip Environment (SCE)

The goal of this part is to make you familiar with the System-On-Chip Environment (SCE) by going through the first part of the tutorial that demonstrates SCE's system specification and analysis capabilities on a GSM Vocoder design example. The tutorial instructions are available as part of the SCE installation (see below) and online at:

http://www.cecs.uci.edu/~cad/publications/tech-reports/2003/TR-03-41.tutorial.pdf

Note, however, that the tutorial is based on an older version of SCE. As such, some steps have changed and communication design steps have been expanded. A list of errata with all modified and added tutorial steps necessary for the current SCE version is available on the class website:

http://www.ece.utexas.edu/~gerstl/ee382v_f09/docs/SCE_Tutorial_Errata.pdf

SCE is installed next to the SpecC tools on the ECE LRC Linux servers. Instructions for accessing and setting up SCE and the tutorial are posted on the class website:

http://www.ece.utexas.edu/~gerstl/ee382v_f09/docs/SCE_setup.pdf

Again, once logged in (e.g. remotely via `ssh -X` and make sure to have an X11 server running locally), you need to run the provided setup script (depending on your $SHELL):

```
source /home/projects/gerstl/sce-20080601/bin/setup.{c}sh
```

Next, setup a local working directory for the tutorial demo, launch the SCE GUI and follow the steps of the tutorial:

```
mkdir demo
cd demo
setup_demo
acroread SCE_Tutorial/sce-tutorial.pdf &
```

(or point your web browser to `SCE_Tutorial/html/`)

```
acroread $SPECC/doc/SCE_Tutorial_Errata.pdf &
sce
```

Read and go through the tutorial up to and including Section 2 (we will go through additional tutorial steps in a later lab).

Part 3: Digital Camera Analysis

We are now ready to load the digital camera specification model into SCE and start the analysis and exploration process:

1. Open SCE in the digital camera directory and create a new project "digicam.sce" (`Project→New`, `Project→SaveAs…`). Adjust the simulator and compiler options as needed. Set the simulation command to

   ```
   ./%e && diff -s test.jpg goldgen.jpg
   ```

   Set the compiler verbosity level to 3 and the warning level to 2.

2. Import the *digicam.sc* specification model into SCE and add it to the project. Rename the model in the project window to *DigicamSpec*.
3. Compile and simulate the model to validate its correctness.
4. Browse the graphical hierarchy chart. Expose all levels of hierarchy and submit a printout of the chart of the complete specification model (`Window→Print…` to file *DigicamSpec.ps*).
5. Profile the model and generate the bar graph for the Computation profile of behaviors in the *JpegEncoder* part of the design. Submit a printout of the computation graph (`Window→Print…` to file *DigicamSpecProfile.ps*).
6. Submit all source, SIR, project and Postscript files. Report on the relative computational complexity of different JPEG encoder blocks. Briefly describe some possible implementation options and opportunities for optimization in design space exploration.