EECS 222: Embedded System Modeling
Winter 2020

Assignment 3

Posted: January 21, 2020
Due: January 29, 2020 at 6pm

Topic: Introduction to SystemC Language and Simulation

1. Setup:

We will use the same Linux account and the same remote servers as for the previous assignments. A recent version of the SystemC library has been installed on the servers in this directory:

```
/opt/pkg/systemc-2.3.1/
```

Since the open-source SystemC simulator consists of only a library and does not have any executable tools, there is no need to configure your search path. It comes in handy, however, to use an environment variable for the path to the System installation.

If you use the **csh** or **tcsh** shell, use this:

```
setenv SYSTEMC /opt/pkg/systemc-2.3.1
```

On the other hand, if you use the **sh** or **bash** shell, then use this:

```
export SYSTEMC=/opt/pkg/systemc-2.3.1
```

For this assignment, create a new working directory, so that you can properly submit your deliverables in the end.

```
mkdir hw3
cd hw3
```

2. Task A: Reference Example

We will use the simple FIFO example discussed in Lecture 6 as an initial reference model. The source code for this example is available with the SystemC package installed on our server. You should create a local copy of this model.

```
cp $SYSTEMC/examples/sysc/simple_fifo/simple_fifo.cpp ./
```

You can then examine the example in your editor and compile it with the regular GNU C++ compiler. Finally, to simulate it, run it as an executable in your Linux shell, as follows:

```
g++ simple_fifo.cpp -I$SYSTEMC/include \
    -L$SYSTEMC/lib-linux64 \
    -Xlinker -R -Xlinker $SYSTEMC/lib-linux64 \
    -lsystemc -o simple_fifo
./simple_fifo
```

If the output produced is not what you expect from reading the source code of the model, then read the source code again. The behavior of this example is somewhat non-deterministic, which means it allows multiple output orderings that are all correct. We will discuss such non-determinism in detail in one of the coming lectures on discrete event simulation semantics.

You may want to modify the example to examine it more closely. However, there is nothing to submit for this part of the assignment.

3. Task B: Producer-Consumer Example

As a deliverable for this assignment, write and simulate a SystemC program of a simple Producer-Consumer example. Since we have modeled this application already in SpecC, you may want to refer to your SpecC model as a reference. You will see that SpecC and SystemC are very similar in their structure, but their syntax is quite different.

As a guiding reference model, we provide a solution file of the previous assignment.

```
cp ~eecs222/public/ProdCons.sc .
```

You can think of this task as to translate the SpecC model into an equivalent SystemC model. Since much of the code can be reused, you may want to start with copying the SpecC code as a starting point for your SystemC model.

```
cp ProdCons.sc ProdCons.cpp
```

Again, your program should contain two parallel modules named **Prod** and **Cons** that communicate via a connecting channel **C**. The producer should send the message **"Apples and Oranges"** to the consumer. This communication should be character by character (a single byte at a time should be sent/received through the channel), using the same simple protocol as in the previous assignment. While sending and receiving the characters, the consumer and producer behaviors should print each sent/received character to the screen. After the entire message is communicated, both consumer and producer should terminate so that the SystemC root thread can also cleanly complete.

Your SystemC program execution should look like this:

```
sc_main starts.
Producer starts.
Producer sends 'A'.
Consumer starts.
Consumer received 'A'.
Producer sends 'p'.
Consumer received 'p'.
Producer sends 'p'.
Consumer received 'p'.
Producer sends 'l'.
Consumer received 'l'.
Producer sends 'e'.
Consumer received 'e'.
[...]
Producer sends 's'.
Consumer received 's'.
Consumer done.
Producer done.
sc_main done.
```

Hint: Use **EventName.notify(SC_ZERO_TIME)** for all event notifications. If you leave out the **SC_ZERO_TIME** argument, the notification becomes non-deterministic and may or may not reach the corresponding **wait** construct. In that case, your simulation will get stuck and terminate early. (We will discuss the reasoning behind this "strange behavior" in detail when we get to the underlying discrete event simulation semantics of SystemC.)

Name your SystemC source code file **ProdCons.cpp**, since you won't be able to submit it otherwise.

To demonstrate that your program simulates as expected, compile and run it. When running, redirect the output into a file named **ProdCons.log**. Again, use exactly this filename, otherwise you cannot submit it.

3. Submission:

For this assignment, turn in the following deliverables:

```
ProdCons.cpp
ProdCons.log
```

To submit these files, change into the parent directory of your **hw3** directory and run the **~eecs222/bin/turnin.sh** script. Again, this command will locate the current assignment deliverables and allow you to submit them, just as before.

Remember that you can use this 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 Dömer (EH3217, x4-9007, doemer@uci.edu)