Embedded System Design and Modeling

EE382V, Fall 2010

Homework #3

Communication Synthesis, SystemC

Assigned: November 4, 2010
Due: November 18, 2010

Instructions:

- Please submit your solutions via Blackboard. Submissions should include a single PDF with the writeup and a single Zip or Tar archive for any supplementary files (e.g. source files, which has to be compilable by simply running 'make' and should include a README with instructions for running each model).
- You may discuss the problems with your classmates but make sure to submit your own independent and individual solutions.
- Some questions might not have a clearly correct or wrong answer. In general, grading is based on your arguments and reasoning for arriving at a solution.

Problem 3.1: Bus Taxonomy

We introduced in class an extremely simplified taxonomy of busses, distinguishing between master/slave and node-based communication systems:

(a) Define the two different communication system types we have differentiated: master/slave and node-based. Name some characteristics of each communication type and one specific example of each communication type that has not been mentioned on the class slide.

(b) What are the system characteristics that suggest using a master/slave-based bus system? Briefly reason about the identified characteristics and give one specific example for an architecture (and maybe application).

(c) What are the system characteristics that suggest using a node-based bus system? Briefly reason about the identified characteristics and give one specific example for an architecture (and maybe application).

Problem 3.2: Communication Primitives

We discussed synchronous and asynchronous communication at the application level:

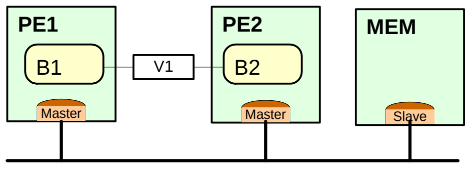
(a) Define in your own words (1-2 sentences) synchronous communication. Show and briefly describe the activity graph/message sequence chart between a sender and a receiver highlighting synchronization and data transfer. Give an example of synchronous communication used in SCE.

(b) Briefly outline an application example which benefits from synchronous communication.

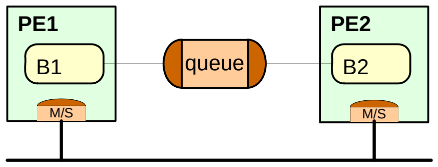
(c) Briefly outline an application example which benefits from asynchronous communication as for example implemented with the *c_queue* in SpecC.

Problem 3.3: Communication Refinement

(d) In the platform on the right side, two PEs (*PE1* and *PE2*) are masters on the bus. In addition, a shared memory is connected to the same bus via a slave interface. The variable *V1* is accessed from both *B1* (mapped to *PE1*) and *B2* (mapped to *PE2*). What are possible mapping options for *V1* and why?



(e) The PEs *PE1* and *PE2* will both be connected to the same bus as shown on the right. As one part of the communication refinement, complex channels have to be mapped to processing elements. *B1* and *B2* communicate through a queue. *B1* sends a total data volume of 1024k to *B2*. Although both behaviors transmit the same mount of data, they differ in the granularity of the data access. *B1* writes into the channel with 128byte messages, while *B2* reads individual bytes (1byte each) out of the queue. Assume that if the queue is mapped to the same PE, the local access is instantaneous and does not incur any bus traffic. To which processing element should the queue be mapped in order to limit the total amount of bus traffic? Briefly explain your reasoning.



Problem 3.4: SystemC Modeling

The SystemC environment is installed on the ECE LRC Linux servers. Instructions for accessing and setting up the tools are posted on the class website:

http://www.ece.utexas.edu/~gerstl/ee382v_f10/docs/SystemC_setup.pdf

In short, once logged in (e.g. remotely via `ssh`), you need to set the $SYSTEMC environment variable (depending on your $SHELL):

`setenv SYSTEMC /usr/local/packages/systemc-2.2.0` ([t]csh)

or

`export SYSTEMC=/usr/local/packages/systemc-2.2.0` ([ba]sh)

The SystemC installation comes with a set of examples, available under `$SYSTEMC/examples`. You can copy an example into a working directory:

```
mkdir hw3
cd hw3
cp /home/projects/gerstl/pkg/systemc-2.2.0/examples/simple_fifo/* .
ls
```

And then use the provided `Makefile` to compile and simulate the example:

```
make
./simple_fifo
```

It is recommended to inspect the sources of the example and the included `Makefile` to understand SystemC compilation and simulation process, experiment with the `Makefile` usage and start modifying the example to experiment with different features of the SystemC language, e.g. to replace the custom `fifo` channel with a corresponding `sc_fifo<char>` channel from the standard SystemC channel library

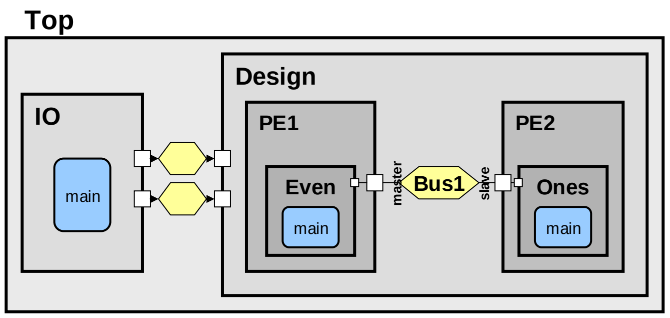
For this problem, we will take the TLM of the SpecC parity generator/encoder example developed in Homework 2, Problem 2.6 and model it in SystemC. As a reference, you can start from the SpecC code posted as part of the solutions to Homework 2 on Blackboard. Following the ideas outlined in reference [25] on the class webpage, we convert the SpecC TLM into an equivalent SystemC one:

(a) Assume again a partitioning where the *Even* process is mapped to *PE1*, the *Ones* process is mapped to *PE2*, and a *Bus1* is connecting *PE1* (master) and *PE2* (slave) using a modified double-handshake protocol. Translate the SpecC TLM of the parity checker design from Problem 2.6(b) into a corresponding SystemC TLM where a top-level `Design` module reflects this partitioning. Matching what we did conceptually in SpecC, follow a loosely timed coding style with blocking transports, active slaves and no temporal decoupling. A SystemC implementation of the *DblHndShk* bus TLM channel is provided for you under:

```
/home/projects/courses/fall_10/ee382v-16985/DblHndShkBus{.h/.cpp}
```

In a straightforward manner, the SpecC behavior hierarchy is converted into a matching hierarchy of SystemC modules where each SpecC behavior becomes a SystemC module with exactly one *main* process. Create layers of modules representing the two PEs and group the processes under these `PE1` and `PE2` modules according to their mapping. Insert a single instance of the `Bus1` *DblHndShk* protocol channel and realize all inter-PE communication to go over this bus instance.

Finally, enclose this `Design` into a typical testbench setup. Implement the top level of the example (`Top` behavior) to describe a proper structure consisting of `IO` and `Design` modules. To make things easier, you are free to convert all communication between `IO` and `Design` into SystemC `sc_fifo<`*T*`>` channels of appropriate template type *T*:



Simulate the model to validate its correctness. Turn in the source files and all input and output test files.

Extra credit: convert the model into one that follows the SystemC TLM2.0 standard, i.e. convert the interface between the PEs and the TLM bus channel into TLM2.0 sockets (for simplicity you can use the *simple* sockets that are part of the `tlm_utils` package). The SystemC TLM library is installed under:

```
/usr/local/packages/systemc-tlm-2.0.1
```

(b) Report on your experiences with SpecC and SystemC modeling. Compare and contrast the languages in terms of ease of use, feature richness, modeling expressiveness, etc. Do the languages meet the goals and requirements for SLDLs outlined early in class? How could the languages be improved?