University of Stuttgart
Germany

Institute of Computer Architecture
and Computer Engineering
Dept. of Embedded Systems

Modeling, Simulation, and Specification

Prof. Dr. Martin Radetzki

Exercise 6: *Transaction Level Modelling with OSCI TLM2 (LT)*

The source files for this exercise are available in the package exercise6.zip on the course webpage. The exercise was designed using OSCI TLM-2.0.1, which can be downloaded from www.systemc.org.

1. OSCI TLM2 base sockets and simple sockets

a) In the directory **ex6_1** you will find an incomplete design using the base sockets. Try to compile them. What error message do you see? Please modify the code such that you can compile and run it successfully. Trace the execution of the program using some debugger (e.g. GNU Debugger gdb – see using_gdb.pdf) to see which function is called when `b_transport()` function is called from the initiator.

b) In the directory **ex6_1** you can also find units using simple sockets (*_simple.{h,cpp}). In main.cpp, uncomment the lines which instantiate units with simple sockets and comment lines which instantiate units with base sockets. Try to compile and run the design successfully.

c) Compare the units with base and simple sockets. List all the differences you can find.

d) Either in the units with base or simple sockets, change the value of the time argument to nonzero when `b_transport()` is called. How does the program behave differently?

e) In `dst_unit` and `dst_unit_simple`, the time argument is set to zero again. Why is this necessary?

2. TLM programming style – untimed and loosely timed

In this part a simple signal processing system shown below will be modeled with different TLM abstraction levels. The system is composed of a source (SRC), a transformation unit (T), and a destination (DST). The source generates and sends the data to transformation unit, which processes and sends the data further to the destination.

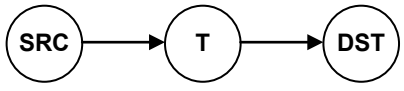

Figure 1: simple signal processing system

a) In the directory **ex6_2/functional** you find the functional modeling of the above system. Compile and run the system, you should see a Penrose triangle. Try to figure out the functionality of the transformation unit.

b) In the directory **ex6_2/utf** you find an incomplete TLM untimed functional implementation of the above system, using blocking transport interface with function b_transport(). Complete the design. Compile and run the system. You should get the same result as in step a). What is the difference of this system compared with the one in step a)?

c) Copy **T** (trans.{h,cpp}) and **main.cpp** of your solution in step b) to the directory **ex6_2/lt**. Use them with the given untimed functional memory model MEM and the incomplete SRC, to build a system shown below. The arrows in Figure 1 show the function call relationships.

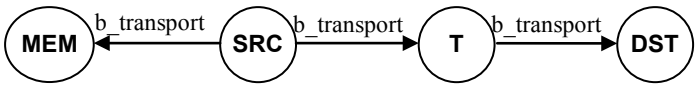

Figure 2: signal processing system with separate memory

Compared to the previous system, input data is no longer stored directly in SRC. Instead SRC needs to fetch one data word (32bit) after the other from a memory block MEM, which is 8192 words large. Address 0x0000 contains the number of words to be processed, followed by the data words to be processed. (Hint: you may need another initiator socket in SRC to communicate with MEM.)

i. Compile and run the system. What is the simulated time at the end of the simulation? (Hint: you need to print out the value of `sc_time_stamp()` in **SRC** at the end of simulation)
ii. Now modify the code such that **MEM** has a read latency of **10 ns**, a write latency of **20 ns**, and **T** has a processing latency of **8 ns**. Recompile and run the system. What is the simulated time at the end of simulation?
iii. Modify the source code such that, there is an additional **1 ns** communication latency between **T** and **DST**. Recompile and run the system. What is the simulated time at the end of simulation? Draw the message sequence chart of this scenario for one iteration with indicated simulation time.