Exercise 6: SystemC and Virtual Prototyping

Exercise on TLM Loosely Timed (LT)

Matthias Jung
WS 2017/2018

The source code to start this execise is available here:
`https://github.com/TUK-SCVP/SCVP.Exercise6`

Task 1

Single Initiator and Target

Figure 1 shows an example for a very simple TLM scenario. The code on github for this exercise already provides a simple initiator called `processor`, which fakes the memory access behavior of an simple 32-bit microcontroller by replaying a tracefile. At the first step you should study the code of this initiator. Do you recognize the *Regular Expressions* for parsing the input trace? In c++ we have to escape the \ by a \, therefore we see for example patterns like \\d+ instead of \d+.

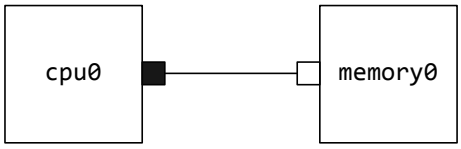

Fig. 1: Initiator and Target

As second part of this task you will implement a simple TLM target called `memory` in a *Loosely Timed* (LT) coding style in the file `memory.h`. The memory size should be provided as a template parameter, and the standard value should be 1024 bytes. If an address higher or equal 1024 is accessed, a proper TLM error handling should be done (see lecture) and the simulation should be stopped with `SC_REPORT_FATAL` by the initiator. Hint: Have a look at the artifacts repository

`https://github.com/TUK-SCVP/SCVP.artifacts/tree/master/tlm_lt_initiator_target`

to see an example for an LT based target. The functions, `nb_transport_fw`, `get_direct_mem_ptr` and `transport_dbg` must be implemented as stubs, i.e.

dummy functions as in the artifacts example. Note that in the target `wait()` should not be called. The time consumed by the memory access (20 ns) should be added to the *delay* variable of the `b_transport` function call.

In order to test your first TLM memory you have to instanciate an object of the class processor (`cpu0`) and an object of the class memory (`memory0`) like this:

```
processor cpu0("cpu0","stimuli1.txt", sc_time(1,SC_NS));
memory<1024> memory0("memory0");
```

The second constructor parameter for the `cpu0` is the input trace to be read, and the third parameter is the inverse of the frequency, in our case 1 GHz. Then bind the initiator socket of the `cpu0` with the target socket of the `memory0` in the `main.cpp` file as shown in the lecture.

The *stimuli1.txt* writes in the beginning some data to the memory and reads again the same data some cycles later. If you implemented your memory in the right way, you will see that the data should be the same as it was written to the memory before.

Task 2

Multiple LT Initiators and Targets

Figure 3 shows an example for a more sopisticated scenario. In this case we have two processors (`cpu0` and `cpu1`) which are connected over an interconnect (`bus0`) to two different memories. The bus should forward the transactions to the differnt memories according to the memory map shown in 3 (routing).

Hint: Have again a look at the artifacts repository

`https://github.com/TUK-SCVP/SCVP.artifacts/tree/master/tlm_lt_initiator_interconnect_target`

to see an example for an LT based interconnect with routing.

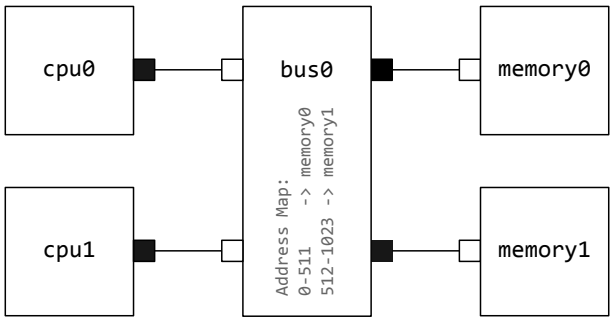

Fig. 2: Initiators Interconnect and Target

Task 3

Temporal Decoupling

Modify the Initiator of Task2 in such a way that you use a quantum keeper for temporal decoupling

Hint: Have again a look at the artifacts repository

`https://github.com/TUK-SCVP/SCVP.artifacts/tree/master/tlm_quantum_keeper`

to see an example for temporal decoupling. In order to see the effects of a quantum we generate two random trace files with the following commands:

```
perl generate.pl 1000000 > random1.txt
perl generate.pl 1000000 > random2.txt
```

Use these two files as input stimuli for the processors. Furthermore, remove all `cout` statements from your code for a more accurate comparison.

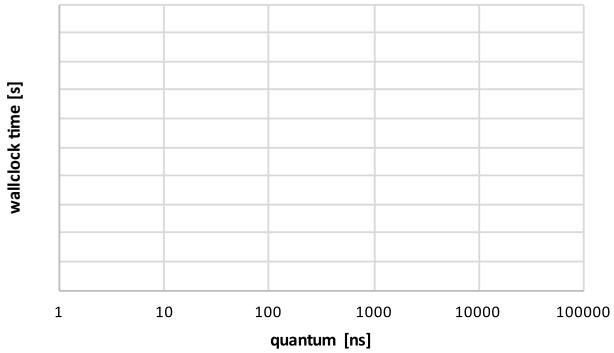

Fig. 3: Quantum vs. Wallclock Time

Execute the code with quantums ranging from 1 ns-100000 ns. For measuring the wallclock time, you can use the time command

```
time ./tlm_lt
```

Put the values into the graph template. You should observe differences in wallclock time around 10%. However, note that the simulation gain comes espacially if you have more than two initiators.