System-Level Design

2.01.334, Summer 2024

Exercise 6
SystemC TLM I

Assigned: June 17, 2024
Due: June 17+24, 2024

Instructions for SystemC Exercises

1. Download the exercise code archive from Stud.IP. It contains code templates for this exercise.
2. Extract the archive to a directory on your machine and complete the exercises in place.
3. As before, you are allowed to work in groups with up to three members. Put the full names of all group members on the cover page of you submitted solution. Add the group members to file `info.txt` in the exercise code directory, too.
4. Please submit your solution via mail to `<sld-dozenten@v.offis.de>`. Submissions should include a single typewritten PDF file with the writeup and a single Zip or Tar archive of the exercise code directory. Clean all code directories with `make clean` before archiving.

Task 1

a) Implement the TLM memory model of module `ram` in `ram.h` and `ram.cpp`.

After adding the missing parts of this memory module, connect up the provided `master` module directly with a single memory module (without a bus) in file `main.cpp` as shown in the lecture and run the simulation by running `make sim-one`.

Note: The address check in module `ram` should be done in function `ram::is_invalid_address`, see provided skeleton files.

Task 2

Extend the model from Task 1 by a simple bus model (see `bus.h`, `bus.cpp`) to connect two masters and two slaves according to Figure 1.

a) Implement a simple *programmers view* model of a bus, using `multi_passthrough_sockets`. As a starting point, you can use the skeletons available in files `bus.h` and `bus.cpp`. Instantiate two masters and two slaves in addition to your bus module in `sc_main()` and simulate your design using `make sim-two`.

Note: The handling of the address map(s) is done in a separate class `address_map` that already provides a `decode()` function and reads the file `mem_map.txt` during elaboration.

b) Refine your bus model from (a) with a mutex-based arbitration scheme to a cycle-approximate model `bus_cx`. Compare the simulation results to the previos scenario.

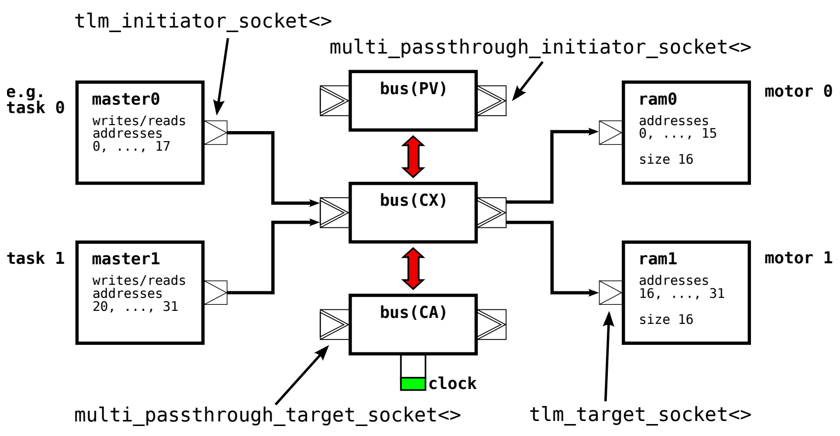
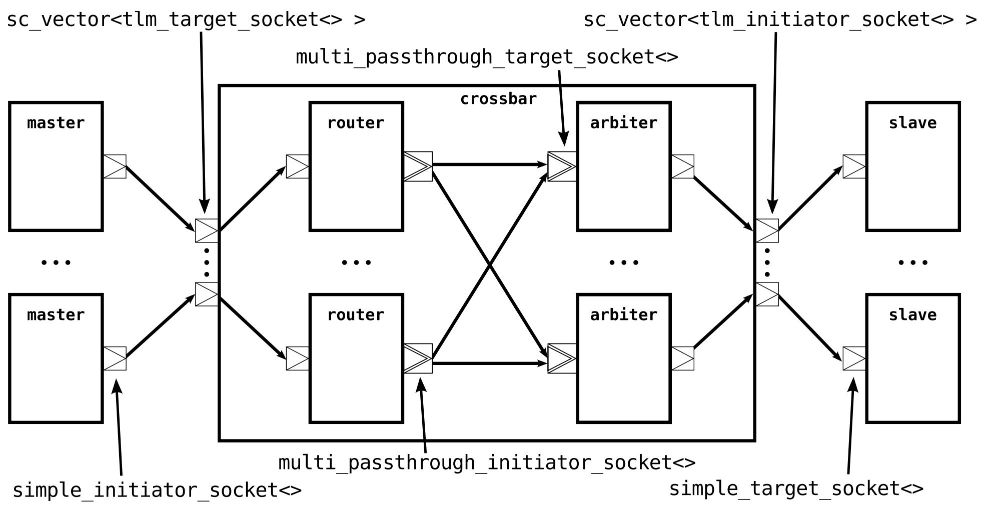


Figure 1: Simple bus system with two masters and two slaves and varying bus models.

c) In an additional refinement step, implement the *cycle accurate* bus model `bus_ca` as shown in the lecture and simulate your design again.

Note: Remember to adjust the `#include` guards if you use an existing file as template for a new module!

Task 3
In the lecture, an alternative communication topology based on a *crossbar* has been discussed (see Figure 2). Implement a *cycle approximate* model of such a crossbar, using an `sc_vector` of `tlm_sockets`.



sc_vector<tlm_target_socket<> >
sc_vector<tlm_initiator_socket<> >
multi_passthrough_target_socket<>
crossbar
master
router
arbiter
slave
...
master
router
arbiter
slave
multi_passthrough_initiator_socket<>
simple_initiator_socket<>
simple_target_socket<>

Figure 2: Crossbar interconnect system, enabling parallel accesses to different slaves.

a) Implement a `router` component, based on the *programmers view* bus model. The router uses a `tlm_target_socket` to receive transactions and a `multi_passthrough_initiator_socket` to transmit the transactions to the addressed arbiters.

b) Build an `arbiter` module, based on the *architecture view* `bus_cx` module, using a `multi_passthrough_target_socket` for receiving transactions from the routers and a `tlm_initiator_` `socket` to transmit the transaction to the corresponding ram module. Arbitration can be modelled via an `sc_mutex` and an optional delay should be given to the constructor.

c) In the file `crossbar.h`, you find a skeleton for the surrounding `crossbar` module, encapsulating N routers (for the masters) and M arbiters (for the slaves). It is modelled as a C++ template class. Fill up the missing parts (like instantiation of the sub-modules, and binding of the initiator and target sockets).

d) Instantiate and bind `crossbar<2,2>` as your interconnect component in `sc_main()`. Run the simulation and compare the results to `bus_cx` (`make sim-three`). What do you observe?

Administrative

Advisors: Kim Grüttner <kim.gruettner@dlr.de>
Jörg Walter <joerg.walter@offis.de>
Henning Schlender <henning.schlender@dlr.de>
Sven Mehlhop <sven.mehlhop@offis.de>