DLR

OFFIS
INSTITUT FÜR INFORMATIK

CARL VON OSSIETZKY universität OLDENBURG
FAKULTÄT II
INFORMATIK, WIRTSCHAFTS- UND RECHTSWISSENSCHAFTEN

System-Level Design

2.01.334, Summer 2024

Exercise 4

SystemC I

Assigned: May 27, 2024
Due: June 3, 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: Create and simulate a SystemC model of a synchronous adder.
Beside its data inputs and outputs, a synchronous adder has a clock and a reset port. In contrast to a combinatorial adder, a synchronous adder calculates the sum of the two inputs only at a rising clock edge. This means that any change on the inputs between two rising clock edges does not change the value of the output port.

a) In the directory `adder`, you can find the files `adder.cpp` and `adder.h`. Extend the already existing code in these files so that it implements the described behaviour of the synchronous adder. Do not forget to correctly initialise the adder during a reset.

b) Compile and create the executable simulation by calling `make` in the respective directory. You can then start the simulation by calling `make sim` or `./adder.x`.

c) Analyse the created waveform traces in `trace.vcd` using *GTKWave*. You can start the program by entering `gtkwave <trace-file-name>`. What do you notice when looking at the traces?

d) What changes if you remove the call to `sc_stop()` from the testbench stimuli process in line `testbench.cpp:73`? How can this behaviour be explained?

Task 2: Create and simulate an accumulator using the synchronous adder from Task 1.
At each rising clock edge the accumulator adds the value of its input to an internal register and writes the result to the data output. In case of a reset the internal value is reset to 0.

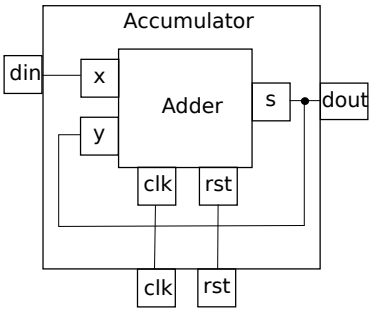

Figure 1: Structure of accumulator

a) Change to directory `accumulator`. Implement the structure shown in Fig. 1 by completing the code given in `accumulator.h` and `accumulator.cpp`.

b) Compile and create the executable simulation by calling `make` in the respective directory. You can then start the simulation by calling `make sim` or by executing `./accumulator.x`.

c) Extend the function `sc_main()` given in in `main.cpp` so that waveform traces of the inputs and outputs of the accumulator are generated. After that, compile and start the simulation again. Examine the created trace file using *GTKWave*.

Task 3: Create an executable functional model of a *line-follower controller* being part of an autonomous car.

The model consists of two SystemC modules being connected via FIFOs. The module `car` models the autonomous car including its sensors and its movement within a virtual environment simulated using the Irrlicht3D engine (open-source 3D engine). The module `car_controller` represents the controller calculating speed and rotation of the car based on the values of the car sensors. The communication between the two modules is done via `sc_fifo`-Channels. The modules alternately exchange sensor data and control data via these channels, i.e. each module is blocked until it receives data from the other module, then calculates and sends back the answer. Therefore, to avoid an initial deadlock it has to make sure that `car` initially sends sensor data to `car_controller`. The structure of the line-follower model is shown in Fig. 2.

Note: The data types for sensor and control data are user-defined classes defined and implemented in files `data_types.h` and `data_types.cpp` which also contain functions for `sc_trace`, the comparison operator (==) and the stream output operator (<<).

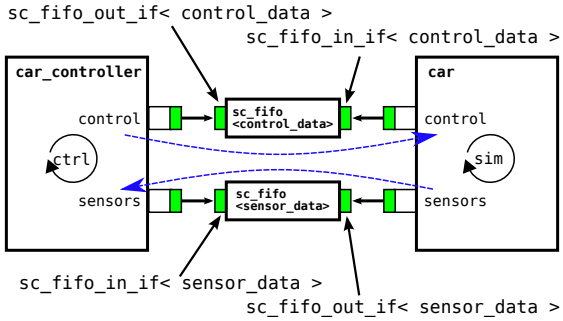

Figure 2: Structure of the `line-follower` model

a) Change to directory `line-follower`. Complete the existing source code to enable correct compilation and simulation of the model.

b) If you look at the output of the simulation you will see that no simulation time is passing. However, within the Irrlicht engine there is indeed some passing simulation time which is used internally to calculate the new position of the car. The current value of the engine timer can be read within the model `car` by using the statement `unsigned t = e.get_timer()`. Extend the process in `car` so that the same time is consumed by the SystemC simulation. (You can choose any reasonable time unit for mapping the returned timer value of the Irrlicht engine to SystemC time.)

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>