System-Level Design

2.01.334, Summer 2024

Exercise 5
SystemC II

Assigned: June 10, 2024
Due: June 10, 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

The SystemC model of the line-follower from last exercise is based on floating point data types for the sensor and control data. As part of a refinement `car_controller` has now been modified to use the final implementation's bit-true integer data types.

To enable further use of the `car` model and the Irrlicht test environment as a testbench for `car_controller`, the floating point values now have to be converted to integer types and vice versa. In a first step, this should be done using specialised converter channels, whose interface is shown in Figure 1.

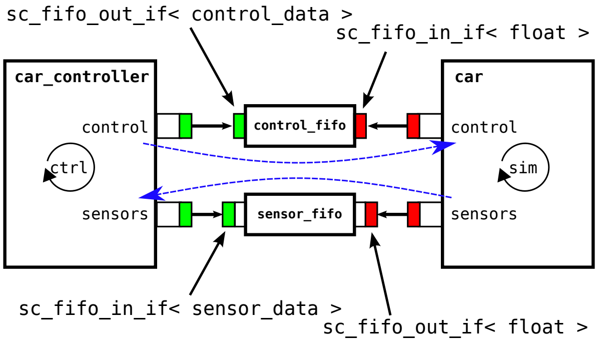

Figure 1: Extended `line-follower` model

a) Modify the definition of the data types in data_types.h so that control data is in the range of $\{z \in \mathbb{Z} : -128 \leq z < 128\}$ and sensor data in the range of $\{n \in \mathbb{N} : 0 \leq n < 256\}$. Choose appropriate SystemC data types.

b) Implement the channel sensor_fifo as a regular SystemC module exporting the interfaces of two internal FIFO channels as in and out and with an internal clocked process which reads one speed value and twelve sensor values from the internal input FIFO one after another, puts these into a sensor_data object and finally writes the object into the internal output FIFO.

What should be the size of the internal FIFOs? Are both FIFOs required? How could it be solved differently?

c) Complete the implementation of the primitive channel control_fifo within the files control_fifo.h and control_fifo.cpp.

This channel shall convert each field of any control data object written by car_controller to float using the to_float() function and afterwards make it available via the FIFO input interface. How can the synchronization be realised cleverly?

Task 2
During the lecture the concept of an *Adapter* for interface conversion has been introduced. More specifically a *Transactor* as a specific type of adapter translates higher level to lower level interfaces making it a suitable tool for selective communication refinement within system models. Subdirectory uart contains an implementation of UART components which could be used in a real implementation of the line-follower for communication between the sensors and the controller of the car. It also contains a suitable testbench; use make sim to execute it.

a) Complete the prepared transactor fifo_tx, which writes incoming floating point values from the FIFO interface to an sc_signal<bool> output port following the serial protocol.

The correct time delay of the transmission of a single bit is stored in the sc_time constant delay. The float value can be converted to sc_bv<8> using the function to_byte(float).

Take care, that the initial state of the serial data lines is initialized correctly. For this you can use the member function start_of_simulation() which is predefined for each SystemC module and called by the simulation kernel for each instantiated module at the start of the simulation. It can be overridden by user-defined modules individually.

b) Extend the prepared adapter fifo_rx_unit to a working transactor between the sc_fifo_in_if<sc_bv<8>> interface and the signal interface of the receiving UART component.

Note: An export of a module can also be bound to the module itself if the module is derived from the export's interface. For this use the self-reference for the binding of the export within the module's constructor.[1].

c) Modify the channel sensor_fifo from Task 1 so that it uses both newly-created transactors internally.

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>

[1] In C++, a self-reference to an object can be obtained by using (*this) in any member function.

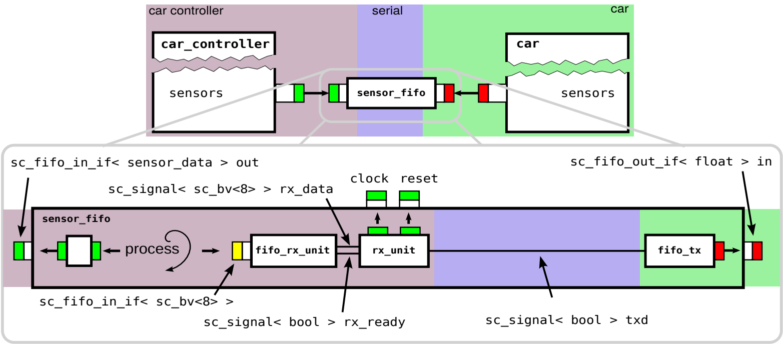

Figure 2: Overview of hierarchical adapter implementation (signals abstracted by lines)