System Level Languages
IL2452

Lab 1

Implementing Systems Using SystemC

Name: ____

Personal Number: ____

Approval:

Date:_____________________________

Sign:_____________________________

1. General information

Each Lab event has been thought of as consisting of 3 sections:

- The first section where students have to accomplish a set of *preparation tasks*. Their main purpose is to put the students in the condition of being able to solve the lab tasks.
- The second section where students will solve the actual lab tasks.
- The third section will be a discussion with the lab assistant about both the preparation and the lab tasks. To get a Pass, students will have to justify their answers.

2. Introduction to Lab1

Prerequisites: before starting working on Lab 1, students should have gone through the material shown during the lectures.
Goal: the main goal of Lab 1 is to allow students to get familiar with SystemC, by experimenting on its basic features.
Skills acquired: at the end of Lab 1, we expect that students will be able to use SystemC to implement simple architectures.

3. Preparation tasks (to be done before the Lab tasks!)

Task 3.1

1) What are modules and what are they for?
2) How many ways do you know to declare a module? Explain them.
3) What are ports? What are they used for?
4) What is the difference between *positional-based* and *name-based* port binding? What are advantages and disadvantages?
5) A simple way to connect modules is by using *sc_signal<T>*. What is *sc_signal<T>*?
6) Describe how the SystemC simulation kernel works, focusing in particular on:
    - elaboration/simulation phase.
    - how concurrency is implemented.
    - evaluate-update concept.

Task 3.2

1) What is a channel? What is an interface and how are they related to each other?
2) How do channels, interfaces and ports interact to provide communication? Illustrate with a figure.
3) What are virtual methods and what are they used for?
4) What is the difference between primitive and hierarchical channels? Explain how to choose when to use one or the other.
5) What is the difference between a module and a hierarchical channel?
6) Where are a primitive and a hierarchical channel derived from?
7) What is a sensitivity list? Explain the difference between static and dynamic sensitivity.
8) What are events and what is their purpose?

4. Lab tasks

Task 4.1

Description: in this task the students will have to test the behaviour of a simple 2-port OR gate. In order to do that, they will need to use a stimuli generator (driver) to create an input for the OR gate, and they will need a monitor to collect the results of the DUT, which is an OR gate in this task (see Fig. 4.1).

The stimuli generator, the OR gate and the monitor are already provided as 3 independent modules. Each module is defined in one header file (*.h) and its methods are implemented in one other file (*.cpp). The main file, containing the instantiation of the modules, their binding and the simulation setup is also provided.

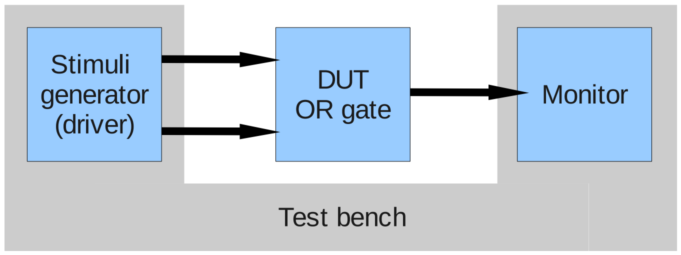


Fig. 4.1

To be done by the student:

1) Get the code for this task from Appendix (You will not be able to just do *copy & paste*, you need to type it by hand. By doing so, you will get a better understanding of code writing). Note that the code in the Appendix is reported sequentially in a unique block. However, this code should be split in different files, which are clearly named at the beginning of each new code section.
2) Run the simulation and check the results.
3) Do the results represent an OR function? Write them in table.
4) Ports binding is done by using *name binding*. Change them to *position-binding*, rerun the program and verify that there is no difference in behaviour.
5) Change modules instantiation, use pointers.
6) The stimuli generator's processes are implemented using a Thread Process. Try to change them to Method Processes and comment the result.

<u>Task 4.2</u>

Description: this task has as a main object the exploration of different solutions to implement system-level communication. The task is based on the “source – sink” principle, where source and sink are thought of as 2 independent modules. The source module produces a lower-case string containing the Latin alphabet and sends it over to the sink module, which converts it to an upper-case string and prints it on the screen. The communication between source and sink occurs over a hierarchical channel. In Fig. 4.2 we report a block diagram. For this task, only partial code is provided and students will have to fill in the missing parts in order to make the system work.

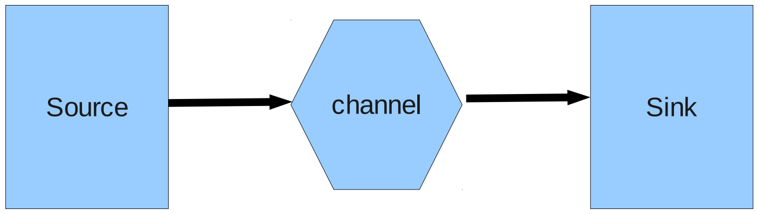


Fig. 4.2

In detail, the code provided is as follows:

- channel.h: all code is missing.
- channel.cpp: all code is missing.
- source.h: partial code is provided.
- source.cpp: all the code is provided.
- sink.h: partial code is provided.
- sink.cpp: all the code is provided.
- main.cpp: all the code is missing.

To be done by the student:

1. Write the code for *channel.h*. This file must contain 3 parts: declaration of an interface for writing to the channel, declaration of an interface to read from the channel, declaration of the channel itself. We remind the student that they have to implement a hierarchical channel.
2. Write the code for *channel.cpp*. This file contains the implementation of the channel's methods declared in *channel.h*.
3. Find what is missing in *source.h* and *sink.h* and complete them.
4. Write the file main.cpp. This file must contain the instantiation of the objects, their binding, code to start simulation. You can assume that the simulation runs indefinitely.
5. Try to compile and run the simulation. Does it work properly?
6. Modify the code to use the primitive channel *sc_fifo* for communication.

Task 4.3

Description: in this task students will have to complete the implementation of a simple bus-based architecture, according to the block diagram shown in Fig. 4.3.

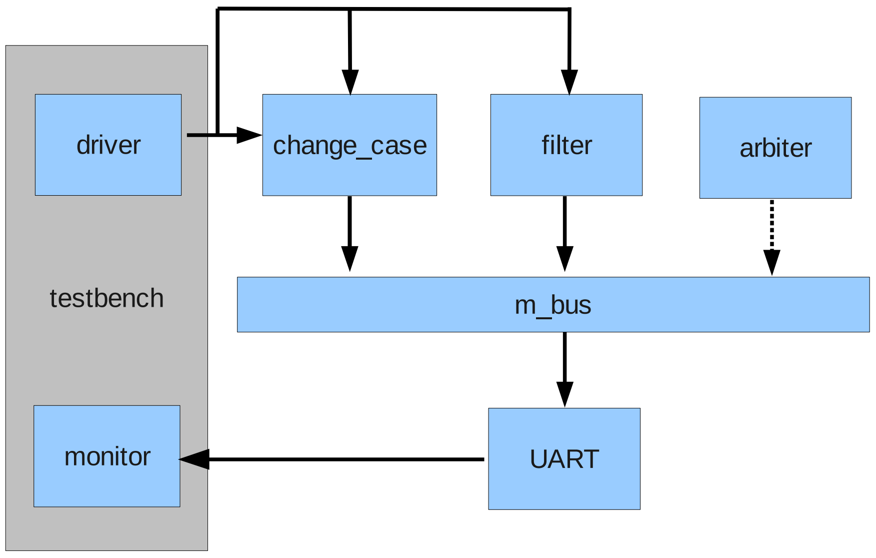


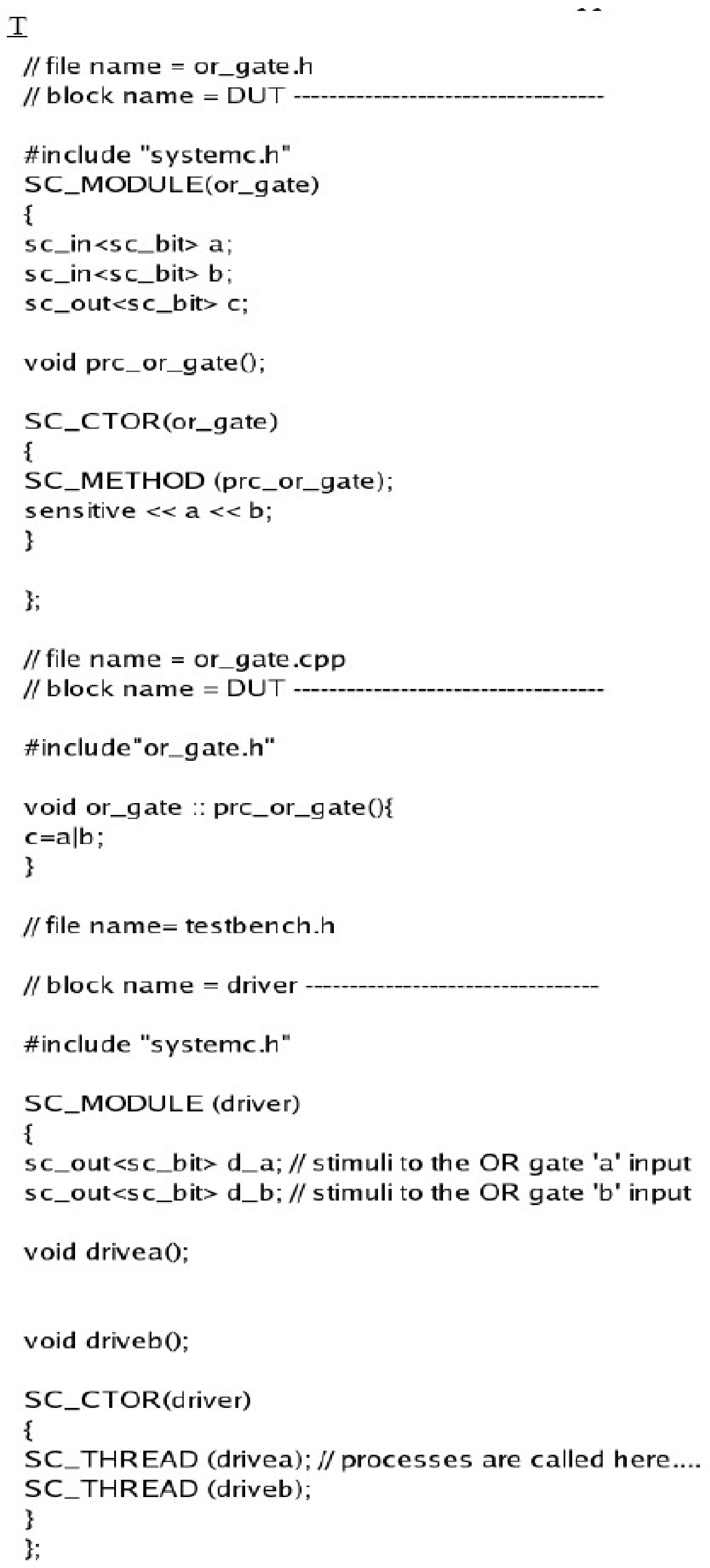
Fig. 4.3

M1 (change_case) is a module that receives as an input a string of lower-case characters, converts it to upper-case and then sends it to the output. M2 (filter) is another module that receives the same string of lower-case characters, changes to '#' all the characters not belonging to the word “systemc”, and then sends the new characters to the output. M3 (arbiter) is a third module that represents the functionality of a simple arbiter. It allocates a different time slot for M1 or M2, so that they are both granted access to the bus at different times. M4 (m_bus) is a module that represents the bus functionality. It is implemented as a simple 2-way multiplexer. Its inputs are the outputs of M1 and M2. M5 (UART) is a module representing UART input/output functionality. It simply reads the data from the bus and sends it to the output.
The SoC behaviour is tested using a simple testbench, which is composed of two parts: T1 (driver) is a module that generates a string containing the English alphabet for an infinite number of times. That is the input to M1 and M2. T2 (monitor) is a module that displays the characters read by the UART.
We provide the students with the code for M1, M2, M5, T1, T2.

To be done by the students:

1. implement M3 (arbiter). As mentioned before, the arbitration policy is timed based, where M1 and M2 are allocated the same amount of time. You are free to choose the width of your time slot. Be sure to have a reference time signal “clock signal” as an input of your arbiter block.
2. write the main file. In this file you will instantiate the modules and bind them according to the block diagram. Choose a reasonable simulation length.
3. Run the simulation and verify that the results are as expected.
4. Try to remove the wait() statement in the driver.cpp file. Run the simulation and justify the new behaviour.

Appendix



```
// file name = or_gate.h
// block name = DUT -----------------------------------

#include "systemc.h"
SC_MODULE(or_gate)
{
sc_in<sc_bit> a;
sc_in<sc_bit> b;
sc_out<sc_bit> c;

void prc_or_gate();

SC_CTOR(or_gate)
{
SC_METHOD (prc_or_gate);
sensitive << a << b;
}

};

// file name = or_gate.cpp
// block name = DUT -----------------------------------

#include"or_gate.h"

void or_gate :: prc_or_gate(){
c=a|b;
}

// file name= testbench.h

// block name = driver ---------------------------------

#include "systemc.h"

SC_MODULE (driver)
{
sc_out<sc_bit> d_a; // stimuli to the OR gate 'a' input
sc_out<sc_bit> d_b; // stimuli to the OR gate 'b' input

void drivea();


void driveb();

SC_CTOR(driver)
{
SC_THREAD (drivea); // processes are called here....
SC_THREAD (driveb);
}
};
```

```
// block name = monitor --------------------------------


SC_MODULE (monitor)
{
sc_in<sc_bit> m_a, m_b;
sc_in<sc_bit> m_c; // both the input and output of the OR gate are to be monitored

void prc_monitor();

SC_CTOR(monitor)
{
SC_METHOD(prc_monitor);
sensitive << m_a << m_b << m_c; // whenever the input/output of the OR gate changes prc_monitor triggers
}
};

// file name = testbench.cpp

// block name = driver --------------------------------

#include"testbench.h"

// these are the two
// processes to generate stimuli for OR gate
void driver :: drivea(){
 d_a.write((sc_bit)false); // b a = 0 0
 wait(5,SC_NS);
 d_a.write((sc_bit)true); // b a = 0 1
 wait(5,SC_NS);
 d_a.write((sc_bit)0); // b a = 1 0 , false=0
 wait(5,SC_NS);
 d_a.write((sc_bit)1); // b a = 1 1 , true=1
 wait(5,SC_NS);
}


void driver :: driveb(){
 d_b.write((sc_bit)0);
 wait(10,SC_NS);
 d_b.write((sc_bit)1);
 wait(5,SC_NS);
}
```

```
// block name = monitor ---------------------------------


void monitor :: prc_monitor(){

 cout<< "AT "<<sc_simulation_time()<<" input is : ";
 cout<<m_a<<" , "<<m_b<<" output is : "<<m_c<<endl;
}

// this is the main program which is used to instantiate all modules and to bind them
// this also starts the simulation

#include"testbench.h"
#include"or_gate.h"
#include"systemc.h"
int sc_main(int argc , char *argv[])
{
sc_signal<sc_bit> t_a,t_b,t_c; // signals used to connect all the modules

or_gate *g1;
g1=new or_gate ("g1");
driver *d1;
d1=new driver("d1");
monitor *m1;
m1=new monitor("m1");

g1->a(t_a);//...............
g1->b(t_b);//   name binding
g1->c(t_c);//................

d1->d_a(t_a);
d1->d_b(t_b);

m1->m_a(t_a);
m1->m_b(t_b);
m1->m_c(t_c);

sc_start(100,SC_NS); // starts the simulation and run for 100 ns

return 0;// return on success
}
```