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 3

SpecC Language, Refinement, SCE

Assigned: April 29, 2024
Due: May 6+27, 2024

General Instructions

1. Please submit your solutions via mail to your advisors (mail addresses can be found at the end of this document). Submissions should include a single typewritten PDF file with the writeup and a single Zip or Tar archive for any supplementary files (e.g. source files, which has to be compilable by simply running `make` and should include a README file with instructions for running each model).
2. You may discuss the problems with your classmates but make sure to submit your own independent and individual solutions. You are allowed to work in groups with up to three members. In this case, please put the full names of all three group members on the cover page of you submitted solutions.
3. Some questions might not have a clearly correct or wrong answer. In general, try to write down your arguments and reasoning how you have arrived at your solutions.

Before you start:
For the SpecC exercises you need to access the SpecC compiler (`scc`), which is installed on an virtual machine image.
Please follow the following steps:

1. Download VirtualBox `https://www.virtualbox.org/` and install it on your computer.
2. Download the VM appliance file `wendelin.ova` from `https://drive.google.com/file/d/1YBykgLiysJvDx4rOCk11BL6n_jHZ2SPZ/view?usp=sharing`
3. Import the VM appliance file `wendelin.ova` into VirtualBox (by opening the file)
4. Start the virtual machine `wendelin`
5. Log in using
   a) Login: `sld`
   b) Password: `sld_lecture`
6. You can then open a terminal window through the menu bar at the top of the screen: `Applications -> System Tools -> Terminal`
7. If you want to change the screen resolution of the virtual machine go in the menu bar to `System -> Preferences -> Display`

Task 1: SpecC Language and Modeling
The SpecC installation includes a comprehensive set of examples showing the features and use of the language. Examples are found in `$SPECC/examples/simple/`. You can copy them into a working directory:

```
mkdir work
cd work
cp $SPECC/examples/simple/* .
```

And then use the provided `Makefile` to compile and simulate all examples:

```
make all
make test
```

It is recommended to inspect the sources of all examples (e.g. using the installed text editors `vi`, `emacs` or `gedit`) and the included `Makefile` to understand the use of `scc` for the compilation and simulation process, and to experiment with the `scc` command-line usage and with the various `sir_xxx` tools. The directory `$SPECC/examples/parity/spec` contains an example of a parity generator written in SpecC. For this assignment, copy the example to a local working directory:

```
cd /home/$USER/work
mkdir parity
cd parity
cp $SPECC/examples/parity/spec/* .
```

You can compile and run the example using:

```
make
./tb
```

a) Draw the SpecC diagram (graphical notation/representation) of the system and briefly describe its functionality.

> **Hint:** You can view and edit all design files by entering
>
> `gedit *.sc &`
>
> in your local `parity` directory.

b) Modify the example to separate computation from communication. Create a new channel that encapsulates basic communication primitives and replace all shared variables and events between `Ones` and `Even` to exclusively use one or more instances of your custom channel. You can look at the example on slides 18-20 of "Lecture 05 - Specification & Refinement (Extended)" for guidance. Simulate the modified code to verify its correctness.

c) Now replace your custom channel with one or more `c_queue` instances out of the SpecC standard channel library. Again, simulate the code to verify correctness. Does it make any difference whether you use a `c_queue` or a `c_double_handshake` channel? Why or why not?

d) Similarly upgrade the communication between the main design and the testbench (`IO`) to use proper communication channels (pick what you feel is appropriate). In the process, modify the parity checker to separate out all communication with the testbench into additional `Start` and `Done` behaviors that execute before and after the `Ones/Even` combination, respectively. Make sure to create a clean SpecC hierarchy that does not mix

different types of behavioral compositions in one parent behavior. Again, simulate and make sure everything works correctly. Does your design exhibit any actual concurrency (in the sense of code that can run truly in parallel)? How could the model be changed to expose additional parallelism (just sketch the graphical diagram; specifically think about situations in which a continuous stream of data items need to be run through the parity checker)?

Hint: Please put the solution of each task b), c) and d) into its own sub-directory. For sending your solution, please prepare a tarball of the whole parity directory and copy it to your ARBI home directory using your ARBI `arbi-login` name:

```
tar -cvzf /home/$USER/work.tar.gz /home/$USER/work
scp /home/$USER/work.tar.gz <<login>>@taifun.informatik.uni-oldenburg.de:.
```

From the ARBI home directory you can send the `work.tar.gz` to us via mail.

Task 2: Bus Taxonomy
We introduced in class an extremely simplified taxonomy of buses, distinguishing between master/slave and node-based communication systems:

a) Define the two different communication system types we have differentiated: master/slave and node-based. Name some characteristics of each communication type and one specific example of each communication type that has not been mentioned on the class slide.

b) What are the system characteristics that suggest using a master/slave-based bus system? Briefly reason about the identified characteristics and give one specific example for an architecture (and maybe application).

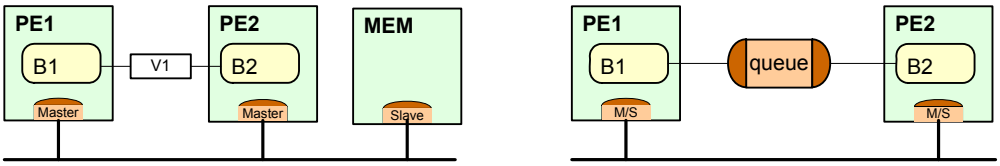
c) What are the system characteristics that suggest using a node-based bus system? Briefly reason about the identified characteristics and give one specific example for an architecture (and maybe application).

Task 3: Communication Semantics
We discussed synchronous and asynchronous communication primitives at the application level:

a) Define in your own words (1-2 sentences) synchronous communication, e.g. by showing and briefly describing the activity graph/message sequence chart (communicating state machines) between a sender and a receiver highlighting synchronization and data transfer. Briefly outline an application example which benefits from synchronous communication as for example implemented with the `c_double_handshake` channel in SpecC.

b) Define in your own words (1-2 sentences) asynchronous communication, e.g. by showing and briefly describing the activity graph/message sequence chart (communicating state machines) between a sender and a receiver highlighting synchronization and data transfer. Briefly outline an application example which benefits from asynchronous communication as for example implemented with the `c_queue` channel in SpecC.

Task 4: Communication Refinement
a) In the platform in Figure 1(a), two PEs ($PE1$ and $PE2$) are masters on the bus. In addition, a shared memory is connected to the same bus via a slave interface. The variable $V1$ is accessed from both $B1$ (mapped to $PE1$) and $B2$ (mapped to $PE2$). What are possible mapping options for $V1$ and why?



(a) (b)

Figure 1: Architecture examples

b) The PEs $PE1$ and $PE2$ will both be connected to the same bus as shown in Figure 1(b). As one part of the communication refinement, complex channels have to be mapped to processing elements. $B1$ and $B2$ communicate through a queue. $B1$ sends a total data volume of 1024k to $B2$. Although both behaviors transmit the same mount of data, they differ in the granularity of the data access. $B1$ writes into the channel with 128byte messages, while $B2$ reads individual bytes (1 byte each) out of the queue. Assume that if the queue is mapped to the same PE, the local access is instantaneous and does not incur any bus traffic. To which processing element should the queue be mapped in order to limit the total amount of bus traffic? Briefly explain your reasoning.

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>