Hands-on Lab
Functional Verification
SEI 3A – CSI 3A – 2016

Objectives

In this lab, you will be in the position of a verification engineer who must check the correctness of a design. You will be given the design specification, and the RTL code. Your job is to ensure that the design is correct or else to report bugs to the designer.

At the end of this practical course, you will know how to:

- Read a specification
- Write PSL properties
- Write and verify a testbench
- Find and correct bugs
  - Using a simulation environment
  - Using a model checking tool (OneSpin)

Notes

A report has to be sent by mail one week after the end of the second lab to:

gplassan@synopsys.com

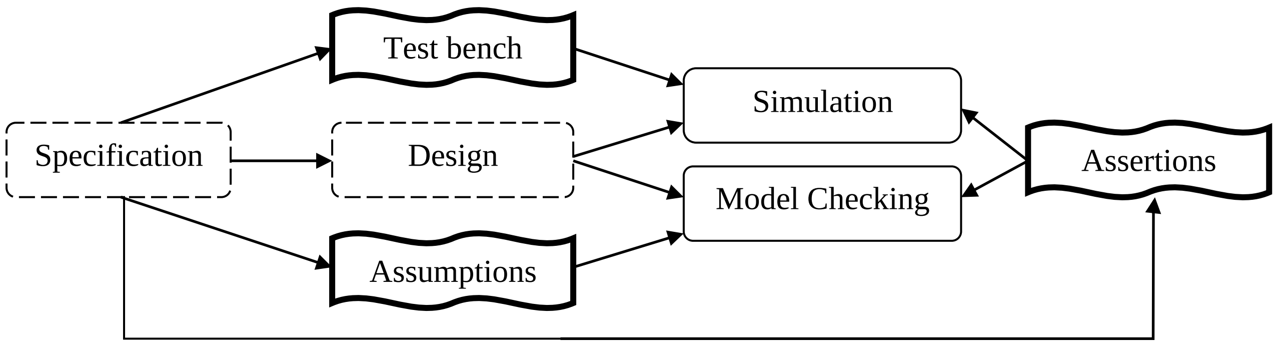
You can use computers freely in your off-time, but be aware that the CIME gates close at 8pm (then, you can get out, but not get in).

General information

Login: "xph3seixxx" xx =200...205

In the main folder, TP_VERIF, you will find 7 sub-folders:

- config	Configuration files for the tools (modelsim, onespin). Always source both `.bashrc_*` right after opening a terminal.
- bench	VHDL of the test bench
- libs	Libraries for VHDL compilation
- onespin	Folder used by the model checking tool
- psl	Formal assertions/properties for the design
- vhd_ko	VHDL sources of the design (with bugs)
- vhd_ok	VHDL sources of the design (without bug)

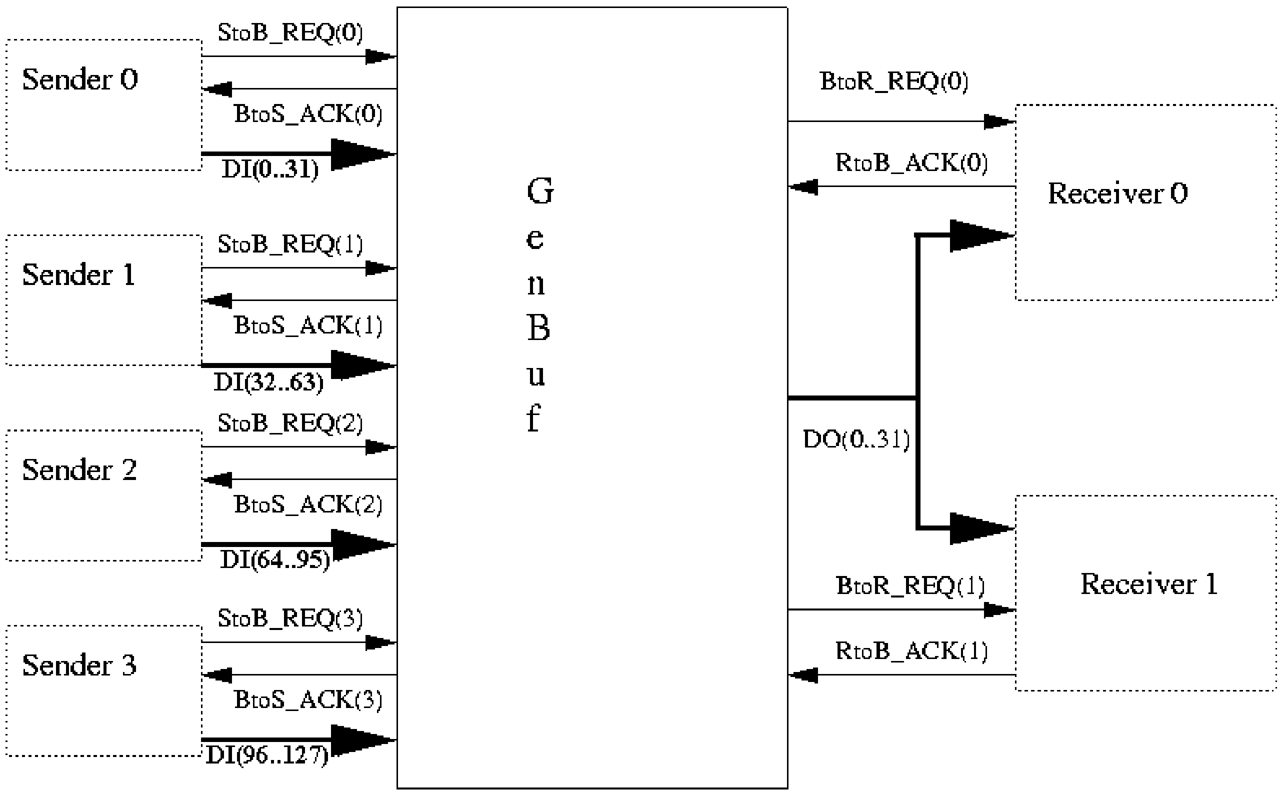


Guillaume Plassan & Katell Morin-Allory – October 2016

Design specification (from IBM)

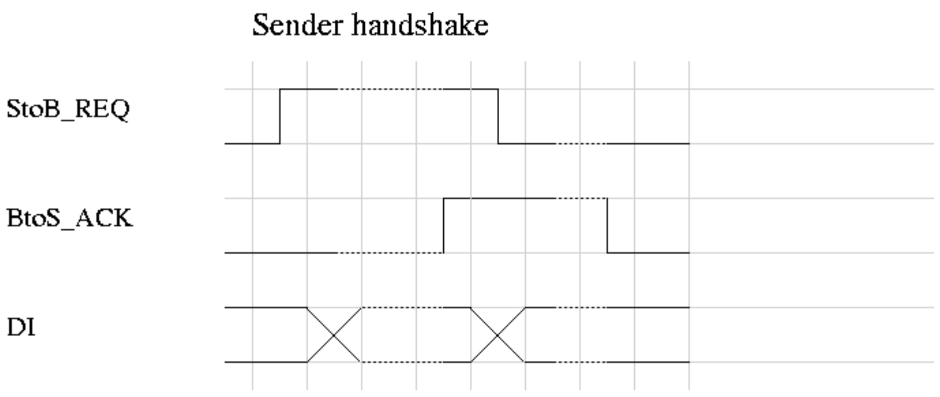
GenBuf is a design block that queues words (32 bits) of data sent by four senders to two receivers. The queue is a depth 4 FIFO. The senders are equivalent, as are the receivers. The interface for each sender consists of a request input (denoted StoB_REQ(i) for the ith sender), an acknowledge output (denoted BtoS_ACK(i)), and one point-to-point data bus (denoted DI(i*32..(i+1)*32-1)). (Note that each sender has its own bus, although we are defining a joint single dimension array).

The interface for each receiver consists of a request output (denoted BtoR_REQ(i)), an acknowledge input (denoted RtoB_ACK(i)), and one output data bus (denoted DO(0..31)), that is shared by both receivers. The following is a block diagram of the design and its interface. Dashed boxes represent the environment.



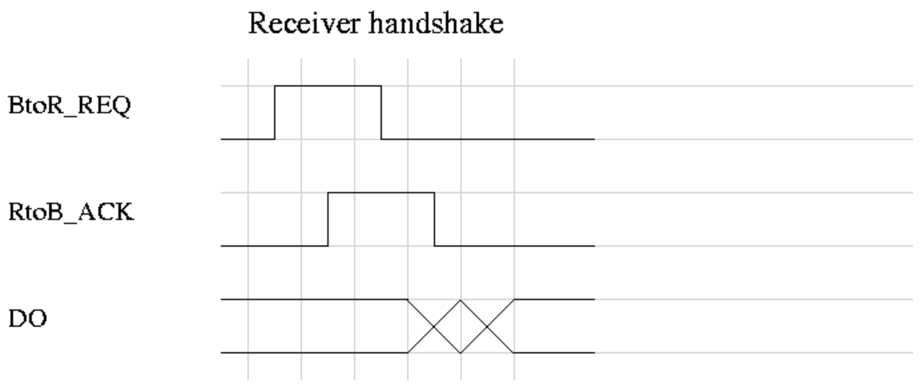
The interface between GenBuf and the senders is a 4-phase handshaking protocol described below:

1. When sender i, has data to send, it initiates a transfer by asserting StoB_REQ(i) (Server to Buffer REQuest).
   One cycle later, the sender puts the data on its data bus (i.e. DI(i*32..(i+1)*32-1)).
2. When GenBuf can service the sender, it reads the data from the data bus and asserts BtoS_ACK(i).
3. In the cycle following the assertion of BtoS_ACK(i), the sender should deassert the signal StoB_REQ(i). From this point onwards, the data on the data bus is considered invalid.
4. The end of the transaction is marked by GenBuf deasserting BtoS_ACK(i). A new transaction may begin a cycle after the deassertion of BtoS_ACK(i). (Note: GenBuf may hold BtoS_ACK(i) asserted for several cycles before eventually deasserting it.)



The interface between GenBuf and the receivers is a 4-phase handshaking protocol described below:

1. When GenBuf has data to send, it chooses one of the receivers, (denoted the jth receiver). GenBuf then initiates a transfer by asserting BtoR_REQ(j).
2. In the cycle following the assertion of BtoR_REQ(j), the receiver asserts RtoB_ACK(j).
3. In the cycle following the assertion of RtoB_ACK(j), GenBuf puts the data on the data bus (i.e. DO(0..31)) and deasserts BtoR_REQ(j). Note: The cycle in which the data on the bus is valid is the same cycle when GenBuf deasserts BtoR_REQ(j).
4. To conclude the transaction, the receiver should deassert RtoB_ACK(j) a cycle after the deassertion of BtoR_REQ(j). GenBuf does not initiate another transaction until a cycle after RtoB_ACK(j) is deasserted.



The following properties are guaranteed:

- GenBuf maintains FIFO order
- Senders are never starved
- Receivers are requested by a round-robin scheme
- GenBuf never requests two receivers at the same time

Test bench

In the **bench** folder, you will work with the file **bench.vhd**, which instantiates the genbuff.

- ➢ Using the design specification, write a bench simulating the behavior of the sender and receiver modules.
- ➢ You should try to maximize the test coverage with an exhaustive simulation.

PSL Properties Specification

Two different kinds of properties have to be written in order to model the system:

- Assertions that the design must follow
- Assumptions that the environment must follow

Handshake Properties:

- A request eventually gets acknowledged.
- When the request is deasserted, acknowledge will eventually be deasserted.
- No acknowledge unless requested.
- The request stays up until it is acknowledged.
- The request is deasserted one cycle after it has been acknowledged.
- Acknowledge cannot be deasserted unless the request is first deasserted.

Genbuf Properties:

- GenBuf does not request two receivers at the same time.
- GenBuf will not make two consecutive requests to the same receiver.
- GenBuf deasserts BtoR_REQ in the cycle that it puts the data on bus.
- Every data word read by GenBuf will be eventually sent out to a receiver.
- Data are kept in order.

Sender/Receiver Properties

- Only one sender can send data at any given time.
- Only one receiver can read data at any given time.

Queue Properties:

- GenBuf does not receive when the queue is full.
- Genbuf does not send when the queue is empty.
- All data received by the receivers has previously been sent by the sender (i.e., the queue is not sending invalid data to the receiver).

- ➢ In **genbuf.psl**, write all these properties.

<u>Question:</u> Can you prove that the Handshake specification is complete (i.e. all possible behaviors are formalized into properties)? You could use a truth table.

<u>Questions (concerning genbuf,psl):</u>

What is the purpose of "**vmode Queue**"?

What is the use of the signals **Q_counter**, **DxRF_***, **DxR**, **DxW** and **DWR**?

Explain the properties: **xx_stable**, **DATA_E_W_***, **GENBUF_F_O_*** and **Not_C_***.

Simulation

In this step, you will simulate an execution of the genbuf in your bench, and check if you can catch a bug with your assertions. As you can see in **script**, the properties must be linked to the VHDL during compilation with the command line: `$ vcom -work * -pslfile *.psl *.vhd`
The properties panel can be opened in Modelsim with: **View → Coverage → Assertions**

- Simulate the **genbuf_ok** in your bench, with the PSL properties. Since the design is correct, if you find bugs, it means your properties are incorrect, and you should fix them.
  Note: Of course, in the real world, you will not have a correct design to check that your PSL properties are correct, so you should be extra-careful when writing properties.

- Simulate the **genbuf_ko** in your bench, with the PSL properties.

Question: Can you find any bug? If you do, describe it so that the designer can correct it.
Question: There are 7 bugs in the vhd_ko. Did you find them all? If not, how would you explain this?

Model Checking

In this step, you will use the model checking tool OneSpin to find potential bugs.
You can import the correct VHDL first in order to learn how to use the tool. However, your final results must be done on the **vhd_ko**.

- Go in the **onespin** directory and run the tool
```
$ onespin&
```
- In **setup/vhd**, select **Version 2008**, then load the VHDL and PSL files.
- Click on **E** to elaborate (building the design hierarchy), then click on **C** to compile.

OneSpin Setup

- Click on **CC** to enter the consistency check mode.
- In the **Auto checks** tab, right click on a signal and select **check all**.
  OneSpin will check for initialization, dead code, lint, etc. ()
- Are the registers properly initialized (**Init** tab)?
  To change the default reset active value, type in the shell:
```
> set_reset_sequence -high RST
```
- Are some FSMs stuck in initial state (**Stick** tab)?
  To add a control on the inputs, type in the shell:
```
> set_mode setup
> set_compile_option -undriven_value input
> set_mode cc
```
- Run the verification again and analyze the results.

Verifying PSL properties

- Click on **MV** to enter the modelchecking mode.
- Run the model checking on each property and analyze the results.

Hint: In order to avoid state explosion, you may try to put a constant on the data.

Question: Can you find any bug? If you do, describe it so that the designer can correct it.
Question: There are 7 bugs in the vhd_ko. Did you find them all? If not, how would you explain this?