VC Formal Lab
Assume Guarantee

Learning Objectives

In this VC Formal lab, you will use a FIFO example to learn to do the following:

- Perform assume-guarantee reasoning

Lab Duration:
40 minutes

Familiarity with the SystemVerilog Assertion (SVA) language and knowledge of basic formal verification concepts are required for this lab.

Files Location

All files for this VC Formal lab are in directory:
$VC_STATIC_HOME/doc/vcst/examples/FPV/Abstraction/Assume_Guarantee

Directory Structure	
FPV/Abstraction/Assume_Guarantee	Lab main directory
README_VCFormal_Assume_Guarantee.pdf	Lab instructions
design/	Verilog RTL code of the Device Under Test (DUT)
sva/	SVA properties to check functionality of the DUT
run/	Run directory
solution/	Solution Directory

Resources

The following resources are available for in-depth guidance regarding VC Formal usage, commands, and variables.

VC Formal User Guide:
$VC_STATIC_HOME/doc/vcst/VC_Formal_Docs/VC_Formal_UG.pdf

VC Formal Apps Quick References Guides:
$VC_STATIC_HOME/doc/vcst/VC_Formal_Docs/Quick_Reference_Guides/

VC Formal Apps Tcl Templates:
$VC_STATIC_HOME/doc/vcst/VC_Formal_Docs/Quick_Reference_Guides/vcf_tcl_templates/

Prepare your Environment

1. Set environment variable pointing to your VC Formal installation directory:

```
%setenv VC_STATIC_HOME /tools/synopsys/vcstatic
```

2. Add path $VC_STATIC_HOME/bin to the PATH environment variable.

3. Change your working directory to FPV/Abstraction/Assume_Guarantee/run:

```
%cd FPV/Abstraction/Assume_Guarantee/run
```

Now you are ready to begin the lab.

Create a run.tcl Setup File

VC Formal has a Tcl-based command interface. It is common to start with a Tcl file to set up and compile a design. In this step, you will create a VC Formal Tcl file for the DUT, a FIFO, used in this lab.

4. Open file run.tcl (any arbitrary name is ok to use) using any text editor:

```
%vi run.tcl
```

5. Add command to enable FPV App mode (default when starting VC Formal) and debug modes:

```
set_fml_appmode FPV
```

6. Specify the width and depth of the FIFO as Tcl variables:

```
set FIFODEPTH 8
set FIFOWIDTH 32
```

7. Specify the directories for the design and sva as Tcl variables. The DUT file is located under directory FPV/Abstraction/Assume_Guarantee/design. The assertion and bind files are located under directory FPV/Abstraction/Assume_Guarantee/sva:

```
set SVA_DIR ../sva
set RTL_DIR ../design
```

8. Specify Formal TB top level module name as Tcl variable:

```
set design fifo_ctrl
```

9. Add command to compile DUT and SVA properties:

```
     set vcs " \
          +define+FIFODEPTH=${FIFODEPTH} \
          +define+FIFOWIDTH=${FIFOWIDTH} \
          ${RTL_DIR}/fifo_ctrl.sv \
          ${SVA_DIR}/fifo_abs.sv \
          ${SVA_DIR}/bind_fifo_abs.sv \
          -assert svaext \
        "
```

```
     read_file -sva -format sverilog -top $design -vcs "$vcs"
```

Note: To use unified usage model to compile design, use these commands instead of `read_file` to compile design and SVA properties:

```
     analyze -format sverilog \
       -vcs "$vcs"
     elaborate $design -sva
```

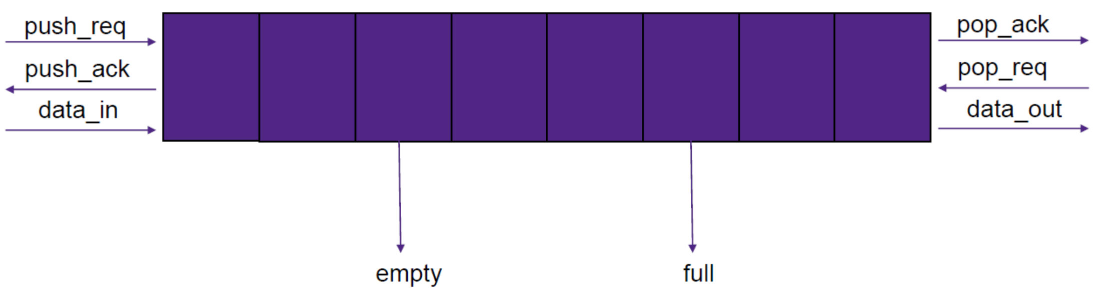
10. Add clock and reset information:

```
     create_clock clk -period 100
     create_reset resetn -low
     sim_run -stable
     sim_save_reset
```

11. Save run.tcl file and exit editor.

FIFO Block Diagram

First In First Out (FIFO)



- Design should write data and read data in a First In First Out order
- When the design has no more free space, the full flag should be raised
- When the design has no data inside, the empty flag should be raised
- When the full flag is high, push_ack must be low
- When the empty flag is high, pop_ack must be low
- When push_req is high and push_ack low, push_req must be kept high and data_in stable
- When pop_req is high and pop_ack low, pop_req must be kept high

Formal Testbench for FIFO

Check fifo_ctrl_checker.sv file to see all assertions and assumptions. There is a scoreboard as well that is used to check the data integrity of the DUT.

12. Start the tool in Verdi GUI mode:

```
    %vcf -f run.tcl -verdi
```

    VC Formal starts in the Verdi GUI mode, with icons, tables, tabs, and windows especially designed for property verification with the FPV App. The App mode is set to FPV by default.

13. Start property verification:

    Click on the Start Check icon ▶

    Assertions "ast_no_overflow" and "ast_data_integrity" fail. Debug "ast_no_overflow" first and try to prove it. Then, try to check the other assertion "ast_data_integrity". It will not converge.

Lab Solution

The assume-guarantee reasoning can be applied in this lab to allow proving the "ast_data_integrity" assertion. To apply this methodology, the problem can be divided as follows:

- Abstract FIFO Full (this is the assume part of the assume-guarantee methodology):
  - Reset abstract WR and RD pointers
  - Constraint wr_ptr == rd_ptr just after reset (for the 1st cycle)
  - Add cutpoint on FIFO full signal
  - Add constraints on FIFO full
  - empty |-> !full
  - wr_ptr == rd_ptr && !empty |-> full
  - full && !pop |-> ##1 full
  - full && pop && push |-> ##1 full
  - full && pop && !push |-> ##1 !full

- Verify these constraints in the original setup with no cutpoints or reset abstractions (this is the guarantee part of the assume-guarantee reasoning).

In order to prove the last assertion using the above reasoning, the following steps can be followed to apply the assume-guarantee methodology:

14. Change your working directory to FPV/Abstraction/Assume_Guarantee/solution/run:

```
    %cd FPV/Abstraction/Assume_Guarantee/solution/run
```

15. Start the tool in Verdi GUI mode:

```
    %vcf -f run.tcl -verdi
```

Check the content of run.tcl using in Verdi. A Tcl variable "ASSUME" is defined and is equal to 1, which allows running the proof while having the above-mentioned constraints as assumptions.

16. Start property verification by clicking on the Start Check icon . Ensure that all assertions are proven.

17. Edit the Tcl file and set ASSUME to 0 and GUARANTEE to 1. This allows verifying the assumptions.

18. Restart the session using and run the proofs again. All assertions should be proven.