VC Formal Lab
Case Splitting

Learning Objectives

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

- Perform case splitting

Lab Duration: 40 minutes

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

©2023 Synopsys, Inc.

Files Location

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

Directory Structure	
FPV/Abstraction/Case_Splitting	Lab main directory
README_VCFormal_Case_Splitting.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/Case_Splitting/run:

```
%cd FPV/Abstraction/Case_Splitting/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 data transfer channel, 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. Set the following Tcl variables to define length, data width, data integrity assertion enable and split mode

```
set LEN_WIDTH 3
set DATA_WIDTH 64
set DATA_INTEG 0
set SPLIT_MODE 0
```

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

```
set ENV_DIR ..
set RTL_DIR ${ENV_DIR}/design
set SVA_DIR ${ENV_DIR}/sva
```

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

```
set design ctrl
```

9. Add command to compile DUT and SVA properties:

```
set vcs "
  +define+DATA_INTEG=${DATA_INTEG}
  +define+SPLIT_MODE=${SPLIT_MODE}
  -pvalue+LEN_WIDTH=${LEN_WIDTH}
  -pvalue+DATA_WIDTH=${DATA_WIDTH}
  ${RTL_DIR}/ctrl.sv
  ${SVA_DIR}/ctrl_checker.sv
  ${SVA_DIR}/bind_ctrl_checker.sv
"
```

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

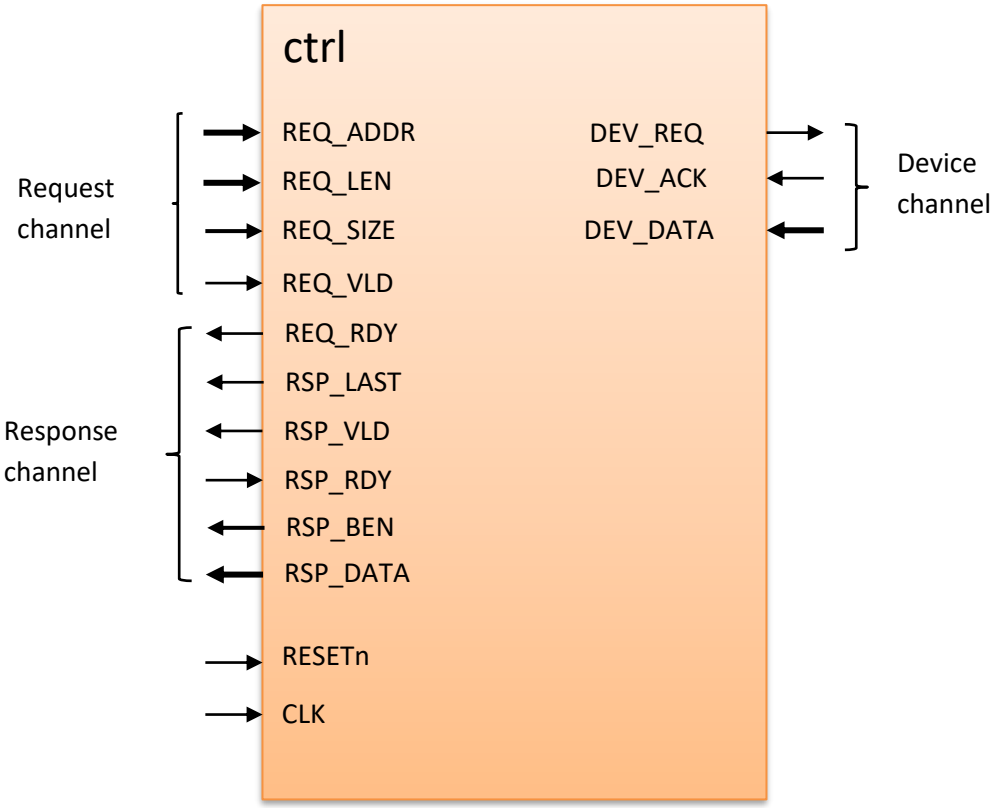
```
create_clock CLK -period 100
create_reset RESETn -low
```

```
sim_run
sim_save_reset
```

11. Save run.tcl file and exit editor.

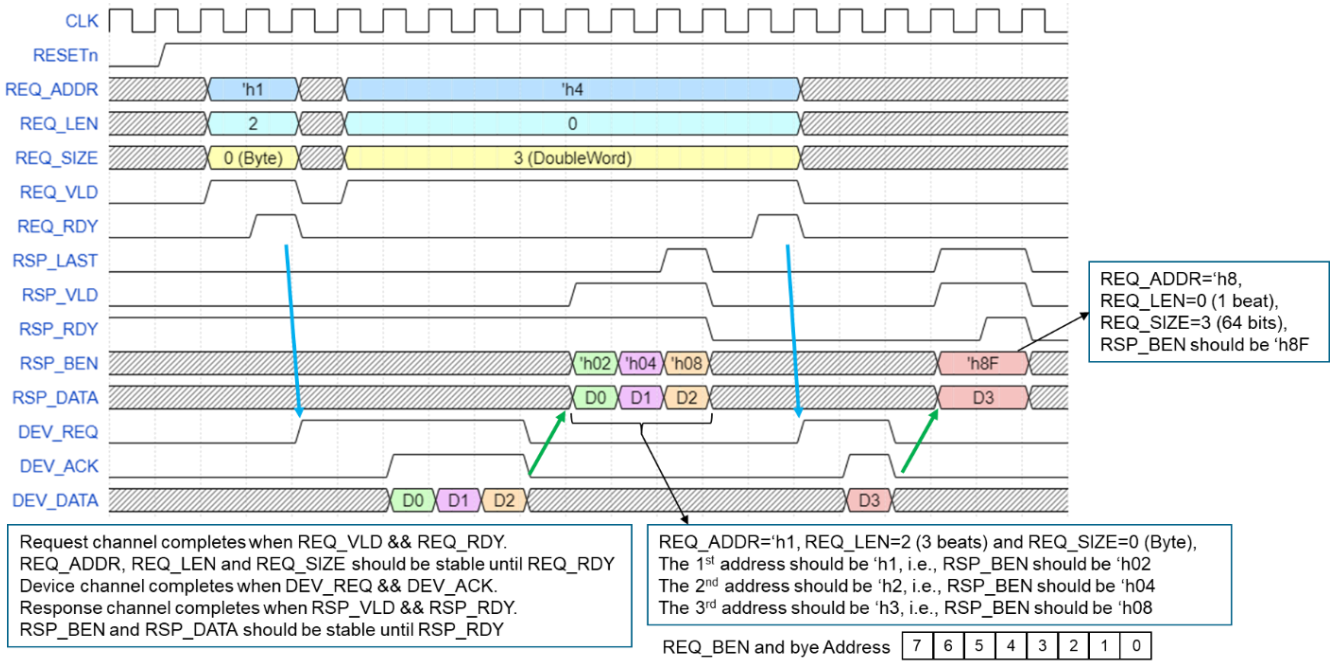
DUT Specification

Simple data transfer with variable length.



Parameter Name	Default		Description
ADDR_WIDTH	20		Address bit width
DATA_WIDTH	64		Data bit width
LEN_WIDTH	3		Length bit width
BEN_WIDTH	(DATA_WIDTH/8)		Byte enable bit width
Port Name	**Direction**	**Bit width**	**Description**
CLK	In	1	Clock
RESETn	In	1	Reset (active Low)
REQ_ADDR	In	ADDR_WIDTH	Request address
REQ_LEN	In	LEN_WIDTH	Request length
REQ_SIZE	In	2	Request access size
REQ_VLD	In	1	Request valid
REQ_RDY	Out	1	Request ready
RSP_LAST	Out	1	The last data of response
RSP_VLD	Out	1	Response valid
RSP_RDY	In	1	Response ready
RSP_BEN	Out	BEN_WIDTH	Response byte enable
RSP_DATA	Out	DATA_WIDTH	Response data
DEV_REQ	Out	1	Device request
DEV_ACK	In	1	Device acknowledgement
DEV_DATA	In	DATA_WIDTH	Device data

The figure below shows the timing diagram of the DUT.



Formal Testbench for DUT

```
localparam RSP_LENGTH = 'b1 << LEN_WIDTH;

bit [LEN_WIDTH-1:0]  req_counts, rsp_counts, dev_counts;
bit [DATA_WIDTH-1:0] dev_data_q [0:RSP_LENGTH-1];
bit [ADDR_WIDTH-1:0] rqst_addr;
bit [1:0]            rqst_size;
bit [BEN_WIDTH-1:0]  exp_rsp_ben;
bit [BEN_WIDTH-1:0]  byte_rsp_ben;
bit [BEN_WIDTH-1:0]  hword_rsp_ben;
bit [BEN_WIDTH-1:0]  word_rsp_ben;
bit [BEN_WIDTH-1:0]  dword_rsp_ben;
bit [DATA_WIDTH-1:0] valid_byte_lanes;
bit [2:0]            symb_bit;
bit [7:0]            data_byte_lanes [0:BEN_WIDTH-1];

always @(posedge CLK or negedge RESETn) begin
   if (!RESETn) begin
      req_counts <= {RSP_LENGTH{1'b0}};
      rsp_counts <= {RSP_LENGTH{1'b0}};
      dev_counts <= {RSP_LENGTH{1'b0}};
      rqst_addr  <= {ADDR_WIDTH{1'b0}};
      rqst_size  <= 2'b0;
      for (int i=0;i<RSP_LENGTH;i++) begin
         dev_data_q[i] <= {DATA_WIDTH{1'b0}};
      end
   end else begin
      if (REQ_VLD && REQ_RDY) begin
         req_counts <= REQ_LEN;
         rqst_addr  <= REQ_ADDR;
         rqst_size  <= REQ_SIZE;
         rsp_counts <= {RSP_LENGTH{1'b0}};
         dev_counts <= {RSP_LENGTH{1'b0}};
      end else begin
         if (RSP_VLD && RSP_RDY) begin
            rsp_counts <= rsp_counts + 'd1;
         end
         if (DEV_REQ && DEV_ACK) begin
            dev_data_q[dev_counts] <= DEV_DATA;
            dev_counts <= dev_counts + 'd1;
         end
      end
   end
end
```

Modeling codes


property p_rsp_data_stable;
@(posedge CLK) disable iff (!RESETn)
(RSP_VLD && !RSP_RDY) |=> $stable((RSP_DATA & valid_byte_lanes));
endproperty
property p_rsp_data_stable_per_byte(idx);
@(posedge CLK) disable iff (!RESETn)
(RSP_VLD && !RSP_RDY) |=> $stable(data_byte_lanes[idx]);
endproperty
property p_rsp_data_integrity;
@(posedge CLK) disable iff (!RESETn)
RSP_VLD |-> (RSP_DATA==dev_data_q[rsp_counts]);
endproperty
property p_rsp_data_integrity_per_beat(beat);
@(posedge CLK) disable iff (!RESETn)
(RSP_VLD && rsp_counts==beat) |-> (RSP_DATA==dev_data_q[beat]);
endproperty
Property to check data stability for whole data
Property to check data stability per byte lane
Property to check data integrity for all beats
Property to check data integrity per data beat
Property definition
ast_rsp_data_stable : assert property(p_rsp_data_stable);
for (genvar i=0;i<BEN_WIDTH;i++) begin : PERBYTE
ast_rsp_data_stable_per_byte : assert property(p_rsp_data_stable_per_byte(i));
end : PERBYTE
ast_rsp_data_integrity : assert property(p_rsp_data_integrity);
for (genvar i=0;i<RSP_LENGTH;i++) begin : DATINTEG
ast_rsp_data_integrity_per_beat : assert property(p_rsp_data_integrity_per_beat(i));
end : DATINTEG
Property instantiation

Case Splitting

- ‘ast_rsp_data_stable’ checks all data bits
  - 1 assertion checks all bits, i.e., all data byte lanes
- ‘ast_rsp_data_stable_per_byte’ checks per byte lane
  - 1 assertion checks 1 byte lane
  - The number of assertions depends on the number of byte lanes, i.e., data bus width
- ‘ast_rsp_data_integrity’ checks all data beats
  - 1 assertion checks data integrity for all burst beats
- ‘ast_rsp_data_integrity_per_beat’ checks per burst beat
  - 1 assertion checks 1 beat during burst
  - The number of assertions depends on the burst length

Run Formal Proofs and Review Results

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  . Ensure that all assertions are proven.

Lab Solution

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

```
% cd FPV/Abstraction/Case_Splitting/solution
```

Several modes are supported in run.tcl. Try each of the following to see the effect of each mode.

15. Execute non case splitting assertions only:

```
% vcf -f run.tcl -x "set SPLIT_MODE 1" -verdi &
```

16. Execute case splitting assertions only:

```
% vcf -f run.tcl -x "set SPLIT_MODE 2" -verdi &
```

17. Execute Data Integrity assertions only:

```
% vcf -f run.tcl -x "set DATA_INTEG 1" -verdi &
```

18. Execute with 16 workers: add 'set WORKERS 16;':

```
% vcf -f run.tcl -x "set DATA_INTEG 1; set WORKERS 16;"
-verdi &
```

19. Execute with different transfer length: add 'set LEN_WIDTH <n>;'
for example, LEN_WIDTH 4 to make 16 beats:

```
% vcf -f run.tcl -x "set LEN_WIDTH 4; set WORKERS 16;"
 -verdi &
```

Other Examples