Using SystemVerilog Assertions for Functional Coverage

Mark Litterick (Verification Consultant)

mark.litterick@verilab.com
Outline

- Demonstrate capability of SVA for implementing a complex functional coverage monitor
- Coding style and syntax applicable to more typical application of SVA coverage: to define designer’s key corner cases
- Related to SNUG Europe 2005 paper (http://www.verilab.com/download.htm):
  Utilizing Vera Functional Coverage in the Verification of a Protocol Engine for the FlexRay™ Automotive Communication System
- Overview of FlexRay™ protocol engine testbench architecture
  - role of functional coverage
  - overview of protocol engine architecture and specification
  - requirements for functional coverage model
- SVA implementation, structure and coding
- Conclusion
Protocol Engine
Testbench Architecture
Protocol Engine Architecture

START

STATE 0

STATE 1

STATE 2

POC

SYNC

MAC

CODEC

SDL

FSP

CHANNEL INTERFACE

HOST INTERFACE

BUS GUARDIAN INTERFACE

POC = Protocol Operation Control
SYNC = Clk Sync and Macrotick Gen
MAC = Media Access Control
FSP = Frame and Symbol Processing
CODEC = Coding/Decoding
Functional Coverage Model Requirements

- Variety of coverage points
  - states
  - state transitions
  - multi-state transition sequences
  - conditional state (path) transitions
  - SDL trigger events
- Check for illegal states and transitions
- RTL implementation independent
- Coding style closely reflects SDL for maintenance and debug
- Capability to query coverage results from testbench environment for closed loop constrained random stimulus or functional checking
State Coverage

\`define DEFAULT_CONFIG (vState =?= 7'b000_xxxx)
\`define INTEGRATION_LISTEN  (vState =?= 7'b111_0101)
// etc

state_DC : cover property ( @(posedge clk) `DEFAULT_CONFIG);
state_IL : cover property ( @(posedge clk) `INTEGRATION_LISTEN);
// etc

property prop_LEGAL_STATE;
@(posedge clk)
( `DEFAULT_CONFIG ||
  `CONFIG ||
  `READY ||
  // etc
  `NORMAL_PASSIVE ||
  `HALT
); endproperty : prop_LEGAL_STATE

assert_LEGAL_STATE : assert property (prop_LEGAL_STATE)
else $error("%m: illegal state");
Transition Coverage

```verilog
property prop_LEGAL_TRANS;
  @(posedge clk)
  disable iff (rst)
  (!$stable(vState) |-> (seq_H_DC.ended ||
    // etc
    seq_CL_IS.ended));
endproperty

assert_LEGAL_TRANS : assert property (prop_LEGAL_TRANS)
else $error("%m: illegal transition");

sequence seq_H_DC;
  @(posedge clk)
  `HALT ##1 `DEFAULT_CONFIG;
endsequence : seq_H_DC

sequence seq_CL_IS;
  @(posedge clk)
  `COLDSTART_LISTEN ##1 `INITIALIZE_SCHEDULE;
endsequence : seq_CL_IS
// etc

trans_H_DC : cover property ( seq_H_DC );
trans_CL_IS : cover property ( seq_CL_IS );
// etc
```
sequence seq_POC_CSA_CSI;
 @(posedge clk)
   `READY        ##1
   `COLDSTART_LISTEN   [*1:$] ##1
   `INITIALIZE_SCHEDULE [*1:$] ##1
   `INTEGRATION_COLDSTART_CHECK [*1:$] ##1
   `COLDSTART_JOIN    [*1:$] ##1
   `NORMAL_ACTIVE    ;
endsequence : seq_POC_CSA_CSI

trans_POC_CSA_CSI : cover property ( seq_POC_CSA_CSI );
Conditional Transition Coverage

```verilog
F07_13_007 : cover property ( @(posedge clk) seq_CCC_IL.ended && (zStartupNodes <= 0) && (vCycleCounter[0] == 1) );

F07_13_009 : cover property ( @(posedge clk) seq_CCC_IL.ended && (zSyncCalcResult != `WITHIN_BOUNDS) && (zStartupNodes > 0) && (vCycleCounter[0] == 1) );

F07_13_011 : cover property ( @(posedge clk) seq_CCC_IL.ended && (vRemainingColdstartAttempts == 0) && (zStartupNodes == 0) && (vCycleCounter[0] == 0) );

//etc
```
SDL Trigger Event Coverage

F07_11_003_A : cover property ( @(posedge clk) 'COLDSTART_LISTEN && $rose(header_received_on_A));

F07_11_003_B : cover property ( @(posedge clk) 'COLDSTART_LISTEN && $rose(symbol_decoded_on_B));

F07_11_005 : cover property ( @(posedge clk) 'COLDSTART_LISTEN && $rose(CHIRP_on_A));

F07_11_011 : cover property ( @(posedge clk) 'COLDSTART_LISTEN && $rose(tStartup));

// etc
SVA Coverage Monitor Architecture

module FRPocCovMon_sva (input clk, input rst, input [6:0] vState, input pWakeupChannel, input [4:0] vRemainingColdstartAttempts, input [1:0] zSyncCalcResult, input header_received_on_A, ...);

// cover property statements
endmodule : FRPocCovMon_sva
SystemVerilog Coverage API

• SystemVerilog Coverage API real-time access system functions

integer num;
if ($coverage_control( `SV_COV_CHECK,
  `SV_COVASSERTION,
  `SV_COV_MODULE,
  $root.tb.pe.poc.trans_POC_CSA_CSI) == `SV_COV_OK )
begin
num = $coverage_get( `SV_COVASSERTION,
  `SV_COV_MODULE,
  $root.tb.pe.poc.trans_POC_CSA_CSI);

case(num)
  `SV_COV_OVERFLOW : $fatal("non-integer coverage value");
  `SV_COV_ERROR : $fatal("error extracting coverage value");
  default : $display("trans_POC_CSA_CSI coverage = %0d",num);
endcase
end
else
$fatal("coverage cannot be obtained for trans_POC_CSA_CSI");
Conclusion

• Possible to generate a complex functional coverage monitor completely in SVA
  – demonstrates coverage capability of SystemVerilog Assertions language

• Pros:
  – mixed language simulators enable SVA to be mixed with any HDL or HVL
  – can be used in testbench environments with no HVL coverage
  – implementation and use requires no OO skills

• Cons:
  – not scalable for all coverage in system, e.g. abstract data in stimulus

• Provides coding guidelines for typical SVA coverage applications

mark.litterick@verilab.com