Simulation-Based FlexRay™ Conformance Testing - An OVM Success Story

Mark Litterick
Consultant & Co-Founder
Verilab

Agenda

- FlexRay overview
- What we mean by conformance testing
- What OVM brings to the party
- Layered sequences
- Using the factory for configuration
- Using TLM ports for scoreboard checker
- Some project development statistics
- Conclusions
**FlexRay™ Overview**

- automotive communications system
- robust, deterministic & fault-tolerant
- supports applications like drive-by-wire

**FlexRay™ Configurations**

- scalable & flexible
- many network topologies
- different data rates
- synchronous and asynchronous transfer
- single or multi-master clock synchronization
- time synchronization across networks
- scalable fault tolerance
- over 60 node and cluster parameters
Conformance Testing

- **FlexRay Conformance Test Specification**
  - defines a set of directed test cases and methods
  - verifies *conformance* of communication controllers with FlexRay Protocol Specification
  - devices must pass these tests to claim conformance
  - test specification is independent of test environment implementation
  - test environment implementation can be hardware or simulation-based

Simulation-Based Testing

- Simulation-based *verification* is standard for *all* complex semiconductor design
- **Advantages include:**
  - validate behaviour before manufacture
  - find and correct defects earlier in design cycle
  - improved controllability and observability
  - lower-cost and lower-risk for supplier and end user
  - improved quality
- **Additional benefits to Conformance Test project include:**
  - concluded prior to any V3.0 controllers being available
  - enabled validation of evolving Conformance Test Specification
  - also found several defects in new Protocol Specification features
  - verification of FlexRay Executable Model
SystemVerilog & OVM

- Conformance Test Environment
- Open Verification Methodology (OVM)
- SystemVerilog IEEE1800
- Questa™
- IUS™
- VCS™

- FlexRay-specific code
- Uses OVM building blocks
- Open source (Apache)
- Class library
- Consistent methodology
- Facilitates interoperability
- Supported by all simulators

- Multi-language simulators
- VHDL, Verilog, SystemVerilog, SystemC

Conformance Test Environment

- UPPER TESTER (UT)
  - CHI Adaption Layer
  - Implementation Under Test (IUT)

- LOWER TESTER (LT)

- Conformance Tests

- IUT Implementation
  - Capabilities
  - Features

- LOG

- RESULTS
  - PASS/FAIL
OVM Architecture

OVM Sequences

- OVM sequences are used for all **stimulus**
- Concise description of test stimulus
- Example:

  In cycle 9, the LT simulates a startup frame in slot 1 with the wrong header CRC...

  ```
  `ovm_do_on_with(error_seq,LT,{
    lt_ch == FR_AB;
    lt_cycle == 9;
    lt_kind == FR_STARTUP_PAYLOAD;
    lt_slot == 1;
    lt_error == FR_HEADER_CRC;
  })
  ```
Sequence Hierarchy

Layered Sequences

```verilog
`ovm_do(preamble3_seq)

fork
  `ovm_do_on_with (ut_exe_seq, 'UT, (count==1;))
  `ovm_do_on_with (lt_error_seq, 'LT, {ch==AB; kind==STARTUP; cycle==8; error==CRC;})
join

fork
  if (ch inside{A, AB}) `ovm_do_on_with (ch_x_seq, 'CHA, {...; error==CRC;})
  if (ch inside{B, AB}) `ovm_do_on_with (ch_x_seq, 'CHB, {...; error==CRC;})
  `ovm_do_on_with (tb_cycle_seq, 'TB, {count==1;})
join

`ovm_create(item)
  item.c_error.constraint_mode(0); // disable
  item.randomize() with {...; item.error==CRC;}
  `ovm_send(item)

if (item.error==CRC) generate_bad_crc();
else generate_good_crc();
ch_interface.RxD = crc[N]; // drive signals
```
Using OVM Factory for Configuration

- FlexRay protocol has more than 60 node and cluster configuration parameters
- Each test must run under multiple configurations
  - 5x basic configurations
  - 3x channel applicability
  - preamble & test-specific modifications
  - require 10164 total runs for 432 conformance tests
- Configuration information used all over the testbench (drivers, monitors, sequencers)

OVM Factory solves the problem...

Configuration Hierarchy

```
// define all 60+ parameters
rand int pKeySlotID;
// define valid range under all conditions
constraint c_range_pKeySlotID {pKeySlotID inside {[0:1023]};}
// invalid constraints must be overloaded in derived classes
constraint c_pKeySlotID {0;}
// redefine constraints for all parameters
constraint c_pKeySlotID {pKeySlotID == 1;}
// redefine only required constraints for each test
constraint c_pKeySlotID {pKeySlotID == 2;}
```

Conformance Test Specification (ODT File)

- test_1_2_3
  - 10 parameters
  - 3 modifications sets
  - 5 basic config
  - 3 ch applicable (A,B,AB)
  - 45x config classes

odt2cfg script

configuration library
**Example Configuration**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Modification to Basic Configuration 1</th>
<th>Modification to Basic Configuration 2</th>
<th>Modification to Basic Configuration 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>gDynamicSlotIdlePhase [bitwidth]</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>gMacroPerCycle [BIT]</td>
<td>2095</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>gNumberOfStaticSlots</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>gPayloadLengthStatic [byte]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pClusterDriftDamping [FS]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pPayloadLengthDynMax [byte]</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

// file
// single line to include configuration
#include "config_lib/config_1_2_3.sv"

---

class test_1_2_3_bc1a_cha_i_config
extends flexray_config_bc1a_p2;
`ovm_object_utils(test_1_2_3_bc1a_cha_i_config)
`fr_config_mod_cha
constraint c_gdDynamicSlotIdlePhase { gdDynamicSlotIdlePhase == 0 ; }
constraint c_gMacroPerCycle { gMacroPerCycle == 2095 ; }
constraint c_gNumberOfStaticSlots { gNumberOfStaticSlots == 2 ; }
constraint c_gPayloadLengthStatic { gPayloadLengthStatic == 0 ; }
constraint c_gCycle { gCycle == 2095 ; }
constraint c_pClusterDriftDamping { pClusterDriftDamping == 0 ; }
constraint c_pListenTimeout { pListenTimeout == 1677721 ; }
constraint c_pMicroPerCycle { pMicroPerCycle == 83886 ; }
constraint c_pOffsetCorrectionStart { pOffsetCorrectionStart == 2085 ; }
constraint c_pRateCorrectionOut { pRateCorrectionOut == 51 ; }
constraint c_pPayloadLengthDynMax { pPayloadLengthDynMax == 1 ; }
endclass

---

OVM Factory

---

// test file
// single line to include configuration
#include "config_lib/config_1_2_3.sv"

define fr_test_all_scdc_i (TEST, INDEX)
  `fr_test_config_ch_i_utils (TEST, bc1a, cha, INDEX)
  `fr_test_config_ch_i_utils (TEST, bc1a, chb, INDEX)
  ...
  (15x fr_test_config_ch_i_utils in total)
  `fr_test_config_ch_i_utils (TEST, bc3, chab, INDEX)
enddefine

---

define fr_test_all_scdc_i (TEST, INDEX)
  `fr_test_config_ch_i_utils (TEST, bc1a, cha, INDEX)
  `fr_test_config_ch_i_utils (TEST, bc1a, chb, INDEX)
  ...
  (15x fr_test_config_ch_i_utils in total)
  `fr_test_config_ch_i_utils (TEST, bc3, chab, INDEX)
enddefine

---

```vhdl
class TEST::"CH"::INDEX extends flexray_base_test;
  `ovm_component_utils(TEST::"CH", INDEX)
  function void build();
    super.build();
    set_type_override_by_type("flexray_config::get_type() :: TEST :: CH :: INDEX :: config::get_type()");
  endfunction : build
endclass
```

---

Whole environment sees the change
Test Structure

- include "config_lib/config_1_2_3.sv"
- class test_1_2_3_seq extends base_sequence_utils;
- virtual task body();
- fork
  - ovm_do (preamble3_seq)
- execute
  - ovm_do_on_with (exe_seq,'UT, {count==1;})
  - ovm_do_on_with (err_seq,'LT, {ch==AB; kind==...})
- join
- postamble
- endtask

run test_1_2_3_bc1a_cha_ii
run_conformance list.all.txt

Scoreboard Checker

- expect + observe => compare
SB Operation & TLM Ports

- **UT & LT sources** *publish* transactions
  - instantiate OVM TLM analysis *port*
  - write expected or observed transactions to port
    - IUT Tx: UT expects buffer to be transmitted, LT observes bus traffic
    - IUT Rx: LT expects traffic to be received, UT observes buffer status

- **Scoreboard** *subscribes* to transactions
  - instantiate OVM TLM analysis *export*
  - implement export-specific write method
  - send transactions to comparator

- **Transaction comparator**
  - compares pairs of transactions
  - comparison rules implemented in transaction

---

Scoreboard Implementation

```vlog
// UT CHI ovm_monitor expected transaction
ovm_analysis_port #(flexray_bus_transaction) m_chi_bus_port;
transaction = generate_transaction();  // expect
m_chi_bus_port.write(transaction);    // publish

// LT CHI ovm_monitor observed transaction
ovm_analysis_port #(flexray_bus_transaction) m_cha_bus_port;
transation.field = ch_interface.TxD;     // observe
m_cha_bus_port.write(transaction);   // publish

// ovm_scoreboard subscribes TLM export & implements write()
ovm_analysis_imp_cha_bus #(flexray_bus_transaction, fr_scoreboard) m_cha_bus_export;

m_chi_iut_tx_comparator.before_export.write(transaction);  // in write_chi_bus()
m_chi_iut_tx_comparator.after_export.write(transaction);    // in write_cha_bus()
write to ovm_in_order_class_comparator

m_upper_tester.m_chi_bus_port.connect(m_scoreboard.m_chi_bus_export);
m_lower_tester.m_cha_bus_port.connect(m_scoreboard.m_cha_bus_export);

// TLM port and export connected in env

// actual comparison defined in transaction
function bit flexray_bus_transaction::comp(flexray_bus_transaction transaction);
if (compare_failed) `fr_report_error(...)
```

---

Copyright © Verilab 2010
Code Distribution

- Small OVM overhead (11%)
- Test-Specific but Easy (7%)
- Very Complex Application-Specific Modelling and Checking (32%)
- Mainly Automatically Generated Code (39%)
- Medium Complexity lots of reuse Sequences (11%)

Project Development Chart

- OVM environment running early
- Forget OVM is even there and focus on application problems
- OVM facilitated parallel test development

Copyright © Verilab 2010
Conclusion

- FlexRay Consortium project goals were met
  - Improved quality of *FlexRay V3.0 Conformance Test Specification*
  - FlexRay *Executable Model* rigorously debugged
  - More cost effective Conformance Test Environment
  - Conformance testing earlier in development cycle
  - Operation validated with multiple target IUTs
- OVM increased development productivity and consistency
  - On-time delivery for all project milestones
- OVM really *facilitates* interoperation
  - Interoperation of non-OVM SystemVerilog code still a lot of effort!
  - Achieved identical conformance results on different simulators