Utilizing Vera Functional Coverage in the Verification of a Protocol Engine for the FlexRay™ Automotive Communication System

Mark Litterick, Verilab
Markus Brenner, Freescale Semiconductor
Outline

• Overview of FlexRay™ communications system
  – key characteristics
  – cluster topology and node architecture
  – frame and communication cycle format and hierarchy
• Testbench architecture
  – role of functional coverage
  – overview of protocol engine
• Derive internal state coverage for protocol operation controller
• Vera implementation, structure and coding
• Problems and limitations
• Conclusion
FlexRay™ Overview

- Flexibility, performance and fault tolerance
- Distributed, adaptive clock synchronization
- Micro-architectural specification using SDL
FlexRay™ Node Architecture

- Single or dual channel
- CHI is responsible for registers, data buffers, interrupts, cmds, etc.
- PE provides OSI Layer 2 (Data Link) functionality
- Bus guardian optional
- Typically implemented as single chip (SoC) solution or dual-chip CPU + CC peripheral
Protocol Engine Architecture

POC = Protocol Operation Control
SYNC = Clk Sync and Macrotick Gen
MAC = Media Access Control
FSP = Frame and Symbol Processing
CODEC = Coding/Decoding
#define COLDSTART_LISTEN 7'b111_0011

event cover_trig_evt;

coverage_group poc_trig_cov {
    sample_event = sync(ALL, cover_trig_evt) async;
    sample poc.vState {
        state F07_11_003_A (COLDSTART_LISTEN) if (poc.header_received_on_A === 1);
        state F07_11_003_B (COLDSTART_LISTEN) if (poc.header_received_on_B === 1);
        state F07_11_005 (COLDSTART_LISTEN) if (poc.CHIRP_on_A === 1);
        state F07_11_011 (COLDSTART_LISTEN) if (poc.tStartup === 1);
    }
}

// called when any SDL trigger is active
coverObject(... obj) {
    trigger(cover_trig_evt);
}
State Transition Coverage

```plaintext
COLDSTART_CONSISTENCY_CHECK
vPOC!StartupState := COLDSTART_CONSISTENCY_CHECK

coldstart consistency
check
SyncCalcResult
(zSyncCalcResult, zStartupNodes, zRefNode)

vCycleCounter ?
Odd
else
zStartupNodes ?

vRemainingColdstartAttempts ?

zColdstartAborted := true;
vPOC!ColdstartNoise = zColdstartNoise;

COLDSTART_GAP

enter
coldstart gap

integration listen

} // called when a state change occurs
coverObject(... obj) {
    trigger(cover_state_evt);
}
```

**event** cover_state_evt;
**coverage_group** poc_state_cov {
    **sample_event** = sync(ALL, cover_state_evt) async;
    **sample** poc.vState {
        **state** CCC (7'b111_1010);
        **state** IL (7'b111_0101);
        **trans** F07_13_007 (“CCC” -> “IL”)
            if ((poc.zStartupNodes <= 0)
                && (poc.vCycleCounter[0] === 1));
        **trans** F07_13_009 (“CCC” -> “IL”)
            if ((poc.zSyncCalcResult !== WITHIN_BOUNDS)
                && (poc.zStartupNodes > 0)
                && (poc.vCycleCounter[0] === 1));
        **trans** F07_13_011 (“CCC” -> “IL”)
            if ((poc.vRemainingColdstartAttempts <= 0)
                && (poc.zStartupNodes === 0)
                && (poc.vCycleCounter[0] === 0));
    }
```
Vera Implementation

```plaintext
Mark Litterick

\[
\text{task} \ coverObject(...) \{
  \text{trigger}(cover\_evt);
\}
\]

coverage_group poc\_state\_cov {
  sample\_event = sync(ALL, cover\_evt) async;
  sample poc.vState {
    wildcard state WL (7'b010_xx01);
    trans DC\_C ("DC" -> "C");
    trans F02\_X ("NP" -> "H") if (poc.a == 1);
    trans POC1 ("DC" -> "C" -> "NP");
  }
}

if (change\_of\_state()) {
  update\_poc\_obj();
  poc\_cov.coverObject(poc\_obj);
}
update\_poc\_obj() {
  if (vera\_is\_bound(port.$sig))
    poc\_obj.member = port.$sig;
}

\[
deactivateBin("bin\_pattern");
// disable bins related to void binds
\]

if (!change_of_state()) {
  update_poc_obj();
  poc_cov.coverObject(poc_obj);
}
update_poc_obj() {
  if (vera_is_bound(port.$sig))
    poc_obj.member = port.$sig;
}

\[
\text{queryHit("bin\_pattern");}
// feedback for constrained random
// checking capability for directed tests
\]
```
Managing Coverage Info

- Total coverage
  - all tests, all configs

- Config coverage
  - all tests, one config

- Test coverage
  - one test, all configs

- Achieved through:
  - database namespace management
  - report merging
  - post processing
Problems and Limitations

• Real-world RTL implementation takes time to calculate & respond
  – SDL diagrams are time-independent: many process steps in zero time
  – coverage implementation more closely tied to RTL than intended

• Inaccessibility of some variables results in missed coverage
  – only a few variables could not be accessed in RTL implementation since stored in RAM (~ 2% coverage)

• Very difficult to reach some protocol corner cases with constrained random stimulus
  – many directed tests were required

• Evolution of FlexRay™ protocol specification and SDL
  – maintenance of spec tags could become an issue
  – auto-generation of coverage statements from SDL feasible, but outside the scope of this project
Conclusion

- Pragmatic solution to internal state coverage for Protocol Operation Controller for FlexRay™ Communication Controller
- Demonstrated the role of internal state coverage within the overall verification environment
- Functional coverage proved invaluable in measuring effectiveness of constrained-random and directed tests for regression suite
  - identifying missing tests and coverage holes
  - steering constrained-random tests towards coverage targets
  - highlighted testbench infrastructure requirements to enable directed tests
- Vera implementation and code provided are scaleable and usable for a number of similar applications

mark.litterick@verilab.com