Using Automation to Close the Loop Between Functional Requirements and Their Verification

Brian Craw – Cypress Semiconductor
David Crutchfield – Cypress Semiconductor
Martin Oberkoenig - Cypress Semiconductor
Markus Heigl - Cypress Semiconductor
Martin O’Keeffe - Cypress Semiconductor
Agenda

• The problems with manual requirements tracking

• The Flow and Tools
  – Requirements Management Tool (RM Tool)
  – Coverage Code Generator (CovGen)
  – Verification Management System (VMS)

• Conclusions
Manual Processes are Error Prone

- Tracking spreadsheets are overly complex and therefore under-utilized

- Possible misses on requirement changes

- Incomplete coverage metric to requirement mapping can be hard to detect
ISO 26262 Compliance

• Quality standard required for Automotive products

• Requires a high level of traceability throughout the development process to guarantee top-level safety requirements have been implemented and verified down to the lowest level

• Difficult to fully achieve without the use of a Requirements Management tool and some level of automation.
The Flow

RM Tool

- Higher Level Requirements
  - "Derived from" relationship
- Verification Requirements
  - "Verified by" relationship
- Test Items
- Coverage and Coverage-Intent Items

CovGen.xls

CovGen

- Coverage report (.xml)
- Coverage Database(s)

Simulator

VMS

- DUT RTL, test bench, and Tests
- SV coverage code (.sv[h])
• Features
  – Usable across business disciplines
    • Marketing, Design, Verification, etc.
  • Every group uses the same tool
  • Want to avoid exporting/importing of data from/to different tools
Requirements Management Tool

• Features
  – Usable across geographies
  • Centralized repository of data
  • No nightly syncing of data
Requirements Management Tool

• Features

  – Customizable
    • Can create custom items and workflows to track more than just requirements such as functional coverage state and test status

  • Embedded scripting for custom report generation

  • Available API for advanced extensions
Requirements Management Tool

- The Requirements
  - High Level Requirements
    - Customer, Marketing, Architecture
  - Verification Requirements
    - Derived from higher levels
Requirements Management Tool

• Test Items
  – Describe functional tests
  – Have a “Verify” relationship to Verification Requirements
  – Can be used to create a Test Plan
Requirements Management Tool

• Coverage and Coverage-Intent Items
  – Describe SystemVerilog coverage constructs: Coverpoints, Crosses, and Assertions
  – Have a “Verify” relationship to Verification Requirements
  – These items are updated with regression results via a custom plug-in
Requirements Management Tool

• How are Requirements Verified?

Test Items  | Verification Requirements  | Coverage/Coverage-Intent Items
---|---|---
Test 1  | Req. 1  | Coverage Item 1
Test 2  | Req. 2  | Assertion Item 1
Test 3  | Req. 3  | Coverage Item 2
...  | ...  | ...
Test T  | Req. R-2  | Coverage Item C-3
   | Req. R-1  | Coverage Item C-2
   | Req. R  | Coverage Item C-1
R  | Coverage Item C

VERIFY requirements with a PASS
Are VERIFIED by test items

VERIFY requirements when 100% or “hit”
Are VERIFIED by coverage items
Coverage Generator (CovGen)

- In-house tool to generate SystemVerilog code from a coverage intent specification
  - Reads in an XLS file and outputs SV code
  - Custom plug-in for RM tool written to export Coverage-Intent items into XLS format
  - Each Coverage-Intent item corresponds to a line in the XLS and represents a Coverpoint or Cross
  - Special keywords are used to describe various types of bins
Coverage Generator (CovGen)

• Benefits

  – Ensures correct SV syntax

  – Maintains naming conventions
    • Important that the names match between RM Tool and test bench so back-annotation is successful

  – Easier to review intent in a spreadsheet format or in the RM Tool than the SystemVerilog code itself
<table>
<thead>
<tr>
<th>TYPE</th>
<th>CATEGORY</th>
<th>REGISTER</th>
<th>FIELD</th>
<th>BIT FIELD QTY</th>
<th>NAME</th>
<th>WGT</th>
<th>BINS</th>
<th>SAMPLE EVENT</th>
<th>CP1</th>
<th>CP2</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>CP</td>
<td>Var</td>
<td>ENABLE</td>
<td>1</td>
<td>2</td>
<td>cp_enable</td>
<td>1</td>
<td>AUTO</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CP</td>
<td>TYPE=CHAN_MODE_T</td>
<td>MODE</td>
<td>MODE</td>
<td>7:0</td>
<td>1</td>
<td>cp_mode_mode</td>
<td>1</td>
<td>TRANS = SINGLE =&gt; DUAL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CP</td>
<td>Reg</td>
<td>CTL</td>
<td>CHAN1</td>
<td>2:0</td>
<td>1</td>
<td>cp_ctl Chan1</td>
<td>1</td>
<td>WALKONES</td>
<td>chan1_en</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CP</td>
<td>Reg</td>
<td>CTL</td>
<td>CHAN2</td>
<td>5:3</td>
<td>1</td>
<td>cp_ctl Chan2</td>
<td>1</td>
<td>MIN=3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>cs_cp_ctlChan1_cp_ctlChan2</td>
<td>1</td>
<td>IGNORE [CP1=MIN &amp; CP2=7]</td>
<td>cp_ctl Chan1</td>
<td>cp_ctl Chan2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example Coverage Generator Output

Covergroup

covergroup SAMPLE_cg;
  type_option.weight = 1;
  type_option.comment = "CovGen example";
  .
  .
endgroup : SAMPLE_cg;

Coverpoint with walking one bins

cp_ctl_chan1: coverpoint bit_3_t'(ctl.peek("CHAN1")) iff (chan1_en) {
  type_option.weight = 1;
  bins bin.walkone_0 = {3'b001};
  bins bin.walkone_1 = {3'b010};
  bins bin.walkone_2 = {3'b100};
}

Coverpoint with transition bin

cp_mode_mode: coverpoint CHAN_MODE_T'(mode.peek("MODE")) iff (sample_cp_mode_mode) {
  // Captures the transition from single channel to dual channel mode
  type_option.weight = 1;
  bins bin.single_fb.dual = (SINGLE => DUAL);
}

Cross with ignore_bins

cs_cp_ctl_chan1_cp_ctl_chan2: cross cp_ctl_chan1, cp_ctl_chan2 {
  type_option.weight = 1;
  ignore_bins bin.cp1_0.cp2_7 = (binsof(cp_ctl_chan1) intersect {0} &&
                             binsof(cp_ctl_chan2) intersect {7});
}

Coverpoint with auto generated bins

cp_enable: coverpoint ENABLE iff (sample_cp_enable) {
  type_option.weight = 2;
  bins bin.0 = {0};
  bins bin.1 = {1};
}
Verification Management System

RM Tool

- Higher Level Requirements
- Verification Requirements

Test Items

Coverage and Coverage-Intent Items

Coverage Report (.xml)

CovGen.xls

CovGen

- DUT RTL, test bench, and Tests
- SV coverage code (.sv[h])

VMS

Coverage Database(s)

Simulator

Custom plug-in

“Derived from” relationship

“Verified by” relationship
Verification Management System

• VMS – Verification Management System
• Established a standard approach to:
  – Design and test bench organization
  – Regression management
  – Regression status / coverage collection

INPUT

- dut.files, tb.files, vms.cfg, test-list, command-line, pre/post scripts

OUTPUT

- UCDB, HTML, XML Reports, logs, email, waves, debug, test-lists
Verification Management System

- VMS will produce coverage databases for all simulations and merge them into a single file.

- Vendor coverage database access utilities are used to generate an XML report containing all coverage metrics.
Coverage Results Import

- Custom plug-in written to import coverage metrics report
- Coverage Intent Items are updated with the coverage results.
  - i.e. percent bin hits, assertion fires
- Test Items are updated with test results.
  - i.e. pass/fail
Conclusions

• The automation and traceability provided by this flow allow us to efficiently achieve the quality demanded by both industry and internal standards

• Current State: Development nearly complete. Pilot projects due to start soon

• Future Work: Plan to broaden the flow to benefit other aspects of development such as mixed-signal IP development and verification, validation, document generation, etc