Coverage based verification for software testing

TVS first started development of the C++ based Functional coverage library when a client BluWireless first approached TVS with a requirement to verify their SystemC based hardware design. As the library was developed it became apparent that this same library which is C++ based could also be used to test software by applying the same concepts of Constraint driven random verification and coverage.

1 Infrastructure development

1.1 Infrastructure for Constraint Driven Random Verification.

Constraint based random verification is enabled through use of external randomization library called CRAVE. The syntax of CRAVE has been designed to naturally fit with C++ and SystemC.

- The CRAVE library allows the users to set constraints in a manner similar to UVM & System Verilog.
- CRAVE allows inline constraints, which can be formulated and changed incrementally at run-time.
- CRAVE is freely available under the MIT license. CRAVE is available via public repositories at GitHub.

In addition other modes of randomization including the C++ inbuilt randomization can be used. If this approach is used then there would be no dependency on the System C library which CRAVE needs. On the other hand C++ does not have inbuilt constraint handling system so constraint setting and over-riding would be an issue faced by the user.

1.2 Coverage through TVS Functional Coverage Library & asureSIGN

TVS has developed a Functional Coverage Library(FCL) to enable Coverage Driven Verification(CDV). This library can be used either with TVM or individually on C++ based environments. The main features of the FCL are

- The functional coverage report is generated in both CSV & XML format (using the "Tiny xml" library add-on). This format is compatible with the TVS asureSign so that coverage can be viewed against requirements.
- The FCL supports: multiple cover groups in a single instance; conditional coverage; exclusion or Inclusion of cross bins (by specific cover bin scope or bin specific); transition Bins (one series of incrementing transition); auto bins (up to 16 bits); cross coverage (for up to 4 cover points).

Once the regression suite of tests has been run with Function Coverage enabled, the results will have the necessary data on the functional coverage. This data can be dumped from FCL into CSV or directly in XML. XML is one of the formats that the asureSIGN tool uses to load the test data

Once asureSIGN is set up with the feature extraction, verification goals and test bench elements, and the mappings between them it can be used to track functional coverage. A functional coverage goal can be mapped to the functional coverage points that implement them.

As tests are run they automatically report their results to asureSIGN via an API and the code and functional coverage results are collected and automatically fed to asureSIGN via XML (as mentioned above). Thus as each regression is run asureSIGN automatically records the results (and the version control tag) so that progress can be monitored.

1.3 Code Coverage

Code coverage allows the user to analyze which portions of the code in the program are executed and how many times.

The Gcov utility provides information on how many times a line of code in a program is covered (i.e code coverage). This utility also has the capability to do branch coverage i.e the utility examines the various branch conditions e.g. conditional statements lie if, case etc. The Gcov utility creates *.gcov files which log the information on code coverage. Gcov comes as a standard utility with the GNU Compiler Collection (GCC) suite.

LCOV is the GUI for viewing the results generated by the Gcov tool. It collects gcov data for multiple source files and creates HTML pages containing the source code annotated with coverage information.

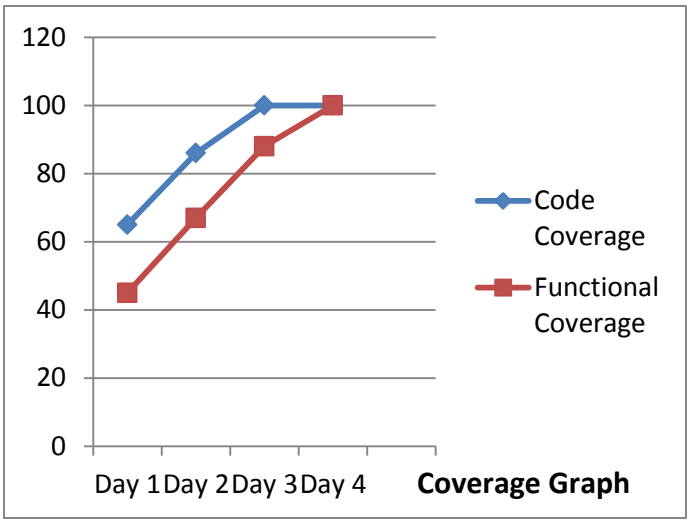
2 Advantages

This approach of using hardware concepts like Functional or Feature driven Coverage for software testing provides the following advantages:

- The Architect can mention Features which need to be covered. These features can then be tracked for implementation and testing.
- Code Coverage ensures that all code in the DUT is covered.
- Management gets a better idea on the current status of the software as a ratio to tested & implemented feature Vs non tested features.
- Testing can be set up such that this coverage can be automatically loaded after every set of regressions to give current status.
- Testing can be done in a more stringent & random way. This can address any corner cases arising from regular tests not covering certain features. This also ensures that the tester does not miss out any testcase set or features.

3 Case Study

We used a bubble sort program since this is easy for most programmers to understand. This Bubble sort program can take different types of data as input. All elements can be integer, short, long, char or string. The sorting can be done in ascending or descending order.



We added the functional coverage to the code. This functional coverage had the following cover points (partial list):

- Direction of Sort.
- Type of data: int, long, char
- Type of data to be sorted in Ascending or Descending order.
- Input data already sorted in Ascending or Descending order.

Code Coverage was also carried out.

The Bubble sort Functional Coverage when viewed in asureSIGN can be seen below: