Code-Test-Verify all for free – Assertions + Verilator

Hemamalini Sundaram, Tech lead
Kasthuri Srinivas, Sr. Verification Engineer
Supriya Ummadisetty, Sr. Verification Engineer
VerifWorks Pvt Ltd

***Abstract*- Protocols such as APB and AHB are widely used in System-on-Chip (SoC) designs. SystemVerilog Assertions (SVA) are excellent candidates to capture these protocol requirements in concise, unambiguous, executable b specification. Though these are freely available standards, lack of opensource tools supporting concurrent assertions has been a significant hindrance to widespread use of SVA and checkers in general. Another challenge is the significant run-times associated with debugging of such complex assertions especially when simulated with large SoC designs. In this work, we present a verification methodology for the AMBA APB /AHB protocol checkers using SystemVerilog Assertions (SVA) developed completely using open-source simulator – Verilator. We also present Unit testing as the way of testing each assertion in isolation for correctness. The entire code base will be made available online for free**

I. INTRODUCTION

Simulation-based design verification is a familiar method of functional verification. For this we need good and efficient simulators. Simulators are software programs which mimic the DUT behavior and will run the verification code. Though the simulation helps in the design analysis without risk, refinement of the design, enhancing the accuracy, etc., cost of which is very high. The primary motivation for this paper is to enable the industries to explore open-source simulators.

In this paper, *Verilator*- a super-fast open-source simulator is used along with advanced verification technologies such as Assertion based Verification, Unit Testing, Debug automation etc. *SVunit* is also an open-source test framework which supports the Verilog/SystemVerilog code development and UVM Verification Testbenches. SVUnit is also used to write Unit Test Cases to verify Pass/Fail scenarios. One more open-source tool, *SV waveterm* is used to debug the waves. The major advantage of this waveform debugging tool is that it provides the waveform in the logfile which will help us save the debug time with a stand-alone waveform viewer.

One good application of this work can be the academia that can now learn SVA for *free using opensource tools*. Other applications would be for teams to use Verilator as a development platform, especially for RTL designers.

II. IMPLEMENTATION APPROACH

The first challenge was to modify existing CIP code to work with the limited support of SVA features in Verilator. Second was to port SVUnit framework (A SystemVerilog class-based Unit test framework) to compile in Verilator.

Once basic SVA is running in Verilator, we describe the challenge of "correctness" of each assertion by developing Unit tests for PASS and FAIL scenarios for each property.

Test cases were developed in SVunit to verify the SVA that is used in checker IP. These test cases are developed for both PASS and FAIL conditions of SVA.

III. CHECKER IP (CIP) IN VERILATOR

Checker IP is an IP that captures properties of a given protocol in the form of SystemVerilog asserts, covers and assumes [T1, R1]. Motivation behind creating a CIP is to focus on the expected outcome of the DUT in terms of expected behaviors, unexpected protocol violations, desired scenarios etc. In a CIP, we separate the "does-it-work", "are-we-done" queries from the means of doing it (either simulation or formal).

For standard protocol, such as ARM's AMBA family [T1, R2], it is common to define a set of compliance checks [T1, R3] from the specification itself. Given the number of systems being built around standard buses, it is imperative for the industry to be able to leverage on a standard CIP that checks for a list of well-defined compliance rules. However, a set of properties and asserts around them is not a reusable piece of CIP – they are simply a collection of properties. There are several techniques and guidelines in making such a list of properties into an IP (CIP). To differentiate from traditional VIP, we use the term Checker IP - CIP. CIP, strictly speaking, is a sub-set of standard VIP. This CIP is designed such that it is Verilator compatible. The table below shows the assertion checks that is developed for APB.

Assertion	Description
a_p_af_low_power	Low Power checks
a_p_af_rst_penable a_p_af_rst_pwrite a_p_af_rst_paddr a_p_af_rst_pwdata	Reset value checks
a_p_af_active_one_clk a_p_af_paddr a_p_af_pwdata a_p_af_pwrite a_p_af_setup_to_enable a_p_af_psel_pen1	Protocol compliance checks

TABLE I: ASSERTION CHECKS DEVELOPED FOR APB

IV. UNIT TEST FOR EACH ASSERTION USING SVUNIT TESTING FRAMEWORK

Assertions are "checkers" of your design, or the core of "verifier" – but who will verify *that* verifier? Given that a comprehensive CIP is a complex code, it requires thorough verification itself. We have developed a series of unit tests inside Go2UVM framework [F3] to tackle this problem which is explained further in this section.

In a nutshell, this involves creating Pass and Fail trace for each assertion with a simple UVM test. These unit tests should be smart enough to be self-checking. We have used a UVM report mocker from open-source SVUnit framework [6] to self-check each unit test around assertions.
Consider APB protocol requirement on signal *psel* and *penable*, as shown in Figure –

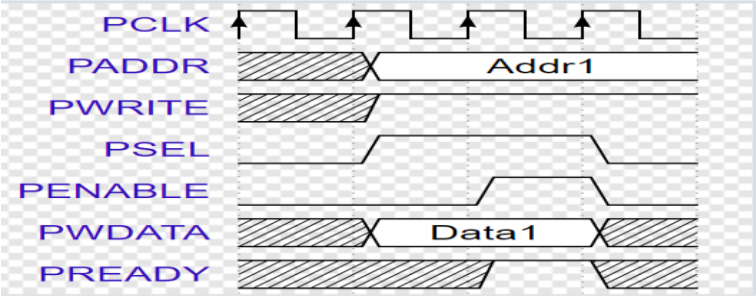

Figure 1. APB PSEL AND PENABLE REQUIREMENT

The scenario here is whenever psel is high, in the following clock cycle penable should be high. The traditional SVA can be coded as seen in the figure (F2) below

```
//property : if psel is high, pen is high in the next clock cycle

a _p_af_psel_pen :assert property
    (disable iff(!preset_n))
        psel && !pen |=>  pen;
    else
        `AF_CIP_ERROR(a_p_af_psel_pen, pen is not HIGH after psel is high)
```

Figure 2. TRADITIONAL SVA CODE FOR ONE OF THE CHECKS

.

Unit test for the above assertion using SVunit along with GO2UVM framework macros is coded as follows

```
//  Desc:
//      Unit test for CIP_ID: a_p_af_psel_pen
//      psel = 1, next clock penable = 0
//      Expected result: FAIL
//*****************************************
***
`SVTEST(fail_a_p_af_psel_pen)

    `g2u_display ("fail_a_p_af_psel_pen")
    idle ();
    `g2u_display ("Driving psel to 1")
    psel = 1'b1;
    wait_for_n_clks (1);
    `g2u_display ("Driving pen to 0")
    penable = 1'b0;
    wait_for_n_clks (1);
    penable = 1'b0;
    idle ();
    wait_for_n_clks (10);
    `FAIL_IF_LOG(1, "Expected FAIL a_p_af_pse
    `g2u_display ("End fail_a_p_af_psel_pen")

`SVTEST_END
```

Figure 3. ASSERTION CHECK FOR psel and penable using SVUNIT.

V. RESULTS

The following screen shots show the results that we have obtained from executing the SVA with cip_enabled Verilator simulator, SVunit test framework and SVwaveterm as a waveform debugging tool which are all open source. Figure 4 and 5 Shows the results of different assertion checks that have been coded for APB scenarios.
Figure 6 shows the waveform in SV Waveterm. We can see how the waveform is visible in the log file for an easy and time saving debug.

Figure 4. PASS / FAIL RESULTS OF THE ASSERTIONS CHECKS

```
UVM_INFO ../src/APB_CIP_src//test_a_p_af_pwrite_stable_bw_setup_and_en.sv(53) @ 2750.000 ns [IVL_GO2UVM] Driving pwrite to
UVM_INFO ../src/APB_CIP_src//af_apb_cip.sv(45) @ 2755.000 ns [IVL_GO2UVM] psel: 1 pen: 1
UVM_INFO ../src/APB_CIP_src//test_a_p_af_pwrite_stable_bw_setup_and_en.sv(56) @ 2760.000 ns [IVL_GO2UVM] Driving pwrite to
UVM_ERROR ../src/APB_CIP_src//af_apb_cip.sv(81) @ 2765.000 ns [a_p_af_idle_pwrite_stable]  pwrite is not stable when APB is
UVM_ERROR ../src/APB_CIP_src//af_apb_cip.sv(155) @ 2765.000 ns [a_p_af_pwrite_stable_bw_setup_and_en] pwrite is not stable
ERROR: [2790.000 ns][apb_slave_ut]: fail_if: 1 [ Expected FAIL a_p_af_pwrite_stable_bw_setup_and_en ] (at ../src/APB_CIP_sr
and_en.sv line:61)
UVM_INFO ../src/APB_CIP_src//test_a_p_af_pwrite_stable_bw_setup_and_en.sv(62) @ 2790.000 ns [IVL_GO2UVM] End fail_a_p_af_pw
INFO:  [2790.000 ns][apb_slave_ut]: fail_a_p_af_pwrite_stable_bw_setup_and_en::FAILED
INFO:  [2790.000 ns][apb_slave_ut]: pass_a_p_af_rst_paddr::RUNNING
UVM_INFO ../src/APB_CIP_src//test_a_p_af_rst_paddr.sv(12) @ 2885.000 ns [IVL_GO2UVM] pass_a_p_af_rst_paddr
UVM_INFO ../src/APB_CIP_src//test_a_p_af_rst_paddr.sv(13) @ 2885.000 ns [IVL_GO2UVM] Driving rst_n to 0
UVM_INFO ../src/APB_CIP_src//test_a_p_af_rst_paddr.sv(15) @ 2885.000 ns [IVL_GO2UVM] Driving paddr to 0
UVM_INFO ../src/APB_CIP_src//test_a_p_af_rst_paddr.sv(20) @ 3010.000 ns [IVL_GO2UVM] End pass_a_p_af_rst_paddr
INFO:  [3010.000 ns][apb_slave_ut]: pass_a_p_af_rst_paddr::PASSED
INFO:  [3010.000 ns][apb_slave_ut]: fail_a_p_af_rst_paddr::RUNNING
UVM_INFO ../src/APB_CIP_src//test_a_p_af_rst_paddr.sv(35) @ 3105.000 ns [IVL_GO2UVM] fail_a_p_af_rst_paddr
UVM_INFO ../src/APB_CIP_src//test_a_p_af_rst_paddr.sv(37) @ 3120.000 ns [IVL_GO2UVM] Driving rst_n to 0
UVM_INFO ../src/APB_CIP_src//test_a_p_af_rst_paddr.sv(39) @ 3120.000 ns [IVL_GO2UVM] Driving paddr to 1
UVM_ERROR ../src/APB_CIP_src//af_apb_cip.sv(121) @ 3125.000 ns [p_af_output_at_reset] paddr is not low at reset
ERROR: [3250.000 ns][apb_slave_ut]: fail_if: 1 [ Expected FAIL a_p_af_rst_paddr ] (at ../src/APB_CIP_src//test_a_p_af_rst_p
UVM_INFO ../src/APB_CIP_src//test_a_p_af_rst_paddr.sv(45) @ 3250.000 ns [IVL_GO2UVM] End fail_a_p_af_rst_paddr
INFO:  [3250.000 ns][apb_slave_ut]: fail_a_p_af_rst_paddr::FAILED
INFO:  [3250.000 ns][apb_slave_ut]: FAILED (8 of 16 tests passing)

INFO:  [3250.000 ns][__ts]: FAILED (0 of 1 testcases passing)
```

Figure 5. LOG FILE OF THE RESULTS

PeterMonsson / sv_waveterm Public

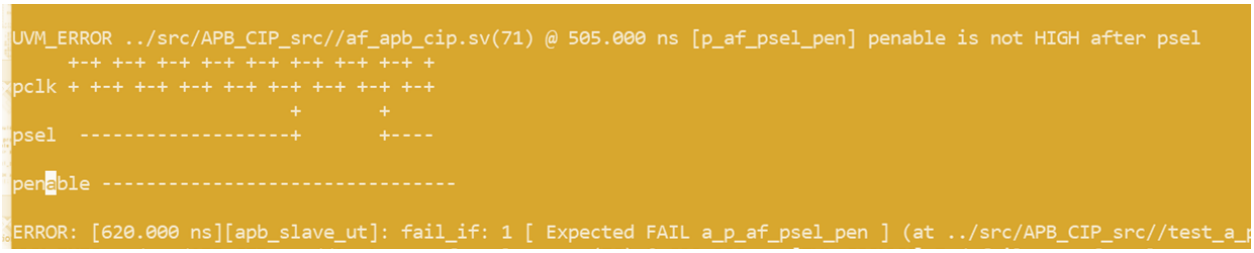

Figure 6. SV WAVETERM – WAVEFORMS IN THE LOG FILE

VI Summary

Design Verification with SystemVerilog Assertions has been popular in the industry for well over a decade. While simple checkers can be developed quickly and used across design entities, a comprehensive CIP (Checker IP) uses a good architecture and set of coding guidelines to keep them reusable. In this paper, we have shared our experience of converting a plain set of properties to a reusable CIP and integrating into Verilator. We also shared how we used a self-checking unit test framework to verify each assertion in a CIP.

REFERENCES

[1] SystemVerilog LRM - http://standards.ieee.org/getieee/1800/download/1800-2012.pdf
[2] ARM AXI specification – https://www.arm.com/products/system-ip/amba-specifications
[3] ARM releases assertion models - https://www.arm.com/about/newsroom/12266.php
[4] Experiencing Checkers for a Cache Controller Design
http://systemverilog.us/DvCon2010/DvCon10_Checkers_paper.pdf
[5] Accellera Open Verification Library (OVL) http://accellera.org/activities/working-groups/ovl
[6] SystemVerilog Assertions handbook, www.systemverilog.us, www.verifnews.org/publications/book
[7] "What are $past compared to on first clock event?" http://bit.ly/2hkb7nV
[8] Go2UVM open-source test layer, www.go2uvm.org.
[9] SVUnit - http://www.agilesoc.com/open-source-projects/svunit/