With ever increasing design complexities, ASIC and SoC (system on chip) design verification has become the biggest challenge for design and verification engineers. Various Hardware Description Languages (HDLs), like Verilog, VHDL, and SystemVerilog, and verification methodologies, like UVM and OVM, have facilitated the task of verifying designs. Despite this, bugs are still missed in the verification/validation phases, which eventually leads to a re-spin of the entire chip. Verification environments that assist with the right set of assertions and coverpoints not only increase the verification efficiency, but also aids the verification engineer to ensure that the functionality of the IP has been met according to design specifications. This article gives insight into how to capture assertions and coverpoints and how they should be written in order to achieve maximum design verification robustness.

INTRODUCTION

Assertions are a set of expressions which indicate an error if some set of conditions are violated. They are used to catch those errors which must not happen. They check a specific behavior and display a message if an error occurs. They monitor for bad behavior in the design under test (DUT), and can be used to generate a warning or an error message. Assertion-based verification (ABV) has been used in past decades. Finding the right set of assertions is critical to avoid any bug escapes in the final SoC and ASIC development chips. Cover points, on the other hand, give a measure of the completeness of the design verification and its progress. Coverage metrics tell us what portion of the design has been activated during simulation. They identify portions of the design that were never activated during the process of simulation, which allows us to adjust and control our input stimulus. Hence, in this article, we cover how to find appropriate assertions and coverpoints by reading the design specification. We have used the Serial ATA (SATA) storage protocol design specification as the reference.

ASSERTION-BASED VERIFICATION

ABV has been successfully applied at multiple levels of design and verification abstraction, ranging from high-level assertions within transaction-level test benches down to implementation-level assertions synthesized into emulation and hardware. They can be grouped into three categories.

Simple assertions: These capture precise lines from the specification. This set signifies those design functionalities that should not be infringed and should be fired immediately in case of any violation. The rules that are stated as “shall be/must be” in the specification should be captured as part of simple assertions.

Advanced assertions: These are interpreted from the reading of the specification. They capture the higher abstraction level of the design that is to be verified. These can also be interpreted as logical assertions.

Harmless assertions: These result in a warning rather than an error. The specification states that the design functionality is not affected even if some of the conditions are violated. Hence, these violations are given as a warning to the user. The rules that are stated as “can be/may be” in the specification should be captured as part of this set.

It is imperative that the verification engineer who is designing and configuring the verification architecture captures all three types of assertions. These sets are required to target different domains. It is in the control of the user which set should be enabled while running the simulation. Users verifying the core IP design can enable all three sets. Users with IP designs of a higher abstraction level can enable only simple and advanced sets; whereas FPGA customers enable only the simple set targeting an exact property. Care should be taken that every design specification is captured, as most of the debugging of a design can be done by capturing the right set of assertions.

Some assertions are not captured while reading the specification. These assertions are then added and planned in the next phase when specification maturity gets developed; for example, after completing the first phase of verification.

ASSERTIONS NAMING AND DESCRIPTION

Writing assertions involves a name along with the description and the section of the specification from where it has been written. The right naming and description is important since it aids in debugging the design efficiently. The name should be as precise as possible and should also convey the design functionality that has been violated. It is advised to use appropriate abbreviations while naming.
This keeps the name short and at the same time conveys its meaning. The name of the assertion can start with the name of the protocol, followed by the layer name (if it is a layered protocol), then followed by the erroneous condition.

The description should match the exact line of the specification along with a brief message for the user. The message should print the actual scenario that has occurred in comparison with the benchmark scenario that is expected in the appropriate format. The section of the specification should be specified so that the user can refer directly to that part of the design specification in case of any mismatch.

The following examples show how the three sets of assertions should be captured and written.

**Simple Assertions**

**Assertion 1**

```
7.7.1.3 COMRESET
COMRESET always originates from the host controller and forces a hardware reset in the device. It is indicated by transmitting bursts of data separated by an idle bus condition.
```

*Assertion name:* SATA_PL_INVLD_COMRST

*Assertion description:* The OOB COMRESET signal shall consist of no less than six data bursts, including inter-burst temporal spacing.

*Assertion section:* 7.7.1.3

In this assertion, the validity of the COMRESET signal is checked. It shall consist of no less than six data bursts. If the COMRESET signal with data burst less than six is received, this assertion will fire. The assertion description informs the user of the actual numbers of bursts received.

**Assertion 2**

```
Figure 1. Serial ATA specification snippet from physical layer (PL) section 7.7.1.3
```

*Assertion name:* SATA_LL_CONT_BYT0_COM

*Assertion description:* CONT primitive detected with byte 0 as COMMA character. ALIGN represents the only primitive that contains COMMA character.

*Assertion section:* 9.5.1

In this assertion, it is specified that only the ALIGN primitive contains the COMMA character. If any other primitive is received with byte 0 as the COMMA character, then the assertion is fired, as this is a violation.

**Advanced Assertions**

**Assertion 1**

```
7.7.1.3 COMRESET
COMRESET always originates from the host controller and forces a hardware reset in the device. It is indicated by transmitting bursts of data separated by an idle bus condition.
```

*Assertion name:* SATA_PL_ALIGN_H2D_BEF_873_US

*Assertion description:* The host should allow for at least 873.8 μs (32 768 nominal Gen1 Dword times) after detecting the release of COMWAKE to receive the first ALIGNP from the device. Hence the host should not start transmitting ALIGNP before this interval.

*Assertion section:* 7.7.1.3

This assertion is an advanced assertion since it is interpreted from the specification. In SATA protocol, the host cannot send the ALIGN primitive before the device sends it. The host should wait for at least 873.8 μs to receive the first ALIGN from the device. It cannot send ALIGN during that interval; hence this condition should be asserted if violated.

**Figure 3. Serial ATA specification snippet from physical layer (PL) section 7.7.1.3**

*Assertion name:* SATA_LL_ALIGN_H2D_BEF_873_US

*Assertion description:* The host should allow for at least 873.8 μs (32 768 nominal Gen1 Dword times) after detecting the release of COMWAKE to receive the first ALIGNP from the device. Hence the host should not start transmitting ALIGNP before this interval.

*Assertion section:* 7.7.1.3

This assertion is an advanced assertion since it is interpreted from the specification. In SATA protocol, the host cannot send the ALIGN primitive before the device sends it. The host should wait for at least 873.8 μs to receive the first ALIGN from the device. It cannot send ALIGN during that interval; hence this condition should be asserted if violated.
Assertion 2

Figure 4. Serial ATA specification snippet from transport layer (TL) section 10.5.9.1

Assertion name: SATA_TL_DMA_STUP_FIS_D_1_WHN_A_1
Assertion description: D bit detected as 1 when A bit is set to 1 in DMA Setup FIS from device to host. D bit should be 0; i.e. data is transmitted from host to device when A is set to 1 in DMA Setup FIS.
Assertion section: 10.5.9.1

This assertion checks the value of D bit in DMA Setup FIS when A bit is set to 1. If D bit is 1, then there is violation of the specification and the assertion should be fired. This is an advanced assertion since the condition is not explicitly specified. It is interpreted from the specification.

Assertion 3

Figure 5. Serial ATA specification snippet from physical layer (PL) section 8.4.2 (Fig 258)

Assertion name: SATA_PL_DEV_CONT_DR_COMINIT
Assertion description: Device is continuously sending COMINIT when host is in HR_AwaitNoCOMINIT state. Possibly it is stuck in DR_COMINIT state.
Assertion section: 8.4.2

This assertion verifies a complex scenario where the device is stuck in a state and is continuously sending the COMINIT signal. It is important to have a set of assertions related to state transitions of the DUT to check whether the DUT is hung up or stuck in a particular state.

Assertion 4

Figure 6. Serial ATA specification snippet from physical layer (PL) section 8.4.3 (Fig 259)

Assertion name: SATA_PL_HOST_CONT_COMRESET
Assertion description: Host is continuously sending COMRESET when device is in DR_Reset state. Possibly it is stuck in HR_Reset state.
Assertion section: 8.4.3 (Fig 259)

This assertion verifies the host where the host is stuck in a reset state and is continuously sending the COMRESET signal to the device which is keeping the device in the reset state. These kinds of situations are erroneous and should be asserted.

Harmless Assertions

Assertion 1

Figure 7. Serial ATA specification snippet from application layer (AL) section 13.6.4.3.1

Assertion name: SATA_AL_RD_FPDMA_QUEUE_REG_D2H_FIS_STS_BIT4_NT_1
Assertion description: When Host transmits READ FPDMA QUEUE command, the Register Device to Host FIS sent by the device with BSY and DRQ set to 0, and error field specifying the type of error, Status Bit 4 may be 1.
Assertion section: 13.6.4.3.1

In this assertion, Bit 4 in the FIS may be set to 1. Hence this is not the condition that should be strictly met and hence its violation will not lead to an error.
COVERAGE BASED VERIFICATION

Coverage based verification ensures that every design aspect has been verified. An input test vector relies on the functionality that has to be verified. Hence, functional coverage minimizes the effort and gives an idea whether the test vector has achieved 100% coverage or whether the stimulus needs to be changed to achieve full coverage. Any functional coverage is characterized by covergroups, coverpoints and crosses. A covergroup defines a coverage model. A coverpoint describes a particular type of coverage event, how it will be counted, and one or more bins (coverage event counting mechanism) to organize the count. A cross defines a new event by combining two or more existing coverpoints.

Coverage can be divided into three levels on the basis of complexity.

**Basic coverage**: This covers those coverpoints and crosses that are required for a very basic level of verification. These scenarios are explicitly mentioned in the specification and can be easily captured. For example, if a packet has a bit which can take both values 0 and 1, then a coverage can be written with that bit as the coverpoint and values 0 and 1 as the user defined bins. Coverage including the state transitions of a design can also be categorized as basic coverage.

**Extensive coverage**: This includes various complex scenarios that are required for an extensive verification. It is not explicitly mentioned in the specification and depends on our interpretation. For example, any complex scenario that is associated with a state change or events associated with the transfer of particular types of packets.

**Erroneous coverage**: Any coverpoint or cross that covers any erroneous events. This gives us a measure of how many erroneous conditions have been encountered while running the simulation.

It should be kept in mind that if any design aspect has been covered under an assertion, then we should not write a coverage for that certain event. Functional coverage maps each functionality (that is specified to be tested in the test plan) to a coverpoint. When it is hit, the corresponding coverpoint is automatically updated. The coverage report can be generated and viewed in a simulator.

**Coverage naming and description**

A coverage plan incorporates the name of the covergroups, coverpoints, and cross and a brief description. The covergroup naming convention follows that of the naming of the assertions, starting with a protocol name followed by the name of the layer (if it is a layered protocol). The name of the coverpoint indicates the event, and the description gives a concise idea about the functionality of the event.

The following examples show how the three levels of coverage should be captured and written.

**Basic Coverage**

Coverage example 1

| D | Direction (D) bit, specifies whether subsequent data transferred after this FIS is from transmitter to receiver or from receiver to transmitter. If set to one the direction is transmitter to receiver. If cleared to zero, the direction is receiver to transmitter. |

**Coverage example 2**

| A | Auto-Activate (A) bit, if set to one, in response to a DMA Setup FIS with data transfer direction of Host to Device, causes the host to initiate transfer of the first Data FIS to the device after the DMA context for the transfer has been established. The device shall not transmit a DMA Activate FIS to trigger the transmission of the first Data FIS from the host. If cleared to zero, a DMA Activate FIS is required to trigger the transmission of the first Data FIS from the host if the data transfer direction is Host-to-Device. |

**Extensive coverage**

This is an imperative event and can be directly interpreted from the specification. It covers if the bit D in the DMA Setup FIS has assumed both the values; i.e., the data transfer has taken place in both the directions with host as the DUT.

Coverage example 2

| Covergroups: sata_tl_host_cvg |
| Coverpoint: dma_stup_bit_d |
| Description: Bit D can take both values 0 and 1; i.e., bidirectional flow of data. |

**Erroneous coverage**

Any coverpoint or cross that covers any erroneous events. This gives us a measure of how many erroneous conditions have been encountered while running the simulation.

**Coverage example 2**

| Covergroups: sata_tl_host_cvg |
| Coverpoint: dma_stup_dma_act_fis |
| Description: DMA Activate FIS has been transmitted or not; i.e., Bit A has been both 0 and 1. |
It is important to cover the value of Bit A in DMA Setup FIS that indicates two different scenarios, whether the DMA Activate FIS is transmitted or not, and it is required to capture both the scenarios. This is a basic coverage and can be easily captured from the reading of the specification.

**Extensive Coverage**

Coverage example 1

**Figure 10. Serial ATA specification snippet from transport layer (TL) section 10.5.5.1**

**Covergroups:** sata_tl_host_cvg  
**Coverpoint:** reg_h2d_fis_pm_port  
**Description:** Port Multiplier can connect up to 15 devices. This covers if all the devices connected to Port Multiplier have been accessed.

This is an extensive coverage which is hit when all the devices connected to the Port Multiplier have been accessed by the host.

Coverage example 2

**Figure 11. Serial ATA specification snippet from transport layer (TL) section 10.5.5.1**

**Covergroups:** sata_tl_host_cvg  
**Coverpoint:** dma_stup_dma_buff_offset  
**Description:** A DMA Buffer Offset can be zero or non-zero.

In SATA protocol, A DMA Buffer Offset with value of 0 indicates that the device does not support out of delivery configuration for NCQ commands. This coverage is not explicitly mentioned in the specification and depends on the understanding of the functioning of the protocol. It is essential to capture this kind of coverage as it verifies the complex functioning of the DUT.

**Erroneous Coverage**

Coverage example 1

**Figure 12. Serial ATA specification snippet from physical layer (PL) section 4.1.1.15**

**Covergroups:** sata_pl_dev_cvg  
**Coverpoint:** host_ill Disp  
**Description:** Covers if disparity violation has taken place in device.

In some cases, it is necessary to cover the erroneous scenarios. It is in the control of the user of the verification IP (VIP) to enable or disable this kind of coverage on the basis of the abstraction level. The above coverage captures an illegal event where the device receives data with a running disparity violation.

**SUMMARY**

Capturing the right set of assertions and coverage for all levels of complexity makes it easy to debug a design of any abstraction level. This process should be followed and practiced across all the protocols; such as PCIe®, AMBA® and Ethernet.

**REFERENCES**

[1] Serial ATA Revision 3.3 specification  
[2] Coverage Cookbook, Verification Academy  
27 Video Courses Available Covering
- SystemVerilog OOP
- Formal Verification
- Intelligent Testbench Automation
- Metrics in SoC Verification
- Verification Planning
- Introductory, Basic, and Advanced UVM
- Assertion-Based Verification
- FPGA Verification
- Testbench Acceleration
- PowerAware Verification
- Analog Mixed-Signal Verification

UVM and Coverage Online Methodology Cookbooks
Discussion Forum with more than 7500 topics
Verification Patterns Library

www.verificationacademy.com