Debug of SystemVerilog Assertions

Mahantesh Udikeri
Synapse Design

1 SystemVerilog Assertions

SystemVerilog Assertions (SVA) are a very critical and important part of the verification flow and the creation of the testbench. There are multiple steps in creating meaningful and successful assertions, as defined below.

1. Identify the protocol which is critical and which needs to be monitored
2. Define the protocol for writing the assertions
3. Write the assertions as per the definition
4. Make sure the assertions are correct and properly check the normal and error protocol situations

An important issue when writing assertions is to check they are giving the correct or expected behavior. During the checking or debugging of the assertions, we need to introduce errors into the protocol. Then we need to make sure the assertions are catching the protocol mismatch or error. Also we need to make sure that in the normal operation of the protocol the assertion is working properly and doesn't issue an error message or doesn't behave improperly.

Debugging the assertions is a very critical step, as we need more options to verify the assertions are behaving correctly. All the EDA companies have some options in their tools specifically to assist in the debug of assertions, and with most of these options, we need to use them in a more graphical based method. However, most of the experienced verification engineers prefer to use the command line and message log files to debug the assertion issues. They will also check the waveforms in combination with the log files to debug any issues.

Let's look at one example to explore assertions and the way we can debug them.

1.1 Identify

This is specific to the chip and the protocols associated with the chip.

1.2 Defining the Assertions

Once the assertion has been identified, an example could be defined as below.

"The sig_2 should fall within 75 clock cycles after the sig_1 is low for 125us and without the sig_3 or rst_n is asserted".

This is the definition of the protocol that the chip has to follow. If it doesn't follow the above protocol, the assertion should display an error message.

1.3 WRITING THE ASSERTIONS

As per the identification of the assertions and definitions, we will write the assertion. Let's write the initial version as shown below.

```
// Here we are defining the property as per the protocol requirement.
property sig_2_should_fall_within_75clks_after_stable_sig_1_for_125us_p;
   @(posedge sys_clk)
      disable iff(!rst_n || !sig_3)
         $stable(sig_1) [*5000] ##1 $rose(sig_1) |-> ##[0:150] $fell(sig_2) ;
endproperty

// We are asserting the property to meet the protocol requirement
// if it fails to meet the requirement, it will display an error message
sig_2_should_fall_within_75clks_after_stable_sig_1_for_125us_a : assert
property(sig_2_should_fall_within_75clks_after_stable_sig_1_for_125us_p)
      $display("@%0d The sig_2 fell within 75 clock cycles after the sig_1 is low for around
125us",$time);
    else $error("@%0d The sig_2 didn't fall within 75 clock cycles after the sig_1 is low for
around 125us",$time);
```

Once we write the assertion, we need to make sure that it is behaving properly and we need to check that it is working correctly during both normal protocol operation and also during the error operation.

The key issue in debugging the assertion is to see that it is behaving properly by using a step-by-step process.

1.4 CHECKING OR DEBUGGING THE ASSERTION

To debug the assertion we need to verify its exact behavior for each step. We can do this with waveform checking for the signals associated with the assertion.

When we try to debug based just on waveforms, it is very cumbersome, as we want to know when each step is met based on time. If the assertion is not behaving properly, then to find the cause using just waveforms to determine where the issue has happened will take much longer.

To debug and find the issues with assertions, it is helpful to have an option to display a message with the time slot for each time the assertions is initiated, and also display a message in each step of the assertion. With this method, it is helpful to collect all the messages in one place and follow them closely in the log files, in addition to also using the waveforms for debug. This will give better clues for why the assertion has failed.

In the next example I am adding a display message for each step of the assertion. By adding these display messages it will be much easier to debug the assertion.

```
property sig_2_should_fall_within_75clks_after_stable_sig_1_for_125us_p;
  @(posedge sys_clk)
    disable iff(!rst_n || !sig_3)
      //Step 1
      $stable(sig_1) [*5000] ##1
      //Step 2
        ($rose(sig_1), $display("@%0d DEBUG_ASSERT sig_1 is zero for 125us with sig_3
      low",$time))
              |-> ##[0:150]
      //Step 3
                ($fell(sig_2), $display("@%0d DEBUG_ASSERT The sig_2 fell within 75 clock
      cycles",$time)) ;
```

```
endproperty
```

The display messages in steps 2 and 3 above will be very helpful when executed with a waveform during the assertion debug process.

It is also helpful when a keyword like DEBUG_ASSERT is added in the display message, or when we dump the display messages directly to a file to review them later. As all the assertion related messages and their associated time will be in one place, we can identify the issues in the assertion if it is failing. If the assertion is passing we can also make sure it is checking for the correct behavior or that the assertion thread was started at the proper time.

1.5 SUMMARY

I have found that following the steps above streamlines the process of checking and debugging assertions in my testbenches, and I believe others will find this to be the case also.

1.6 FOLLOW-ON DISCUSSION

For some protocols we need to wait for long durations of time for a set of signals to transition to certain values, and only then can we define the next step of the assertion. These type of assertions are very critical as they will take considerable compute resources and slow down the simulations. This will adversely affect the overall time needed in the verification schedule.

Please follow my next whitepaper on how to improve the performance of such assertions.