Verification Plan

Ethan Sarif-Kattan
LH Lee

December 18, 2020

Imperial College London

Contents

1 Introduction **3**

2 UART TX **3**

3 UART RX **4**

4 FIFO **4**

5 Parity Gen **4**

6 Parity Check **5**

7 AHBUART **5**

8 AHB LITE SYS **5**

1 Introduction

In this document we outline what we plan on testing and why. The goal is to achieve full verification for the AHB UART perpheral. Each testbench will use constrained random testing to cover a range of values without having to hard code them.

Each testbench will be properly structured to make it more extensible; they will contain:

- Testbench: The top level module responsible for instantiating the design under test, generating the clock, and initialising signals so it runs correctly.
- Test: A wrapper for the environment for extensibility purposes; multiple environments can be added down the line.
- Environment: Sets up and instantiates all the correct fields of the other components of the test, such as mailboxes. Responsible for running the monitor, driver, generator, and scoreboard.
- Interface: The blueprint for the virtual interface that will be used to communicate between the testbench and the design under test.
- Transaction: A class that specifies the important fields that will be used for comparison of input and output throughout the testbench.
- Generator: Randomly generates transaction instances using the transaction class with constraints and randomized values.
- Driver: Sends transactions created by the generator to the virtual interface used to communicate with the design under test
- Monitor: Reads output from the virtual interface that was sent by the design under test. Converts this data into a transaction that can be sent to the scoreboard for final scoring.
- Scoreboard: Collects output transactions and compares the inputs with the outputs and prints the pass or fail status to the console. Also tallies up the number of tests that pass and fail.

2 UART TX

- Check that the 9 bit input data vector is correctly transformed to serial data at the correct baud rate. This is to make sure overall primary functionality works.
- Generate code coverage to guarantee enough code has been put under test in our testbench.
- Ensure tx_start goes high before transmitting the data, tx_done goes high after the data is transmitted.
- For functional coverage, ensure input data is tested with both odd and even values, and test the distribution of values used is balanced enough by using low, medium-low, mediums-high, and high bins. Then test a cross combination of all of these. This will guarantee we have tested over a satisfactory range of inputs and gain confidence in our testbench.

- Write assertions for important transitions, such as tx_done being asserted after the current state enters the stop state. We can also check that the start and stop bit are correct by asserting tx high for the stop state and tx low for the start state. This gives confidence that the module behaves correctly.

3 UART RX

- Read the serial Rx input data at the correct baud rate and check it matches the output vector. This is to make sure overall primary functionality works.
- Ensure rx_done is high after the receiving is done.
- For functional coverage, ensure input data is tested with both odd and even values, and test the distribution of values used is balanced enough by using low, medium-low, mediums-high, and high bins. Then test a cross combination of all of these. This will guarantee we have tested over a satisfactory range of inputs and gain confidence in our testbench.
- Write assertions for important transitions, such as rx_done being asserted after the current state enters the stop state. This gives confidence that the module behaves correctly.
- Generate code coverage to guarantee enough code has been put under test in our testbench.

4 FIFO

- Write SystemVerilog Assertions for important properties that must be true. For example, empty should not be true on the cycle following rd & wr, and full should not be true on the cycle following wr & rd. Another example is full and empty should not both be true at the same time. This guarantees the module behaves correctly over time and meets the specification.

5 Parity Gen

- Check that the bottom 8 bits of the output data match the input data to make sure overall primary functionality works.
- Confirm the parity bit is correct under both even and odd parity (randomly choose one) and with an occasional parity fault injection with a custom distribution.
- For functional coverage, ensure even and odd parity is tested, as well as parity fault injection. Also make sure input data is tested with both odd and even values. Then test a cross combination of all of these. This will guarantee we have tested over a satisfactory range of inputs and gain confidence in our testbench.
- Generate code coverage to guarantee enough code has been put under test in our testbench.
- Write assertions to make sure that the data is still correct after a parity bit is added. The bottom 8 bits should be the same in both input and output.

6 Parity Check

- Check that PARITYERR is asserted when the parity bit is incorrect for both even and odd parity, and with an occasional parity fault injection with a custom distribution. This makes sure overall primary functionality works.
- Generate code coverage to guarantee enough code has been put under test in our testbench;
- For functional coverage, ensure even and odd parity is tested, as well as parity fault injection. Also make sure input data is tested with both odd and even values. Then test a cross combination of all of these. This will guarantee we have tested over a satisfactory range of inputs and gain confidence in our testbench.
- Generate code coverage to guarantee enough code has been put under test in our testbench.

7 AHBUART

- Test overall transmission and receiving through the UART. Check the bottom 8 bits of HWDATA equate to the first 8 bits sent at the transmitter end. Then feed that output back into the Rx receiver terminal simultaneously and ensure HRDATA matches the original HWDATA.
- Push multiple data vectors through the UART at once to ensure the FIFO mechanism works as intended. When the HREADYOUT is low, stop sending data through until it becomes high again (at which point we know the FIFO has more space to hold more data).
- Check that parity is generated and received correctly for both even and odd parity, and with an occasional parity fault injection with a custom distribution.
- Ensure the AHBUART works at these baud rates: 110, 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 38400, 57600, 115200, 128000 and 256000
- For functional coverage, ensure even and odd parity is tested, as well as parity fault injection. Also make sure input data is tested with both odd and even values. Test the distribution of values used is balanced enough by using low, medium-low, mediums-high, and high bins. Then test a cross combination of all of these. Then test a cross combination of all of these. This will guarantee we have tested over a satisfactory range of inputs and gain confidence in our testbench.
- Generate code coverage to guarantee enough code has been put under test in our testbench.
- Write assertions to ensure certain properties of the module are met. For example, check the necessary conditions for uart_rd and uart_wr and their values on the next cycle. This guarantees the module behaves correctly over time and meets the specification.

8 AHB LITE SYS

- Drive transactions through the RsRx terminal of an AHBLITE SYS that is running an assembler program that waits for the receive buffer to have content, and sends the value back through the UART.

- Monitor the output of the RsTx terminal and check that it matches what was sent in.
- For functional coverage, ensure even and odd input data is tested. Test the distribution of values used is balanced enough by using low, medium-low, mediums-high, and high bins. Then test a cross combination of all of these. Then test a cross combination of all of these. This will guarantee we have tested over a satisfactory range of inputs and gain confidence in our testbench.