Qué son las metodologías de verificación y cómo crear la tuya propia

Creador: David Rubio G.

Entrada: https://soceame.wordpress.com/2025/02/09/que-son-las-metodologias-de-verificacion-y-como-crear-la-tuya-propia/

Blog: https://soceame.wordpress.com/

GitHub: https://github.com/DRubioG

Fecha última modificación: 13/02/2025

En una entrada anterior ya comenté una forma avanzada de estructurar los testbench.

https://soceame.wordpress.com/2024/12/12/testbenchs-avanzados-momento-de-evolucionar/

Bien, pues a raíz de esta entrada me he ido encontrando la existencia de diferentes metodologías que se basan en el principio que comenté en esta anterior entrada.

Estos métodos se basan en la reutilización de módulos Open Source que se utilizan para hacer simulaciones, y después genera un informe. Estos módulos contemplan un motón de situaciones de posible simulación, Además, estos módulos son simplemente una simplificación y la creación de unas interfaces que confrontan a las del DUT.
Entonces, al final lo que haces es crear un módulo que agrupa todos los módulos que puedes requerir para hacer la comprobación.
Las principales metodologías son la UVM, la OSVVM y la UVVM.

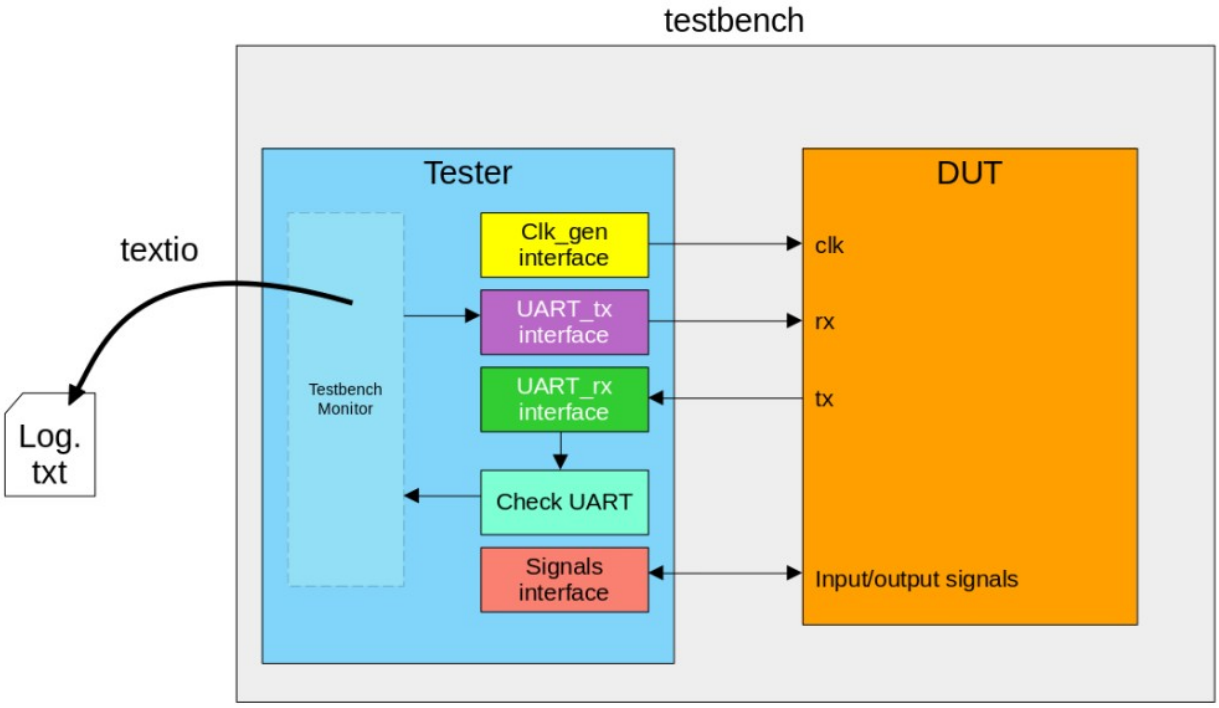
- La **UVM (Universal Verification Methodology)** es simplemente unas normas de verificación para poder ascender a capas superiores de abstración y llegar a sistemas de clases y objetos, como en un lenguaje de programación. El problema es que este enfoque está más preparado para SystemVerilog que para VHDL.
- El **UVVM (Universal VHDL Verification Methology)** es un proyecto Open Source que permite mediante el seguimiento de un procedimiento y utilizando las librerías Open Source de UVVM realizar la verificación.
- El **OSVVM (Open Source VHDL Verification Methodology)** es otro proyecto Open Source que permite mediante el seguimiento de un procedimiento y utilizando las librerías Open Source de OSVVM realizar la verificación.

La realidad es que estos dos últimos si revisas el código fuente que utilizan son infumables.

Entonces, debido a la complejidad que ofrecen estos estándares y que muchas veces no se ajustan a la capacidad o al flujo de trabajo de un programador FW en VHDL, ahora te voy a explicar con los principios de los estándares anteriores, cómo se puede crear tu propio método de verificación.

Cómo crear tu propia metodología de verificación

Crear tu propia metodología de verificación no es difícil, solo se necesita estructurar los testbench para llegar a una estructura como la siguiente.



En esta estructura de testbench se puede ver dentro que está compuesto de diferentes módulos. Tenemos el módulo que queremos simular, el DUT, y también tenemos el modulo "**Tester**", este módulo lo único que hace es agrupar todas las interfaces de comunicación con el DUT y gestionar todo el sistema de verificación.

Dentro del *Tester* se ubica todo lo necesario para Testear el módulo DUT y además, también cuenta con un módulo interno que es el encargado de controlar todos los test que se le pasan al DUT, y además genera informes en un fichero sobre el comportamiento del DUT hacia el mundo exterior. Esto es muy útil, porque ya no es necesario que el verificador esté comprobando todas las señales que se simulan, si no que solo es necesario que **mire el informe que ha generado el Tester al final de la simulación**. Tampoco es necesario que el *Tester* solo genere un informe, puede generar tantos como se requiera, pero sí que tiene que haber uno que es el principal que tiene que contener qué pruebas se le han hecho al DUT y qué resultado de obtiene. Para ello se hace uso de la librería **textio**.

NOTA: VHDL no permite la lectura y la escritura sobre un mismo fichero, por eso se recomienda que se escriba el fichero de informe desde un solo sitio y no cerrarlo hasta terminar debido a que cada vez que se abre un fichero con textio lo empieza desde cero, o sea, borra todo lo que hay dentro.

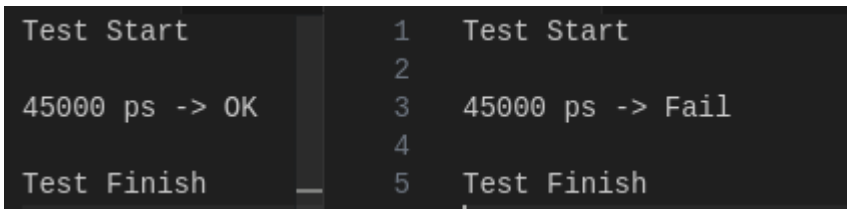
https://soceame.wordpress.com/2024/07/01/como-usar-las-librerias-textio-en-vhdl/

Un ejemplo práctico de un fichero Tester sería algo como lo siguiente, donde tenemos un fichero que primero actualiza un dato y después bloquea su ejecución con un *wait until* hasta que *input* tenga un valor, esta entrada la genera el módulo que comprueba que la ejecución es correcta o fallida, y dependiendo de ese valor se escribe una cosa u otra en el fichero *Test_report.txt*.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use std.textio.all;
use ieee.std_logic_textio.all;

entity tester is
    Port (
            input : in std_logic;
            data_out : out std_logic_vector(3 downto 0)
    );
end tester;

architecture tester_arch of tester is

file fichero_salida : text open write_mode is "Test_report.txt";

begin

    process
        variable linea_salida : line;
     begin
        write(linea_salida, string'("Test Start"));
        writeline(fichero_salida, linea_salida);
        writeline(fichero_salida, linea_salida);

        data_out <= x"A";

        report "Comienzo test";

        wait until input = '1' or input = '0';
        write(linea_salida, time'image(now));
        write(linea_salida, string'(" -> "));
        if input = '1' then
            write(linea_salida, string'("OK"));
        elsif input = '0' then
            write(linea_salida, string'("Fail"));
        end if;
        writeline(fichero_salida, linea_salida);
        writeline(fichero_salida, linea_salida);

        write(linea_salida, string'("Test Finish"));
        writeline(fichero_salida, linea_salida);
        file_close(fichero_salida);

        report "Fin " severity failure;

    end process;
```

***Nota**: no se pueden usar el tipo string como entrada por los puertos, pero se puede crear un alias que facilite los nombres.*

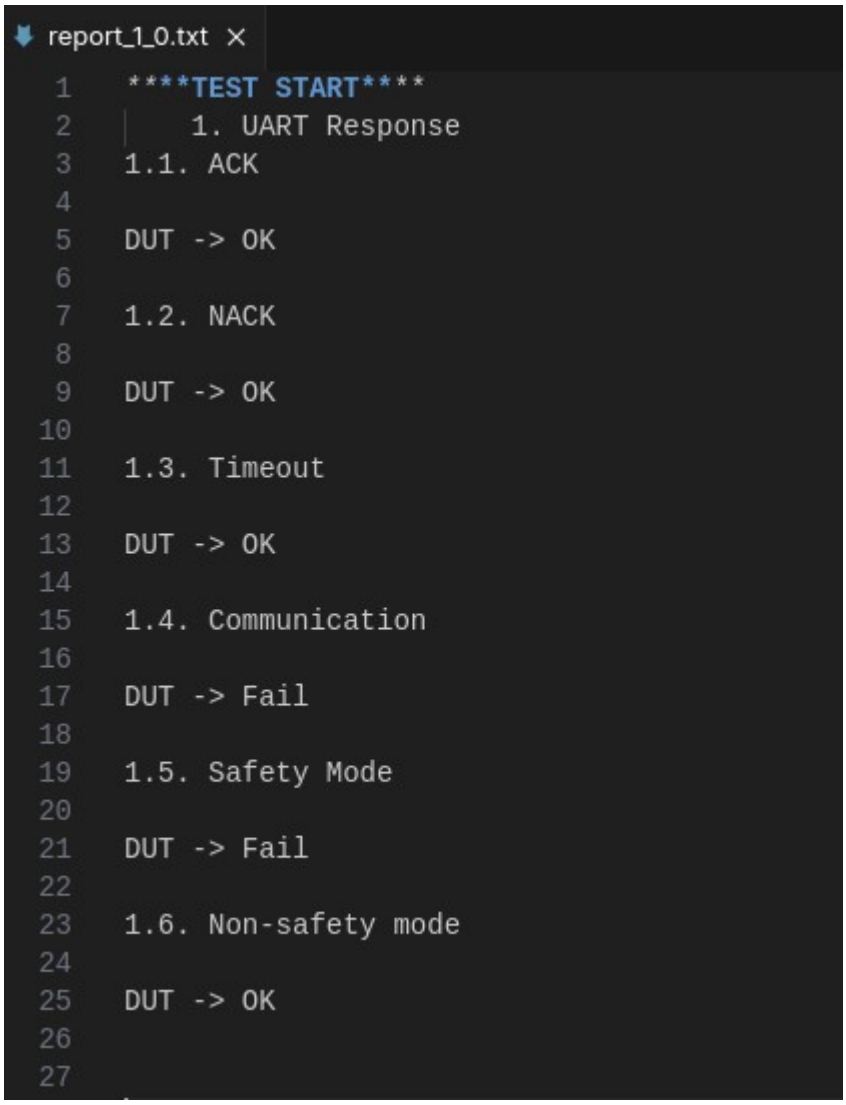
El fichero que se generaría tendría la siguiente forma.



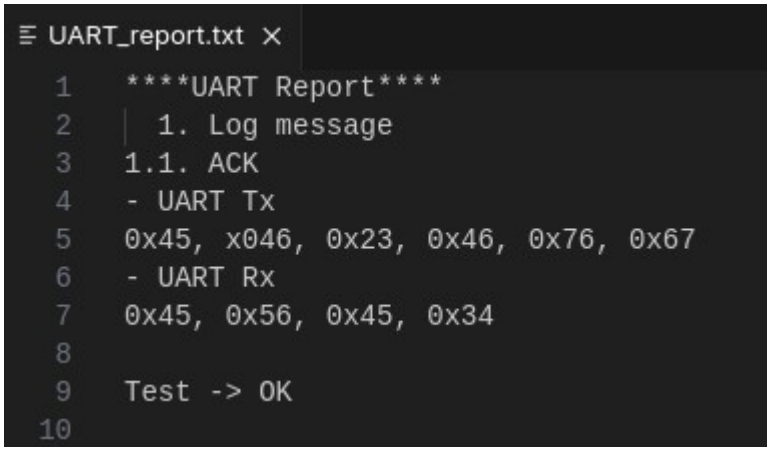
NOTA: la librería textio no tiene forma de escribir un salto de línea salvo que se utilice el *writeline* vacío. Y todos los write se escriben en la misma línea sin espacios, para meterlos tienes que meter una sentencia específica. Y por último, si no se escribe el último writeline no escribe la última línea.

Esto genera también la ventaja de que no tienes que basar tus simulaciones en ver los *report* o los *assert* del código, porque estos van en un fichero que se puede leer de forma limpia.

Un informe podría ser algo parecido a esto, donde se puede apreciar el test que se está pasando con la comprobación de los otros módulos para verificar si responde de la forma deseada o no. Se recomienda tener un documento donde figuren los test numerados para poder conocer el funcionamiento del test de forma escrita más allá del informe que genera el simulador.



Y en otro documento puedes tener el informe de la UART, donde figuren los mensajes enviados.



La ventaja es que tanto los test como las comprobaciones son automáticos.

Ventajas de este sistema de verificación

- **Reusabilidad**: debido a que todos los submódulos internos son reutilizables entre proyectos, incluso pueden llegar a ser los mismos que los del FW que quieres testear, como por ejemplo la UART, puedes utilizar el módulo TX contra el módulo RX, solo que esté esta en el Tester.
- **Velocidad de desarrollo**: al tener los módulos ya desarrollados, lo único que hay que hacer es crear un fichero que los maneje, el Tester.
- **Escalabilidad**: al tener todo dividido en pequeños módulos y que estén controlados por un módulo que solo sirve para pasar test, lo único que hay que hacer es añadir el nuevo test.
- **Facilidad de comprobación de errores**: cuando solo haces simulaciones viendo señales puedes perderte detalles necesarios de la simulación que se te pueden haber pasado o tener que comprobar señales de módulos totalmente distintos. Al usar este método puedes ver todo lo que esté interrelacionado fácilmente, porque es el propio FW el que lo comprueba.
- **Velocidad de simulación**: las simulaciones cuanto más señales a simular son más lentas, con este método no hace falta mirar ninguna señal, solo hace falta mirar el informe generado al final.
- **Verificación avanzada**: es posible que el DUT que se quiera utilizar no sea el que se utiliza para hacer el Tester, entonces, al tener un sistema que simula todos los posibles casos de forma fácil es posible cambiar el DUT de forma relativamente fácil y solo tener que comprobar el informe para ver si el DUT se ajusta a la aplicación real.

Sé que este método al principio puede ser difícil de implementar, pero a la larga facilita mucho la vida del diseñador FW, debido a que automatiza la tarea y permite hacer simulaciones más largas de forma fácil.

Nota final

Al igual que se puede crear un fichero con un informe, también se puede crear una tabla tipo CSV o incluso un fichero .md (MarkDown) con la información deseada.

Además también puedes insertar marcas de tiempo en el informe, relativas a la ejecución de la simulación, utilizando un timer externo que te cree las marcas de tiempo, o utilizando marcas de tiempo con la palabra clave "*now*".

https://soceame.wordpress.com/2025/02/09/como-utilizar-la-alabra-clave-now-en-vhdl/