Simulação e teste em VHDL

No desenvolvimento em VHDL é comum definirmos uma estrutura que chamamos de “Modelo U Invertido” para testar nossos componentes, conforme mostrado na Fig. 1. Esse modelo tem três entidades principais: a entidade que queremos testar (device under test, ou DUT), o golden model e o testbench, além de um *top module* que faz a ligação entre esses componentes.

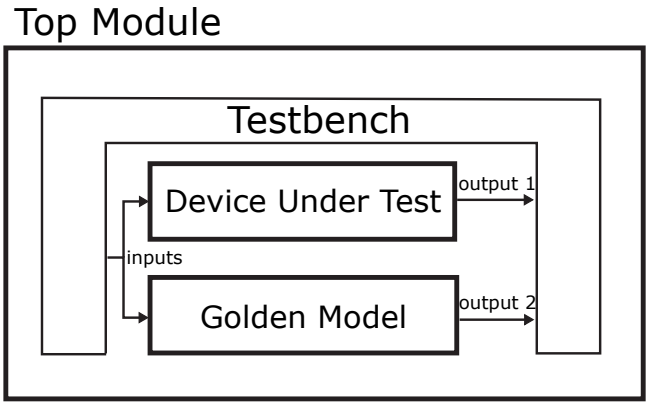

Fig. 1: O Modelo em “U” Invertido

O golden model pode ser definido como uma *implementação perfeita* do módulo que queremos testar. Ele pode ser apenas a tabela verdade do módulo, ou pode ser feito de maneira comportamental. Em alguns casos, o golden model pode ser implementado em uma outra linguagem de programação, que escreve os dados em arquivo. A entidade VHDL então abriria este arquivo e usaria os dados encontrados como referência.

O testbench tem a função de gerar todas as entradas possíveis que o modelo requer, mostrar essas entradas tanto para o DUT quanto para o golden model, e verificar se as saídas estão corretas. Caso alguma saída esteja incorreta, o testbench deve avisar sobre o problema.

VHDL fornece dois comandos para facilitar esse processo:

assert boolean-expression
    report string-expression **severity** severity-expression;

report string-expression **severity** severity-expression;

O comando **report** simplesmente escreve no console uma string. O comando **assert** testa a condição booleana e, se esta condição for falsa, escreve no console a string ao lado. O comando “integer’image(i)” é utilizado para converter a variável *i* para string, para que ela possa ser escrita no console. O operador “&” é usado para concatenar duas strings. Com isso, podemos escrever os três componentes para testar o circuito de maioria:

Algoritmo 1 Top module

```
1: library IEEE;
2: use IEEE.STD_LOGIC_1164.ALL;
3:
4: entity topmodule is
5: end topmodule;
6:
7: architecture topmodule_arch of topmodule is
8:
9:         component maioria_dut is
10:        port (A,B,C: in STD_LOGIC;
11:                F: out STD_LOGIC);
12:        end component;
13:
14:        component maioria_golden_model is
15:        port (A,B,C: in STD_LOGIC;
16:                F: out STD_LOGIC);
17:        end component;
18:
19:        component testbench is
20:        port (f_dut,f_gm: in STD_LOGIC;
21:                a,b,c: out STD_LOGIC);
22:        end component;
23:
24:        signal a, b, c, f_gm, f_dut: STD_LOGIC;
25: begin
26:        U0: maioria_dut port map(a, b, c, f_dut);
27:        U1: maioria_golden_model port map(a, b, c, f_gm);
28:        U2: testbench port map(f_dut, f_gm, a, b, c);
29: end topmodule_arch;
```

Algoritmo 2 Device under test (DUT)

```
1: library IEEE;
2: use IEEE.STD_LOGIC_1164.ALL;
3:
4: entity maioria_dut is
5:   port ( A,B,C: in STD_LOGIC;
6:           F: out STD_LOGIC);
7: end maioria_dut;
8:
9: architecture dut_arch of maioria_dut is
10:        signal P1,P2,P3 : STD_LOGIC;
11: begin
12:        P1 <= A and B;
13:        P2 <= A and C;
14:        P3 <= B and C;
15:        F <= P1 or P2 or P3;
16: end dut_arch;
```

Algoritmo 3 Golden model

```
1: library IEEE;
2: use IEEE.STD_LOGIC_1164.ALL;
3:
4: entity maioria_golden_model is
5:   port ( A,B,C: in STD_LOGIC;
6:           F: out STD_LOGIC);
7: end maioria_golden_model;
8:
9: architecture gm_arch of maioria_golden_model is
10:         signal ABC : STD_LOGIC_VECTOR (2 downto 0);
11: begin
12:         ABC <= A & B & C;
13:
14:         with ABC select
15:             F <=  '0' when "000",
16:                   '0' when "001",
17:                   '0' when "010",
18:                   '1' when "011",
19:                   '0' when "100",
20:                   '1' when "101",
21:                   '1' when "110",
22:                   '1' when "111",
23:                   '0' when others;
24:
25: end gm_arch;
```

Algoritmo 4 Testbench

```
1: library IEEE;
2: use IEEE.STD_LOGIC_1164.ALL;
3: use IEEE.STD_LOGIC_ARITH.ALL;
4: use IEEE.STD_LOGIC_UNSIGNED.ALL;
5:
6: entity testbench is
7:   port ( f_dut,f_gm: in STD_LOGIC;
8:           a,b,c: out STD_LOGIC);
9: end testbench;
10:
11: architecture testbench_arch of testbench is
12: begin
13:         process
14:             variable contbin : STD_LOGIC_VECTOR (2 downto 0);
15:         begin
16:
17:             report "Iniciando teste..." severity NOTE;
18:
19:             contbin := "000";
20:
21:             for i in 1 to 8 loop
22:
23:                 a <= contbin(2);
24:                 b <= contbin(1);
25:                 c <= contbin(0);
26:                 wait for 200 ns;
27:
28:                 assert (f_gm = f_dut) report "Falhou: i = " & integer'image(i) severity ERROR;
29:
30:                 contbin := contbin + 1;
31:
32:             end loop;
33:
34:             report "Teste finalizado!" severity NOTE;
35:
36:             wait;
37:
38:         end process;
39: end testbench_arch;
```

Os códigos do top module, do device under test e do golden model são simples, mas é importante comentar alguns detalhes do código do testbench.

Na simulação, quando o processo (no testbench) muda os valores dos sinais *a, b* e *c*, os componentes (no top module) respondem e mudam os valores de *f_dut* e *f_gm*.

Na **linha 26** do algoritmo do testbench, temos uma declaração *wait for 200 ns*. Essa instrução faz com que a simulação espere por 200 ns. Em geral, não é tão importante o tempo dessa espera (a não ser que o seu circuito tenha sido projetado com atraso), mas essa instrução é extremamente importante para o funcinamento do testbench. Nas linhas 23, 24 e 25, assinalamos o valor da **variável** *contbin* para os **sinais** *a, b* e *c*. Dentro de um processo, essa declaração não é instantânea: o VHDL só considera que os sinais *a, b* e *c* tem os seus novos valores quando o processo termina, **ou** quando este processo é colocado em espera, que é o que acontece quando executamos a declaração *wait for*.

Na **linha 36** temos uma declaração *wait*. Essa instrução efetivamente coloca o processo em espera, e trava o processo completamente. É importante que essa declaração esteja aqui - caso contrário, o processo é executado em loop infinito, o que pode causar problemas dependendo do simulador utilizado.