ESCOLA POLITÉCNICA DA UNIVERSIDADE DE SÃO PAULO

PCS 3225 – SISTEMAS DIGITAIS II

ATIVIDADE FORMATIVA 2 - TESTBENCHES

Professor:
Dr. Marco Tulio Carvalho de Andrade

Grupo 10	**NUSP**
Guilherme Fortunato Miranda	13683786
João Pedro Dionizio Calazans	13673086
Lucas Pates Rodrigues	11545304
Matheus Tierro de Paula	11804021
Pedro Kooshi Ito Sartori	13683661

SÃO PAULO
2023

SUMÁRIO

1. APRESENTAÇÃO DOS CASOS DE TESTE

Os casos de teste do multiplicador binário são encontrados na tabela 1.

Tabela 1: Casos de teste

Multiplicando em decimal	Multiplicando em binário	Multiplicador em decimal	Multiplicador em binário	Resultado em decimal	Resultado em binário
3	0011	6	0110	18	00010010
15	1111	11	1011	165	10100101
15	1111	0	0000	0	00000000
1	0001	11	1011	11	00001011

Fonte: De autoria própria.

2. EXPLICAÇÃO DOS CASOS DE TESTE

A figura 1 ilustra um caso de teste no código.

Figura 1: Caso de teste.

```
    ---- Caso de teste 1: A=3, B=6
    Va_in <= "0011";
    Vb_in <= "0110";
    -- Reset inicial (1 periodo de clock) - não precisa repetir
    rst_in <= '1'; start_in <= '0';
    wait for clockPeriod;
    rst_in <= '0';
    wait until falling_edge(clk_in);
    -- pulso do sinal de Start
    start_in <= '1';
    wait until falling_edge(clk_in);
    start_in <= '0';
    -- espera pelo termino da multiplicacao
    wait until ready_out='1';
    -- verifica resultado
    assert (result_out/="00010010") report "1.OK: 3x6=00010010 (18)"
severity note;

    wait for clockPeriod;
```

Fonte: De autoria própria.

Cada caso de teste é composto por um bloco de “instruções”, no qual, essencialmente, são atribuídos os sinais de entrada desejados, e por fim, feito o monitoramento da saída, a fim de verificar se o DUT funciona corretamente para as entradas testadas.

Inicia-se o bloco atribuindo o valor das entradas para o caso de teste em questão, a partir dos sinais Va_in e Vb_in. Em sequência, tem-se o reset inicial e a atribuição de 0 a start, por uma duração de um clock inteiro. Esse reset inicial é feito apenas no primeiro caso de teste, ilustrado na figura 1.

Na próxima borda de descida do clock – não a de subida para que a UC se inicie logo na próxima de subida –, finalmente inicia-se o sinal start, mantendo-o assim por um clock inteiro. A multiplicação acaba na chegada de sinal Ready em HIGH/’1’. A função assert apresenta um report sempre que a operação em parênteses é falsa. Como a afirmação obtida no report é de funcionamento correto, deseja-se que o sinal de assert seja falso quando o sinal de result_out, vindo da DUT, seja exatamente igual a multiplicação, daí a operação de desigualdade.

Quanto a codificação de casos adicionais, após o fim desse bloco de código, é possível apenas adicionar uma estrutura em bloco semelhante atentando-se à mudança de valores nos campos de entrada e saídas esperadas para o novo teste, além do report com a mensagem apropriada.

3. MODIFICAÇÃO PARA VETOR DE TESTES

Enviado em arquivo ZIP.

4. EXECUÇÃO DO TESTBENCH COM VETOR DE TESTES

A execução da simulação após alteração tem resultados que atestam para o correto funcionamento do DUT (multiplicador binário) nos casos testados.

5. RESULTADOS OBTIDOS COM VETOR DE TESTES

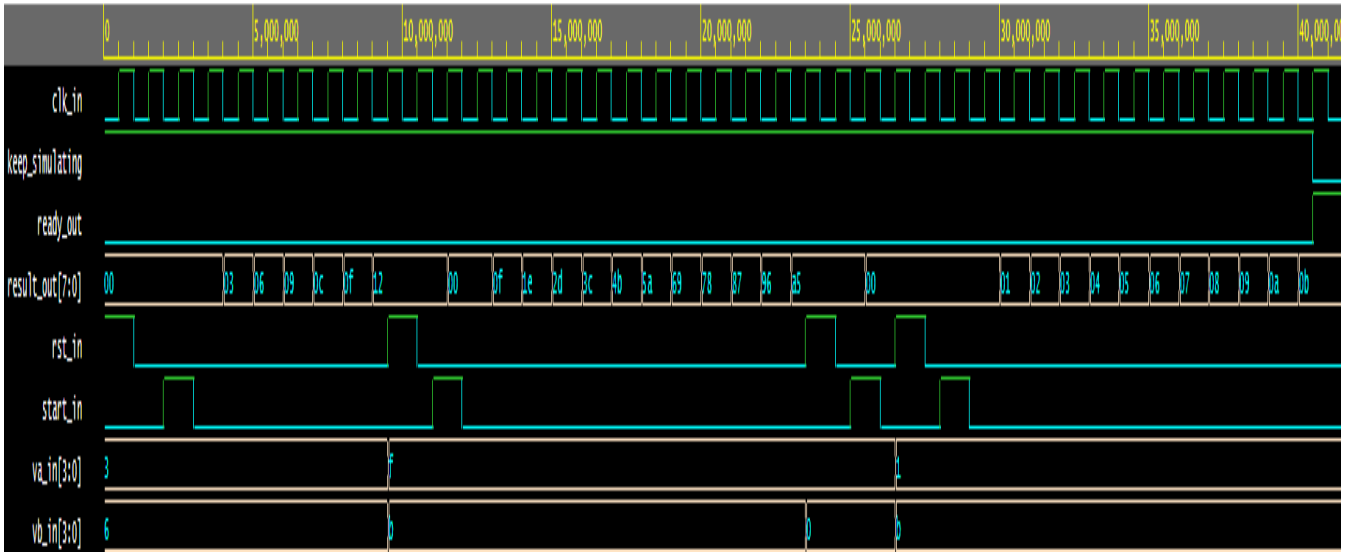
As figuras 2 e 3 apresentam os resultados da simulação.

Figura 2: Mensagens de execução da simulação.

```
testbench.vhd:84:5:@0ms:(assertion note): simulation start
testbench.vhd:108:5:@40500ps:(assertion note): simulation end
```

Fonte: De autoria própria via EDA Playground.

Figura 3: Sinais dos testes.



Fonte: De autoria própria via EDA Playground.

Como indica a figura 2, o programa apresenta apenas dois reports, de início e de fim da simulação, pois em divergência ao código original explicado na seção 2, as mensagens de controle aqui desempenham papel de alerta, surgindo quando a expressão booleana 'saída esperada = saída do DUT' for falsa, ou seja, quando o sinal recebido estiver diferente do que deveria ser.

6. MODIFICAÇÃO PARA LEITURA DE ARQUIVO

Enviado em arquivo ZIP.

7. EXECUÇÃO DO TESTBENCH COM LEITURA DE ARQUIVO

A execução da simulação após alteração tem resultados que atestam para o correto funcionamento do DUT (multiplicador binário) nos casos testados.

8. RESULTADOS OBTIDOS COM LEITURA DE ARQUIVO

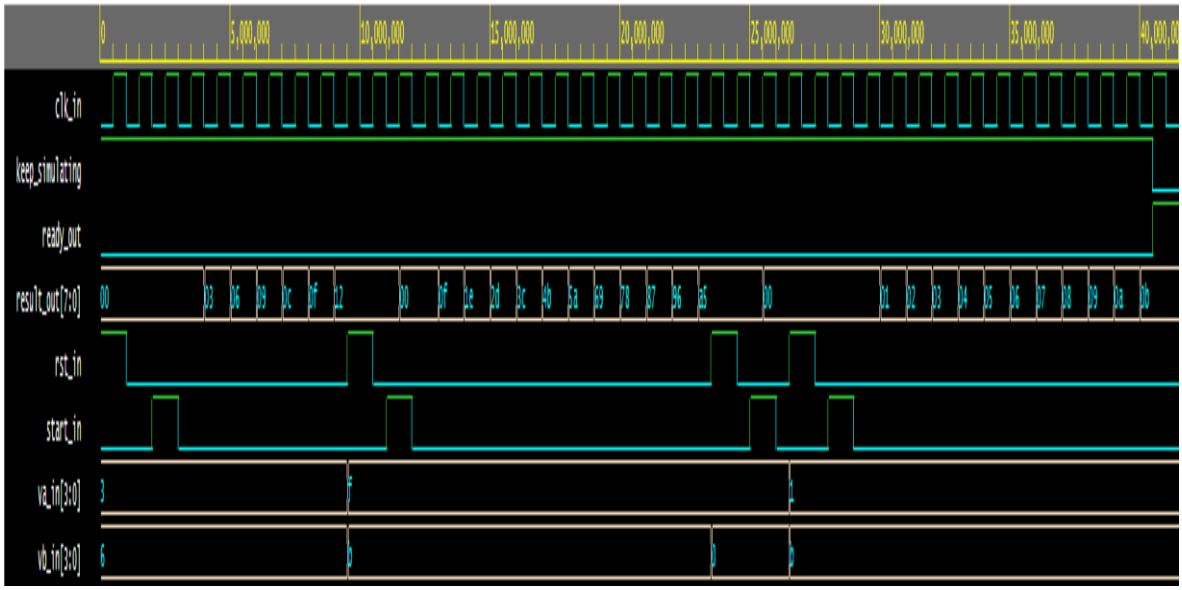
As figuras 4 e 5 apresentam os resultados da simulação.

Figura 4: Mensagens de execução da simulação

```
testbench.vhd:73:5:@0ms:(assertion note): simulation start
testbench.vhd:102:5:@40500ps:(assertion note): simulation end
```

Fonte: De autoria própria via EDA Playground.

Figura 5: Sinais dos testes.



Fonte: De autoria própria via EDA Playground.

Diferentemente dos códigos anteriores, nos quais as entradas e saídas dos casos de testes são explicitadas dentro do código, este utiliza os sinais de entrada e saída recebidos do arquivo .dat para conferir se a saída do DUT corresponde à saída esperada. Como pode ser visto no log da figura 4, não há mensagens de erro, o que atesta o correto funcionamento do DUT nos casos testados.

9. CONSIDERAÇÕES FINAIS

Percebe-se que, apesar de todas as 3 formas de codificar o *testbench* estarem corretas, existe uma diferença considerável entre elas no quesito praticidade.

Por exemplo, a fim de adicionar 15 novos casos de teste no testbench codificado da maneira programática, seria necessário adicionar 15 novos blocos de código semelhantes à figura 1, contendo não apenas valores de entradas e saídas do DUT, mas também todas as considerações no tempo de execução com o comando *wait,* bem como as mensagens de aviso com o comando *assert report.*

Em comparação, caso *o testbench* esteja codificado com vetor de testes, bastaria anexar as novas entradas e saídas ao final do vetor. De modo semelhante, a codificação com leitura de arquivo permite que a adição de novos casos de teste seja feita apenas adicionando os valores de entrada e saída esperada ao fim do arquivo especificado, de modo a diminuir consideravelmente o atrito para adicionar novos casos de teste.