

## PSI-3451 Projeto de CI Lógicos Integrados

### Aula 8- Conceitos relacionados aos circuitos modelados em VHDL

Nesta aula o aluno receberá 3 arquivos de testbenches a serem utilizados com blocos relacionados ao projeto *Snake*, já testados em aulas anteriores:

- *Testbench* para simulação de circuitos combinacionais (*testbench\_comb*): módulo topo que instancia o módulo de lógica combinacional a ser testado e o gerador de estímulos.
- *Testbench* para simulação de circuitos sequenciais simples (*testbench\_seq\_1*): módulo topo que instancia o módulo de lógica sequencial a ser testado, o módulo gerador de clock e o gerador de estímulos.
- *Testbench* para simulação de circuitos sequenciais com protocolo (*testbench\_seq\_2*): módulo topo que instancia o módulo de lógica sequencial a ser testado, o módulo gerador de clock e o gerador de estímulos adaptado para o protocolo estabelecido pelo circuito.

Nesta experiência, os *testbenches* serão escritos em VHDL (outras linguagens também são admissíveis) e as simulações farão uso do Modelsim.

#### 1. *Testbench*

O *testbench* é um código em linguagem de programação ou de descrição de hardware, como o VHDL, que descreve um conjunto de estímulos a ser aplicado nos portos (terminais) de entrada do (modelo do) circuito a fim de se verificar seu correto funcionamento através da correspondente simulação. O *testbench* pode também efetuar a comparação entre os resultados das simulações e um gabarito, indicando de automaticamente se o modelo do circuito está operando corretamente ou se contém erros. Desta forma é possível testar cada um dos modelos do circuitos através de um grande número de estímulos de entrada sem a necessidade de se descrever manualmente tais estímulos através dos recursos fornecidos pelos simuladores, reduzindo o risco de erros durante as simulações.

Não se deve subestimar a importância da elaboração de um *testbench*. Dada a necessidade de se certificar de que um projeto/circuito tenha sido realizado corretamente como forma de garantir a sua qualidade, é comum que se gaste mais tempo no desenvolvimento de um *testbench* do que no projeto do circuito propriamente dito. A tarefa de verificação funcional, constituída basicamente da construção e execução de *testbenches*, consome até 60-70% dos recursos humanos e materiais dentro do fluxo do projeto nos projetos profissionais correntes.

O modelo básico de um *testbench* é ilustrado na figura 1. O código VHDL do *testbench* é hierárquico. Ele consiste de um módulo topo que contém 2 submódulos principais: o “Gerador de estímulos” e o “Circuito a ser verificado” (DUV, do inglês, Design Under Verification). Existe um terceiro módulo “Detecção e análise automática de erro”, que não será considerado nas simulações desta aula. O conjunto todo é compilado e rodado em simulador, sendo que o resultado pode ser, por exemplo, uma carta de tempos a ser analisada pelo projetista com o intuito de verificar se o funcionamento do circuito foi adequado. O *Gerador de estímulos* representa o ambiente de operação do circuito; para circuitos complexos, não é possível que todas as condições de operação sejam estimuladas, portanto, deve-se ter um planejamento cuidadoso para a geração dos estímulos.

Apesar de não utilizado nesta experiência, é muito comum que os *testbenches* sejam automatizados (em forma de CAD) também para não ser necessária por parte do projetista a inspeção visual para a análise de erros. Desta forma, costuma ser adicionado um módulo de detecção e análise automáticas de erros a partir da simulação realizada e da análise do grau de

progressão dos testes. Este módulo é apresentado na Figura 1 por linhas pontilhadas e não é implementado nesta aula. É comum que tal módulo seja implementado em linguagem de verificação diversa do VHDL ou Verilog, por exemplo, em SystemVerilog, C++, SystemC, etc.

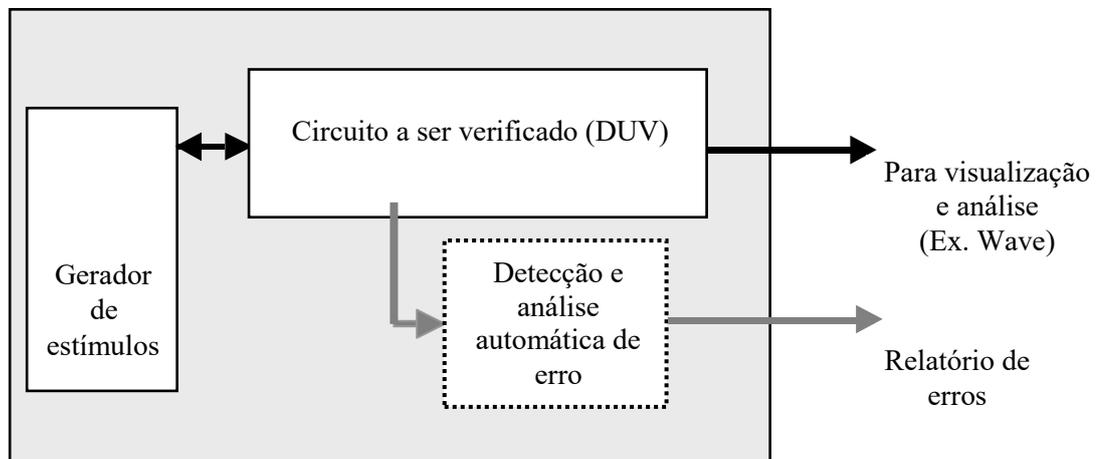


Fig.1: Modelo básico de um *testbench*

Até o momento da disciplina, o(a) aluno(a), ao aplicar vetores de testes em uma simulação manual, tem a intenção de emular uma ou várias situações de operação de um circuito. Da mesma forma, quando se projeta o gerador de estímulos, tem-se tal objetivo em vista, porém, automatizando a aplicação de testes. Apesar do fato de que cada circuito específico implica na construção de um gerador de estímulos específico, existem algumas técnicas a serem seguidas de acordo com o tipo do DUV, combinacional, sequencial simples ou com protocolo.

## 2. DUV- Circuito Combinacional

Circuitos combinacionais não apresentam comportamentos associados a estados. Qualquer estímulo particular aplicado às entradas de um circuito combinacional sempre gerará as mesmas respostas (após algum tempo de atraso) independente do momento da aplicação. Desta forma, espera-se a geração de estímulos através de uma sequência do seguinte tipo (seguido de captura em carta de tempos para visualização):

1. Aplicar um vetor 1 de valores às entradas IN\_1, IN\_2, etc.  
Capturar, para visualização, o resultado 1 junto às saídas OUT\_1, OUT\_2, etc.
2. Aplicar um vetor 2 de valores às entradas IN\_1, IN\_2, etc.  
Capturar, para visualização, o resultado 2 junto às saídas OUT\_1, OUT\_2, etc.
3. Segue mais vezes...

É importante perceber que a aplicação de vetores acima é realizada sequencialmente, enquanto que, realtivamente a cada vetor, as indicações de valores para IN\_1, IN\_2, etc. são feitas concorrentemente. Desta forma, um processo (process) VHDL é a forma ideal para a modelagem de uma geração de vetores, como ilustrado no exemplo da Figura 2.

Pela figura, as designações de valores ocorrem todos em tempo de atraso Delta e há um espaçamento de 20ns entre cada vetor aplicado. É comum que este tempo de wait seja dado de forma parametrizada: por exemplo, pode-se usar "wait for TIME\_DELTA", com o parâmetro definido por *generic*.

```

1    process -- Assumindo que entradas IN_1, etc. são do tipo natural
2    begin
3        IN_1 <= 2; IN_2 <= 5; IN_3 <= 8;
4        wait for 20ns;
5        IN_1 <= 8; IN_2 <= 4; IN_3 <= 6;
6        wait for 20ns;
--        -- etc.
k    end process;

```

Figura 2. Exemplo de processo para DUV de circuito combinacional

Pode ocorrer frequentemente a situação de se realizar a simulação, gerando-se um número grande de N vetores; para tal, seria necessária a repetição em N vezes do código correspondente às linhas 2 e 3. Além de ser visualmente inadequado, esta forma pode aumentar a ocorrência de erros. Há uma técnica simples e bastante consagrada de se utilizar procedimentos (procedures) para a aplicação de vetores de testes. Desta forma o processo de aplicação seguiria a forma apresentada na Figura 3.

```

1    process          -- Assumindo que entradas IN_1, etc. são do tipo natural
2        procedure check_routine (constant a, b, c : in natural) is
3        begin
4            IN_1 <= a;
5            IN_2 <= b;
6            IN_3 <= c;
7            wait for TIME_DELTA;
8        end procedure check_routine;
9    begin
10       check_routine (2, 5, 8);
11       check_routine (8, 4, 6);
12       -- etc.
13    end process;

```

Figura 3. Código de processo VHDL com procedimento

Pode-se observar que o novo código prevê a mesma aplicação de vetores da Figura 2, entretanto o novo formato com procedimento permite uma visualização mais limpa dos vetores a serem aplicados, como pode se observar nas linhas 10 e 11. O aluno pode observar também que, neste caso, a construção parametrizada "wait for TIME\_DELTA" foi utilizada.

### 3. DUV- Circuito Sequencial Simples

Circuitos sequencias apresentam comportamentos associados a estados. Desta forma, qualquer estímulo particular aplicado às entradas de um circuito sequencial poderá gerar respostas diferentes se aplicados em momentos diferentes. Denominamos aqui circuitos sequenciais simples aqueles que apresentam uma característica síncrona, ou seja, a cada ciclo de relógio, a aplicação de vetores à entrada produz resultados à saída. A transição de um estado à outro é realizado internamente e não depende de sinais provenientes do ambiente emulado.

Para um DUV de circuito sequencial , sempre há a necessidade de se gerar um sinal de relógio, o que é feito, em geral, por um módulo contendo um processo especialmente codificado para tal. Este gerador de clock faz parte então do gerador de estímulos como está ilustrado na Figura 4. Um exemplo de código para tal processo de geração é apresentado na Figura 5. Pode-se observar que o valor do período do clock é parametrizado pela variável CLK\_PERIOD.

Assim como ocorre no caso de circuitos combinacionais, algumas atividades de simulação devem ser realizadas muitas vezes, entre elas, a ativação do sinal de reset (ou set) e a aplicação de vetores de teste. Desta forma, os comandos VHDL correspondentes a estas tarefas podem ser encapsuladas em procedimentos.

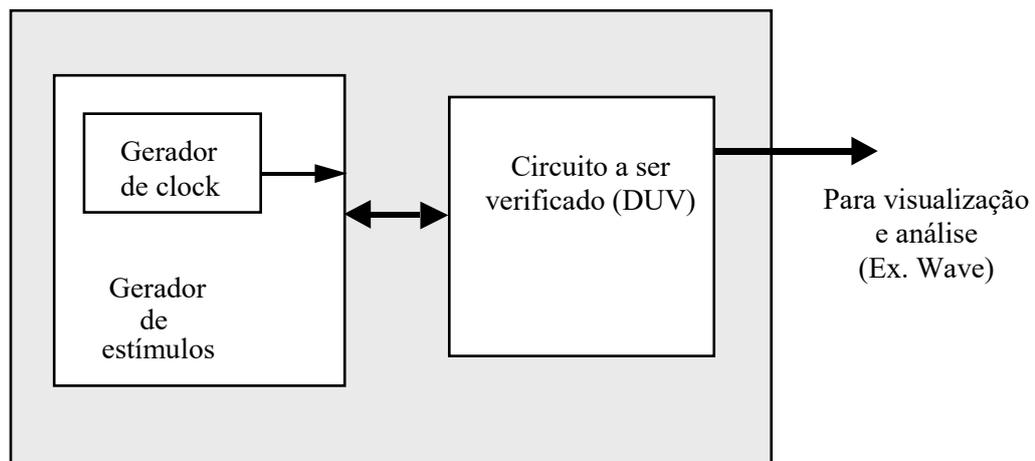


Figura 4. Gerador de estímulos para DUV de circuitos sequenciais

```

process
begin
    CLK <= '1';
    wait for CLK_PERIOD / 2;
    CLK <= '0';
    wait for CLK_PERIOD / 2;
end process clk_generation;

```

Figura 5. Processo gerador de clock

A Figura 6 apresenta um exemplo de procedimento para a ativação do sinal de *reset* síncrono; neste caso , o sinal deve estar pronto antes da borda de subida do clock (assumindo que os registradores são sensíveis à borda de subida do clock). Portanto, o sinal de *reset* é ativado e desativado nas bordas de descida do clock.

Para o procedimento (procedure) de geração de estímulos, a forma apresentada na Figura 3 pode ser seguida, apenas trocando a instrução da linha 7 ( *wait for TIME\_DELTA*) por algo do tipo *wait until rising\_edge (CLK)*; tal comando indica que a aplicação de vetores de entrada será realizada a cada ciclo na borda de subida do clock.

```

procedure reset_activate is
begin
    wait until falling_edge(CLK);
    reset <= '1';
    wait for CLK_PERIOD;
    reset <= '0';
end procedure reset_activate;

```

Figura 6. Procedimento de ativação de reset.

#### 4. DUV- Circuito Sequencial com Protocolo

Neste texto, denominamos de circuitos sequencias com protocolo aqueles em que o sequenciamento de estados depende da resposta do usuário (emulando o ambiente de operação), sendo que a ação deste depende de eventos reportados pelo DUV. Estes eventos são apresentados em forma de variação de *flags* ou de valores de dados de saída do DUV. Neste caso, um estudo pormenorizado da comunicação do DUV deve ser realizado para a codificação correta da aplicação dos vetores de teste.

A Figura 7 exemplifica o caso de um processo de um *testbench*, onde fica explicitado um protocolo de comunicação com o DUV. Por este protocolo, o DUV aguarda um *flag write\_enable=1* em conjunto com um vetor de entradas (dado pelo procedure *check\_routine*). Após o recebimento, o DUV realizará as suas tarefas e, eventualmente, enviará o *flag write\_acknow=1*, indicando que o gerador de estímulos está livre para enviar um novo vetor.

```

Process
    variable i: integer;
Begin

    write_enable <= '0';
    check_routine (0, 0, 0);    -- Assumir como condição de iniciação

    reset_activate;

    wait until CLK_PERIOD;

    for i in 1 to 20 loop
        write_enable <= '1';

        check_routine (i, i+1, i+2);

        write_enable <= '0';

        wait until CLK_PERIOD;

        while (write_acknow /= '1') loop
            wait until CLK_PERIOD;
        end loop;
    end for;
End Process

```

Figura 7. Processo de geração de estímulos em testebench de circuitos com protocolo