Experimento 3:
INTRODUÇÃO À LINGUAGEM VHDL

1 OBJETIVO

O objetivo deste experimento é realizar um estudo introdutório da linguagem de descrição de hardware VHDL. São realizados projetos de circuitos combinacionais simples, em seguida descritos nessa linguagem. Aborda-se inclusive a geração em VHDL das combinações de entrada necessárias à simulação desses circuitos (programas do tipo *test-bench*). Objetiva-se ainda o projeto e a análise de um somador total de duas palavras binárias, tendo como bloco fundamental o somador completo de 3 bits.

2 INTRODUÇÃO TEÓRICA:

O VHDL é uma linguagem empregada na descrição de circuitos digitais. Circuitos digitais cuja complexidade varia desde a dos circuitos combinacionais mais simples até a de um microprocessador completo podem ser projetados, simulados e sintetizados usando essa linguagem.

A estrutura básica de uma descrição de uma unidade em VHDL é relativamente simples, composta apenas de duas partes. A primeira, denominada *entity declaration*, é a definição estrutural do circuito. Nessa etapa, são apenas designados os sinais de entrada e de saída, sem que se definam relações entre esses sinais e sem qualquer detalhamento acerca do funcionamento do dispositivo. O modelo básico dessa parte da descrição é apresentado a seguir.

```
entity nome-da-estrutura is
   port (nome-de-sinais : modo tipo-de-sinal;
         nome-de-sinais : modo tipo-de-sinal;
         ...
         nome-de-sinais : modo tipo-de-sinal);
end nome-da-estrutura;
```

Neste modelo, as designações dos sinais de entrada e saída são definidas pelo programador, sendo válidas quaisquer sequências de caracteres alfanuméricos que comecem por letras e que não constituam palavras reservadas da linguagem. O modo atribuído classifica cada sinal como de entrada (IN) ou saída (OUT ou BUFFER); eventualmente, o sinal poderá ainda ser ora de entrada, ora de saída (INOUT). Finalmente, define-se para cada sinal o seu tipo, por exemplo, um bit (BIT) ou um vetor de bits (BIT VECTOR).

Por exemplo, considere um circuito que implementa a função implicação, descrita na tabela 1.

Tabela 1 - Tabela Verdade da Função de Implicação.

X	Y	X → Y
0	0	1
0	1	1
1	0	0
1	1	1

Como se trata de uma função com dois bits de entrada e um bit de saída, a primeira parte da descrição VHDL fica assim esquematizada:

```
-------------------- Definição da porta x->y --------------------
entity porta_implica is              -- definição da estrutura
                                     -- de entradas e saídas.

   port (x,y: in BIT;
         z: out BIT);
   end porta_implica;
```

Note que, em VHDL, os comentários são introduzidos por dois hífens consecutivos. Todo o texto entre este sinal e o fim da linha é desconsiderado pelo compilador.

A segunda parte da estrutura é a denominada *architecture definition*. Nessa seção, define-se uma implementação particular da unidade introduzida na primeira parte. Seu modelo básico é apresentado a seguir.

```
architecture nome-da-descrição-operacional of nome-da-estrutura is
   delcarações de tipos
   declarações de sinais
   declarações de constantes
   declarações de funções
   declarações procedimentos
   declarações de componentes
begin
   declaração
   declaração
   ...
   declaração
end nome-da-descrição-operacional;
```

Entre a primeira linha (`architecture`) e o `begin`, definem-se todas as funções, constantes, procedimentos, sinais e tipos de sinais a serem utilizados durante a descrição operacional do circuito. Também são declarados os componentes que serão referenciados no corpo da seção. Isso permite, por exemplo, que um dispositivo já descrito em VHDL seja uma das partes de um dispositivo maior, levando a uma estruturação hierárquica dos programas.

Em seguida, entre o `begin` e o `end` principais da descrição operacional, estabelecem-se as relações entre os sinais de entrada e de saída do circuito. Podem estar presentes atribuições diretas de valores lógicos, funções, expressões booleanas, etc.

No caso da implementação da função $x \rightarrow y$, note, por exemplo, que a saída z só é 0 quando x é 1 e y é 0. Assim, pode-se fazer a seguinte descrição:

```
-------------------- Definição da porta x->y --------------------
entity porta_implica is            -- definição da estrutura
                                   -- de entradas e saídas.
   port (x,y: in BIT;
         z: out BIT);
end porta_implica;

architecture porta_implica_op of porta_implica is    -- descrição
                                                     -- operacional.
begin
   z <='0' when x='1' and y='0' else
       '1';
end porta_implica_op;
```

Uma outra alternativa é utilizar a descrição da função analisada na forma soma de produtos minimizada: $z = \overline{x} + y$; Assim, as linhas

```
z <='0' when x='1' and y='0' else
    '1';
```

podem ser substituídas por:

```
z <= not (x) or y;
```

Um dos aspectos mais importantes da linguagem é a possibilidade de incluir uma unidade VHDL na descrição de uma outra unidade, levando à modularização dos programas. Como exemplo,

suponha que se deseja implementar um circuito de detecção de pariade (a saída deve ser 1 se e somente se houver um número ímpar de entradas iguais a 1). Seja a seguinte descrição de uma porta ou-exclusivo:

```
---------------- Definição da porta ou-exclusivo ----------------
entity porta_ou_exclusivo is            -- definição da estrutura
                                        -- de entradas e saídas.
   port (x,y: in BIT;
         z: out BIT);
end porta_ou_exclusivo;

architecture porta_ou_exclusivo_op of porta_ou_exclusivo is
                                      -- descrição operacional.
begin
   z <= (not(x) and y) or (x and not(y));
end porta_ou_exclusivo_op;
```

Como um detector de paridade pode ser obtido a partir da associação de portas do tipo ou-exclusivo, é feita então, imediatamente antes do `begin` que abre sua descrição operacional, uma declaração da estrutura apresentada acima, que pode então ser livremente empregada:

```
---------------- Detector de paridade de 3 bits -----------------
entity detec_paridade_3bits is        -- definição da estrutura
                                      -- de entradas e saídas.
   port (a,b,c: in BIT;
         s: out BIT);
end detec_paridade_3bits;

architecture detec_paridade_3bits_op of detec_paridade_3bits is
                                          -- descrição operacional.
component porta_ou_exclusivo is
   port (x,y: in BIT;
         z: out BIT);
end component;
signal d: BIT;
begin
   U1: porta_ou_exclusivo port map(a,b,d);
   U2: porta_ou_exclusivo port map(d,c,s);
end detec_paridade_3bits_op;
```

Uma vez elaborada uma primeira descrição VHDL de um circuito, chega-se à fase de simulação. A forma mais sistemática de fazê-lo é criar uma outra unidade VHDL, sem sinais de entrada ou de saída, apenas com uma referência ao circuito a ser simulado e com um processo que gere todas as possíveis situações em que ele deve ser testado. No caso de um circuito combinacional, isto significa gerar todas as possíveis combinações dos bits de entrada, conforme exemplificado a seguir para a função x → y implementada anteriormente. Note que as 4 combinações possíveis das entradas x e y são geradas entre as linhas `process` e `end process`, com um intervalo de 20 ns entre elas.

```
-------------- Simulação da porta x->y implementada -------------
entity sim_impl is
end sim_impl;
architecture sim_impl_op of sim_impl is
component porta_implica is
   port (x,y: in BIT;
         z: out BIT);
end component;
signal x_sim,y_sim,z_sim: BIT;
begin
   U1: porta_implica port map(x_sim,y_sim,z_sim);
   process
   begin

      x_sim <= '0';
      y_sim <= '0';
      wait for 20 ns;

      x_sim <= '0';
      y_sim <= '1';
      wait for 20 ns;

      x_sim <= '1';
      y_sim <= '0';
      wait for 20 ns;

      x_sim <= '1';
      y_sim <= '1';
      wait for 20 ns;

      x_sim <= '0';
      y_sim <= '0';
      wait;

   end process;
end sim_impl_op;
```

A partir da unidade de simulação `sim_impl` descrita acima, pode-se então analisar o funcionamento do circuito `porta_implica` com o auxílio de um programa de simulação adequado. Os diagramas temporais apresentados na figura 1 foram gerados pelo compilador/simulador VHDL da *Blue Pacific Computing*. Note que a saída z_{sim} relaciona-se aos dois sinais x_{sim} e y_{sim} exatamente como inidica a tabela 1.

É preciso notar que, nos diagramas da figura 1, a saída z responde instantaneamente a quaisquer variações dos bits de entrada. Em outras palavras, não se considera o atraso de propagação das portas lógicas, já que não se fez qualquer referência a isso na descrição VHDL da unidade porta implica.

É fundamental, no entanto, que se possa considerar este atraso durante o processo de simulação, já que ele jamais pode ser totalmente eliminado de um circuito digital real, e já que em algumas situações práticas ele é de grande relevância, sobretudo quando se consideram circuitos sequenciais. A linguagem VHDL permite incluir implicitamente na descrição de uma unidade um atraso quando da atribuição de um valor lógico a um sinal qualquer.

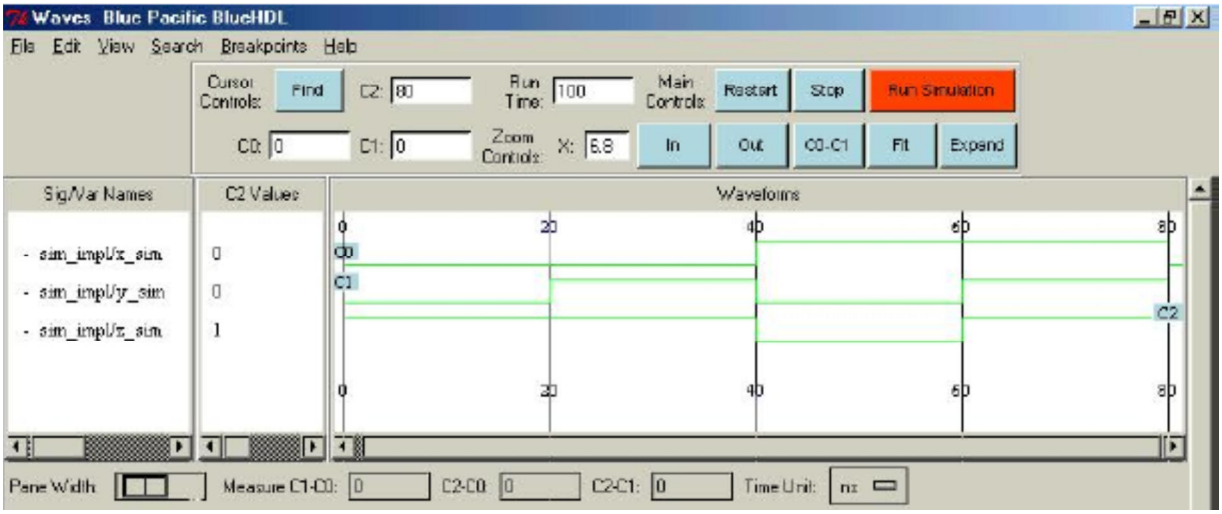


Figura 1 - Resultado da simulação em VHDL da função implicação, desconsiderando-se os tempos de atraso das portas lógicas.

No exemplo da unidade `porta_implica`, pode-se, por exemplo, introduzir um tempo de 10 ns anteriormente a qualquer mudança de nível da saída z, por meio da palavra-chave `wait` (isto representa um atraso de propagação total de 10 ns, mas o atraso pode ser atribuído individualmente às portas). Obtém-se então a seguinte descrição:

```
-------------------- Definição da porta x->y --------------------
entity porta_implica is                  -- definição da estrutura
                                         -- de entradas e saídas.
   port (x,y: in BIT;
         z: out BIT);
end porta_implica;

architecture porta_implica_op of porta_implica is    -- descrição
                                                     -- operacional.
begin
   z <='0' after 10 ns when x='1' and y='0' else
       '1' after 10 ns;
end porta_implica_op;
```

A figura 2 apresenta os diagramas temporais obtidos quando da simulação da nova unidade porta_implica, considerando um tempo de propagação total de 10 ns. Observe-se que, sempre que há uma mudança no nível lógico z, ela ocorre exatamente 10 ns após a variação correspondente do bit dos bits de entrada.

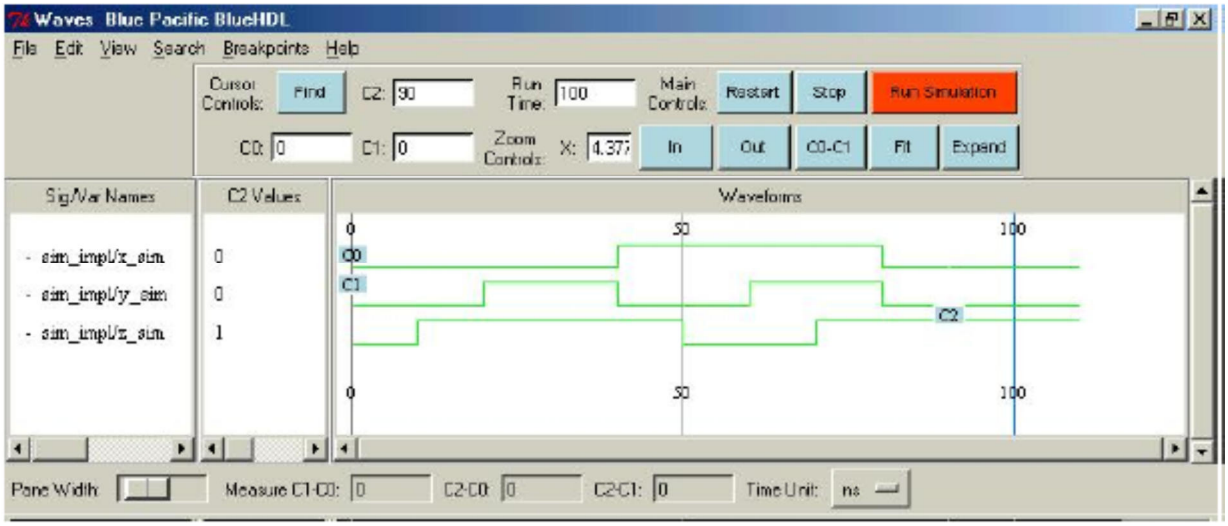


Figura 2 - Resultado da simulação em VHDL da função implicação, considerando-se um atraso de propagação total de 10 ns.

3 INSTRUÇÕES PARA UTILIZAÇÃO DO SOFTWARE DE SIMULAÇÃO VHDL SIMILI

Faça download do software VHDL Simili no Moodle e instale-o, seguindo as instruções fornecidas no arquivo `__Leia-me.txt`.

Para utilização do software, siga os seguintes passos:

1) Crie uma pasta em C:\Alunos com o seu número de matrícula. Exemplo: C:\Alunos\0912345\

2) No Desktop, clique no ícone VHDL Simili 3.1.

3) Se houver algum projeto aberto, clique em File\Close Workspace, e a seguir em Window\Close All

4) Clique em File\New Workspace. Em Workspace location, digite o nome do diretório que você criou (C:\Alunos\0912345), ou use o ícone de "explorar". Em Workspace name, digite um nome com até 8 caracteres, todos em letra minúscula. Exemplo: minhaxor.

5) Clique em File\New. Isto abrirá um editor de texto. Digite seu código VHDL.

6) Clique em File\Save As, e salve na pasta que você criou. Use a extensão .vhdl.

7) Clique em Compile\Compile file 'minhaxor.vhdl' (ou o nome que você usou para o arquivo). O programa perguntará se você deseja adicionar o arquivo ao seu projeto. Responda 'Yes'.

8) Verifique se houveram erros de compilação. Corrija os erros e tente novamente.

9) Clique em Simulate\Restart. O programa perguntará qual a entidade "toplevel". Esta deve ser a entidade principal, ou seja, aquela que utiliza as outras entidades. Por exemplo, se o elemento simular_minhaxor utiliza a entidade minhaxor (para testar todas as combinações de entrada e saída), então a entidade toplevel deve ser simula_minhaxor, e não minhaxor. Além disso, Tome cuidado para selecionar a entidade (ícone vermelho c/ a letra 'e'), e não a arquitetura (ícone verde c/ a letra 'a').

10) Na janela 'Scope', onde aparece uma lista de sinais ('Signals'), selecione os sinais que deseja visualizar (todos?) e arraste-os para a lista de signals próxima à tela preta (onde será desenhada a forma de onda).

11) Clique em Simulate\Run All (Ctrl-F6) para simular todo o seu sistema. Você também pode utilizar Simulate\Run (F6) para executar a sequência de combinações passo-a-passo. Para isso, dê um 'Restart' novamente (F5).

12) Confira se as formas de onda estão de acordo com o esperado. Caso contrário, corrija o seu código e tente novamente.

13) Feche seu projeto e todas as janelas. Clique em File\Close Workspace, e a seguir em Window\Close All.

4 PARTE EXPERIMENTAL

4.1 Função X→Y

Simule o código da função X→Y apresentado na seção 2.

4.2 NÃO-OU-EXCLUSIVO

Escreva na forma soma de produtos minimizada a função $\overline{X \oplus Y}$ descrita pela tabela-verdade a seguir (tabela 2). Em seguida, obtenha a implementação correspondente em VHDL, nas seguintes situações:

- considerando todas as portas lógicas presentes ideais, sem atraso de propagação; e

- considerando um tempo de atraso de 8 ns associado a cada porta.

Finalmente, simule as duas realizações da função. Para isso, escreva o código VHDL que gera todas as combinações dos dois sinais de entrada, na sequência 00 - 01 - 10 - 11 e com uma duração de 50 ns para cada combinação.

Compare os diagramas de tempo que descrevem a relação entre os sinais de entrada e de saída nas duas situações.

Tabela 2 - Tabela-verdade da NÃO-OU-EXCLUSIVO.

X	Y	$\overline{X \oplus Y}$
0	0	1
0	1	0
1	0	0
1	1	1

4.3 Somador parcial

Um somador parcial, ou meio somador, é um circuito que recebe dois bits na entrada e retorna na saída a soma binária correspondente, com dois bits.

1) Preencha a tabela verdade que descreve o somador parcial.

2) Descreva em VHDL o somador parcial. Designe os sinais de entrada por A_i e B_i e os de saída por C_j e S_i, com C_j o mais significativo. Considere um tempo de atraso de 10 ns associado a cada porta.

3) Simule o circuito a partir da descrição obtida no item anterior. Para isso, gere em VHDL todas as combinações de entrada possíveis, com duração de 0,8 μs para cada uma delas. Analise os diagramas de tempo obtidos.

4.4 Somador completo

A figura 3 ilustra um somador completo. Trata-se de um circuito que recebe na entrada três bits, designados por A_i, B_i e C_i, e retorna na saída sua soma, representada pelos bits $C_{i+1} = C_j$ e S_i.

1) Preencha a tabela verdade que descreve o somador completo da figura 3.

2) Apresente o esquemático de um circuito somador completo com apenas seis portas lógicas.

3) Descreva em VHDL o somador completo obtido no item anterior. Adote para os sinais de entrada e saída as mesmas designações da figura 3. Considere um atraso de 10 ns associado a cada porta.

4) Simule o circuito a partir da descrição do item anterior. Para isso, gere em VHDL todas as combinações de entrada possíveis, com duração de 1 μs para cada uma delas. Analise os diagramas de tempo obtidos.

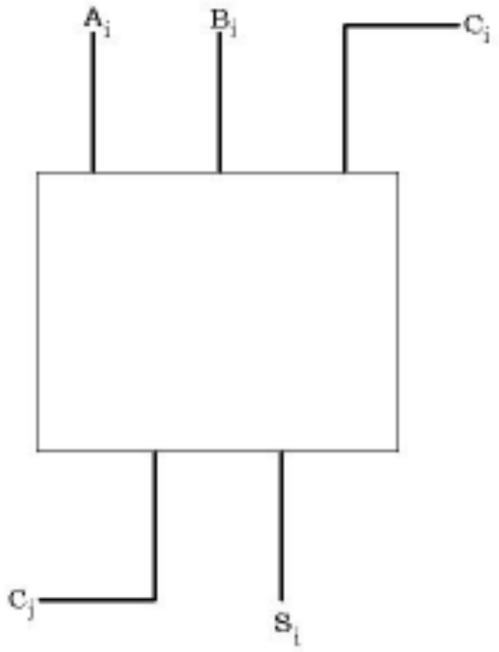


Figura 3 - Representação esquemática de um circuito somador completo.

4.5 Somador total de duas palavras de 3 bits

Um somador total de duas palavras de *n* bits pode ser obtido conectando-se exatamente *n* somadores completos como o ilustrado na figura 3.

1) Apresente o esquemático de um circuito que soma duas palavras A e B de 3 bits cada; utilize no projeto apenas os blocos descritos pela figura 3. Para isso, note que o bit $C_j = C_{i+1}$ deve ser interpretado como um "vai-um" enviado para a casa binária imediatamente mais significativa; da mesma forma, C_i é o bit de "vai-um" recebido da casa imediatamente menos significativa.

2) Descreva em VHDL o circuito representado pelo esquemático no item anterior. Use como componente VHDL o módulo básico "somador completo" implementado na seção 4.4 (figura 3). O circuito descrito deve apresentar como entrada, além dos 6 bits de A e B, um bit de "vai-um" (C_{in}), bem como deve fornecer na saída os 3 bits da soma, S, além de um quarto bit, também de "vai-um" (C_{out}). O ojetivo é permitir sua ligação a outros circuitos de mesmo tipo, para constituir somadores de mais bits.

3) Simule o circuito a partir da descrição anterior. Na simulação, faça pelo menos 5 combinações diferentes. Utilize pelo menos 3 casos em que ocorra a utilização do "vai-um".

5 INSTRUÇÕES PARA A REALIZAÇÃO DO EXPERIMENTO

5.1 Pré-relatório

O projeto a ser apresentado no início da aula deve conter:

1) Leia a Introdução. Digite (copie) e entenda o código VHDL que implementa a porta x→y descrita no roteiro. Traga o código em um pen drive ou CD. Lembre-se de incluir tanto a entidade básica (porta_implica) quanto a entidade de simulação (sim_impl), e de digitar o código referente tanto à descrição das entidades quanto das arquiteturas. No início da aula, simule esse código no VHDL Simili. Este será o primeiro visto do experimento.
2) Apresente a equação booleana que implementa a função $\overline{X \oplus Y}$ descrita na tabela 2, na forma de soma de produtos minimizada. Apresente também o diagrama lógico do circuito que você montaria na protoboard para implementar essa equação.
3) Apresente a tabela verdade do somador parcial (2 entradas e 2 saídas), descrito no item 4.3. Quais são as portas lógicas (and, or, nand, nor, xor, etc.) que implementam cada uma das saídas?
4) Apresenta a tabela verdade do somador completo (3 entradas e 2 saídas) descrito no item 4.4. Apresente o mapa de Karnaugh de cada uma das saídas, e as equações booleanas minimizadas. Se não for possível minimizar algum dos dois mapas de Karnaugh, verifique a possibilidade de se utilizar portas OU-EXCLUSIVO de 2 entradas. Apresente o esquemático desse circuito, usando no máximo 6 portas lógicas.
5) Pense e responda: quantos bits de entrada e saída deve ter um circuito capaz de somar duas palavras de 3 bits cada, sem que haja overflow? A seguir, apresente o diagrama de blocos do somador de palavras de 3 bits descrito na seção 4.5. Utilize no projeto apenas os blocos descritos na figura 3 (nenhuma porta lógica adicional). Dica: Você poderia implementar esse somador usando apenas somadores completos (como é solicitado acima), ou utilizando tanto somadores completos quanto somadores parciais. Na verdade, você pode transformar o somador completo em um somador parcial. Para isso, basta eliminar uma de suas entradas, conectando a ela um valor fixo que não altere o resultado final. Assim, é possível implementar o circuito usando apenas somadores completos.

5.2 Visto

Para conseguir o visto o aluno deve realizar o procedimento descrito nos itens 4.1, 4.2, 4.3, 4.4 e 4.5, e apresentar os códigos VHDL. Cada item terá 20% do valor do visto do experimento. Para a realização deste experimento, o aluno pode utilizar o programa VHDL Simili, disponibilizado no Moodle. A seção 3 traz instruções de uso desse programa. Instruções mais detalhadas estão disponíveis no capítulo 8 da documentação do programa (incluído com o instalador). Se desejado, o grupo pode optar por outro programa para simulação de VHDL.

5.3 Relatório

O relatório é individual, deve ser feito à mão, e consiste em responder ao questionário abaixo. Não é necessário entregar um relatório formal, com introdução, metodologia, resultados, etc.

1) Explique o que é a linguagem VHDL e descreva algumas vantagens da mesma para o projeto de circuitos.

2) Cite ao menos três situações nas quais o VHDL e o uso de lógicas programáveis podem ser úteis.

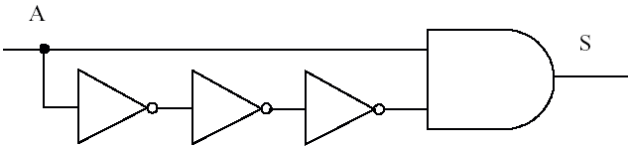
3) Explique resumidamente as partes de um código VHDL (entidade e arquitetura).

4) Suponha que você já criou um componente com a seguinte interface:

```
entity ou_exclusivo is
   port(A,B: in BIT; S: out BIT);
end ou_exclusivo;
```

Usando somente esse componente e alguns sinais, desenvolva agora o código VHDL de uma entidade (com a respectiva arquitetura) que implemente a porta OU-EXCLUSIVO de 4 entradas.

5) Desenvolva em VHDL um componente com um 1 pino de entrada e 1 pino de saída cuja arquitetura implemente o circuito abaixo. Não se esqueça de indicar um atraso de 10 ns para cada porta. Para tal, utilize sinais onde for necessário.



6) Implemente agora uma entidade para testar o componente da questão 5. Faça com que o pino A fique 100 ns em nível baixo, mais 100 ns em nível alto e mais 100 ns em nível baixo novamente. Não se esqueça de incluir um comando para que o simulador pare ao final do processo, ao invés de voltar para o início.

7) Se você rodasse a entidade da questão 6 no VHDL Simili, quais formas de onda você esperaria ver na simulação? Faça um esboço.

8) Suponha que o somador parcial implementado por você tenha a seguinte interface:

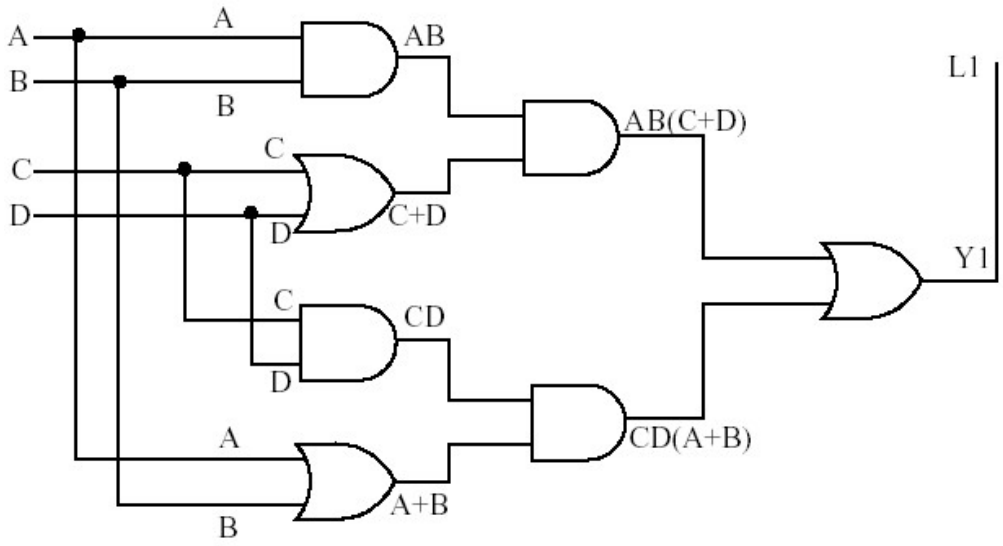
```
entity somador_parcial is
  port(A,B: in BIT; S,Co: out BIT);
end somador_parcial;
```

a) Mostre, na forma de diagrama de blocos, como você poderia implementar um somador completo usando apenas somadores parciais e alguns sinais. Obs.: não é necessário nenhuma porta lógica adicional.
b) A seguir, complete o código VHDL abaixo:

```
entity somador_completo is
   port(A,B,Ci: in BIT; S,Co: out BIT);
end somador_completo;

architecture somador_completo_op of somador_completo is
component
   ___________________________________
end component;
signal ________________
begin
   U1: _____________ port map(_________);
   U2: _____________ port map(_________);
   U3: _____________ port map(_________);
end somador_completo_op;
```

9) Desenvolva em VHDL um código com 4 pinos de entrada e 1 pino de saída cuja arquitetura implemente o circuito a seguir. Para tal, utilize sinais onde for necessário.



10) Suponha agora que você disponha dos componentes `and` e `or` descritos pelas interfaces a seguir:

```
entity and2 is
   port(A,B: in BIT; S: out BIT);
end and2;

entity or2 is
   port(A,B: in BIT; S: out BIT);
end or2;
```

Desenvolva um código usando esses componentes para implementar o circuito da questão 9.

Atenção: Alunos que apresentarem códigos iguais serão reprovados na disciplina e o fato será comunicado à coordenação do curso para que as medidas disciplinares cabíveis sejam tomadas.