Experimento 2 – Material Auxiliar INTRODUÇÃO À LINGUAGEM VHDL E AO MODELSIM

1 – Linguagem VHDL

1.1 – Estrutura básica e atribuições concorrentes

VHDL é uma linguagem empregada na descrição de circuitos digitais. Ela pode ser utilizada para projetar, simular e sintetizar circuitos digitais desde os mais simples até microprocessadores completos.

A estrutura básica da descrição de uma unidade em VHDL é bem simples e é dividida em duas partes. Na primeira, *entity declaration*, definimos as entradas e as saídas dos circuitos, sem especificar ainda quais as relações entre elas. Na segunda parte, *architecture definition*, especificamos o comportamento do circuito. Para os nossos primeiros experimentos, vamos utilizar apenas entradas e saídas dos tipos STD_LOGIC e STD_LOGIC_VECTOR que são usados para representar bits e vetores de bits, respectivamente. Estes tipos estão definidos no módulo STD_LOGIC_1164 da biblioteca IEEE, e, portanto, temos que importar o módulo para usá-los

Como um primeiro exemplo, vamos escrever um código para implementar um circuito com três bits de entrada (A, B e C) e dois bits de saída (X e Y) que realize as funções lógicas $X = \bar{A}C + AB$ e $Y = \bar{A}C + \overline{AB}$. Para a entidade, podemos usar o código mostrado na Listagem 1.

Listagem 1 - Código VHDL para a declaração da entidade do primeiro exemplo

```
1  LIBRARY IEEE;
2  USE IEEE.STD_LOGIC_1164.ALL;
3
4  -- Declaração da entidade
5  ENTITY MEUCIRCUITO IS
6      PORT (A, B, C: IN STD_LOGIC;
7            X, Y: OUT STD_LOGIC);
8  END MEUCIRCUITO;
```

Note que as duas primeiras linhas na Listagem 1 são usadas para importar todas (ALL) as estruturas do módulo STD_LOGIC_1164 da biblioteca IEEE. Em seguida, é declarada a entidade com nome MEUCIRCUITO e definidas as entradas e as saídas. Atenção: o código VHDL não diferencia letras maiúsculas de minúsculas e os comentários são iniciados por dois traços, como na linha 4.

Para definir a relação entre as entradas e saídas, vamos usar atribuições concorrentes na arquitetura. O código completo é mostrado na Listagem 2. Note que os nomes da entidade, da arquitetura e das variáveis são arbitrários, desde que comecem com letra, contenham apenas letras e números e não sejam palavras reservadas.

Listagem 2 - Código completo para o primeiro exemplo

```
1  LIBRARY IEEE;
2  USE IEEE.STD_LOGIC_1164.ALL;
3
4  ENTITY MEUCIRCUITO IS
5      PORT (A, B, C: IN STD_LOGIC;
6            X, Y: OUT STD_LOGIC);
7  END MEUCIRCUITO;
8
9  ARCHITECTURE MEUCIRCUITO_ARCH OF MEUCIRCUITO IS
10 BEGIN
11     X <= (NOT(A) AND C) OR (A AND B);
12     Y <= (NOT(A) AND C) OR NOT(A AND B);
13 END MEUCIRCUITO_ARCH;
```

As atribuições serem concorrentes significa que elas ocorrem ao mesmo tempo. Portanto, se invertêssemos as linhas 11 e 12 da Listagem 2, o resultado seria exatamente o mesmo.

1.2 – Signals

Poderíamos obter o mesmo resultado da Listagem 2 usando sinais (*signals*) dentro da arquitetura. Eles são variáveis que podem ser acessadas dentro da arquitetura e não são nem saídas nem entradas da entidade.

Por exemplo, poderíamos substituir a arquitetura MEUCIRCUITO_ARCH da Listagem 2 pela arquitetura MEUCIRCUITO_ARCH2 da Listagem 3. Note que os sinais são declarados antes da palavra BEGIN e são do tipo STD_LOGIC, assim como as entradas e saídas. Novamente, note que as atribuições entre as palavras BEGIN e END são concorrentes, ou seja, as linhas 4, 5, 6 e 7 podem ser colocadas em qualquer ordem e o resultado será o mesmo.

Listagem 3 - Arquitetura usando sinais

```
1  ARCHITECTURE MEUCIRCUITO_ARCH2 OF MEUCIRCUITO IS
2  SIGNAL FIO1, FIO2: STD_LOGIC;
3  BEGIN
4      FIO1 <= NOT(A) AND C;
5      FIO2 <= A AND B;
6      X <= FIO1 OR FIO2;
7      Y <= FIO1 OR NOT(FIO2);
8  END MEUCIRCUITO_ARCH2;
```

Para entender melhor a lógica usada para escrever o código da Listagem 3, veja a Figura 1. Note que, neste circuito que implementa as mesmas funções lógicas que o código VHDL, “Fio 1” e “Fio 2” não são entradas nem saídas do circuito, mas sinais internos.

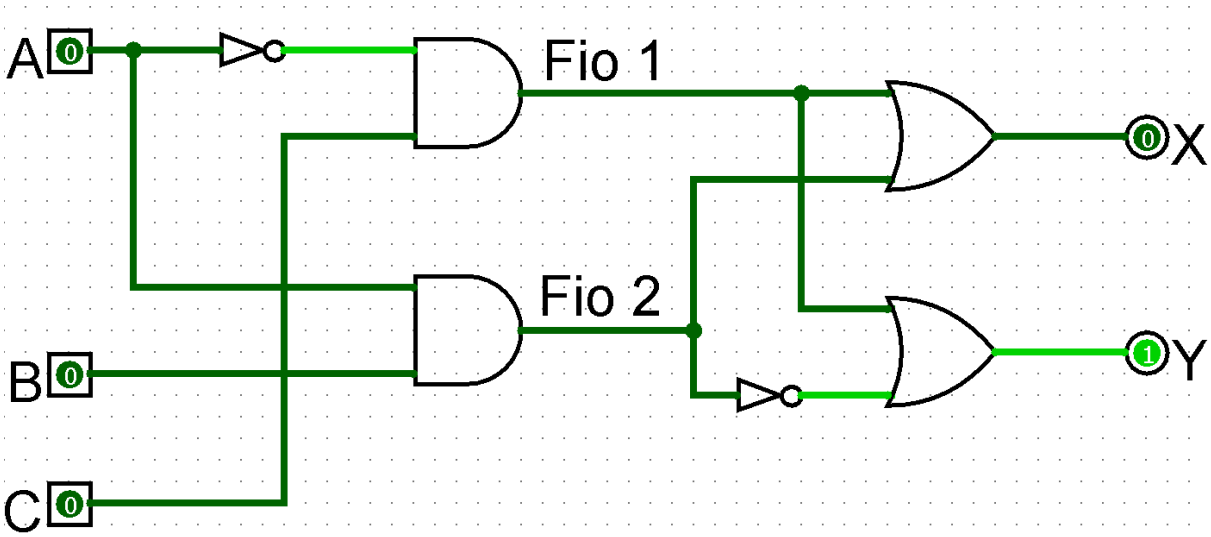

Figura 1 - Esquemático do circuito usando portas lógicas

1.3 – Vetores

Até agora, todas as nossas variáveis foram de um bit apenas, mas podemos ter palavras de vários bits usando o tipo STDO_LOGIC_VECTOR. Como exemplo, considere o código da Listagem 4.

Listagem 4 - Exemplo com vetores

```
1  LIBRARY IEEE;
2  USE IEEE.STD_LOGIC_1164.ALL;
3
4  ENTITY MEUCIRCUITO2 IS
5      PORT (A, B, C, D, E, F: IN STD_LOGIC;
6            Y: OUT STD_LOGIC_VECTOR(2 DOWNTO 0));
7  END MEUCIRCUITO2;
8
```

```
9  ARCHITECTURE MEUCIRCUITO2_ARCH OF MEUCIRCUITO2 IS
10 SIGNAL AUX1: STD_LOGIC_VECTOR(2 DOWNTO 0);
11 SIGNAL AUX2: STD_LOGIC_VECTOR(0 TO 2);
12 BEGIN
13     AUX1 <= A & B & C;
14     AUX2 <= D & E & F;
15     Y <= AUX1 OR AUX2;
16 END MEUCIRCUITO2_ARCH;
```

Há vários detalhes a se comentar na Listagem 4. Primeiro, note que as variáveis AUX1, AUX2 e Y são vetores de três bits, porém, AUX2 é declarada com índices crescentes (0 TO 2) e as outras são declaradas com índices decrescentes (2 DONWTO 0). Isso significa que o bit mais à esquerda de AUX1 é acessado com índice 2 (AUX1(2)), enquanto o bit mais à esquerda de AUX2 é AUX2(0). Mais ainda, o caractere & é usado para concatenar bits. Por fim, operações lógicas como AND, OR e NOT são aplicadas bit a bit. Portanto, o código da Listagem 4 é equivalente ao da Listagem 5.

Listagem 5 - Código equivalente ao da Listagem 4, mas operando em um bit por linha

```
1  LIBRARY IEEE;
2  USE IEEE.STD_LOGIC_1164.ALL;
3
4  ENTITY MEUCIRCUITO2 IS
5      PORT (A, B, C, D, E, F: IN STD_LOGIC;
6            Y: OUT STD_LOGIC_VECTOR(2 DOWNTO 0));
7  END MEUCIRCUITO2;
8
9  ARCHITECTURE MEUCIRCUITO2_ARCH2 OF MEUCIRCUITO2 IS
10 SIGNAL AUX1: STD_LOGIC_VECTOR(2 DOWNTO 0);
11 SIGNAL AUX2: STD_LOGIC_VECTOR(0 TO 2);
12 BEGIN
13     AUX1(2) <= A;
14     AUX1(1) <= B;
15     AUX1(0) <= C;
16     AUX2(0) <= D;
17     AUX2(1) <= E;
18     AUX2(2) <= F;
19     Y(2) <= AUX1(2) OR AUX2(0);
20     Y(1) <= AUX1(1) OR AUX2(1);
21     Y(0) <= AUX1(0) OR AUX2(2);
22 END MEUCIRCUITO2_ARCH2;
```

Também podemos atribuir valores 1 e 0 (não variáveis) para as variáveis. Por exemplo, AUX1(2) <= ‘1’ ou AUX1 <= “110”. Note que são usadas aspas simples para bits e aspas duplas para vetores. Note que, como AUX1 é declarado com índices decrescentes (2 DOWNTO 2), então AUX1 <= “110” faz com que o valor de AUX1(2) seja

1 e o de AUX1(0) seja 0. De forma similar, AUX2 <= “110” faz com que AUX2(0) seja 1 e AUX2(2) seja 0, pois AUX2 foi declarado com índices crescentes.

2 – ModelSim

Vamos usar o ModelSim para simular nossos códigos VHDL. Siga esta seção como um tutorial para aprender o básico do programa.

Após abrir o ModelSim, siga os seguintes passos para criar e simular um projeto em VHDL:

1. Conforme a Figura 2, clique em File > New > Project...

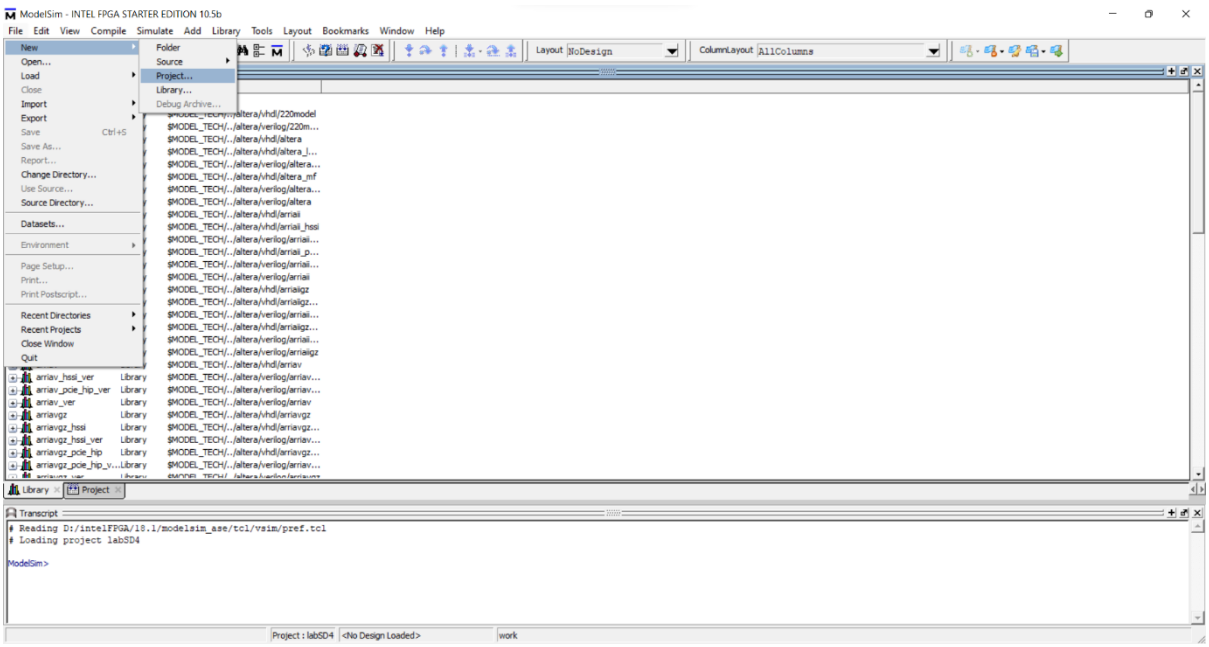

Figura 2 - Criar novo projeto

2. Escolha o nome do projeto e a pasta onde ele será salvo, veja a Figura 3, não mexa nas duas últimas configurações. Clique em OK.

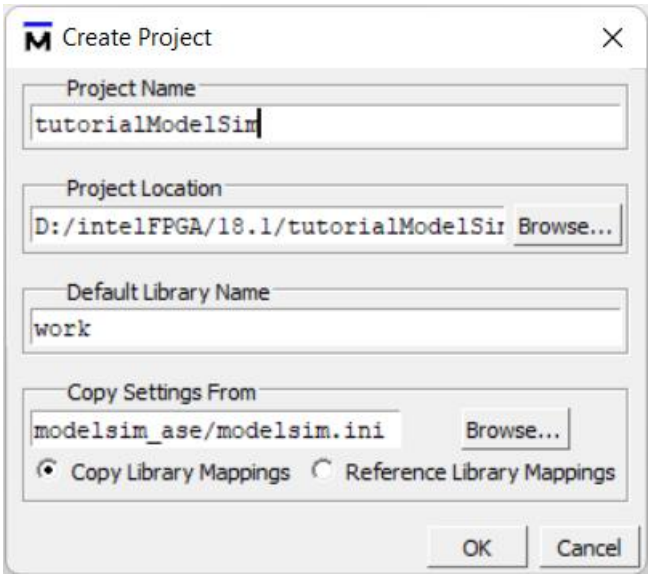

Figura 3 - Escolher o nome e a pasta do projeto

3. Uma nova janela será aberta. Escolha “Create New File” (veja a Figura 4).

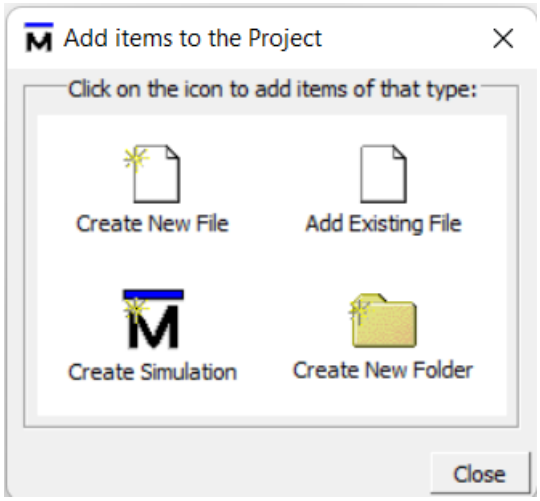

Figura 4 - Inserir itens no projeto

4. Escolha o nome do arquivo e escolha o tipo VHDL conforme a Figura 5.

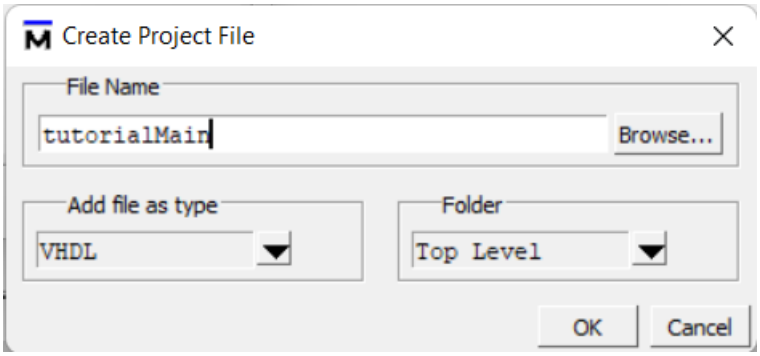

Figura 5 - Criar arquivo VHDL

5. O arquivo deve aparecer na aba “Project”. Abra-o e escreva o código VHDL conforme a Figura 6 (neste exemplo foi usado o código da Listagem 4). **Atenção**: a partir deste ponto, se uma das abas mostradas nas figuras não aparecer para você, clique no menu “View” na barra superior e marque as abas que você quer visualizar

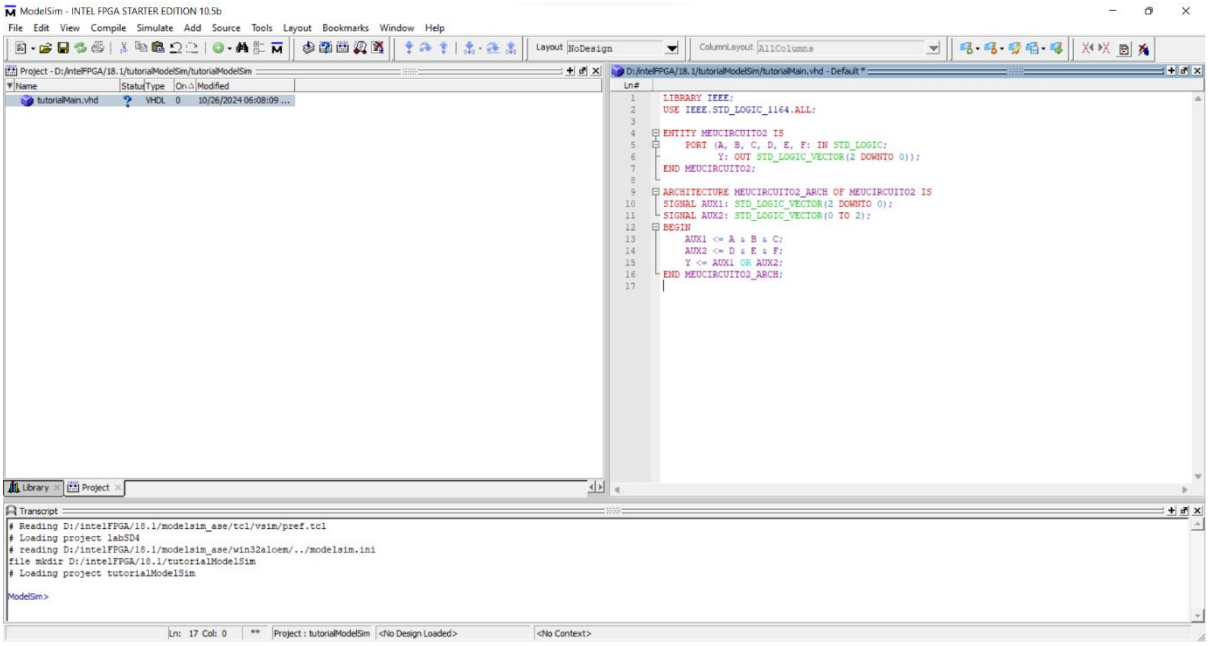

Figura 6 - Edição do código VHDL

6. Salve o arquivo, clique nele com o botão direito e selecione a opção “Compile > Compile Selected” conforme a Figura 7.

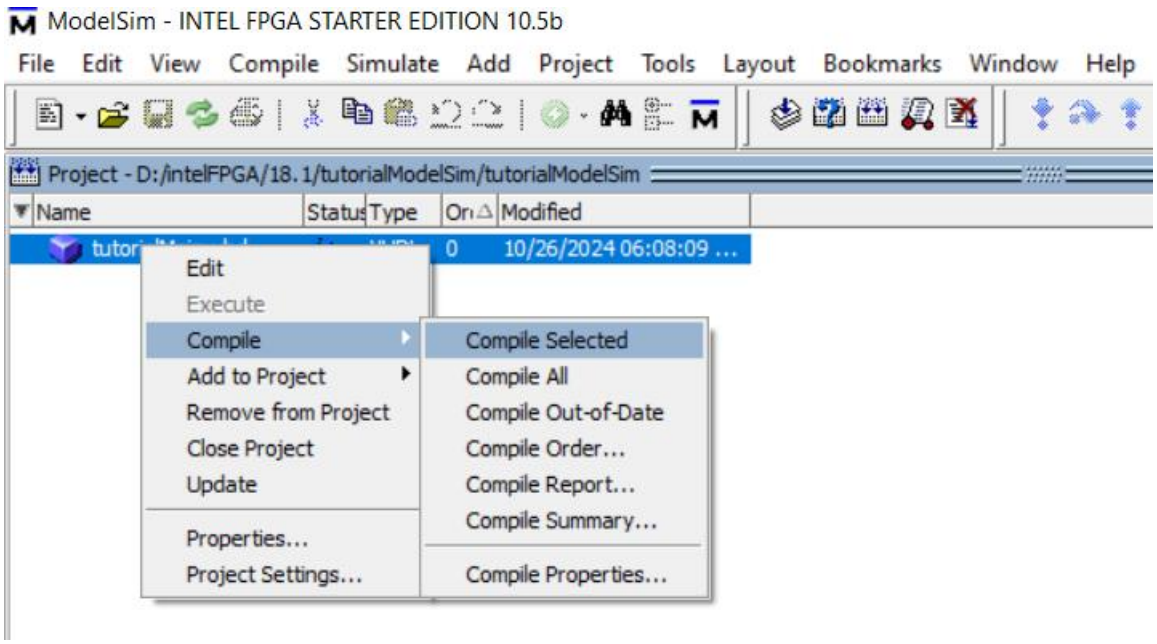

Figura 7 - Compilar arquivo

7. Se tudo estiver certo, você deve ver uma mensagem de sucesso na aba “Transcript”, como na Figura 8. Caso contrário, clique em “Compile > Compile Report...” para ver o que houve de errado, o mais provável é um erro de digitação no código VHDL (lembre-se de salvar o arquivo antes de compilar), e tente de novo.

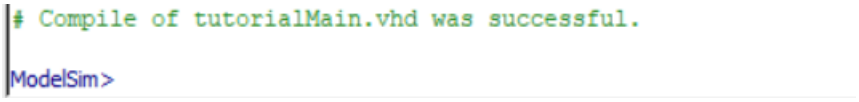

Figura 8 - Mensagem de compilação bem-sucedida

8. Agora, vamos simular o código. Cique em “Simulate > Start Simulation...” conforme a Figura 9.

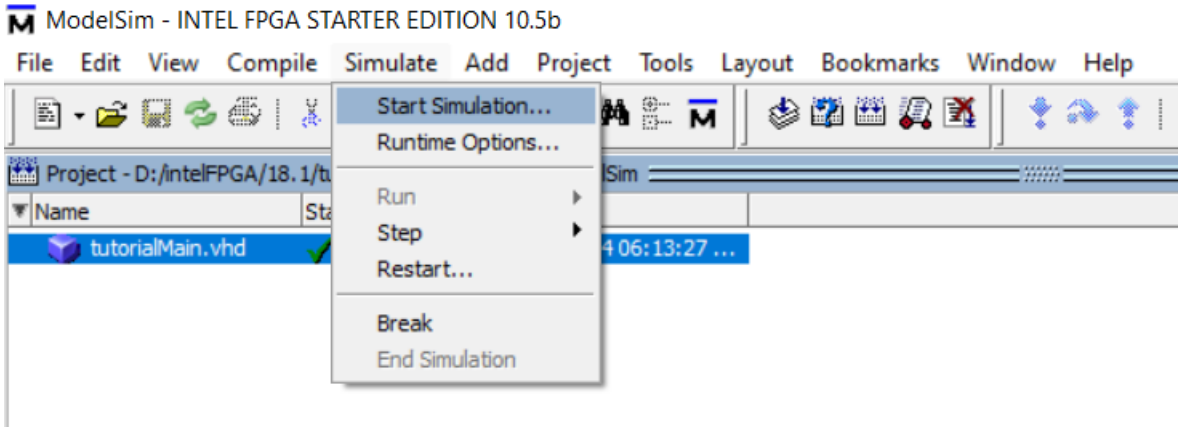

Figura 9 - Iniciar simulação

9. Escolha a arquitetura que você quer simular, por padrão ela está na biblioteca “work” (veja a Figura 10) e clique em OK.

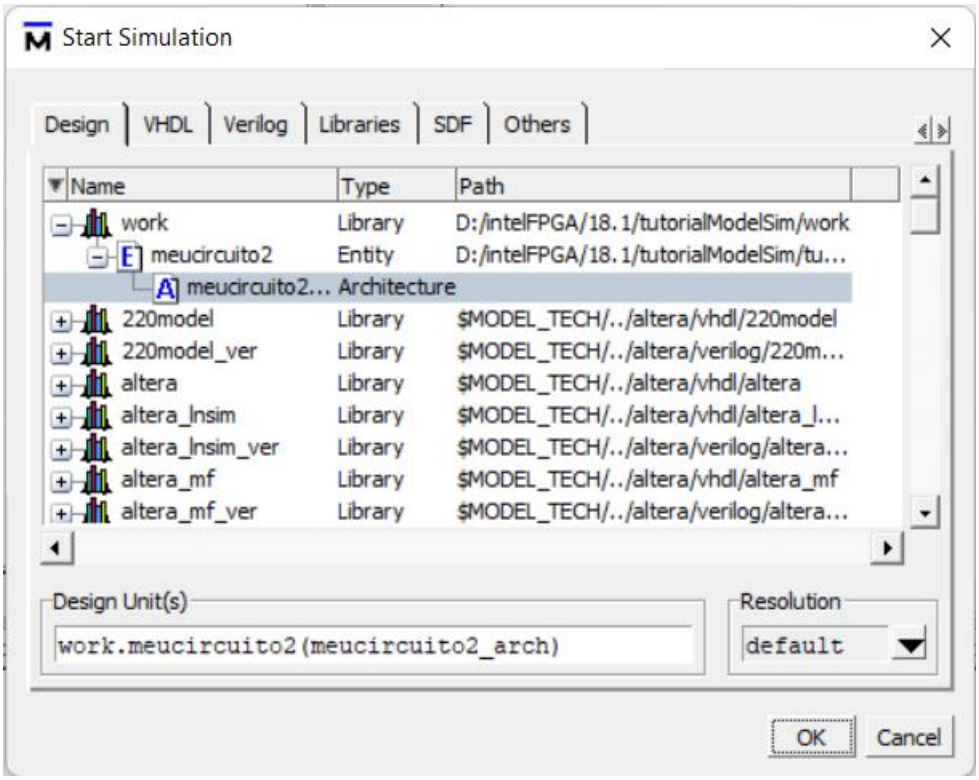

Figura 10 - Escolher arquitetura para simular

10. A janela do ModelSim deve ficar como na Figura 11. Talvez você tenha que clicar na aba “Wave” para ver a ela em vez do código. O valor 100 ps que aparece acima e no centro é o “Run Length”, ele define o tempo que será simulado (talvez o seu esteja com um valor diferente).

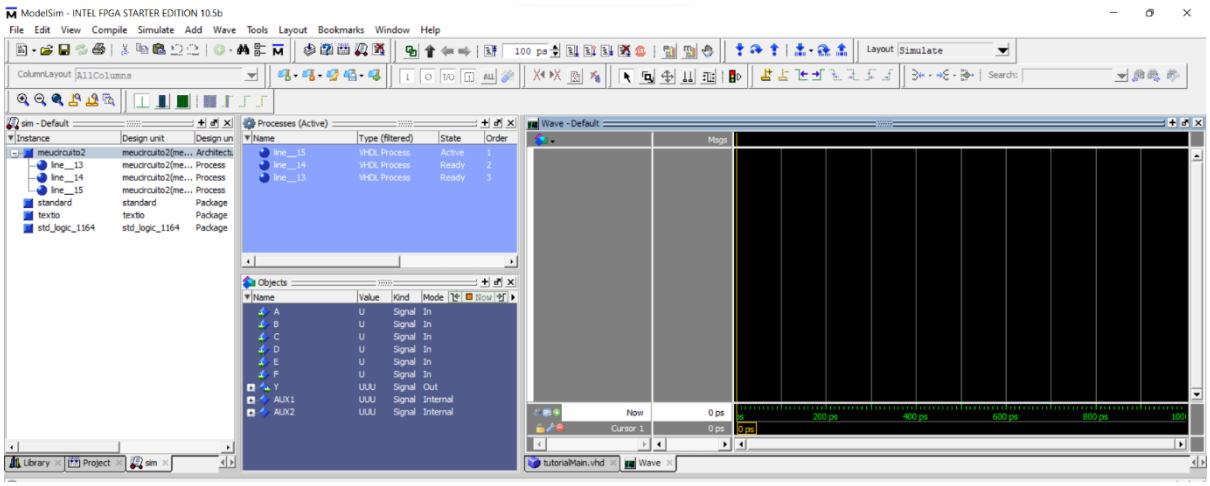

Figura 11 - Tela após escolher o arquivo para simular

11. Os itens na aba “Objects” são as entradas, saídas e sinais do seu circuito. Arraste as variáveis que você quer visualizar para a coluna esquerda da aba “Wave” conforme a Figura 12. Note que as variáveis vetoriais podem ser expandidas para mostrar os bits de forma individual.

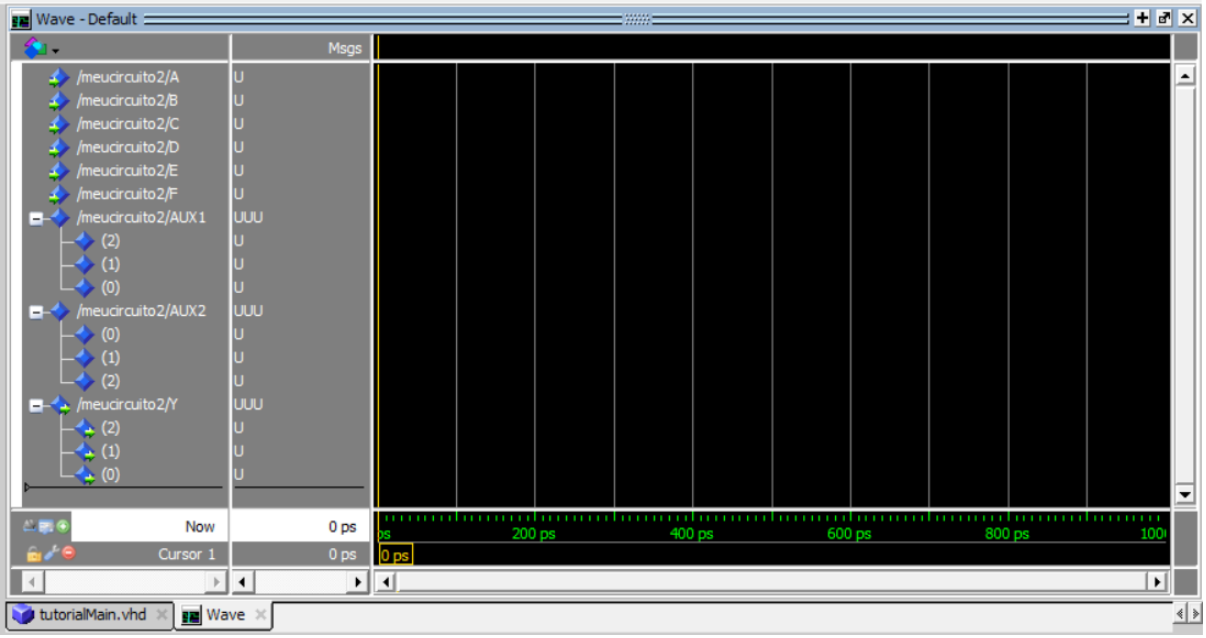

Figura 12 - Aba "Wave" com as variáveis já adicionadas

12. Agora, vamos escolher valores para as entradas. Clique com o botão direito em uma das entradas na aba “Wave” para ver algumas opções (Figura 13). Podemos usar “Force...” para colocar um valor fixo na variável ou “Clock...” para um valor variável no tempo. Observação: na opção “Radix” você pode escolher ver a variável, por exemplo, como um valor decimal em vez de um vetor de binários e na opção “Properties” você pode mudar a cor que será usada para a forma de onda da variável.

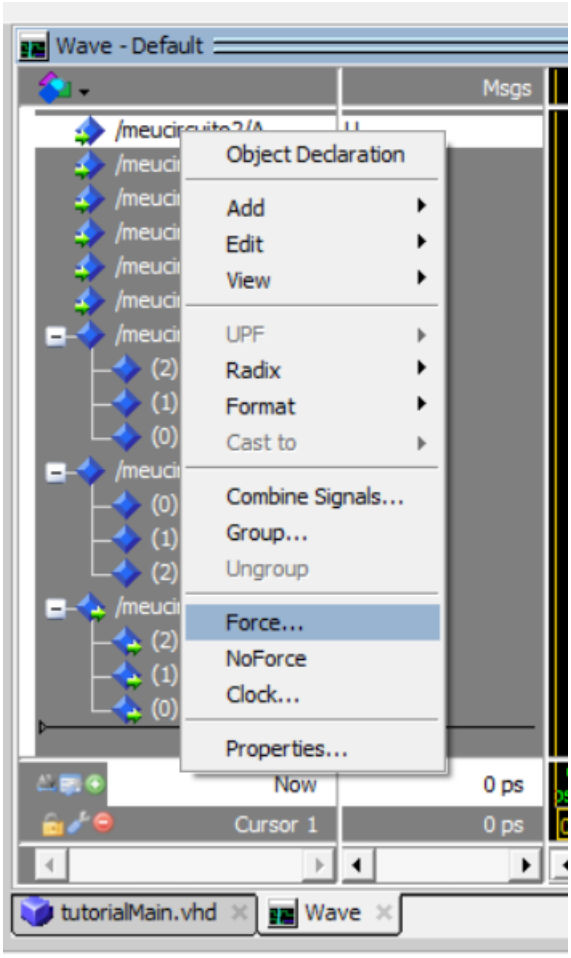

Figura 13 - Opções para as entradas na aba "Wave"

13. Neste exemplo, vamos colocar valores fixos nas entradas ‘A’, ‘B’, ‘C’, ‘D’ e ‘E’ (‘C’ igual a 0 e a outras iguais a 1) e um valor variável na entrada ‘F’. Veja as figuras 14 e 15. Escolha apenas as propriedades “Value” (0 ou 1) e “Period”, deixe as outras nos valores padrão conforme as figuras. O período é medido em picosegundos (ps).

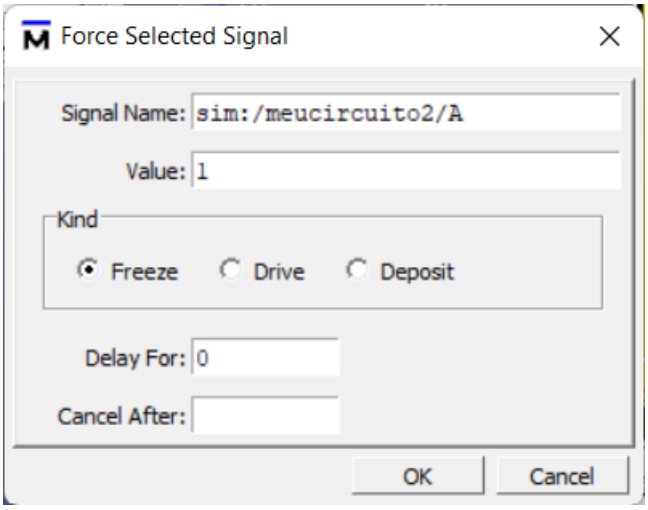

Figura 14 – Entrada com valor fixo

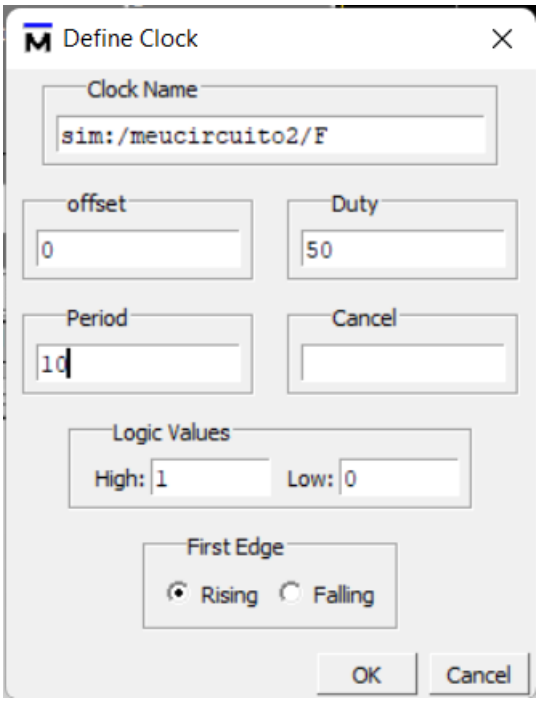

Figura 15 - Entrada com valor variável

14. Para rodar a simulação, clique em “Simulate > Run > Run 100” conforme a Figura 16. Isso vai simular o circuito funcionando pelo tempo definido como “Run Length” (veja o item 10).

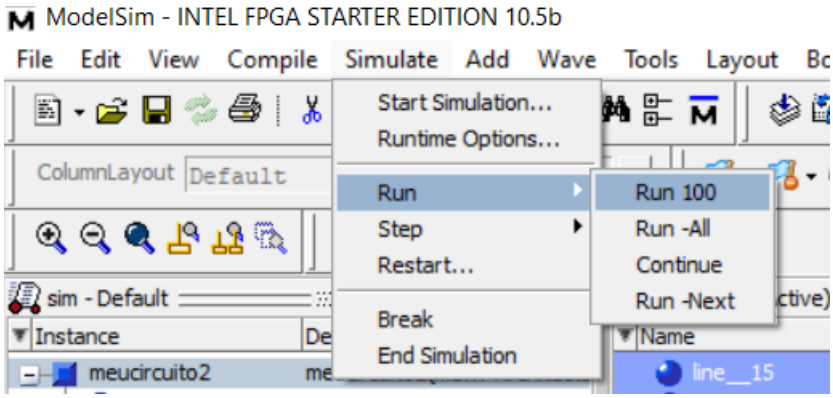

Figura 16 - Rodar simulação

15. As formas de onda para cada variável na aba “Wave” devem aparecer (Figura 17).

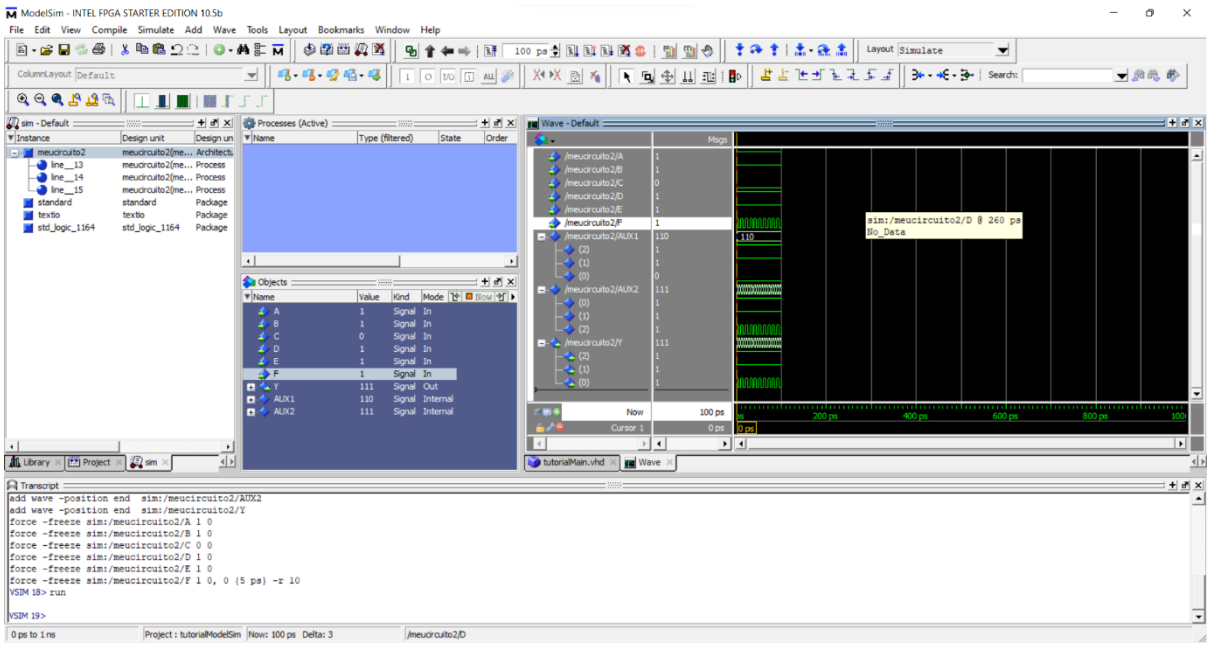

Figura 17 - Tela após rodar a simulação

16. Clique com o botão direito na área vazia da aba “Wave” para ver as opções de zoom (Figura 18).

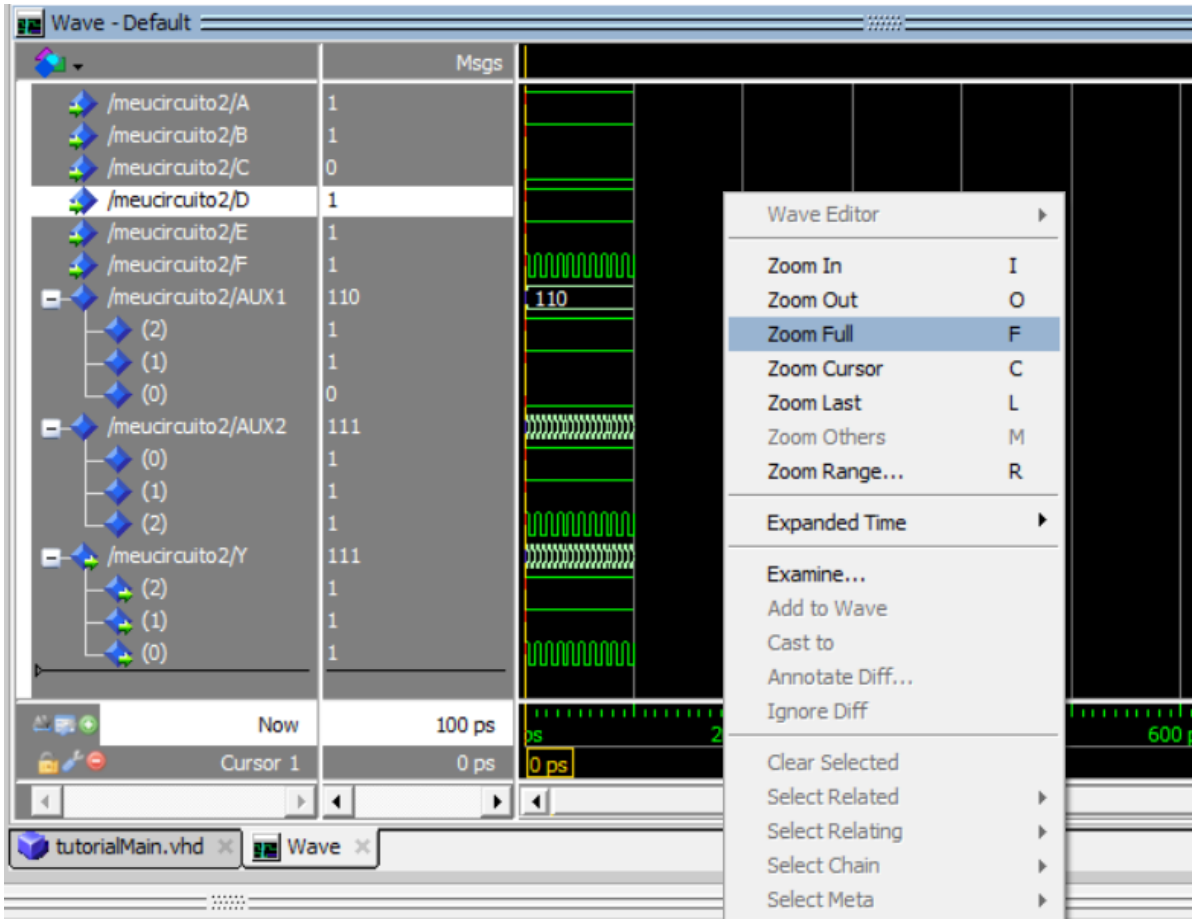

Figura 18 - Opções de zoom

17. Clique em “Simulate > End Simulation” quando quiser encerrar a simulação.

ATENÇÃO! Nos relatórios, as simulações do ModelSim devem ser apresentadas de forma que fique claro o que está acontecendo e como essas simulações mostram que o código desenvolvido faz o que foi pedido pela atividade. Você pode, por exemplo, fazer várias simulações, cada vez mostrando apenas algumas variáveis, e colocar variáveis diferentes com cores diferentes, ou exportar os resultados para criar os gráficos com outro programa (pesquise como fazer). Descreva o que está acontecendo em cada intervalo de tempo caso haja variação temporal dos valores. Em alguns casos, pode ser melhor visualizar um vetor como um hexadecimal ou um decimal em vez de uma sequência de bits.

Dica: pense em como usar “clocks” para as entradas (itens 12 e 13) de forma a mostrar todas as combinações de entradas possíveis em apenas uma simulação.