



Pontifícia Universidade Católica do Rio Grande do Sul
Faculdade de Informática - FACIN

LABORG

Prof. Dr. Rafael Garibotti

OBJETIVOS

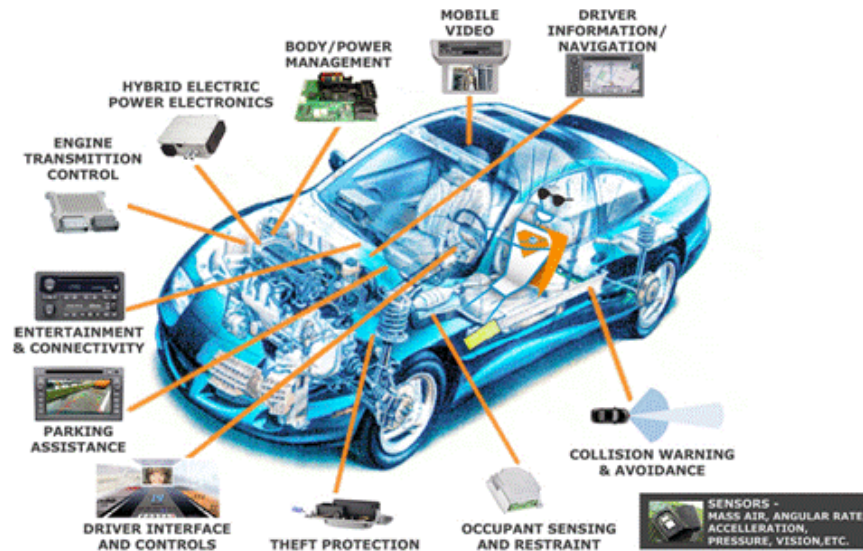
O cumprimento da disciplina busca dar ao aluno, ao final do semestre, condições de:

1. Empregar uma linguagem de descrição de hardware específica para capturar, validar, implementar e prototipar sistemas digitais.
2. Utilizar ferramentas de captura, validação, implementação e prototipação de sistemas digitais computacionais com o auxílio de ferramentas de apoio computacionais.
3. Distinguir bloco de dados e bloco de controle de um computador e/ou processador, quanto à função, composição e construção.

MOTIVAÇÃO



- Software
 - ✓ Aplicações
 - ✓ Sistemas Operacionais
 - ✓ Device Drivers
- Hardware



FPGA Engineer Salary? (Average)

- ✓ Entry level: **\$100,000**
- ✓ Senior: **\$140,000**

COMPETÊNCIAS GLOBAIS

Complementares às competências técnicas (objetivos da disciplina), será incentivado o desenvolvimento de determinadas competências globais nos alunos, tais como:

1. Expressão escrita
2. Expressão oral
3. Autonomia e independência
4. Trabalho em equipe
5. Criticidade
6. Postura e compreensão de seu papel na faculdade e na sociedade



CONTEÚDO

1. Introdução à simulação de circuitos VHDL
2. Introdução a prototipação de hardware com FPGAs
3. VHDL: processos, paralelismo e o comando process
4. Programação em linguagem de montagem de processador MIPS
5. Projeto de um circuito digital de média complexidade
 - ✓ Relógio de Xadrez
6. Prototipação de um processador capaz de executar a maioria das instruções da arquitetura MIPS
 - ✓ Prototipação MR4

AVALIAÇÃO

- Avaliação: **G1** ≥ 7.0, com **G2**

$$G1 = \frac{T1A + T1B + T2A + T2B + 3 * (T3A + T3B)}{10}$$

- As notas dos trabalhos são computadas a partir do desenvolvimento dos trabalhos práticos da disciplina, conforme percentuais divulgados acima.
- Os trabalhos serão realizados **em duplas**, mas a avaliação de conhecimento será individual. Logo, o professor irá questionar cada integrante para a obtenção da nota final de cada trabalho.
- Frequência mínima para aprovação: 75%.
- **PLÁGIO E FRAUDE NÃO SERÃO TOLERADOS!!!**

CRONOGRAMA DOS TRABALHOS

Entrega dos trabalhos:

T1A. 20/08/2017

T1B. 03/09/2017

T2A. 17/09/2017

T2B. 01/10/2017

T3A. 29/10/2017

T3B. 26/11/2017

Observação: Entrega atrasada do trabalho **não será tolerada**. Ou seja, se não entregar até o prazo estipulado, a nota do aluno é **zero**.

RELATÓRIOS

Todos os **trabalhos** são entregues com um relatório. As informações mínimas que devem constar estão citados abaixo:

1. Identificação dos autores.
2. Descrição do problema.
3. Solução encontrada pelo(s) autor(es). Neste item, deve constar comentários pessoais de como resolver o problema, das diversas soluções encontradas se este for o caso, etc.
4. Implementação da solução com descrições passo-a-passo de como foi feito o trabalho. Por exemplo: mostrar a solução VHDL.
5. Simulação com capturas de tela e formas de ondas.

AVALIAÇÃO DOS RELATÓRIOS

10% - Organização na entrega dos arquivos

- ✓ Verificar se os nomes, formatos estão corretos, além de verificar se o relatório está em PDF.

20% - Funcionalidade PARCIAL do trabalho

- ✓ Neste item será avaliado o quanto do trabalho foi efetuado. Em outras palavras, o trabalho será avaliado e considerado as partes corretas, resultando em uma nota proporcional ao trabalho efetuado. Por exemplo, se está funcionando 70% do trabalho final, o aluno receberá por isso, 70% desta nota.

10% - Funcionalidade FINAL do trabalho

- ✓ Verificar se o trabalho final está de acordo com o esperado, se ele está correto. Neste item, não está sendo avaliado os detalhes da implementação.

60% - Qualidade do relatório

- ✓ Neste item será avaliado a qualidade do relatório. Se ele atende aos requisitos mínimos, e o quão detalhado e de fácil entendimento ele está. Será avaliado cada seção, como segue:
 - Descrição do problema e solução encontrada pelo (s) autor (es). Neste item, deve constar comentários pessoais de como resolver, das diversas soluções encontradas se este for o caso, etc.
 - Implementação da solução com descrições passo-a-passo de como foi feito o trabalho. Por exemplo: mostrar a solução VHDL.
 - Simulação com capturas de tela e formas de ondas.

INFORMAÇÕES

- Material disponível somente no Moodle
- Trabalhos serão entregues via Moodle
- Atendimento (dúvidas, provas, trabalhos):
 - ✓ Por email: rafael.garibotti@pucrs.br
 - ✓ Favor, IDENTIFICAR A TURMA acrescentando antes do assunto: **[LABORG590] Assunto**
 - ✓ Fórum Moodle da disciplina
- Metodologia de ensino e andamento das aulas:
 - ✓ Conteúdo e cronograma (Apresentação da teoria)
 - ✓ Aulas práticas visando a entrega dos trabalhos propostos
 - ✓ Comportamento esperado

BIBLIOGRAFIA

BÁSICA:

1. ASHENDEN, P. J. The student's guide to VHDL. Morgan Kaufmann Publishers, Inc. San Francisco, CA, 1998.
2. PATTERSON, D. A., HENNESSY, J. L. Organização e Projeto de Computadores: A interface hardware/software. Livros Técnicos e Científicos S.A., Rio de Janeiro, RJ, 2000.

COMPLEMENTAR:

1. RUSHTON, A. VHDL for logic synthesis. John Wiley & Sons, Inc. Chichester, NY, 1998.
2. CHANG, K. C. Digital design and modeling with VHDL and synthesis. IEEE Computer Society Press. Los Alamitos, CA, 1997.
3. ASHENDEN, P. J. The Designer's Guide to VHDL. Morgan Kaufmann Publishers, Inc. San Francisco, CA, 1996.
4. MAZOR, S., LANGSTRAAT, P. A guide to VHDL. Boston: Kluwer Academic Publishers. Norwell, MA, 1996.

AULA SOBRE:

INTRODUÇÃO A SIMULAÇÃO EM VHDL

INTRODUÇÃO A VHDL

- Uma linguagem para *descrever* sistemas digitais.
- Existem dezenas de outras: **SystemC**, **Verilog**, **Handel-C**, etc...
- Origem:
 - ✓ **VHDL**: **V**HSIC **H**ardware **D**escription **L**anguage.
 - Criado no programa americano “*Very High Speed Integrated Circuits*” (VHSIC), de 1980.
 - Padrão do “*Institute of Electrical and Electronics Engineers*” (IEEE) em 1987, revisado em 1993, 2000, 2002 e 2004.
 - ✓ Feita para *especificar* hardware. Atualmente serve para **simulação** e **síntese**!
 - ✓ Linguagem utilizada mundialmente por empresas de CAD, onde tem grande adoção na Europa. Já **Verilog** é muito usado nos EUA.

BENEFÍCIOS

➤ Especificação do sistema digital:

- ✓ Projetos independentes da tecnologia (implementação física é postergada).
- ✓ Ferramentas de CAD compatíveis entre si.
- ✓ Permite explorar, em um nível mais alto de abstração (em relação a diagramas de esquemáticos) diferentes alternativas de implementação.
- ✓ Permite, através de simulação, verificar o comportamento do sistema digital.

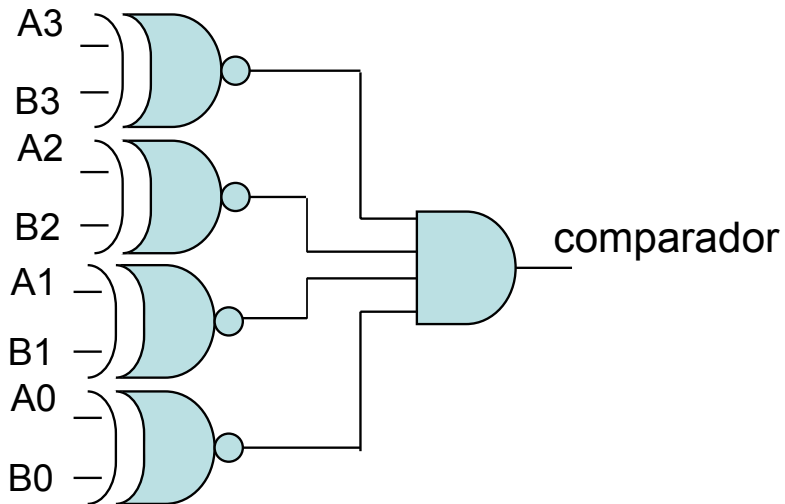
➤ Nível físico:

- ✓ Reduz tempo de projeto (favorece níveis abstratos de projeto).
- ✓ Reduz custo do projeto.
- ✓ Elimina erros de baixo nível (se usado como base de ferramentas automatizadas).
- ✓ **Consequência:** reduz “*time-to-market*” (tempo de chegada de um produto ao mercado).

DESVANTAGENS

- Em relação a diagramas de esquemáticos:
 - ✓ Hardware gerado pode ser menos otimizado.
 - ✓ Controlabilidade / Observabilidade de projeto reduzidas.

➤ Circuito Lógico:



X

➤ VHDL:

```
A : in  std_logic_vector(3 downto 0);  
B : in  std_logic_vector(3 downto 0);  
comparador : out  std_logic;  
...  
comparador <= '1' when A=B else '0';
```

VHDL É UMA LINGUAGEM DE PROGRAMAÇÃO?

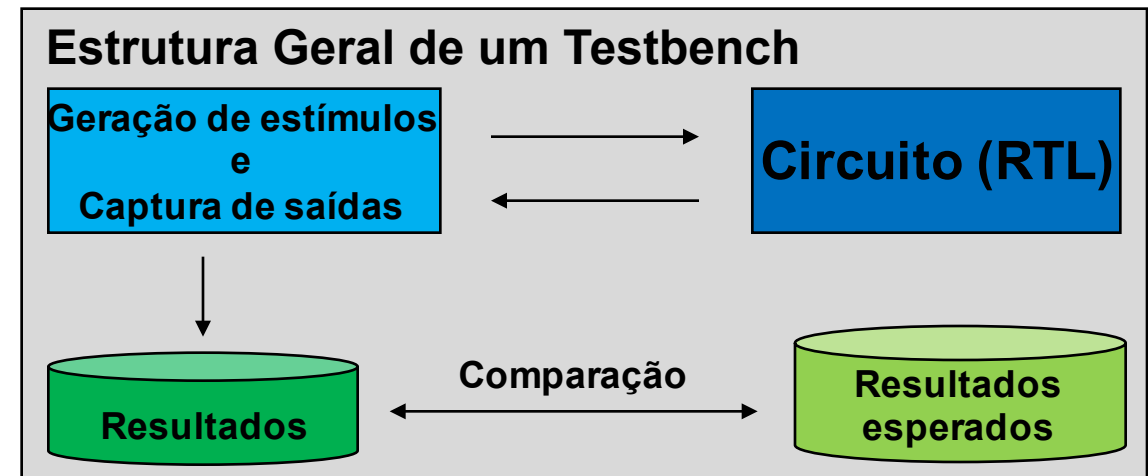
➤ **NÃO.** Pois é uma linguagem de descrição de hardware!

➤ Código é executado em um *simulador*.

- ✓ Não se enxerga o “compilador” de VHDL, não há um “código executável” visível.

➤ Projeto do usuário.

- ✓ Especificado em nível de abstração de transferência em registradores (em inglês, Register – Transfer Level ou **RTL**), mas não apenas neste nível!



➤ **Testbench**: descrição em VHDL ou outra linguagem do procedimento de teste de um circuito.

- ✓ Especificação comportamental do ambiente externo ao projeto (estímulos externos).
- ✓ Interage com o projeto.
- ✓ Não precisa ser descrito em VHDL (o projeto VHDL pode ser validado em ambiente C/C++!!).

TIPO PADRÃO PARA SÍNTESE

➤ **standard_ulogic**

- ✓ Tipo não “resolvido”.
- ✓ Enumeração com 9 níveis lógicos.

```
-- Fonte: stdlogic.vhd
-----
-- logic state system  (unresolved)
-----

TYPE std_ulogic IS ( 'U',  -- Uninitialized
                    'X',  -- Forcing  Unknown
                    '0',  -- Forcing  0
                    '1',  -- Forcing  1
                    'Z',  -- High Impedance
                    'W',  -- Weak      Unknown
                    'L',  -- Weak      0
                    'H',  -- Weak      1
                    '-'   -- Don't care
                );
```

TIPO PADRÃO PARA SÍNTESE

➤ standard_logic

- ✓ Tipo “resolvido” - permite por exemplo implementação de barramentos.
- ✓ Tipo mais utilizado em descrições de hardware.

```
SUBTYPE std_logic IS resolved std_ulogic;
TYPE std_logic_vector IS ARRAY ( NATURAL RANGE <>) OF std_logic;
-----
-- resolution function
-----
CONSTANT resolution_table : stdlogic_table := (
--
--      |  U    X    0    1    Z    W    L    H    -    |
--      -----
--      ( 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U' ), -- | U |
--      ( 'U', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X' ), -- | X |
--      ( 'U', 'X', '0', 'X', '0', '0', '0', '0', 'X' ), -- | 0 |
--      ( 'U', 'X', 'X', '1', '1', '1', '1', '1', 'X' ), -- | 1 |
--      ( 'U', 'X', '0', '1', 'Z', 'W', 'L', 'H', 'X' ), -- | Z |
--      ( 'U', 'X', '0', '1', 'W', 'W', 'W', 'W', 'X' ), -- | W |
--      ( 'U', 'X', '0', '1', 'L', 'W', 'L', 'W', 'X' ), -- | L |
--      ( 'U', 'X', '0', '1', 'H', 'W', 'W', 'H', 'X' ), -- | H |
--      ( 'U', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X' )  -- | - |
--      );
```

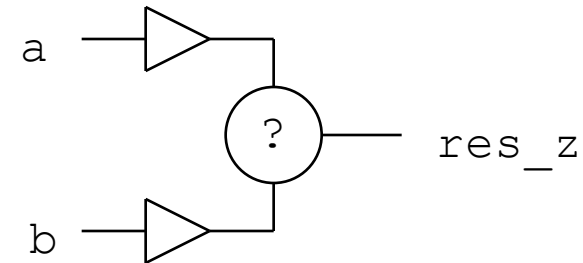
TIPO PADRÃO PARA SÍNTESE

```
signal a, b, z :<tipo pré-definido, não “resolvido”>;  
signal res_z : <tipo_resolvido>;
```

```
z <= a;  
z <= b;
```

X

✓ Erro de “*multiple drivers*”,
ou seja, curto-circuito

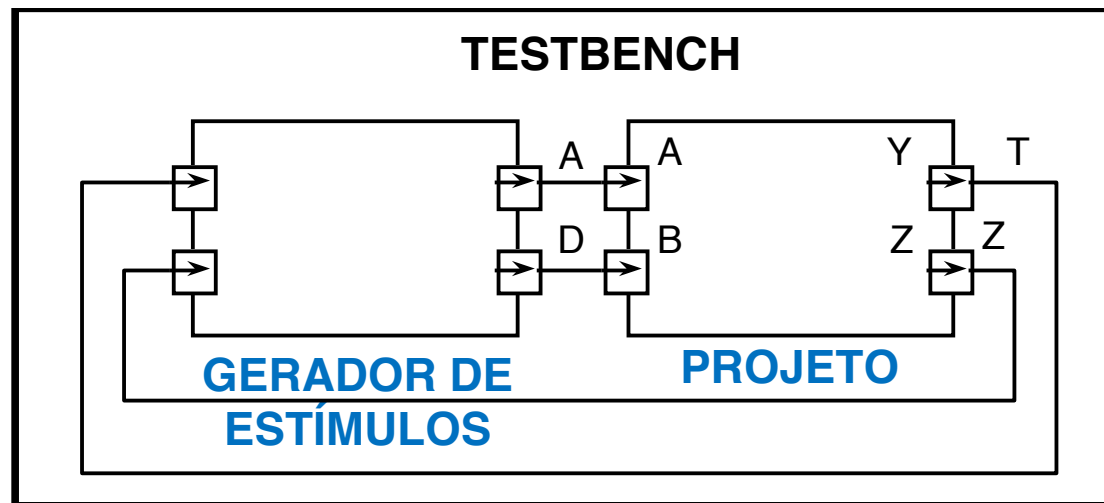


```
res_z <= a;  
res_z <= b;
```

✓ Resolução por tabela de resolução

VALIDAÇÃO POR SIMULAÇÃO

- **Testbenches** - Uma forma simples de testar o projeto.
 - ✓ Na sua forma mais simples, um *testbench* consiste em um ou mais “processos geradores de estímulos” e uma instância do projeto que se quer testar.
 - ✓ O *testbench* é construído como um módulo VHDL que não contém portas de entrada/saída. Ou seja, trata-se de um sistema fechado, ou autônomo.



SUMÁRIO DE DESCRIÇÃO DE HARDWARE

➤ **Entity** – Interface Externa.

- ✓ Especifica somente a interface entre o hardware e o ambiente.
- ✓ Não contém definição do comportamento ou da estrutura internos.

Código VHDL da entidade:

```
entity halfadd is
port (A, B: in std_logic;
      Sum, Carry: out std_logic);
end halfadd;
```

Esquemático:

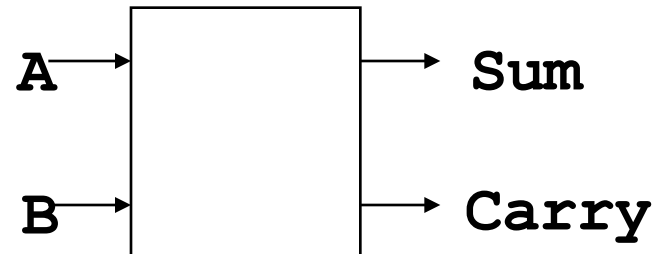


Tabela Verdade:

A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

SUMÁRIO DE DESCRIÇÃO DE HARDWARE

➤ **Architecture** – Comportamento.

- ✓ Especifica o comportamento e/ou a estrutura internos da *entity*.
- ✓ Deve ser associada a uma *entity* específica.
- ✓ Uma *entity* pode ter associada a si várias **architecture** (representando diferentes formas de implementar um mesmo módulo).

Código VHDL da entidade:

```
architecture comp of halfadd is  
begin  
    Sum    <= A xor B;  
    Carry <= A and B;  
end comp;
```

DESCRIÇÃO COMPLETA DO SOMADOR

```
library IEEE;
use IEEE.std_logic_1164.all;

entity halfadd is
port (A, B: in std_logic;
      Sum, Carry: out std_logic);
end halfadd;

architecture comp of halfadd is
begin
    Sum    <= A xor B;
    Carry <= A and B;
end comp;
```

Esta biblioteca e pacote definem o tipo `std_logic`, o qual não faz parte da linguagem VHDL!

EXEMPLO DE TESTBENCH PARA O SOMADOR

```
library IEEE;  
use IEEE.std_logic_1164.all;
```

```
entity halfadd_tb is  
end halfadd_tb;
```

} Testbench não tem pinos
externos (ports in ou out)

```
architecture TB_ARCHITECTURE of halfadd_tb is  
    signal aa, bb, soma, vaium : std_logic;  
begin
```

```
    UUT : entity work.halfadd  
          port map ( A => aa, B => bb,  
                    Sum => soma, Carry => vaium );
```

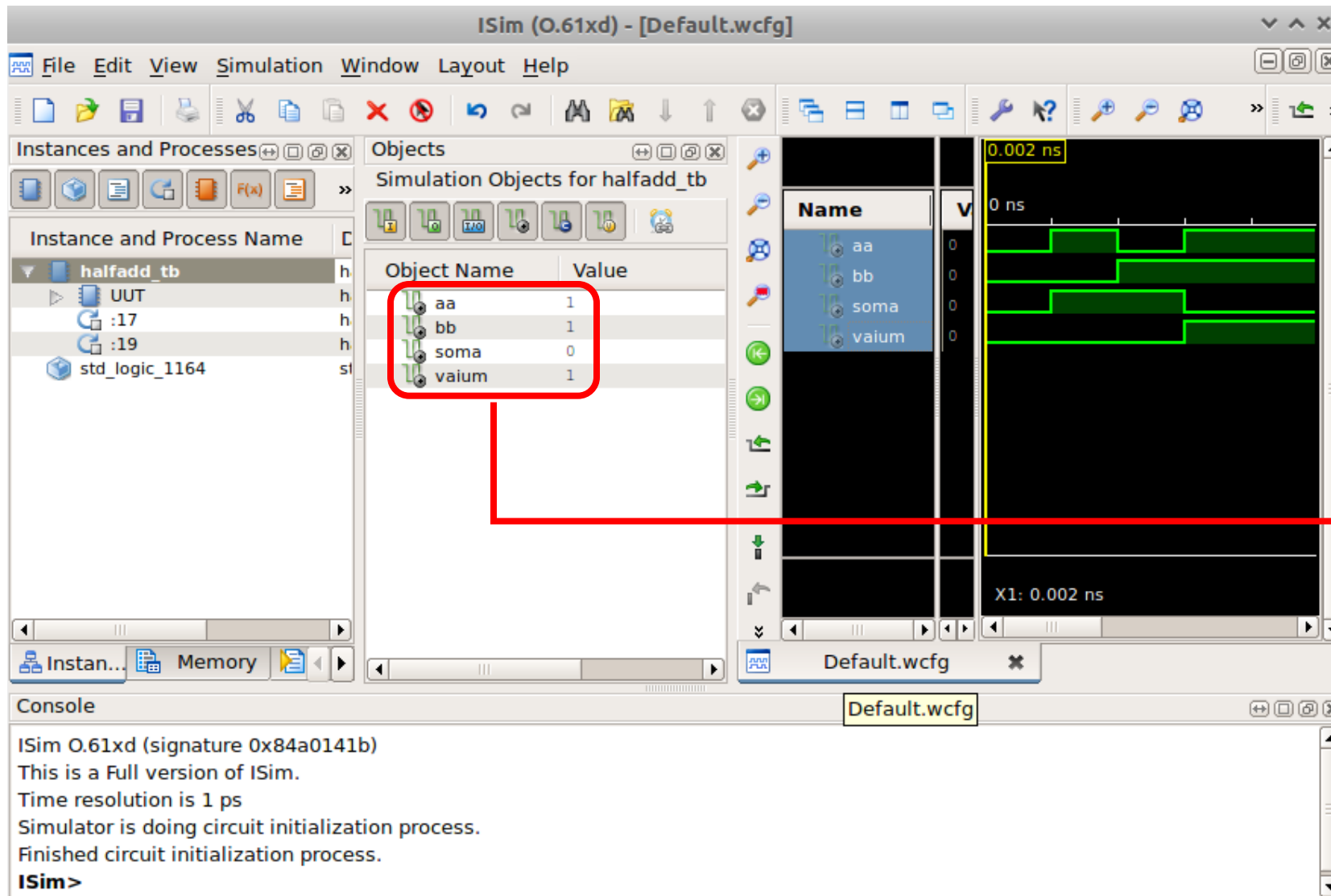
```
    aa <= '0', '1' after 10 ns, '0' after 20 ns, '1' after 30 ns;  
    bb <= '0', '1' after 20 ns;
```

} Instanciação do projeto,
conectando pinos do
projeto aos fios do testador

} Geração dos estímulos,
dizendo como fios se
comportam

```
end TB_ARCHITECTURE;
```


EXEMPLO DE SIMULAÇÃO DA DESCRIÇÃO



Testbench:

- aa** – fio ligado no pino A
- bb** – fio ligado no pino B
- soma** – fio ligado no pino Sum
- vaium** – fio ligado no pino Carry

USANDO O SIMULADOR ISE DA XILINX

USANDO O SIMULADOR ISE DA XILINX

➤ Preparando-se para usar o ambiente ISE:

1. Logar-se no Sistema Operacional Linux (Mint) do laboratório.
2. Executar o comando abaixo para ter acesso às ferramentas de CAD
 - ✓ `Prompt> source /soft64/source_gaph`
3. Executar o comando para configura o uso do ambiente ISE
 - ✓ `Prompt> module load ise`
4. Executar o ISE (em background)
 - ✓ `Prompt> ise &`

USANDO O SIMULADOR ISE DA XILINX

➤ Criando os arquivos do projeto:

1. Criar um diretório de trabalho. Exemplo [half_adder](#).
2. Criar os dois arquivos fonte, com extensão VHDL:

✓ somador1.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;

entity halfadd is
port (A, B: in std_logic;
      Sum, Carry: out std_logic);
end halfadd;

architecture comp of halfadd is
begin
    Sum    <= A xor B;
    Carry <= A and B;
end comp;
```

✓ somador1_tb.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;

entity halfadd_tb is
end halfadd_tb;

architecture TB_ARCHITECTURE of halfadd_tb is
    signal aa, bb, soma, vaium : std_logic;
begin

    UUT : entity work.halfadd
        port map ( A => aa, B => bb,
                  Sum => soma, Carry => vaium );

    aa <= '0', '1' after 10 ns, '0' after 20 ns, '1' after 30 ns;

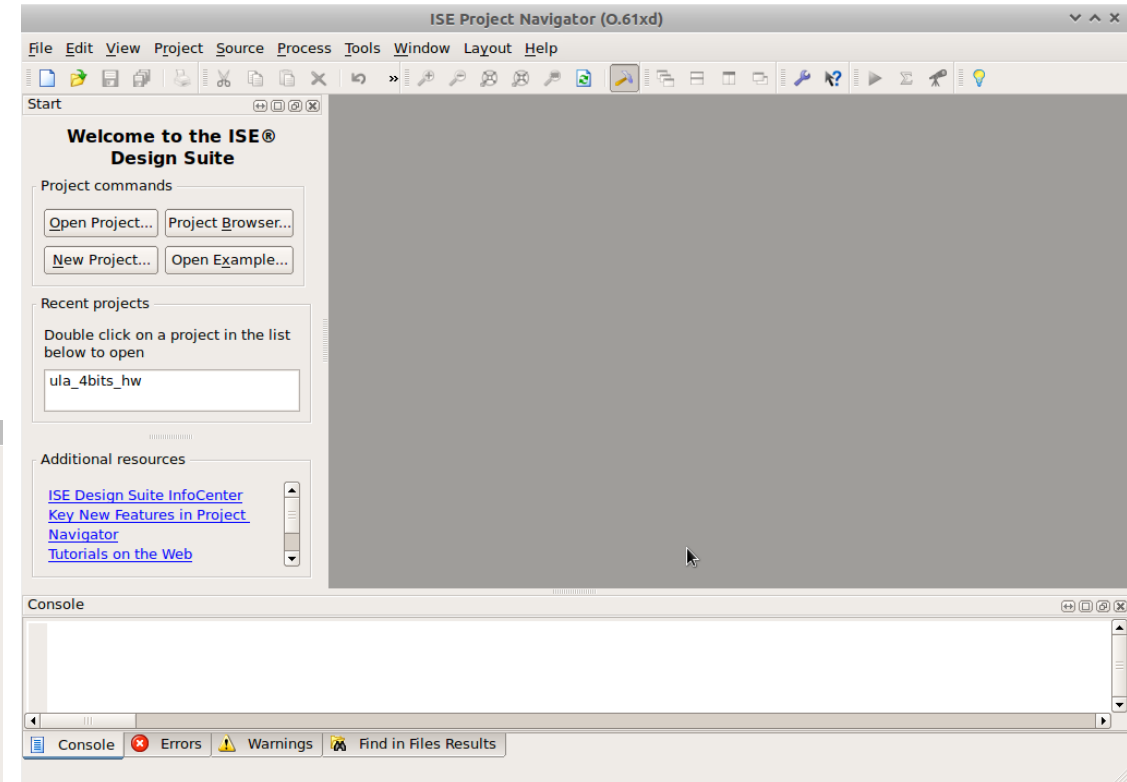
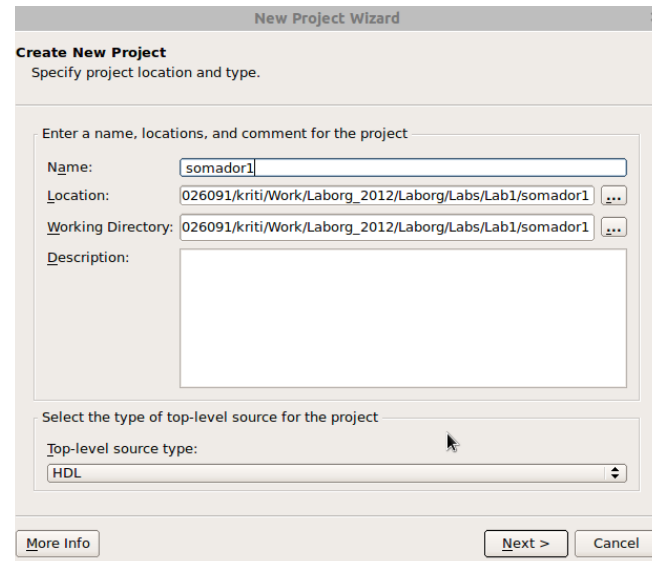
    bb <= '0', '1' after 20 ns;

end TB_ARCHITECTURE;
```

USANDO O SIMULADOR ISE DA XILINX

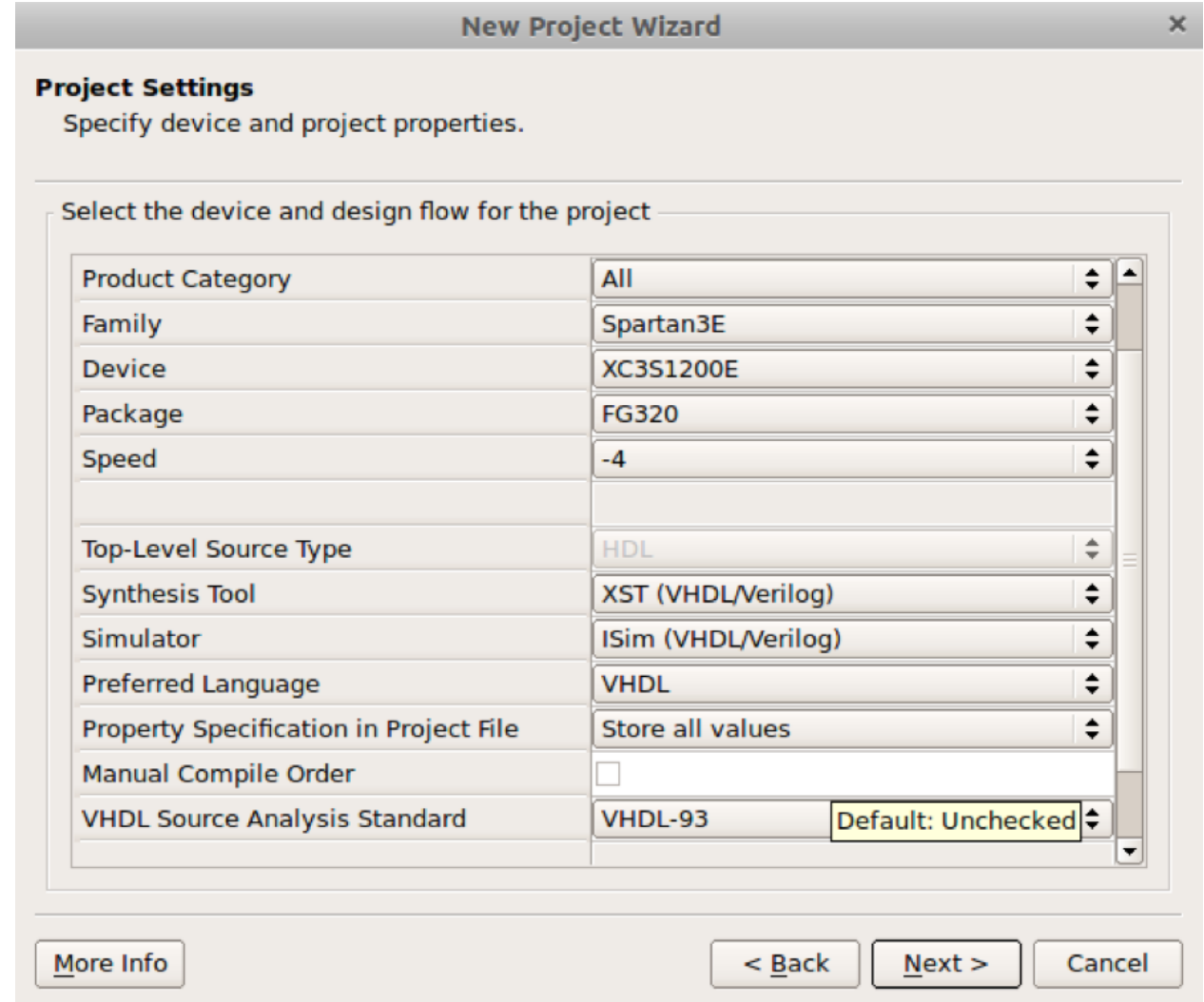
➤ Criando um projeto no ambiente ISE para o FPGA da Nexys2:

1. Caso algum projeto seja aberto ao lançar o ISE, feche-o com a opção de menu **File→Close Project**.
2. Crie um novo projeto de nome **somador1** apertando o botão **New Project**. (Lembre: escolha um diretório no qual tens direito de escrita).
3. Depois de preencher os campos, clique no botão **Next**. Obs: Não usem caracteres especiais no nome do projeto nem no nome do caminho (acentos, brancos, etc.).



USANDO O SIMULADOR ISE DA XILINX

4. Crie um projeto com características abaixo (dados importantes: escolha do dispositivo e do **ISE Simulator - ISIM** na opção **Simulator**). Depois clique no botão **Next**.



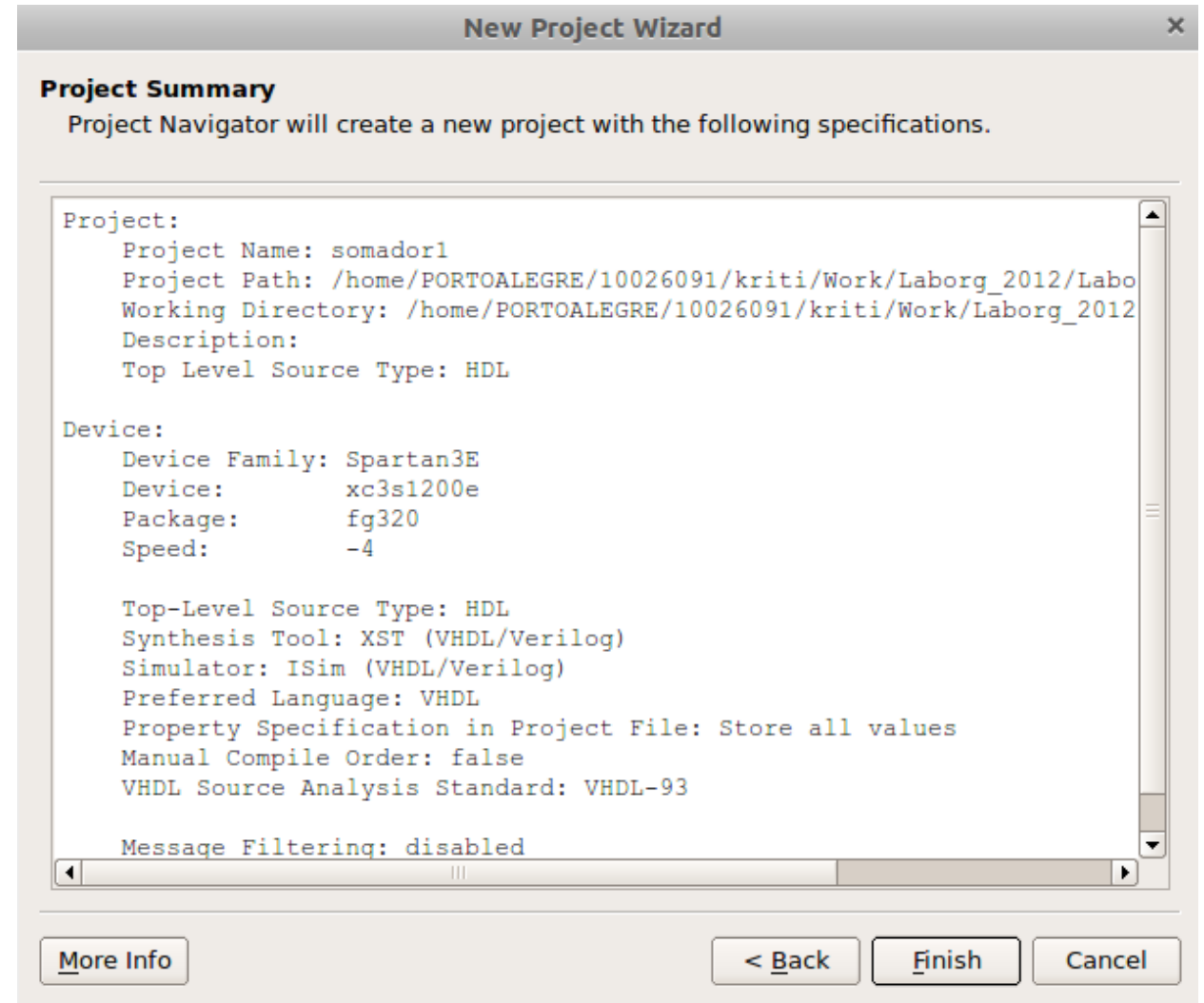
The screenshot shows the 'New Project Wizard' dialog box in Xilinx ISE. The 'Project Settings' section is active, with the instruction 'Specify device and project properties.' Below this, a group box titled 'Select the device and design flow for the project' contains a list of settings:

Setting	Value
Product Category	All
Family	Spartan3E
Device	XC3S1200E
Package	FG320
Speed	-4
Top-Level Source Type	HDL
Synthesis Tool	XST (VHDL/Verilog)
Simulator	ISim (VHDL/Verilog)
Preferred Language	VHDL
Property Specification in Project File	Store all values
Manual Compile Order	<input type="checkbox"/>
VHDL Source Analysis Standard	VHDL-93

At the bottom of the dialog, there are three buttons: 'More Info', '< Back', and 'Next >', along with a 'Cancel' button on the far right. The 'Next >' button is highlighted, indicating the next step in the wizard.

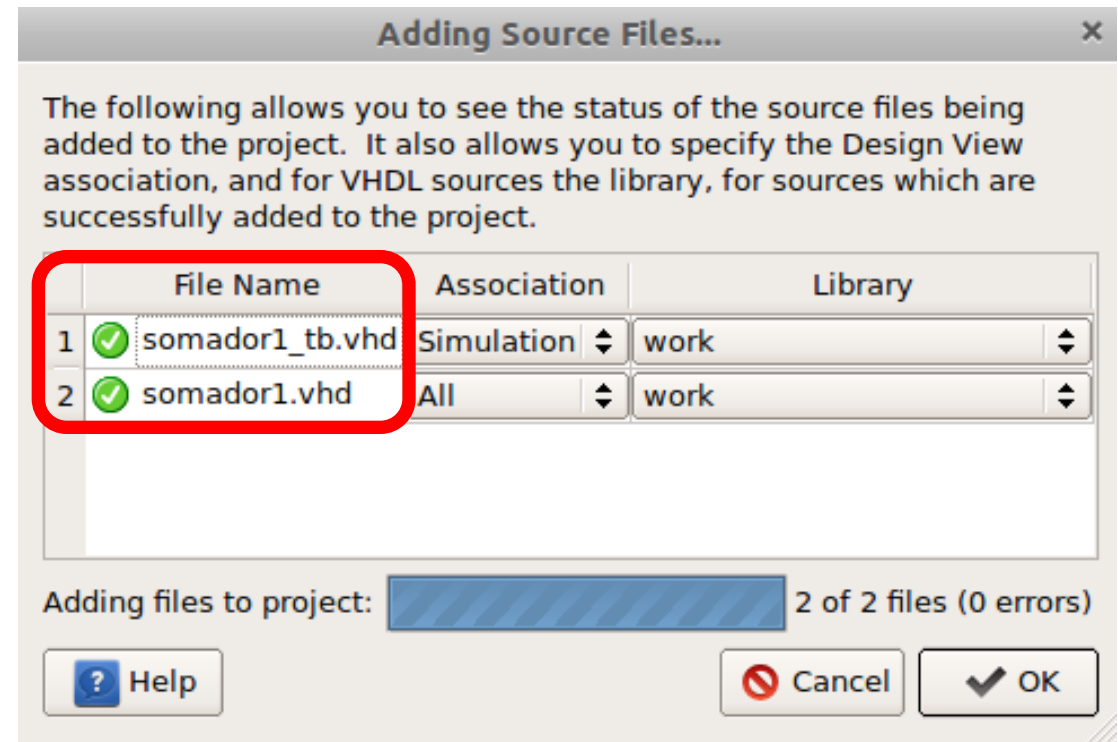
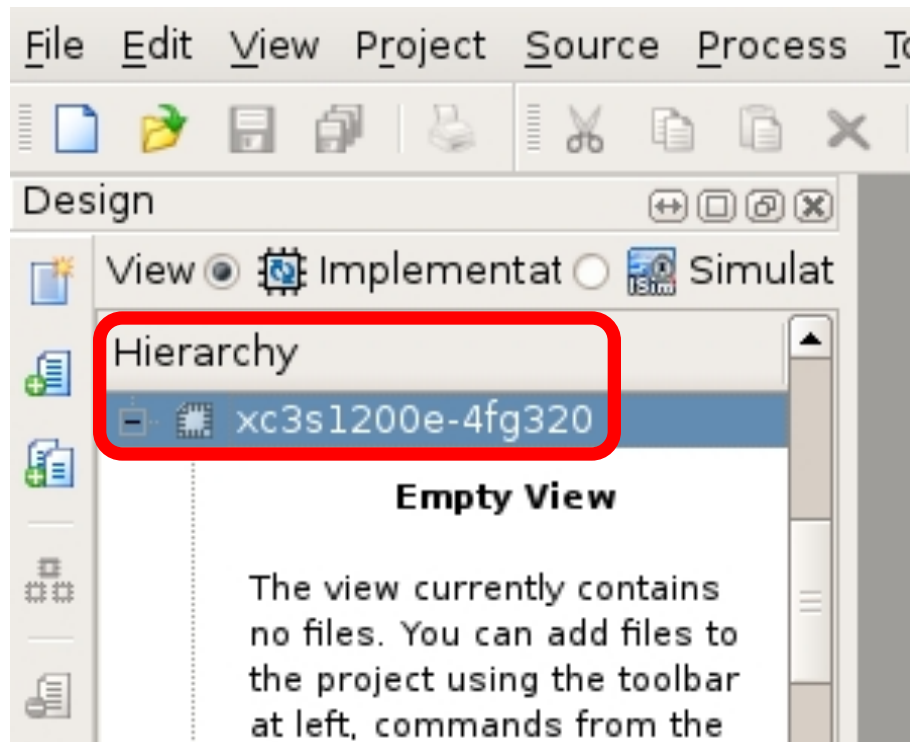
USANDO O SIMULADOR ISE DA XILINX

5. Na janela final, confira os dados do projetos e depois simplesmente clique em **Finish**.



USANDO O SIMULADOR ISE DA XILINX

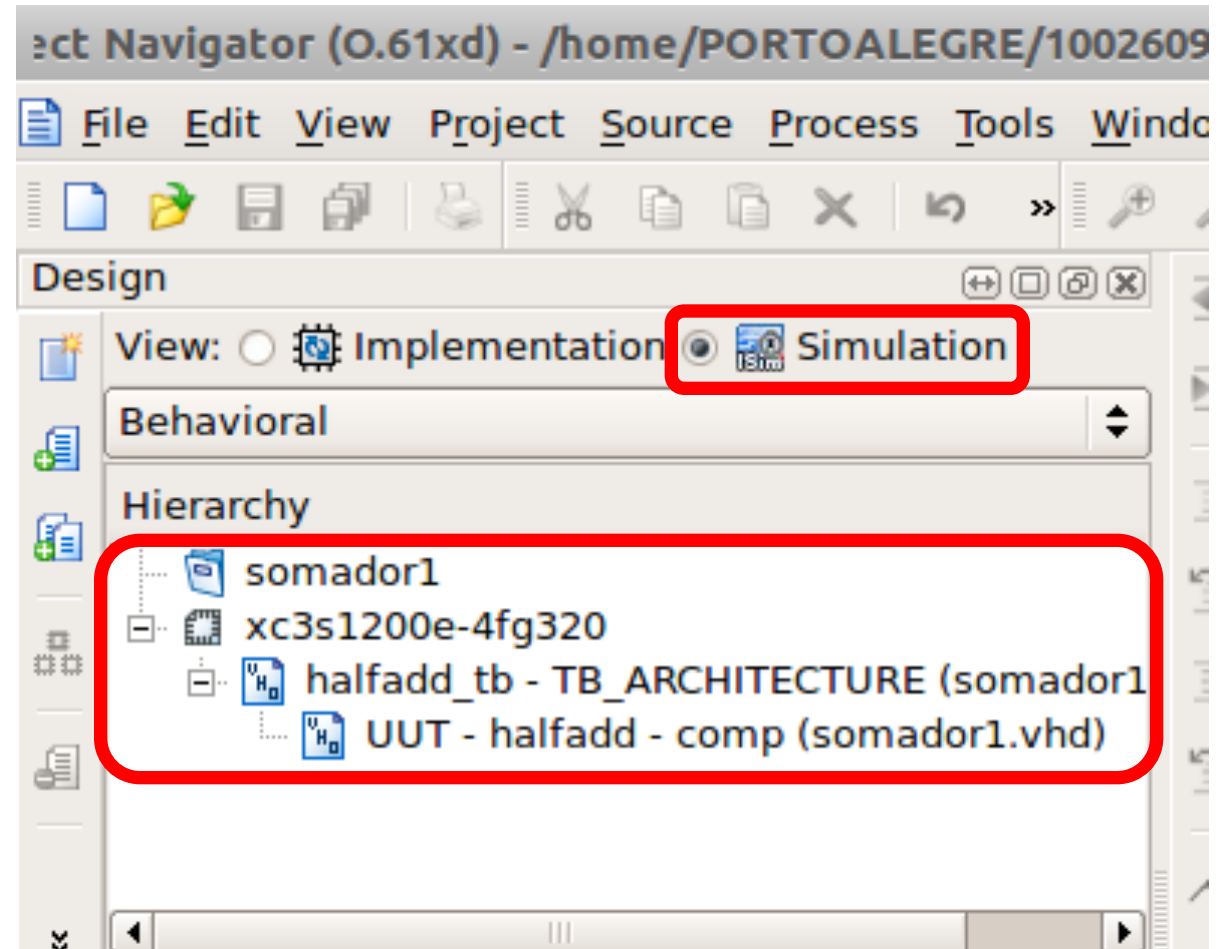
6. No ISE, clique com o botão direito no FPGA (Janela **Hierarchy**) – **Add Copy of Source**. Procure os 2 arquivos VHDL citados respectivamente como **somador1.vhd** e **somador1_tb.vhd**, no diretório acima do diretório principal do projeto (criado pelo ISE). A seguir, clique em **OK**.



USANDO O SIMULADOR ISE DA XILINX

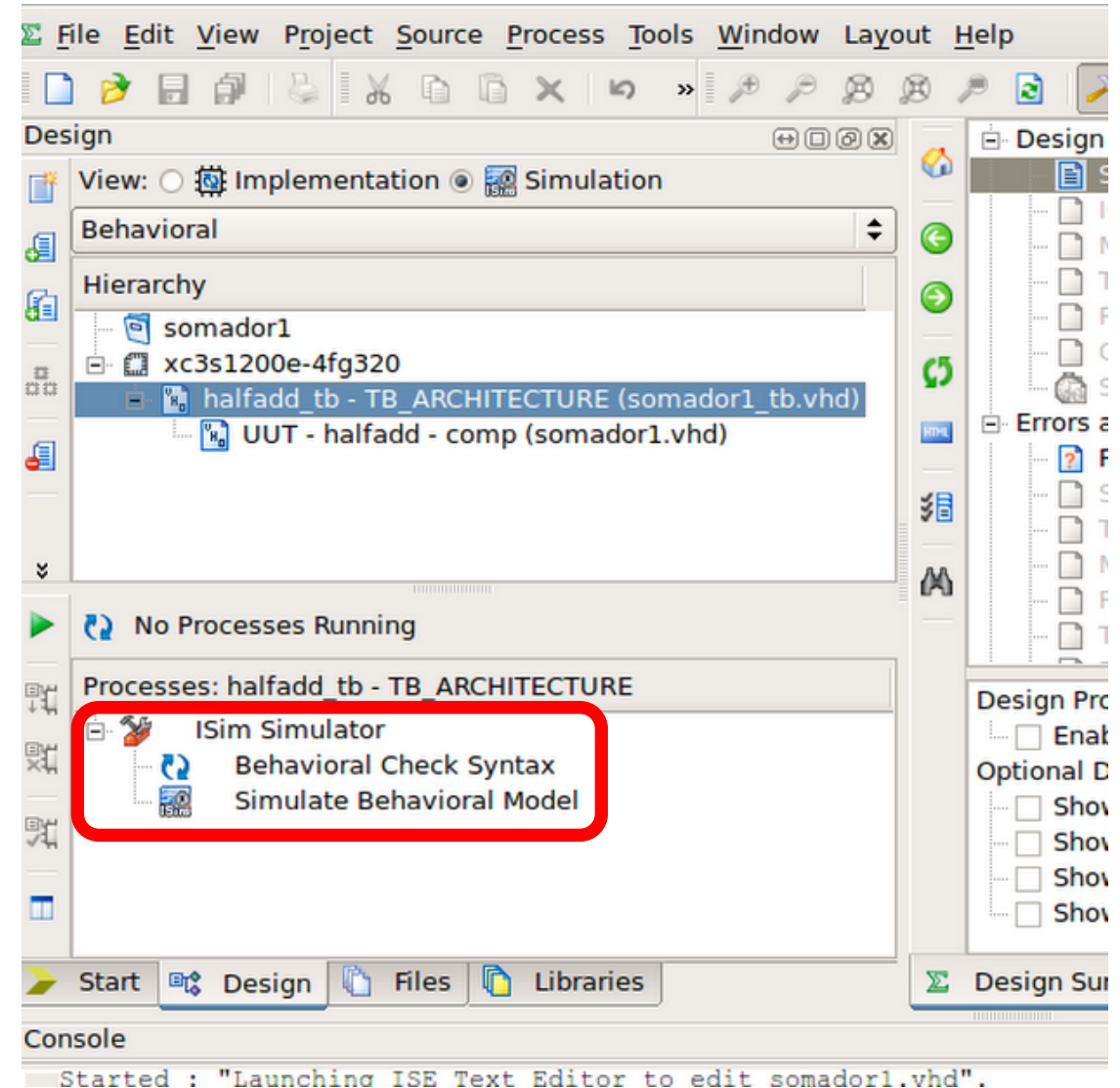
7. Note na janela anterior que o ambiente detectou corretamente os arquivos como sendo: um a ser usado apenas em simulação (o **testbench**, classificado como **Simulation**) e o outro como geral (**All**), significando que ele contém VHDL sintetizável.

No Browser principal do projeto, escolha a visão de simulação (**View Simulation**), pois por omissão a visão escolhida é a de implementação.



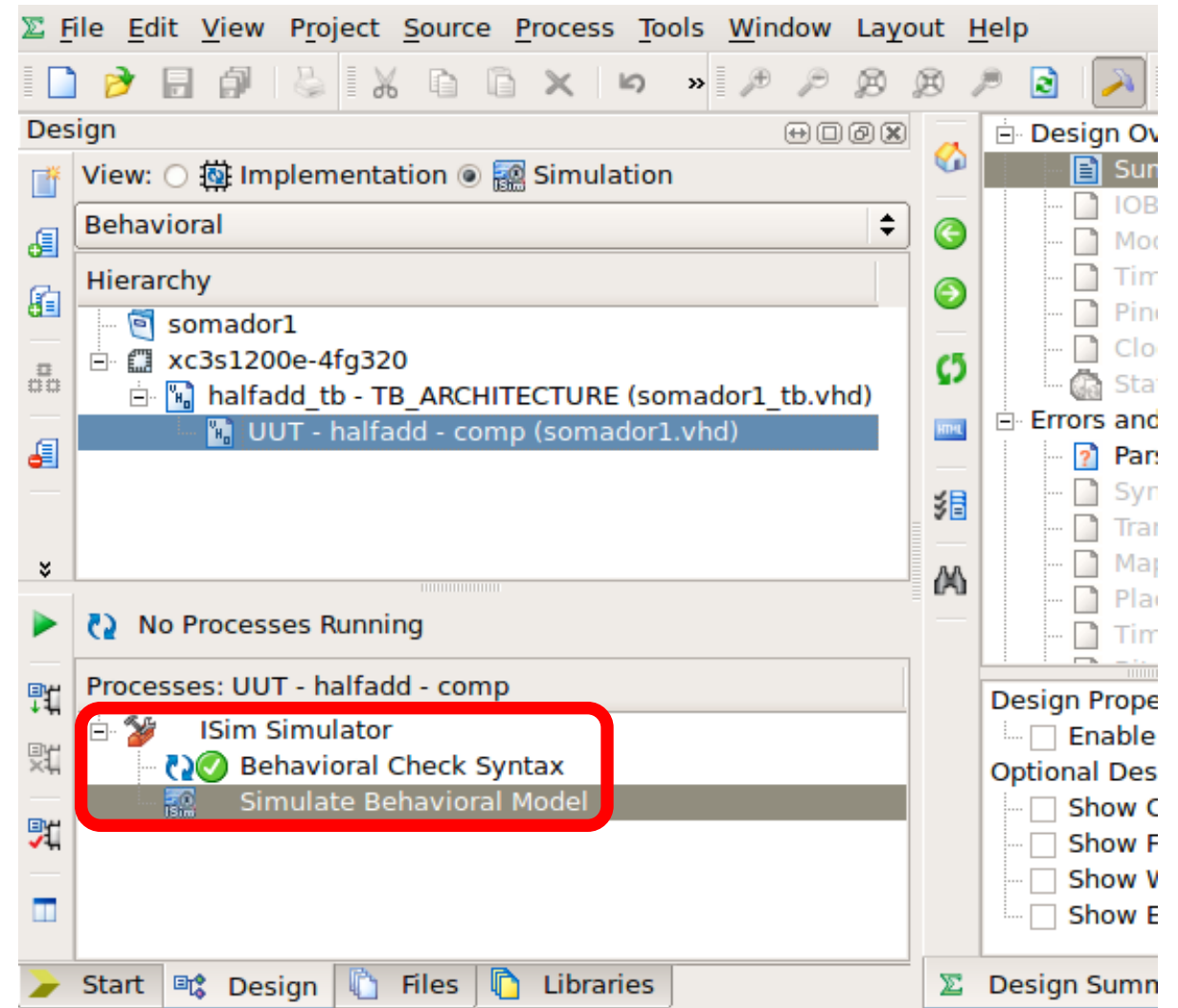
USANDO O SIMULADOR ISE DA XILINX

8. Selecione o arquivo `somador1_tb.vhd` na janela **Hierarchy**. A janela do ISE deve ficar como ao lado. Note que na janela de processos (**Processes**) vê-se o acesso ao simulador.



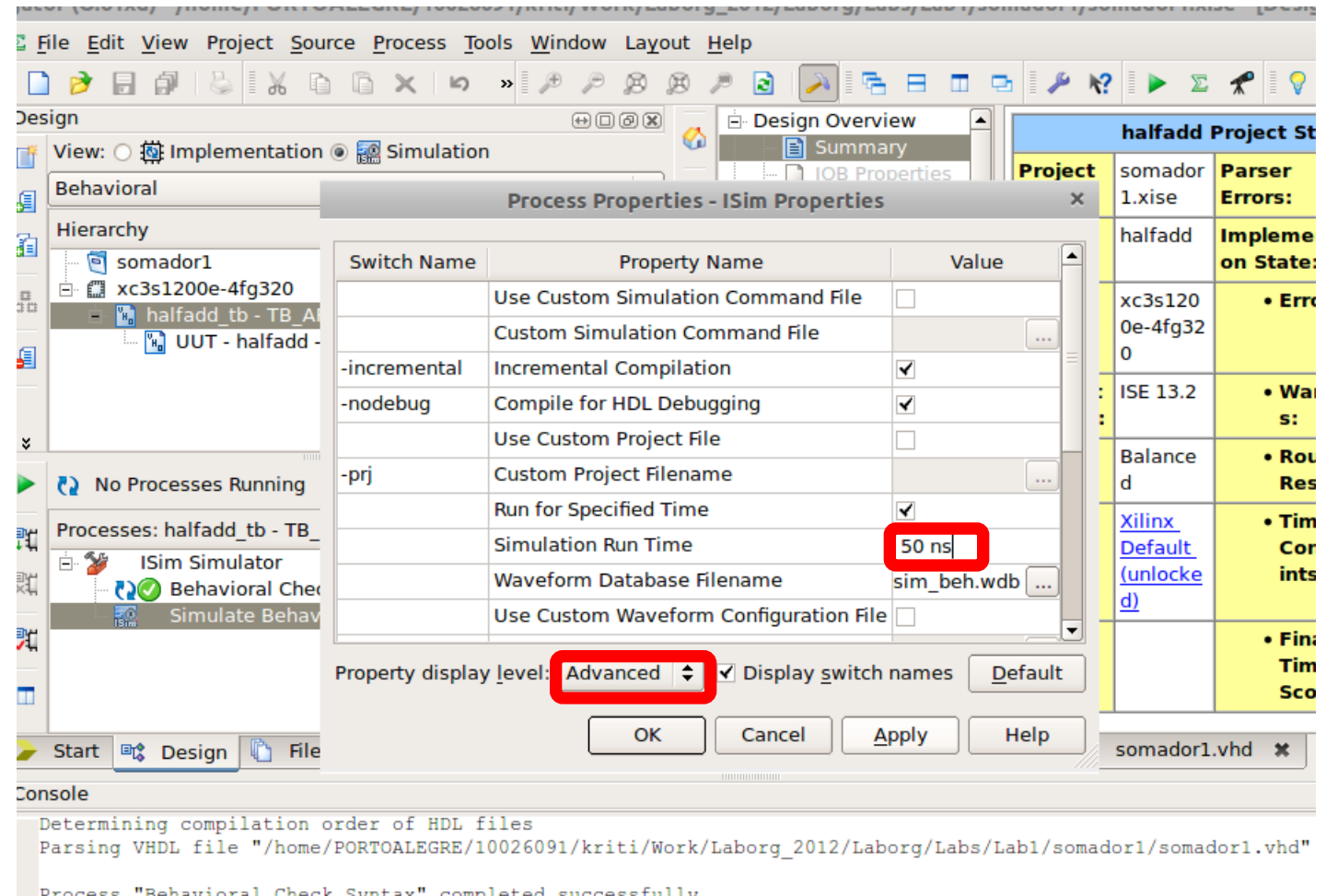
USANDO O SIMULADOR ISE DA XILINX

9. Na janela **Processes**, dê duplo clique na opção **Behavioral Check Syntax**, para verificar que o testbench não possui erros de sintaxe.
10. Abra a hierarquia de projeto, clicando no sinal **+** ao lado do arquivo **somador1_tb.vhd** na janela **Hierarchy**.
Selecione o arquivo **somador1.vhd** e repita o processo de verificação sintática.



USANDO O SIMULADOR ISE DA XILINX

11. Para preparar a simulação, selecione de novo o arquivo `somador1_tb.vhd` (janela [Hierarchy](#)), e clique com o botão direito do mouse na opção **Simulate Behavioral Model – Process Properties** (janela [Processes](#)). Na janela que se abrir, selecione (se já não estiver selecionada) a opção **Advanced** no item **Property Display Level** e mude o **Simulation Run Time** para **50ns**.

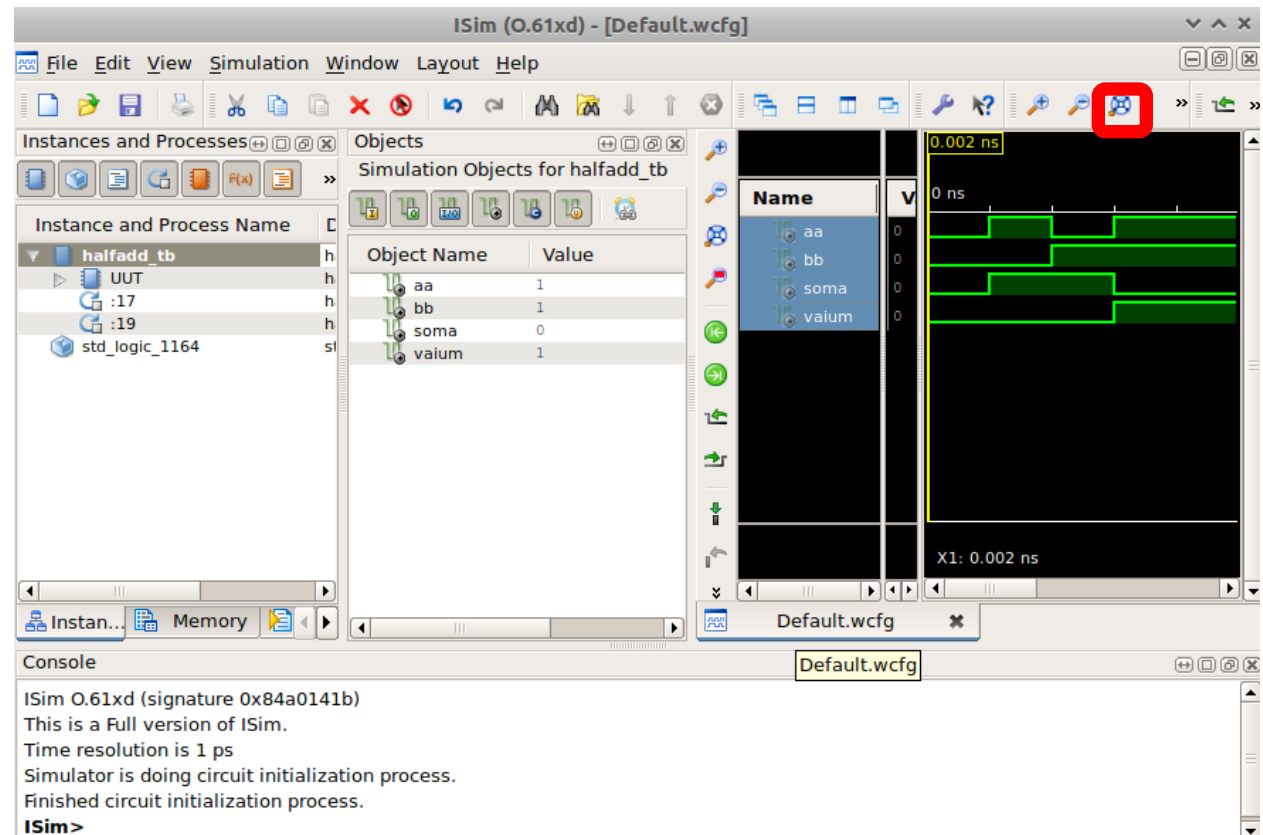


USANDO O SIMULADOR ISE DA XILINX

12. Para simular, basta **fechar a janela Process Properties** clicando em **OK** e dar um **duplo clique** na opção **Simulate Behavioral Model** da janela **Processes**. Surge uma nova janela similar a esta ao lado. Note a janela de formas de onda que mostra os 50ns de simulação do circuito. Experimente usar os diversos botões do simulador.

Note: Esta forma de onda mostra todos os casos da tabela verdade do somador.

Obs.: Lembrem-se de simular o testbench e não o projeto, selecionando o arquivo no topo da hierarquia. Para ver toda a onda na janela clique no botão “Zoom to Full View”.



TRABALHO

TRABALHO

- O **Trabalho 1A (T1A)** consiste em 3 projetos executados no simulador ISE da Xilinx:
 - ✓ **Projeto 1**
 - Adaptação ao ambiente de simulação através do desenvolvimento de um meio-somador de 1 bit ([half_adder](#)). Este projeto já foi desenvolvido nestes mesmos slides (Seção “usando o simulador ISE da Xilinx”).
 - ✓ **Projeto 2**
 - Desenvolvimento de um somador de 4 bits sem vai-um ([full_adder](#)).
 - ✓ **Projeto 3**
 - Desenvolvimento de um flip-flop mestre-escravo assíncrono ([ffd](#)).

PROJETO 2

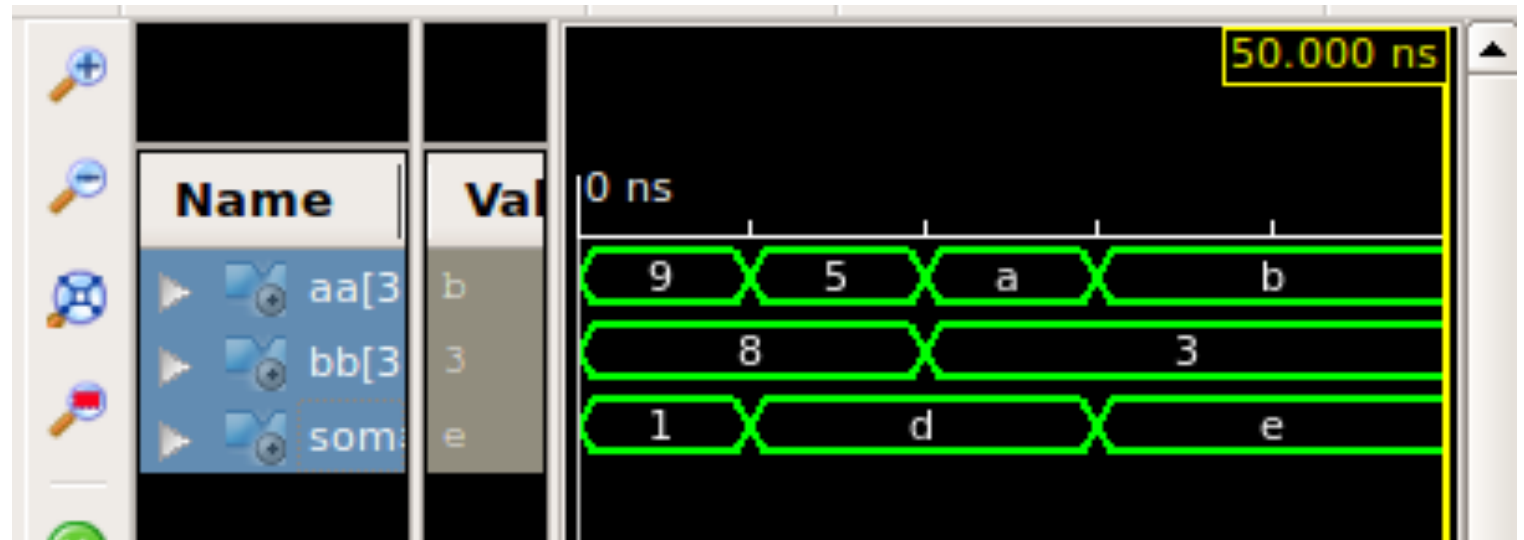
- Escreva um novo código (faça um novo projeto VHDL. Use a opção *Copy Project* do ISE) utilizando sinais não de 1 bit mas de 4 bits (`std_logic_vector(3 downto 0)`). O efeito é transformar o meio somador de 1 bit em um somador de 4 bits.
 - ✓ `std_logic`→fio e `std_logic_vector`→vários fios (barramento).
- A nova entidade VHDL deve parecer com:

```
entity soma is
port (A, B: in std_logic_vector(3 downto 0);
      Soma: out std_logic_vector(3 downto 0));
end soma;
```
- O corpo da arquitetura VHDL deverá conter apenas `soma <= a + b;`.
- Deve-se incluir a linha: `use IEEE.std_logic_unsigned.all;` (Porquê?)
- Escreva o *testbench* correspondente.

PROJETO 2

- Os 4 bits podem ser representados como um único dígito em hexadecimal. No *testbench* pode-se escrever:
 - ✓ `aa <= x"9", x"5" after 10 ns, x"A" after 20 ns, x"B" after 30 ns;`
 - ✓ `bb <= x"8", x"3" after 20 ns;`

- Ao simular este circuito, aparece uma forma de onda similar a apresentada na Figura a direita. Você deverá testar outros valores no seu *testbench*.



- O comportamento é o esperado? **Explique.**

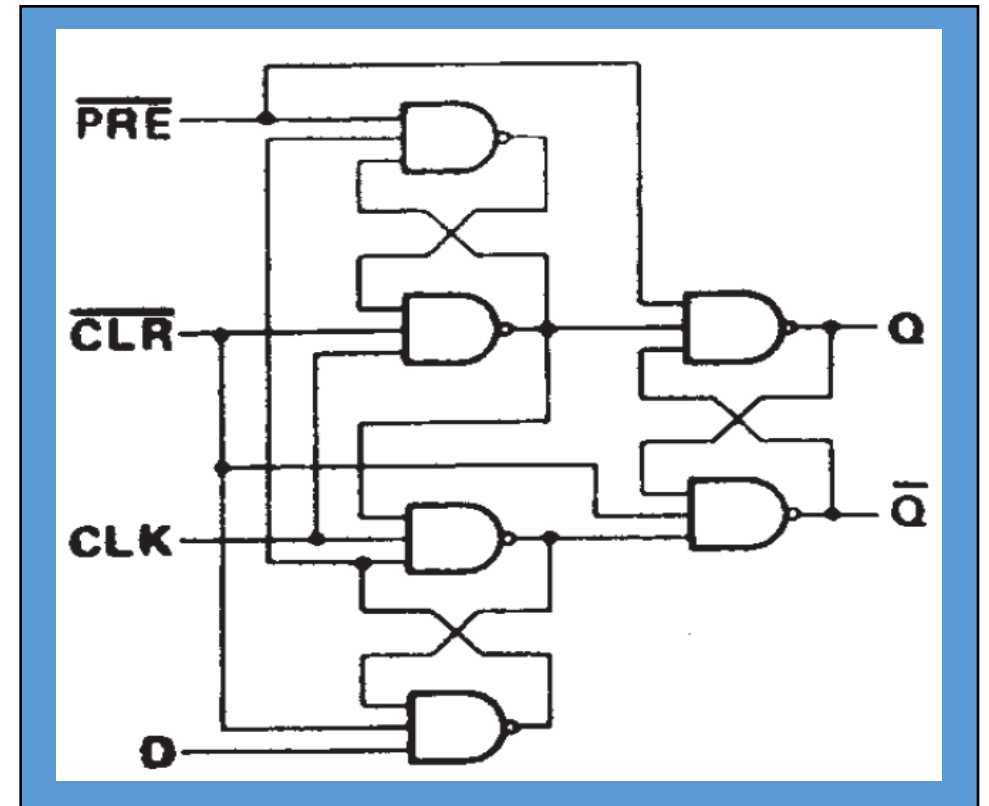
PROJETO 2

- Para gerar todas as combinações possíveis de entradas deste circuito seria necessário produzir 256 padrões (combinando os 16 valores distintos de A com os 16 valores distintos de B). Um trecho de código VHDL capaz de gerar estes estímulos é dado abaixo:

```
signal aa : std_logic_vector(3 downto 0) := "0000"; -- valor inicial
signal bb : std_logic_vector(3 downto 0) := "0000"; -- valor inicial
...
process (aa)
begin
    if (aa/=x"F") then
        aa <= aa+x"1" after 10ns; -- a cada 10ns
    else aa <= x"0" after 10ns; -- incrementa A
    end if;
end process;
process (bb)
begin
    if (bb/=x"F") then
        bb <= bb+x"1" after 160ns; -- a cada 16
    else bb <= x"0" after 160ns; -- valores de A,
    end if; -- incrementa B
end process;
```

PROJETO 3

1. Implemente o circuito de seis portas lógicas ao lado em VHDL.
2. Crie um projeto com o simulador do ISE que contenha o par entidade-arquitetura correspondente a este circuito.
3. Crie um arquivo que instancie este circuito, que gere estímulos para todas suas 4 entradas e os aplique à instância, isto é, crie um *testbench* para o circuito.
4. Simule o *testbench* e descreva textualmente como o circuito funciona.



PROJETO 3

➤ Informações adicionais:

- ✓ Trata-se de um **flip-flop tipo D sensível à borda de subida do relógio** (sinal **CLK**), com **sinais assíncronos** (isto é independentes do, e prioritários em relação ao sinal de relógio) de **preset** (sinal **\overline{PRE}**) e **reset** (sinal **\overline{CLR}**).
- ✓ As saídas são os sinais **Q** e **\overline{Q}** , tipicamente assumindo valores opostos, e mostram o valor do bit armazenado no **FF**.
- ✓ O sinal **\overline{PRE}** em **0** força a saída **Q** para **1** (e **\overline{Q}** para **0**), enquanto **\overline{CLR}** em **0** força **Q** para **0** (e **\overline{Q}** para **1**). Eles não devem nunca ser **0** ao mesmo tempo. Qualquer um em **0** faz com que as bordas do relógio sejam ignoradas (elimina o efeito destas).
- ✓ Seguindo a regra geral de circuitos síncronos, **nenhuma entrada deve mudar** ao mesmo tempo que o relógio.
- ✓ Cada borda de subida de **CLK** sem **\overline{PRE}** nem **\overline{CLR}** ativados provoca a amostragem e o armazenamento do valor de **D**.

PROJETO 3

- Notem que o circuito a implementar é assíncrono (devido aos laços de realimentação combinacionais), e cria um componente básico em projetos síncronos, o flip-flop D.
- Cuidados a tomar:
 - ✓ Ao fazer o *testbench* respeitem a relação de temporização entre sinais de um circuito síncrono:
 - Quando o clock mudar na sua borda sensível ($0 \rightarrow 1$, borda de subida), **NENHUM** sinal amostrado por ele (*D*, neste caso) pode mudar ao mesmo tempo! Lembrem-se dos conceitos de tempo de *setup* e *hold*!
 - Os sinais CLR e PRE também devem respeitar esta regra!
 - ✓ Nomes de sinais no *testbench* podem ser idênticos aos nomes dos pinos dos objetos que o *testbench* instancia.

RESUMO DO TRABALHO 1A

- O Trabalho 1A (T1A) compreende 3 projetos ISE:
 - ✓ Meio-somador de 1 bit (`half_adder`).
 - ✓ Somador de 4 bits sem vai-um (`full_adder`).
 - ✓ Flip-flop mestre-escravo assíncrono (`ffd`).
- O que deve ser entregue:
 - ✓ Arquivo compactado (formato **ZIP**): `<aluno1_aluno2>.zip`, contendo **7 arquivos internos**:
 - Um relatório em PDF com formas de onda, descrevendo como cada circuito funciona e sua implementação de acordo com o modelo apresentado.
 - Para cada 1 dos 3 projetos descritos acima, somente o **código VHDL de implementação** e o **testbench que valida a implementação**. Por exemplo:
 - `half_adder/half.vhd`
 - `half_adder/tb_half.vhd`
 - `full_adder/full.vhd`
 - ...