MC613
Laboratório de Circuitos Digitais

VHDL Crash Course

Prof. Dr. Eng. Isaías Bittencourt Felzmann

isaias@ic.unicamp.br

HDL vs Programação: Uma Mudança Mental

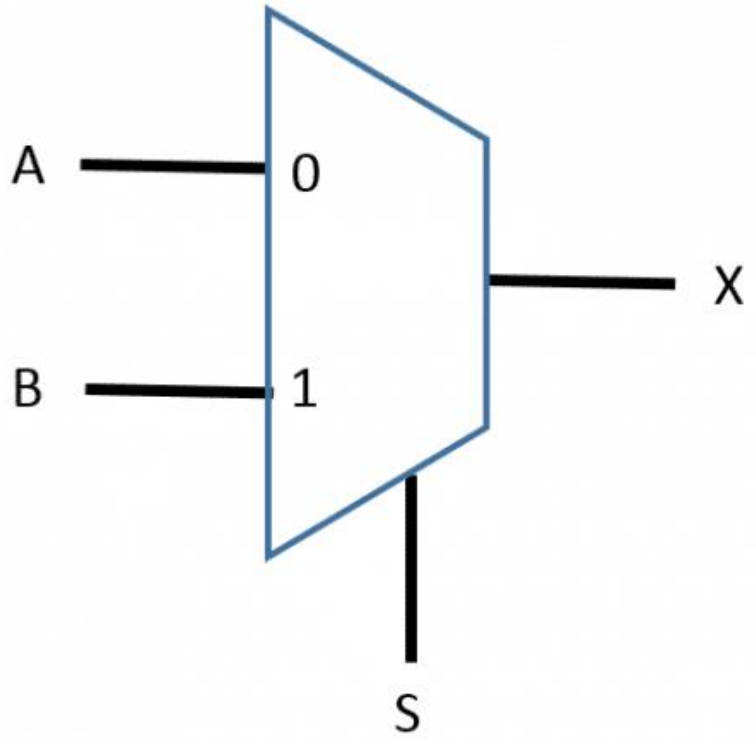
- **HDL:** Linguagem de **Descrição** de Hardware
  - **Descreve** um componente de hardware conhecido.
  - **Não é programação!**

Software (C, Python, Assembly)	Hardware (VHDL, Verilog)
Execução Sequencial (CPU).	Execução **Concorrente**.
Recurso: Tempo de Processamento.	Recurso: Área de Silício (Portas).
Variáveis: Endereços de Memória.	Sinais: Fios Físicos e Tensões.
"E depois..."	*"Ao mesmo tempo..."*

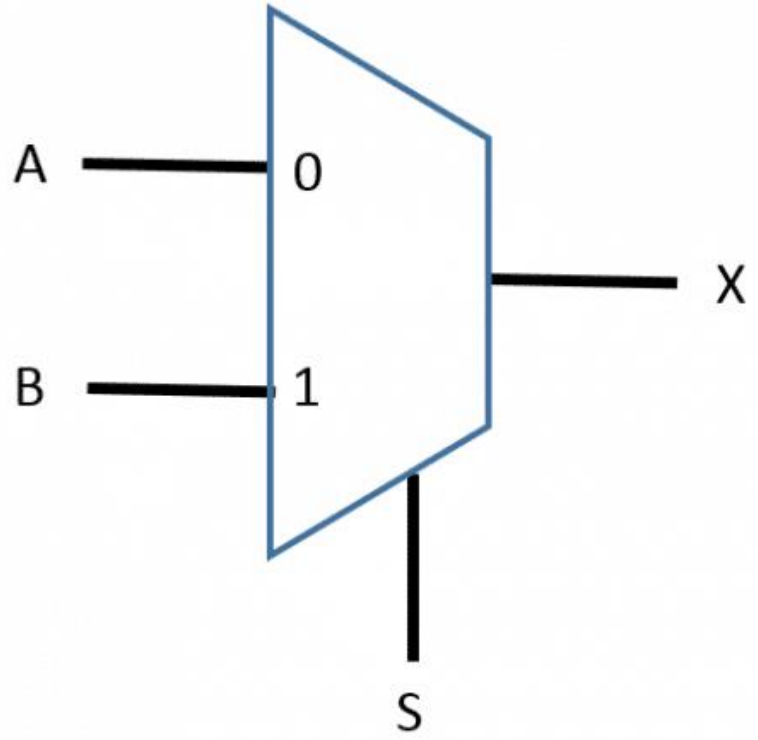
- Não *deveríamos* programar software antes de pensar no algoritmo...

- **Também não devemos descrever hardware antes de pensar no componente**.

Exemplo 1: O que você espera deste circuito?



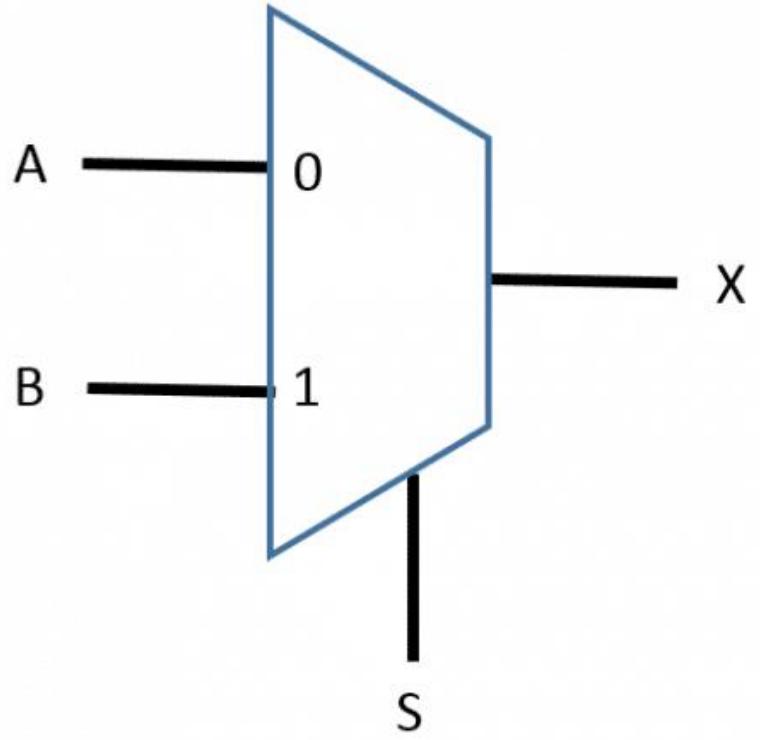
Exemplo 1: O que você espera deste circuito?



- Nome: Multiplexador 2x1

- Funcionalidade esperada:
  - Selecionar na saída X...
  - uma das entradas A ou B...
  - de acordo com a entrada S.

Exemplo 1: O que você espera deste circuito?



- **Descrição comportamental:**
  - A saída X recebe o valor A quando S é igual a 0, senão recebe B.

Exemplo 1: O que você espera deste circuito?

- **Descrição comportamental:**
  - A saída X **recebe** o valor A **quando** S **é igual a 0**, **senão** recebe B.

```
X <= A when ( S = '0') else B;
```

- Observe:
  - Não existe noção de **tempo**.
  - X **sempre** será ou A, ou B.
    - X simplesmente existe, e tem esse valor.
    - Não foi criado aqui. Não foi atribuído agora.
    - Não pode ter outro valor.

Exemplo 1: O multiplexador completo

```
-- 1. Bibliotecas
library IEEE ;
use IEEE . STD_LOGIC_1164 . ALL ; -- Define std_logic (9 valores )

-- 2. Entidade ( Interface / Pinagem )
entity Mux2to1 is
   port ( A, B : in STD_LOGIC ;
      S : in STD_LOGIC ;
      X : out STD_LOGIC );
   end Mux2to1 ;

-- 3. Arquitetura ( Comportamento Interno )
architecture Behavioral of Mux2to1 is
begin
   -- Atribuição Concorrente
   X <= A when ( S = '0') else B;
end Behavioral ;
```

Observação: VHDL não é *case sensitive*.

STD_LOGIC: Representando Bits

A convenção VHDL para representar **sinais lógicos** é **STD_LOGIC**.

	Nome	**Explicação**
‘U’	Não inicializado	Valor desconhecido no início da simulação.
‘X’	Desconhecido	**Erro:** valor desconhecido durante a simulação.
‘0’	Lógico 0	
‘1’	Lógico 1	
‘Z’	Alta impedância	Circuito aberto, desconectado.
‘-’	*Don’t care*	Utilizado para otimização.

Atenção: é comum escrever código desconsiderando valores diferentes de ‘0’ e ‘1’, porém não podemos esquecer que eles existem.

STD_LOGIC_VECTOR: Barramentos de Bits

STD_LOGIC_VECTOR é um vetor de STD_LOGIC.

Tamanho definido pelos índices declarados.

Dois vetores de 4 bits:

`signal V : std_logic_vector(3 downto 0);`
 	
Índices: (3 2 1 0)
 	
`V <= "0101";`
	`signal V : std_logic_vector(0 to 3);`
	
	Índices: (0 1 2 3)
	
	`V <= "0101";`

V(0) = '1' **V(2) = '1'** **V(1) = '0'** **V(3) = '0'**	**V(0) = '0'** **V(2) = '0'** **V(1) = '1'** **V(3) = '1'**

DOWNTO é mais intuitivo para dados numéricos e acaba sendo o padrão *de facto*.

Exemplo 2: Primo ou não primo?

Projete em VHDL o circuito:

- Funcionalidade esperada:
  - A saída prime deve indicar com 1 se o número de 4 bits X é primo.
  - (E 0 se não for).

Exemplo 2: Primo ou não primo?

- Descrição comportamental:
  - A saída prime recebe 1 quando X é igual a 2,
  - senão recebe 1 quando X é igual a 3,
  - senão recebe 1 quando X é igual a 5,
  - senão recebe 1 quando X é igual a 7,
  - senão recebe 1 quando X é igual a 11,
  - senão recebe 1 quando X é igual a 13,
  - senão recebe 0;

Exemplo 2: Primo ou não primo?

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity prime4 is
   port (
      X     : in  std_logic_vector(3 downto 0);
      prime : out std_logic
   );
end entity prime4;
```

```vhdl
architecture rtl of prime4 is
begin

   prime <= '1' when X = "0010" else  -- 2
            '1' when X = "0011" else  -- 3
            '1' when X = "0101" else  -- 5
            '1' when X = "0111" else  -- 7
            '1' when X = "1011" else  -- 11
            '1' when X = "1101" else  -- 13
            '0';

end architecture rtl;
```

WITH/SELECT: Quando o número de alternativas aumenta

- WHEN/ELSE faz uma atribuição condicional.
  - Porém prejudica legibilidade quando número de possibilidades aumenta.

- A construção WITH/SELECT traduz diretamente uma tabela de atribuições com base em um seletor em comum.

Entrada X		Saída Y
0	0	A
0	1	B
1	0	C
1	1	D

Com X como seletor,
Y é A quando 0,
B quando 1,
C quando 2,
D quando 3;

```
with X select
  Y <= A when 0,
       B when 1,
       C when 2,
       D when 3;
```

Exemplo 3; Decodificador one-hot

Entrada X				Saída Y
0	0	0	0	0000000000000001
0	0	0	1	0000000000000010
0	0	1	0	0000000000000100
0	0	1	1	0000000000001000
0	1	0	0	0000000000010000
0	1	0	1	0000000000100000
0	1	1	0	0000000001000000
0	1	1	1	0000000010000000
1	0	0	0	0000000100000000
1	0	0	1	0000001000000000
1	0	1	0	0000010000000000
1	0	1	1	0000100000000000
1	1	0	0	0001000000000000
1	1	0	1	0010000000000000
1	1	1	0	0100000000000000
1	1	1	1	1000000000000000

```
with X select
    Y <= "0000000000000001" when "0000",
         "0000000000000010" when "0001",
         "0000000000000100" when "0010",
         "0000000000001000" when "0011",
         "0000000000010000" when "0100",
         "0000000000100000" when "0101",
         "0000000001000000" when "0110",
         "0000000010000000" when "0111",
         "0000000100000000" when "1000",
         "0000001000000000" when "1001",
         "0000010000000000" when "1010",
         "0000100000000000" when "1011",
         "0001000000000000" when "1100",
         "0010000000000000" when "1101",
         "0100000000000000" when "1110",
         "1000000000000000" when "1111",
         "0000000000000000" when others;
```

?

Por que WHEN OTHERS?

- Lembre-se da definição de STD_LOGIC:
  - ‘0’ e ‘1’ não são as únicas possibilidades.
  - Portanto, as 16 combinações “0000” -> “1111” não cobrem tudo.

- VHDL correto deve cobrir todas as combinações!
  - O simulador vai exibir um erro caso não seja coberto.

SIGNAL: Valores intermediários

- Componentes maiores não têm só entradas e saídas.
  - Eles precisam computar resultados intermediários.

- Em VHDL, valores intermediários são signals.
  - Semelhantes a variáveis em linguagens de programação.
  - Porém, não existe noção de atribuição sequencial.
    - Ou seja, um signal não recebe um valor após a linha anterior.
    - Assim como as saídas, o signal simplesmente existe e assume um valor.
  - Analogia em hardware: Signals são fios.
    - Ligações entre outros componentes internos.

Exemplo 4: ALU simples

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity alu4 is
   port (
      A  : in  std_logic_vector(3 downto 0);
      B  : in  std_logic_vector(3 downto 0);
      op : in  std_logic_vector(1 downto 0);
      Y  : out std_logic_vector(3 downto 0)
   );
end entity alu4;
```

```
architecture rtl of alu4 is
   signal add_r : std_logic_vector(3 downto 0);
   signal sub_r : std_logic_vector(3 downto 0);
   signal and_r : std_logic_vector(3 downto 0);
   signal or_r  : std_logic_vector(3 downto 0);
begin
   add_r <= std_logic_vector(unsigned(A) + unsigned(B));
   sub_r <= std_logic_vector(unsigned(A) - unsigned(B));
   and_r <= A and B;
   or_r  <= A or B;

   with op select
      Y <= add_r when "00",
           sub_r when "01",
           and_r when "10",
           or_r  when others;
end architecture rtl;
```

Exemplo 4: ALU simples – As atribuições são concorrentes!

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity alu4 is
   port (
      A  : in  std_logic_vector(3 downto 0);
      B  : in  std_logic_vector(3 downto 0);
      op : in  std_logic_vector(1 downto 0);
      Y  : out std_logic_vector(3 downto 0)
   );
end entity alu4;
```

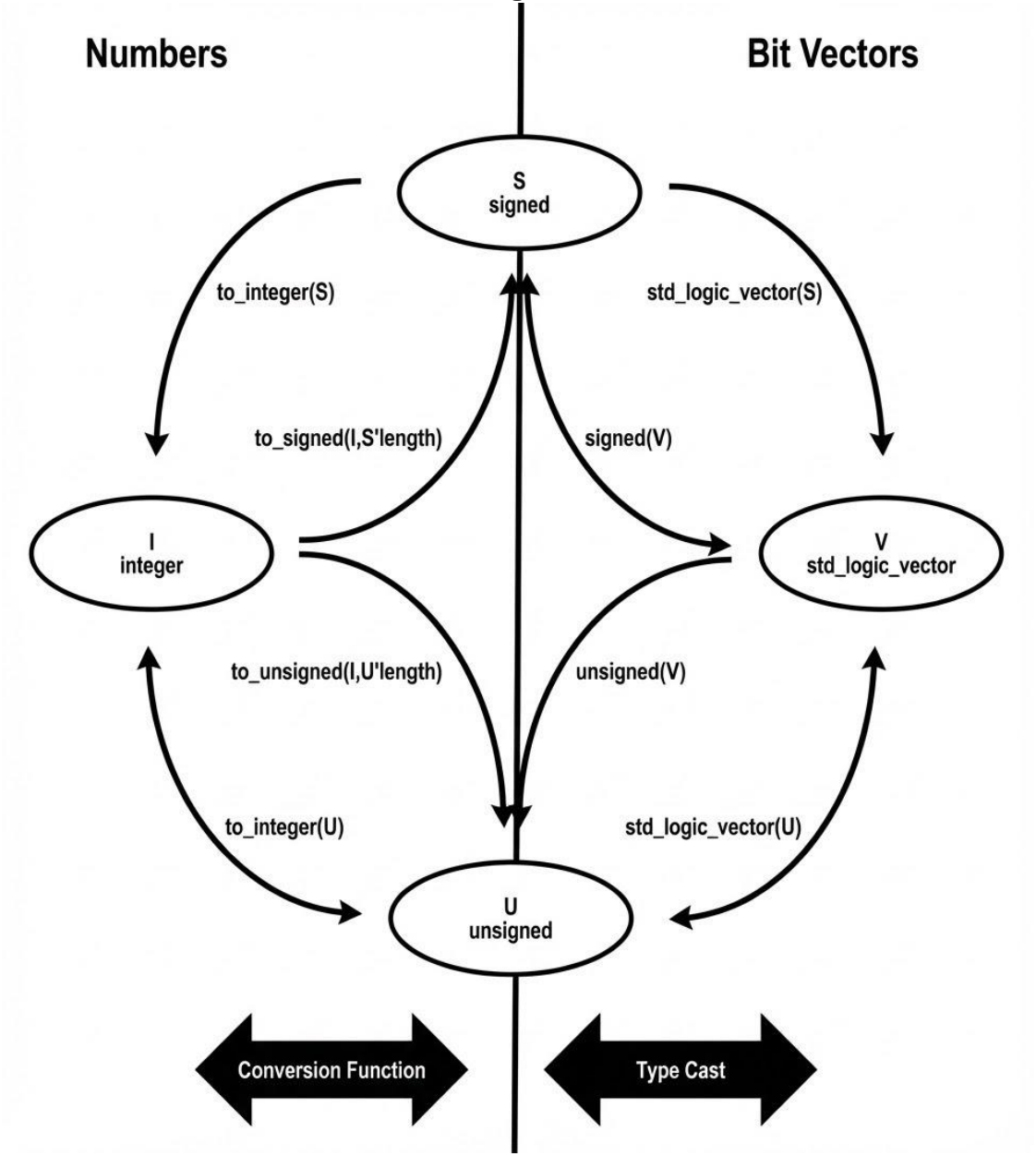
```
architecture rtl of alu4 is
   signal add_r : std_logic_vector(3 downto 0);
   signal sub_r : std_logic_vector(3 downto 0);
   signal and_r : std_logic_vector(3 downto 0);
   signal or_r  : std_logic_vector(3 downto 0);
begin
   with op select
      Y <= add_r when "00",
           sub_r when "01",
           and_r when "10",
           or_r  when others;

   add_r <= std_logic_vector(unsigned(A) + unsigned(B));
   sub_r <= std_logic_vector(unsigned(A) - unsigned(B));
   and_r <= A and B;
   or_r  <= A or B;
end architecture rtl;
```

NUMERIC_STD: Tipos numéricos

- `ieee.numeric_std` introduz tipos numéricos padrão.
  - `signed`: inteiros com sinal;
  - `unsigned`: inteiros sem sinal;
  - `integer`: inteiro utilizado como índice em arrays.

- Esses tipos serão úteis para operações envolvendo números.
  - Porém, é necessário fazer conversões para interação com `std_logic`.

NUMERIC_STD: Conversões entre tipos numéricos



Abordagem hierárquica: instanciando componentes

- Entidades VHDL podem instanciar outras entidades como componentes.
  - Para isso, deve-se nomear a entidade e conectar as portas.

```
u_alu : entity work.alu4
    port map (
        A  => SW(7 downto 4),
        B  => SW(3 downto 0),
        op => SW(9 downto 8),
        Y  => LEDR(3 downto 0)
    );
```

- Todas as entradas e saídas precisam aparecer e ser conectadas.
- `work` aqui significa “a biblioteca atual”.
  - É uma referência de contexto, semelhante a `this`/self em Java/Python.

Exemplo 5: ALU na DE1-SoC

```
library ieee;
use ieee.std_logic_1164.all;

entity alu4_de1soc is
    port (
        SW   : in  std_logic_vector(9 downto 0);
        LEDR : out std_logic_vector(9 downto 0)
    );
end entity alu4_de1soc;
```

```
architecture rtl of alu4_de1soc is
begin

    u_alu : entity work.alu4
        port map (
            A  => SW(7 downto 4),
            B  => SW(3 downto 0),
            op => SW(9 downto 8),
            Y  => LEDR(3 downto 0)
        );

    LEDR(9 downto 4) <= (others => '0');

end architecture rtl;
```

PROCESS: O motor da lógica sequencial

- Lógica sequencial precisa introduzir noção de tempo e ciclos de clock.
  - Isso é feito dentro da estrutura `process`.

- Dentro de um process, as linhas são interpretadas de forma sequencial.
  - Lembre-se: Você ainda está descrevendo um hardware!
  - O process descreve o comportamento no tempo de uma lógica sequencial.
  - Não existe “execução” como em um programa.

PROCESS: Estrutura de um registrador

```
process(clk, reset)
begin
        if reset = '1' then
                Q_reg <= (others => '0');
        elsif clk'event and clk = '1' then
                if enable = '1' then
                        Q_reg <= D;
                end if;
        end if;
end process;
```

Lista de sensibilidade: “gatilho” para interpretar process

Reset assíncrono

Evento borda de subida do clock

Registro entrada

Exemplo 6: Registrador completo

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity reg4 is
   port (
      clk    : in  std_logic;
      reset  : in  std_logic;
      enable : in  std_logic;
      D      : in  std_logic_vector(3 downto 0);
      Q      : out std_logic_vector(3 downto 0)
   );
end entity reg4;
```

```vhdl
architecture rtl of reg4 is
   signal Q_reg : std_logic_vector(3 downto 0);
begin

   process(clk, reset)
   begin
      if reset = '1' then
         Q_reg <= (others => '0');
      elsif clk'event and clk = '1' then
         if enable = '1' then
            Q_reg <= D;
         end if;
      end if;
   end process;

   Q <= Q_reg;

end architecture rtl;
```

VARIABLE: Variáveis auxiliares em um process

- O process admite a criação de variáveis auxiliares.
  - `variable` é ideal para um cálculo intermediário dentro de um process.
- Distinção importante:
  - `variable` atualiza o valor imediatamente.
  - `signal` agenda a atualização para o final do process.

```
signal A,B,C: std_logic_vector(7 downto 0);
--[...]
process(clk)
begin
 if rising_edge(clk) then
   A <= B;
   C <= A;
 end if;
end process;
```

```
signal B,C: std_logic_vector(7 downto 0);
--[...]
process(clk)
 variable A : std_logic;
begin
 if rising_edge(clk) then
   A := B;
   C <= A;
 end if;
end process;
```

VARIABLE: Variáveis auxiliares em um process

```
signal A,B,C: std_logic_vector(7 downto 0);
--[...]
process(clk)
begin
 if rising_edge(clk) then
   A <= B;
   C <= A;
 end if;
end process;
```

- A é visível fora do process;
- A recebe o valor de B;
- C recebe o valor antigo de A.

```
signal B,C: std_logic_vector(7 downto 0);
--[...]
process(clk)
 variable A : std_logic_vector(7 downto 0);
begin
 if rising_edge(clk) then
   A := B;
   C <= A;
 end if;
end process;
```

- A não é visível fora do process;
- A recebe o valor de B;
- C recebe o valor de B...
  - Que é o mesmo novo recém atribuído a A.

Exemplo 6: Acumulador com saturação

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity accum_sat is
  port (
    clk : in  std_logic;
    D   : in  std_logic_vector(7 downto 0);
    Q   : out std_logic_vector(7 downto 0)
  );
end;
```

```
architecture rtl of accum_sat is
  signal q_reg : unsigned(7 downto 0);
begin
  process(clk)
    variable tmp : unsigned(8 downto 0);
  begin
    if rising_edge(clk) then
      tmp := ('0' & q_reg) + ('0' & unsigned(D));
      if tmp(7 downto 0) > unsigned(x"64") then
        q_reg <= unsigned(x"64");
      else
        q_reg <= tmp(7 downto 0);
      end if;
    end if;
  end process;

  Q <= std_logic_vector(q_reg);
end;
```

Cuidado com o uso de Process!

- A estrutura de process incentiva a escrever código de maneira procedural.
  - É fácil colocar o design inteiro dentro de um process...
  - E “programar” como se estivesse programando um *software*.

- **Atenção:** Isso pode causar **muita** frustração.

- Tenha sempre em mente:
  - Você está descrevendo o comportamento de um hardware.
  - Você não está descrevendo os passos de um algoritmo.
  - É sempre bom visualizar mentalmente como o componente se parece.
    - E só então escrever o código VHDL que representa ele.

- Na dúvida, se a lógica é combinacional, não use process!
  - Só use process para lógica dependente de clock.

Cuidado com o uso de Process!

- Process permite utilizar a estrutura familiar if/else.
  - Exemplo: Queremos subir uma flag quando o registrador está saturado.

```
architecture rtl of accum_sat is
  signal q_reg : unsigned(7 downto 0);
  signal sat : std_logic := '0';
begin
  process(q_reg)
  begin
    if q_reg = unsigned(x"64") then
        sat <= '1';
    end if;
  end process;

  --[...]

end;
```

O código está correto?

Cuidado com o uso de Process!

- Process permite utilizar a estrutura familiar if/else.
  - Exemplo: Queremos subir uma flag quando o registrador está saturado.

```
architecture rtl of accum_sat is
  signal q_reg : unsigned(7 downto 0);
  signal sat : std_logic := '0';
begin
  process(q_reg)
  begin
    if q_reg = unsigned(x"64") then
        sat <= '1';
    end if;
  end process;

  --[...]

end;
```

- O que acontece quando `q_reg <> x"64"`?
  - Segundo o código, “nada”.
  - “Nada” significa “mantenha o valor anterior”.
    - Isso é uma memória.
    - Mais especificamente, um latch.

- Lembrança de teoria:
  - Latches são memórias implícitas quase sempre indesejadas em FPGA.
  - **Na prática, algo vai dar errado!**

Lidando com Latches

- Quase sempre, você não quer latches.
  - E nessa disciplina, arrisco dizer que nunca quer latches.
- Como evitar:
  - Todo `if` precisa de um `else`;
  - Todo `when` precisa de um `else`;
  - Todo `with/select` precisa cobrir todos os casos (`when others`).
    - É muito mais fácil pecar no **if/else** com muitas condições aninhadas.
    - **Evite process combinacional!**

  - Regra um pouco mais forte e mais geral:
    - “Todo sinal atribuído em um process combinacional deve receber valor em todos os caminhos possíveis.”

Lidando com Latches

`if` com `else`

```
process(q_reg)
 begin
  if q_reg = unsigned(x"64") then
     sat <= '1';
  else
     sat <= '0';
  end if;
 end process;
```

ou nem use process

```
sat <= '1' when q_reg = unsigned(x"64") else '0';
 -- ou
 with q_reg select
 sat <= '1' when unsigned(x"64"),
          '0' when others;
```

Síntese vs Simulação: Quando VHDL vira Software

- Os códigos e estruturas apresentadas descrevem um hardware desejado.
  - O processo de transformar o código em hardware se chama **síntese**.
  - O hardware sintetizado pode ser configurado na placa FPGA...
    - E verificado funcionalmente utilizando os periféricos dela.

  - Porém, é muito difícil fazer uma verificação robusta na placa.
    - Especialmente com um clock de 50 MHz.

  - Aí entra o fluxo de **simulação**.
    - Para simulação, escrevemos um **testbench** para verificar o circuito.
    - Um testbench é um código VHDL que instancia o componente e gera o comportamento de entradas para ele.
      - Aqui, finalmente, você vai “programar” como as entradas se comportam no tempo.

Exemplo 7: Testbench da ALU4

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity tb_alu4 is
end entity;

architecture sim of tb_alu4 is
  signal A, B, Y  : std_logic_vector(3 downto 0);
  signal op : std_logic_vector(1 downto 0);
begin
  -- DUT
  dut : entity work.alu4
    port map (A=>A, B=>B, op=>op, Y=>Y);

  -- Stimulus
  stim : process
  begin
    A <= x"0"; B <= x"0"; op <= "00"; wait for 10 ns; -- add
    A <= x"3"; B <= x"5"; op <= "00"; wait for 10 ns; -- add
    A <= x"3"; B <= x"5"; op <= "01"; wait for 10 ns; -- sub
    A <= x"A"; B <= x"5"; op <= "10"; wait for 10 ns; -- and
    A <= x"A"; B <= x"5"; op <= "11"; wait for 10 ns; -- or
    A <= x"F"; B <= x"1"; op <= "00"; wait for 10 ns; -- add
    A <= x"0"; B <= x"1"; op <= "01"; wait for 10 ns; -- sub

    wait; -- termina (mantém simulação rodando)
  end process;

end architecture;
```

Exemplo 8: Testbench do Acumulador Saturado

```
library ieee;
use ieee.std_logic_1164.all;

entity tb_accum_sat is end;
architecture sim of tb_accum_sat is
  signal clk : std_logic := '0';
  signal D   : std_logic_vector(7 downto 0);
  signal Q   : std_logic_vector(7 downto 0);
  signal sat : std_logic;
begin
  dut: entity work.accum_sat
    port map (clk=>clk, D=>D, Q=>Q, sat=>sat);

  clk <= not clk after 1 ns;
```

```
stim: process
  begin
    D <= x"30"; wait for 2 ns;  -- 48
    D <= x"30"; wait for 2 ns;  -- 96
    D <= x"30"; wait for 2 ns;  -- 144 → deve saturar em
x"64"

    assert Q = x"64"
      report "Erro: acumulador não saturou corretamente"
      severity error;

    assert sat = '1'
      report "Erro: flag sat deveria estar ativa"
      severity error;

    wait;
  end process;
end;
```

Erros clássicos dos primeiros projetos

- “Ler” uma saída (porta out):
  - Utilize sinais auxiliares internos.
- Latch acidental:
  - Usou if/when sem else ou with/select sem when others.
- Comparações/atribuições de tipos diferentes:
  - Comparou std_logic_vector com integer/signed/unsigned sem conversão.
- Usar process para tudo:
  - Prefira usar process apenas para lógica sequencial.
- Instanciar componentes dentro de process:
  - Ou pior, dentro de um if -> Criar um componente não pode ser condicional.
- Código simula mas não sintetiza:
  - Usou estruturas únicas de testbench -> wait, after, etc.