Sistemas Digitais

Aprendendo VHDL por Exemplos

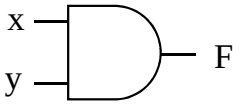
Embedded System Design – Vahid e Givargis

Logic and Computer Design Fundamentals – M. Mano e C. Kime

FPGA Prototyping by VHDL Examples – Pong P. Chu

VHDL para Porta AND
Descrição Comportamental

```
----------------------------------------------------
-- AND gate (ESD book figure 2.3)
----------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;

----------------------------------------------------

entity AND_ent is
port(   x: in std_logic;
    y: in std_logic;
    F: out std_logic
);
end AND_ent;

----------------------------------------------------

architecture behav2 of AND_ent is
begin

    F <= x and y;

end behav2;

----------------------------------------------------
```



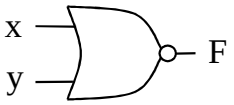
F = x y
AND

x	y	F
0	0	0
0	1	0
1	0	0
1	1	1

- Para colocar comentários num arquivo VHDL, basta iniciar a linha com dois traços (--)
- Um comentário se estende desde os dois traços (--) até o fim da linha
- A ***entity*** define as entradas e saídas do circuito
- A ***architecture*** define o que o circuito faz

VHDL para Porta NOR
Descrição Comportamental

```
-----------------------------------------
-- NOR gate (ESD figure 2.3)
-----------------------------------------

library ieee;
use ieee.std_logic_1164.all;

-----------------------------------------

entity NOR_ent is
port(   x: in std_logic;
    y: in std_logic;
    F: out std_logic
);
end NOR_ent;

-----------------------------------------

architecture behv2 of NOR_ent is
begin

    F <= x nor y;

end behv2;

-----------------------------------------
```



F = (x+y)'
NOR

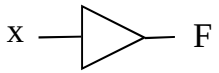
x	y	F
0	0	1
0	1	0
1	0	0
1	1	0

A ***library*** faz com que uma determinada biblioteca (neste caso **ieee**) seja carregada na compilação atual

O ***use*** delimita quais itens da biblioteca devem estar visíveis para a subsequente unidade de projeto (***entity*** + ***architecture***)

VHDL para Porta Driver
Descrição Comportamental

```
----------------------------------------
-- driver (ESD book figure 2.3)
----------------------------------------

library ieee;
use ieee.std_logic_1164.all;

----------------------------------------

entity Driver is
port(   x: in std_logic;
    F: out std_logic
);
end Driver;

----------------------------------------

architecture behv2 of Driver is
begin

    F <= x;

end behv2;

----------------------------------------
```

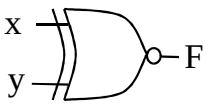


x	F
0	0
1	1

F = x
Driver

VHDL para Porta XNOR
Descrição Comportamental

```
-------------------------------------
-- XOR gate (ESD figure 2.3)
-------------------------------------

library ieee;
use ieee.std_logic_1164.all;

-------------------------------------

entity XNOR_ent is
port(   x: in std_logic;
    y: in std_logic;
    F: out std_logic
);
end XNOR_ent;

-------------------------------------

architecture behv2 of XNOR_ent is
begin

    F <= x xnor y;

end behv2;

-------------------------------------
```



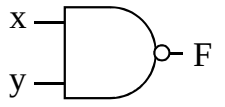
F = x ⊙ y
XNOR

x	y	F
0	0	1
0	1	0
1	0	0
1	1	1

Como XNOR é uma palavra reservada do VHDL, ela não pode ser usada como nome de uma entidade ou arquitetura (por isso foi escolhido XNOR_ent)

VHDL para Porta NAND
Descrição Comportamental

```
-----------------------------------------
-- NAND gate (ESD figure 2.3)
-----------------------------------------

library ieee;
use ieee.std_logic_1164.all;

-----------------------------------------
entity NAND_ent is
port(   x: in std_logic;
    y: in std_logic;
    F: out std_logic
);
end NAND_ent;

-----------------------------------------
architecture behv1 of NAND_ent is
begin

    process(x, y)
    begin
        if (x='1' and y='1') then
        F <= '0';
    else
        F <= '1';
    end if;
    end process;

end behv1;

-----------------------------------------
architecture behv2 of NAND_ent is
begin

    F <= x nand y;

end behv2;

-----------------------------------------
```



F = (x y)'
NAND

x	y	F
0	0	1
0	1	1
1	0	1
1	1	0

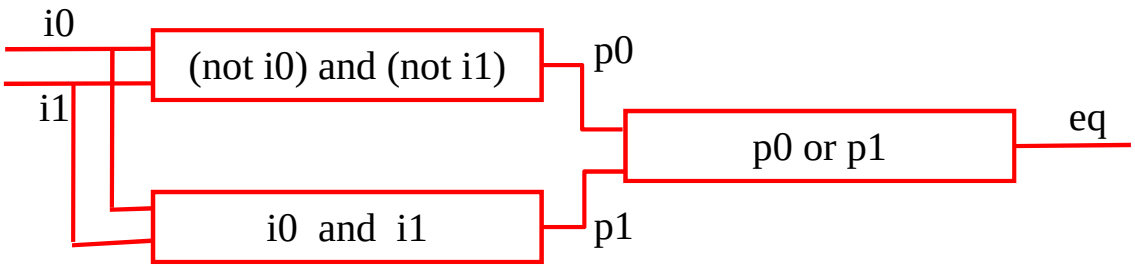
Uma *entity* pode ter mais de uma *architecture* associada a ela, mas durante a simulação apenas uma delas estará ativa

A decisão será feita pelo simulador baseada numa declaração no final do arquivo, chamada *configuration* (ou na ausência de uma *configuration* será utilizada a última *architecture* compilada)

Exemplo de Comparador de 1 Bit

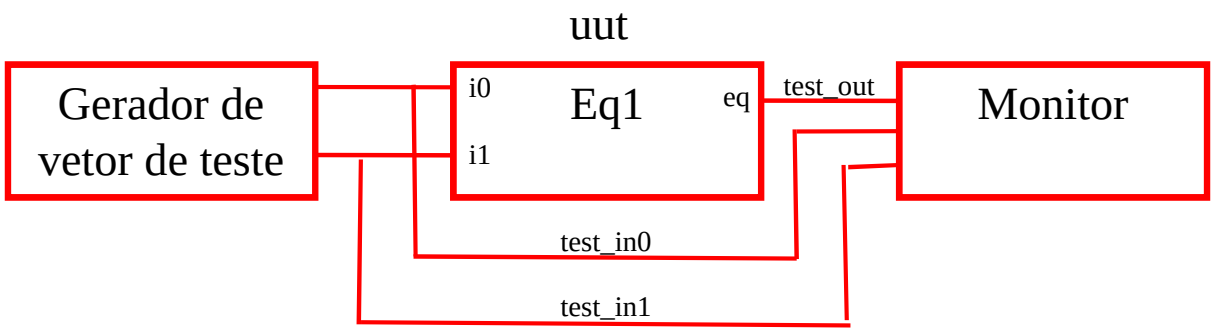
entrada	**saída**
i0i1	*eq*
00	1
01	0
10	0
11	1

eq = i0.i1 + i0 ´.i1 ´




```
-- Listing 1.1
library ieee;
use ieee.std_logic_1164.all;
entity eq1 is
  port(
    i0, i1: in std_logic;
    eq: out std_logic
  );
end eq1;

architecture sop_arch of eq1 is
  signal p0, p1: std_logic;
begin
  -- sum of two product terms
  eq <= p0 or p1;
  -- product terms
  p0 <= (not i0) and (not i1);
  p1 <= i0 and i1;
end sop_arch;
```

Testbench para o Comparador de 1 *Bit*




```
-- Listing 1.1b - Testbench (adaptação)
library ieee;
use ieee.std_logic_1164.all;
entity eq1_testbench is
end eq1_testbench;

architecture tb_arch of eq1_testbench is
  signal test_in0, test_in1: std_logic;
  signal test_out: std_logic;
begin
  -- instancia o circuito sob test (uut - unit under test)
  uut: entity work.eq1(sop_arch)
    port map(i0=>test_in0, i1=>test_in1, eq=>test_out);
  -- gerador dos vetores de teste
  process
  begin
    -- vetor de teste 1
    test_in0 <= '0';
    test_in1 <= '0';
    wait for 200 ns;
    -- vetor de teste 2
    test_in0 <= '0';
    test_in1 <= '1';
    wait for 200 ns;
    -- vetor de teste 3
    test_in0 <= '1';
    test_in1 <= '0';
    wait for 200 ns;
    -- vetor de teste 4
    test_in0 <= '1';
    test_in1 <= '1';
    wait for 200 ns;
    wait;
  end process;
end tb_arch;
```

Um *testbench* é usado para produzir os sinais de teste de um dispositivo modelado em VHDL

Processo em VHDL

- Um processo em VHDL é usado nas descrições comportamentais (*behavioral*)
- Exemplo simples de processo em VHDL:

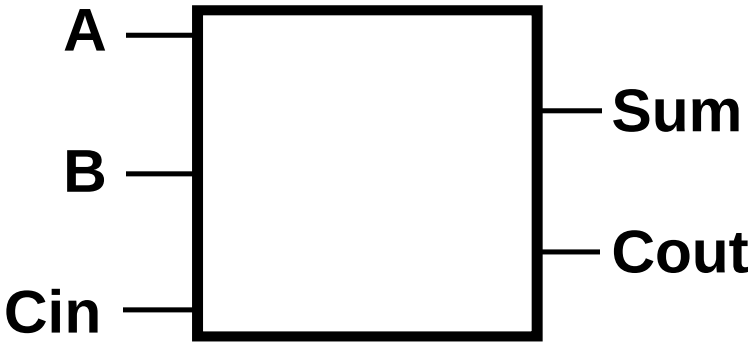
```
ARCHITECTURE behavioral OF clock_component IS
BEGIN
  PROCESS
    VARIABLE periodic: BIT := '1';
       BEGIN
         IF en = '1' THEN
           periodic := not periodic;
         END IF;
         ck <= periodic;
         WAIT FOR 1 us;
  END PROCESS;
END behavioral;
```

Sintaxe de um Processo

```
[ process_label : ] PROCESS
[( sensitivity_list )]

  process_declarations

BEGIN

  process_statements

END PROCESS [ process_label ] ;
```

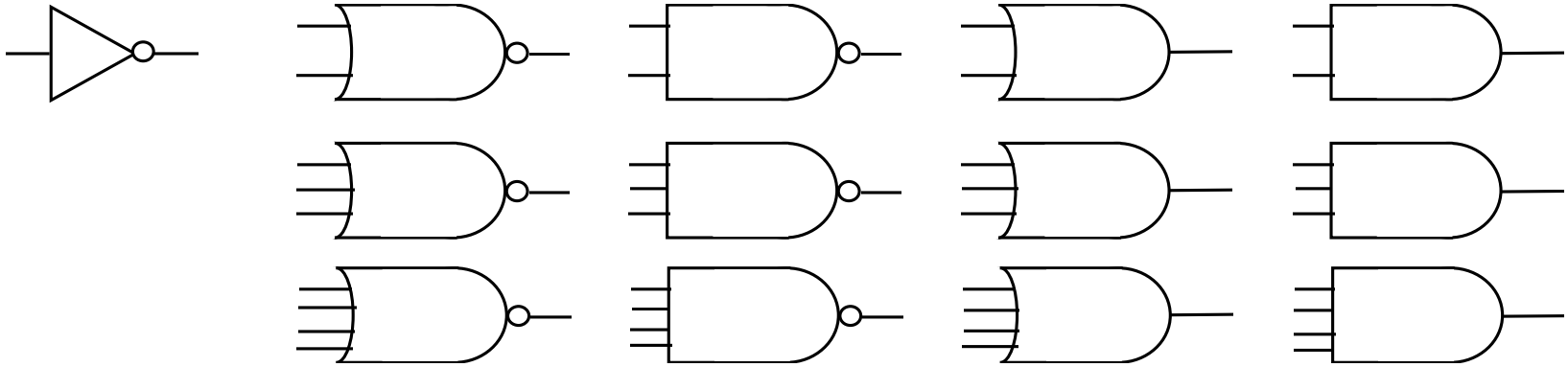
Nenhuma DECLARAÇÃO de SIGNAL dentro de um Processo!

Um Modelo de VHDL ...




```
ENTITY full_adder IS
   PORT ( A, B, Cin : IN BIT;
          Sum, Cout : OUT BIT );
END full_adder;
```

É possível construir a Arquitetura de um Somador Completo (Full Adder) usando estas portas?



Arquitetura do Somador Completo (Full Adder)

A	B	Cin	Sum	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

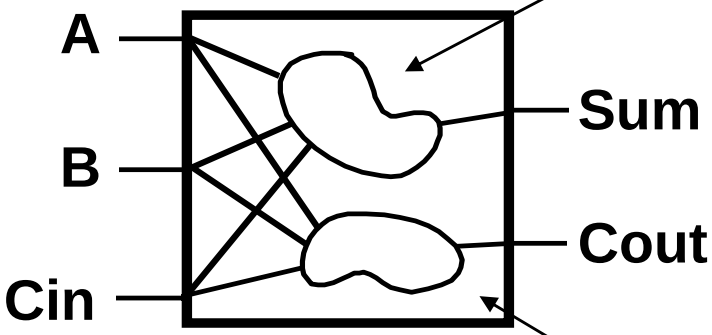
for Cout (I.e. Carry Out):

	Cin (I.e. Carry In)	
A B	0	1
0 0	0	0
0 1	0	1
1 1	1	1
1 0	0	1

for Sum:

	Cin (I.e. Carry In):	
A B	0	1
0 0	0	1
0 1	1	0
1 1	0	1
1 0	1	0

Dois Processos para o *Full Adder*




```
Summation:
PROCESS( A, B, Cin)
BEGIN
    Sum <= A XOR B XOR Cin;
END PROCESS Summation;
```

```
Carry:
PROCESS( A, B, Cin)
BEGIN
    Cout <= (A AND B) OR
            (A AND Cin) OR
            (B AND Cin);
END PROCESS Carry;
```

Arquitetura Completa

```
ARCHITECTURE example OF full_adder IS
    -- Nothing needed in declarative block...
BEGIN

 Summation:  PROCESS( A, B, Cin)
    BEGIN
        Sum <= A XOR B XOR Cin;
 END PROCESS Summation;

 Carry:  PROCESS( A, B, Cin)
    BEGIN
        Cout <= (A AND B) OR
                (A AND Cin) OR
                (B AND Cin);
 END PROCESS Carry;

END example;
```

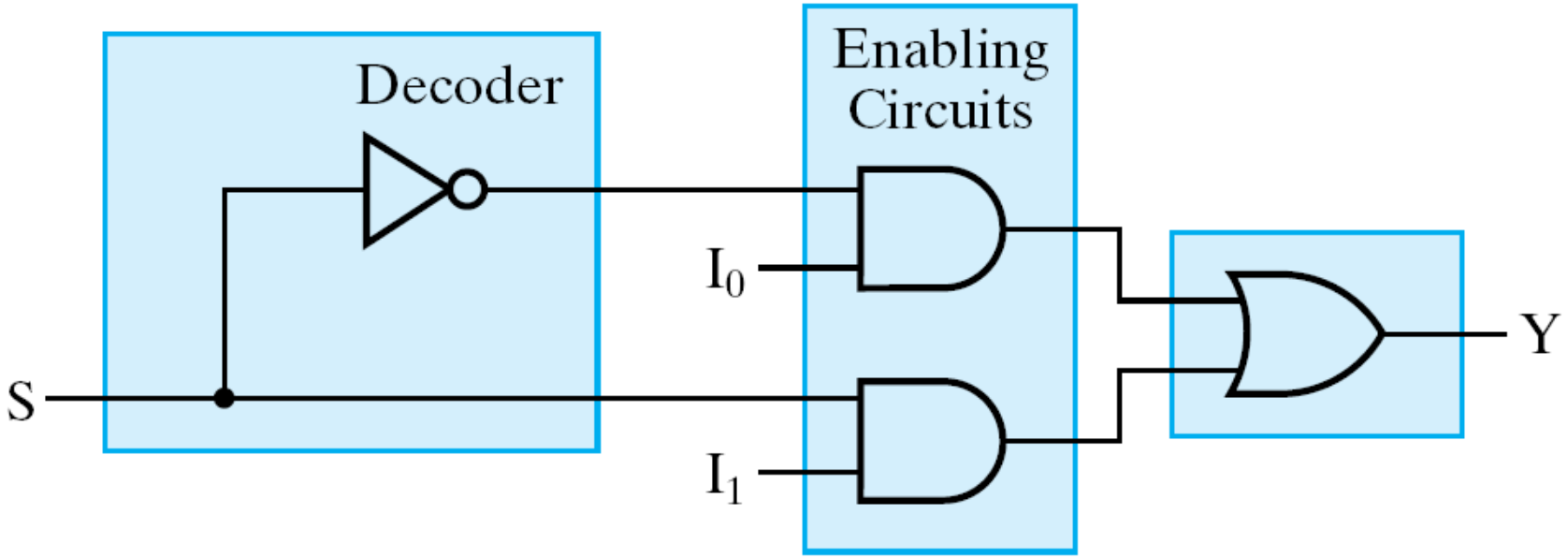
Processo Alternativo para o Carry (Cout)

```
Carry:  PROCESS( A, B, Cin)
   BEGIN
   IF ( A = '1' AND B = '1' ) THEN
     Cout <= '1';
   ELSIF ( A = '1' AND Cin = '1' ) THEN
     Cout < = '1';
   ELSIF ( B = '1' AND Cin = '1' ) THEN
     Cout <= '1';
   ELSE
     Cout <= '0';
   END IF;
 END PROCESS Carry;
```

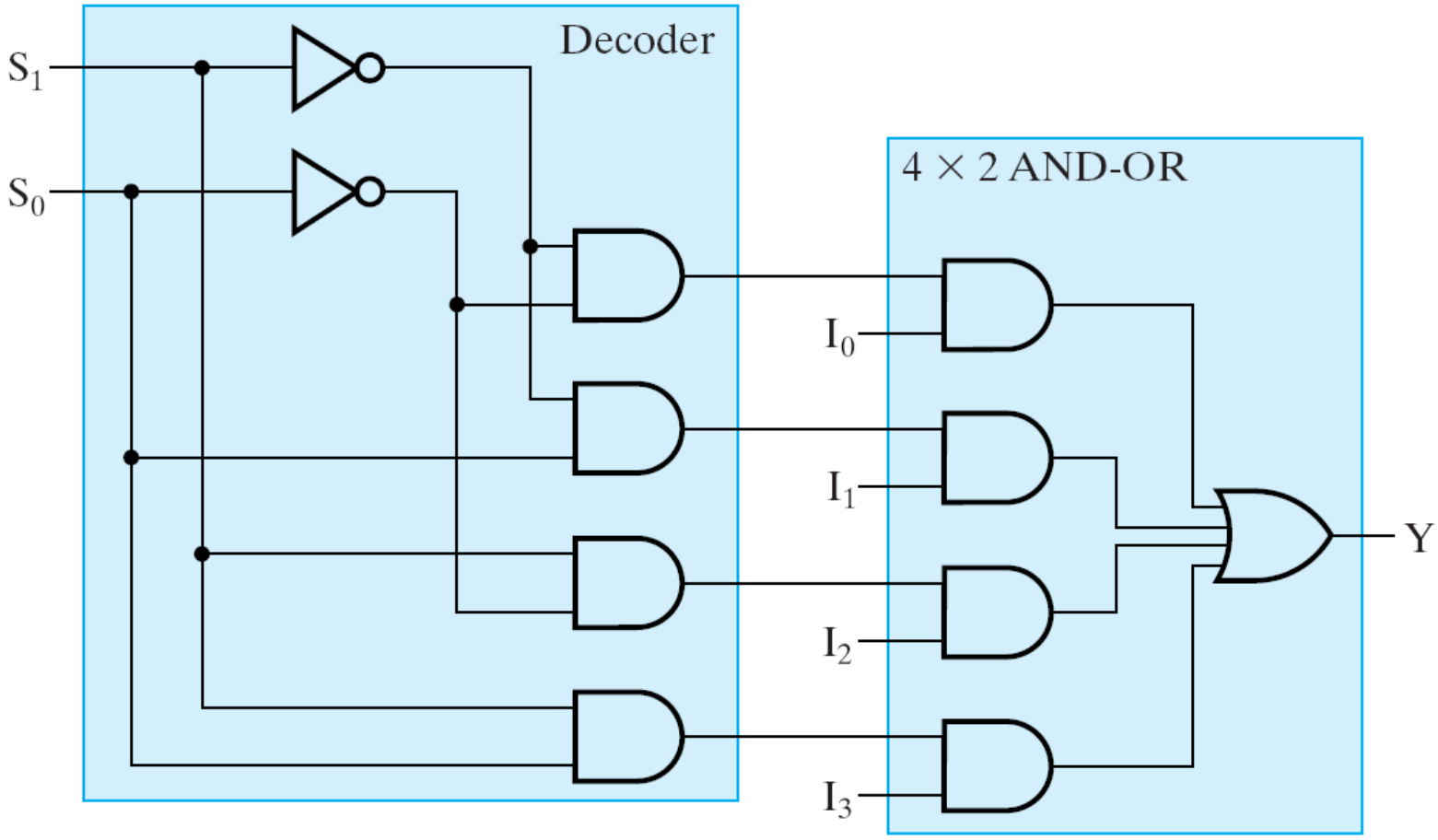
for Cout (I.e. Carry Out):

	Cin (I.e. Carry In)	
A B	0	1
0 0	0	0
0 1	0	1
1 1	1	1
1 0	0	1

Multiplexador
Exemplo 1 de Circuito Combinacional



Multiplexador 4 x 1



S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

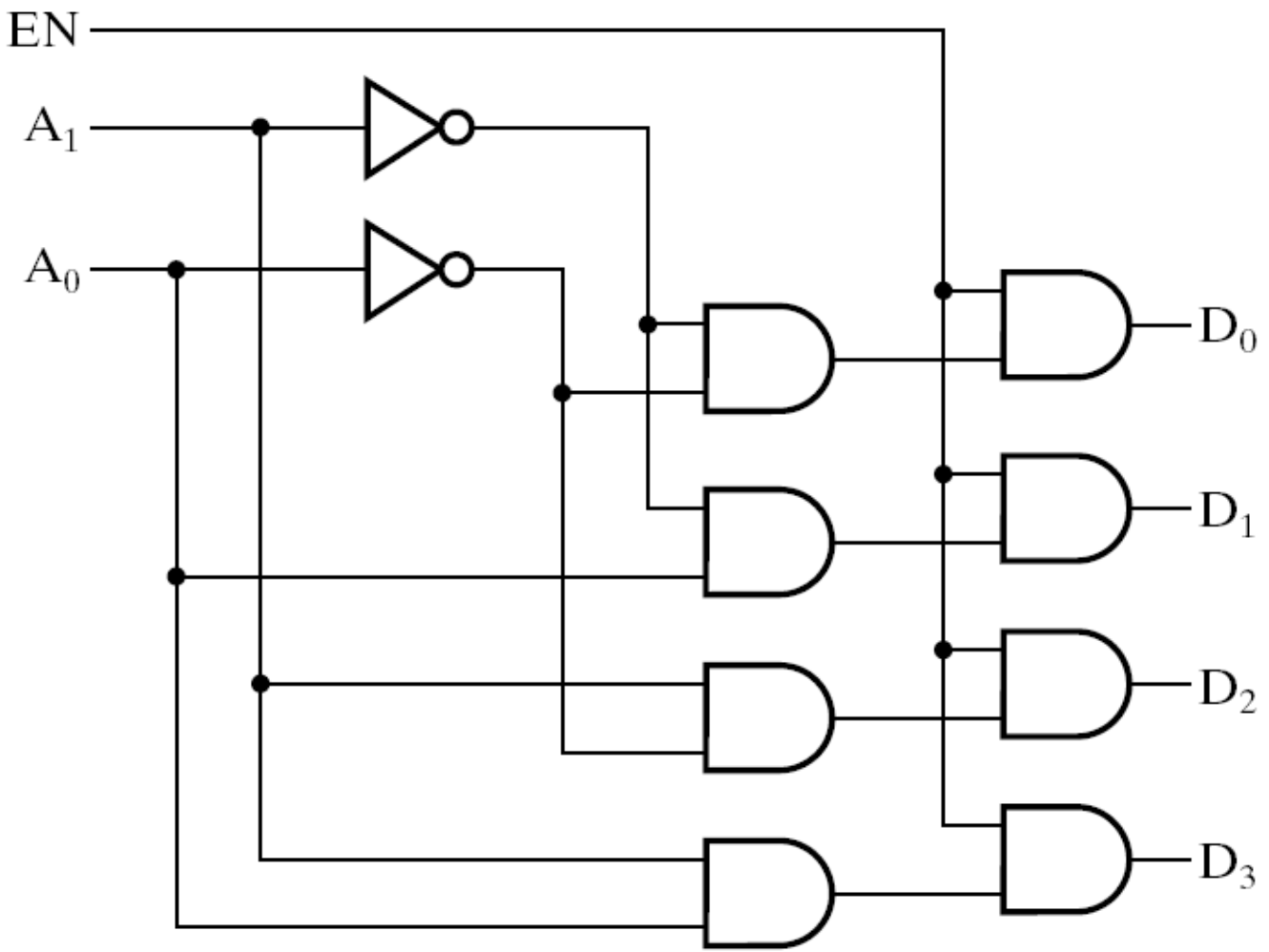
VHDL para Multiplexador 4 x 1
Descrição Fluxo de Dados

```
-- 4-to-1 Line Mux: Conditional Dataflow VHDL Description --  1
-- Using When-Else (See Table 4-7 for function table)     --  2
library ieee;                                             --  3
use ieee.std_logic_1164.all;                              --  4
entity multiplexer_4_to_1_we is                           --  5
port (S : in std_logic_vector(1 downto 0);                --  6
I : in std_logic_vector(3 downto 0);                      --  7
Y : out std_logic);                                       --  8
end multiplexer_4_to_1_we;                                --  9
                                                          -- 10
architecture function_table of multiplexer_4_to_1_we is   -- 11
begin                                                     -- 12
Y <= I(0) when S = "00" else                              -- 13
I(1) when S = "01" else                                   -- 14
I(2) when S = "10" else                                   -- 15
I(3) when S = "11" else                                   -- 16
'X';                                                      -- 17
end function_table;                                       -- 18
```

S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

Decodificador 2 x 4
Exemplo 2 de Circuito Combinatório

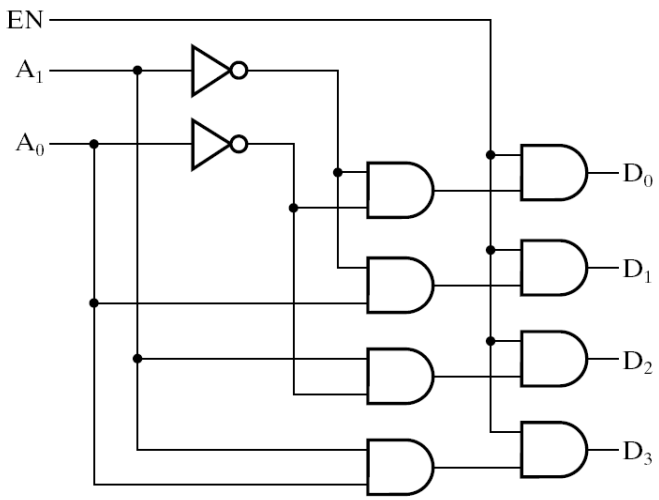
EN	A_1	A_0	D_0	D_1	D_2	D_3
0	X	X	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1



VHDL para Decodificador 2 x 4
Descrição Fluxo de Dados

```
-- 2-to-4 Line Decoder: Dataflow VHDL Description
-- (See Figure 4-10 for logic diagram)
Use library, use, and entity entries from 2_to_4_decoder_st;

architecture dataflow_1 of decoder_2_to_4_w_enable is

signal A0_n, A1_n: std_logic;
begin
A0_n <= not A0;
A1_n <= not A1;
D0 <= A0_n and A1_n and EN;
D1 <= A0 and A1_n and EN;
D2 <= A0_n and A1 and EN;
D3 <= A0 and A1 and EN;
end dataflow_1;
```

EN	A_1	A_0	D_0	D_1	D_2	D_3
0	X	X	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1



VHDL para Decodificador 2 x 4
Descrição Fluxo de Dados

```
-- 2-to-4 Line Decoder: Dataflow VHDL
Description
-- (See Figure 4-10 for logic diagram)

library ieee, lcdf_vhdl;
use ieee.std_logic_1164.all,
lcdf_vhdl.func_prims.all;

entity decoder_2_to_4_w_enable is
   port(EN, A0, A1: in std_logic;
        D0, D1, D2, D3: out std_logic);
end decoder_2_to_4_w_enable;
```

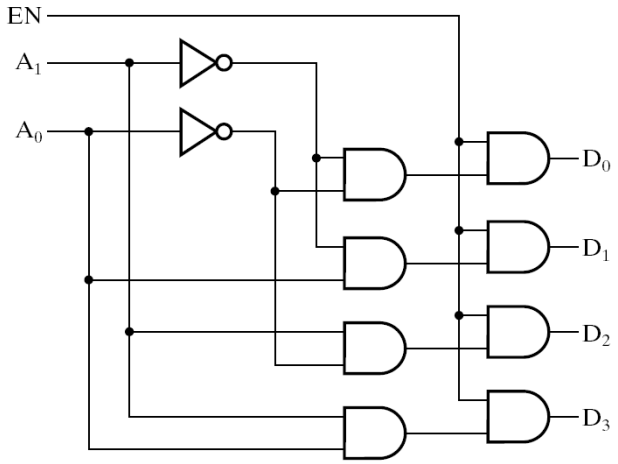
```
architecture dataflow_1 of
decoder_2_to_4_w_enable is

signal A0_n, A1_n: std_logic;
begin
   A0_n <= not A0;
   A1_n <= not A1;
   D0 <= A0_n and A1_n and EN;
   D1 <= A0 and A1_n and EN;
   D2 <= A0_n and A1 and EN;
   D3 <= A0 and A1 and EN;
end dataflow_1;
```

VHDL para Decodificador 2 x 4
Descrição Estrutural

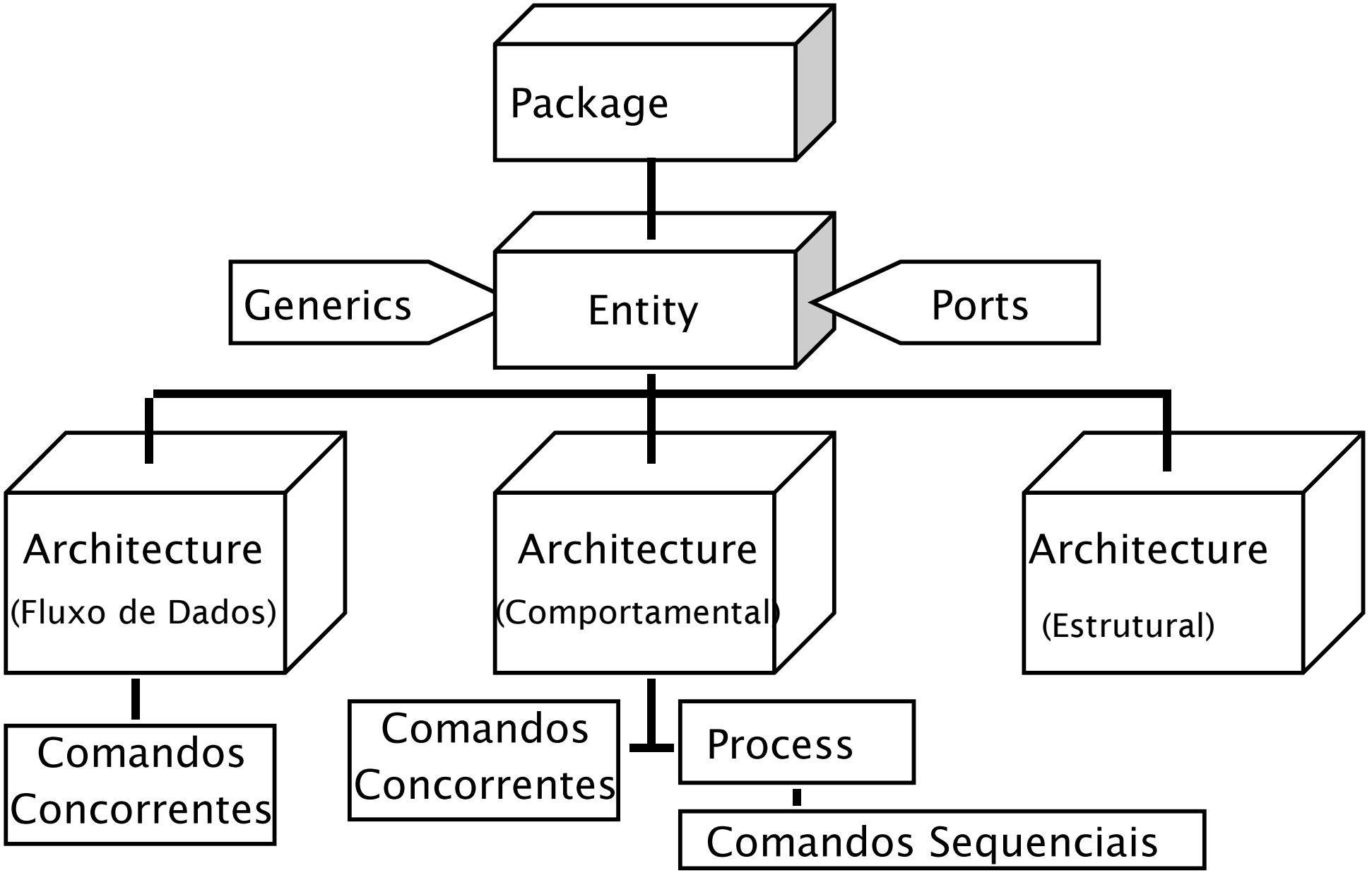
```
library ieee, lcdf_vhdl;
use ieee.std_logic_1164.all, lcdf_vhdl.func_prims.all;
entity decoder_2_to_4_w_enable is
   port(EN, A0, A1: in std_logic;
        D0, D1, D2, D3: out std_logic);
end decoder_2_to_4_w_enable;

architecture structural_1 of decoder_2_to_4_w_enable is
component NOT1
   port(in1: in std_logic;
        out1: out std_logic);
end component;
component AND2
   port(in1, in2: in std_logic;
        out1: out std_logic);
end component;
signal A0_n, A1_n, N0, N1, N2, N3: std_logic;
begin
   g0: NOT1 port map (in1 => A0, out1 =>A0_n);
   g1: NOT1 port map (in1 => A1, out1 => A1_n);
   g2: AND2 port map (in1 => A0_n, in2 => A1_n, out1 => N0);
   g3: AND2 port map (in1 => A0, in2 => A1_n, out1 => N1);
   g4: AND2 port map (in1 => A0_n, in2 => A1, out1 => N2);
   g5: AND2 port map (in1 => A0, in2 => A1, out1 => N3);
   g6: AND2 port map (in1 => EN, in2 => N0, out1 => D0);
   g7: AND2 port map (in1 => EN, in2 => N1, out1 => D1);
   g8: AND2 port map (in1 => EN, in2 => N2, out1 => D2);
   g9: AND2 port map (in1 => EN, in2 => N3, out1 => D3);
end structural_1;
```

```
-- 2-to-4 Line Decoder with Enable: Structural VHDL Description
-- (See Figure 4-10 for logic diagram)
```

EN	A_1	A_0	D_0	D_1	D_2	D_3
0	X	X	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1



Visão de Conjunto



RASSP – http://www.eda.org/rassp/modules/abstracts.html

Test Bench para o MUX 4 x 1

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

entity TestBench_MUX4 is
end;

architecture BENCH of TestBench_MUX4 is
component MUX_4_to_1_WE
port (S :in std_logic_vector(1 downto 0);
      I :in std_logic_vector(0 to 3);
      Y :out std_logic);
end component;
signal S: std_logic_vector(1 downto 0);
signal I: std_logic_vector(0 to 3);

begin
   DUT: MUX_4_to_1_WE port map (S => S, I(0) => I(0), I(1) => I(1), I
(2) => I(2), I(3) => I(3));
   S <= "00", "01" after 30 NS, "10" after 60 NS,
   "11" after 90 NS, "XX" after 120 NS,
   "00" after 130 NS;
   I(0) <= 'X', '0' after 10 NS, '1' after 20 NS;
   I(1) <= 'X', '0' after 40 NS, '1' after 50 NS;
   I(2) <= 'X', '0' after 70 NS, '1' after 80 NS;
   I(3) <= 'X', '0' after 100 NS, '1' after 110 NS;
end BENCH;
```

Um *testbench* é usado para produzir os sinais de teste de um dispositivo modelado em VHDL

VHDL para Circuitos Sequenciais

Normalmente o corpo de um processo implementa um programa sequencial

No entanto, os valores atribuídos a **sinais** só mudam no final do processo

Considere as atribuições, com A = 6, B = 9 e C = 4:

```
B <= A;
C <= B;
```

No final do processo,
B = 6 e C = 9 !!!!!!

Considere agora que A, B e C tivessem sido declarados com **variáveis**, então:

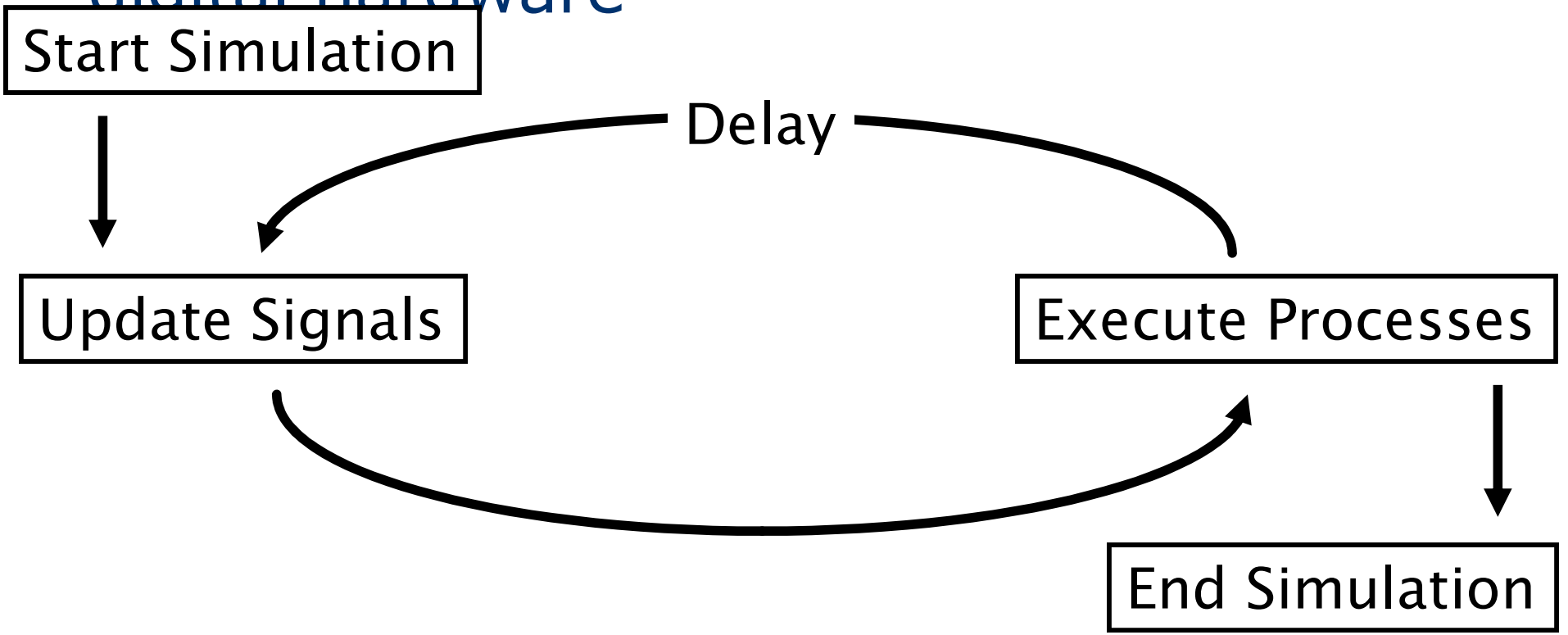
```
B := A;
C := B;
```

Como para variáveis os valores são transmitidos instantaneamente,
B = 6 e C = 6 !!!!!

Variáveis só aparecem dentro de processos !

Timing Model

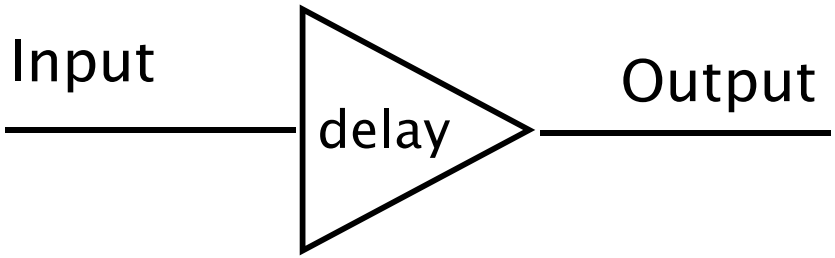
- VHDL uses the following simulation cycle to model the stimulus and response nature of digital hardware



RASSP – http://www.eda.org/rassp/modules/abstracts.html

Delay Types

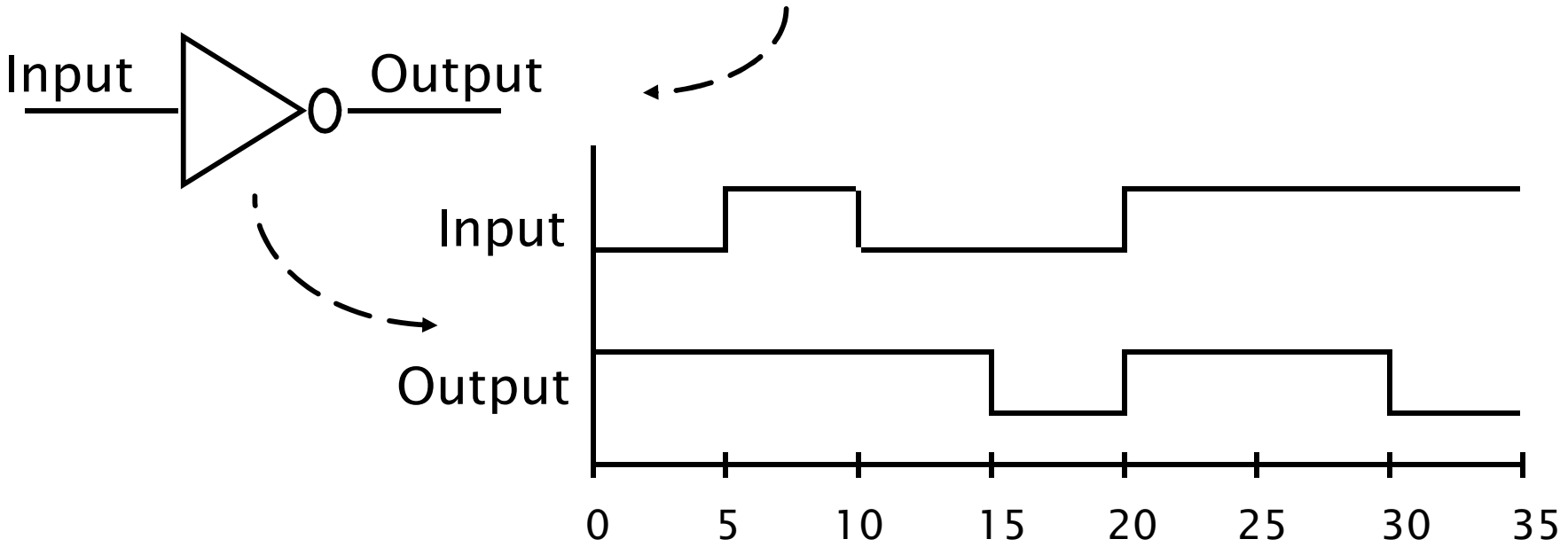
- All VHDL signal assignment statements prescribe an amount of time that must transpire before the signal assumes its new value
- This prescribed delay can be in one of three forms:
  - Transport -- prescribes propagation delay only
  - Inertial -- prescribes propagation delay and minimum input pulse width
  - Delta -- the default if no delay time is explicitly specified



Transport Delay

- Transport delay must be explicitly specified
  - I.e. keyword “TRANSPORT” must be used
- Signal will assume its new value after specified delay

```
-- TRANSPORT delay example
Output <= TRANSPORT NOT Input AFTER 10 ns;
```



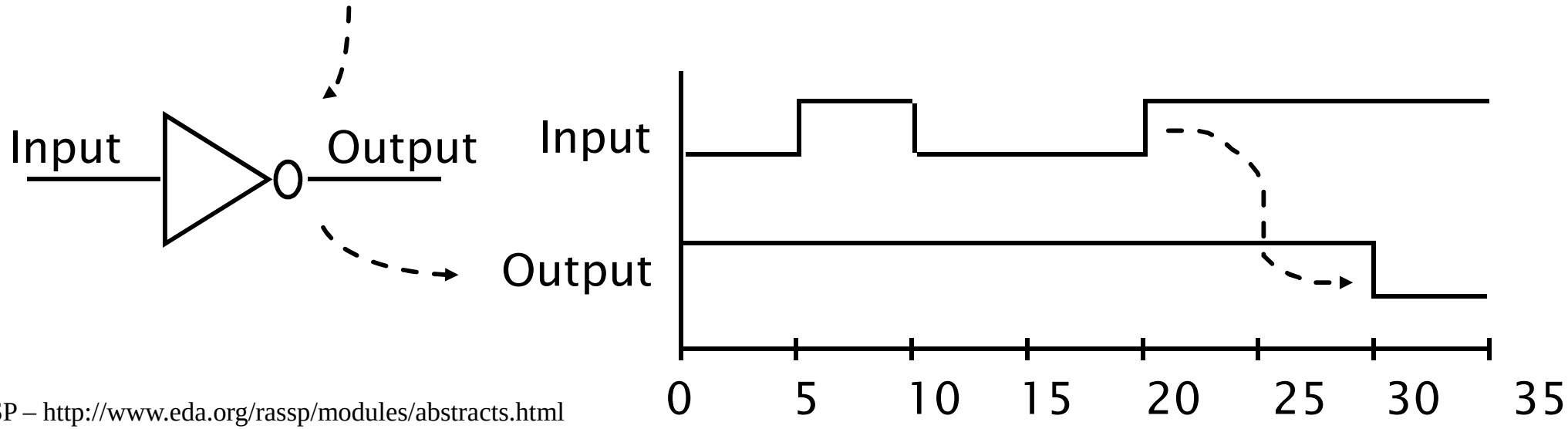
Inertial Delay

- Provides for specification propagation delay and input pulse width, i.e. 'inertia' of output:

```
target <= [REJECT time_expression] INERTIAL waveform;
```

- Inertial delay is default and REJECT is optional :

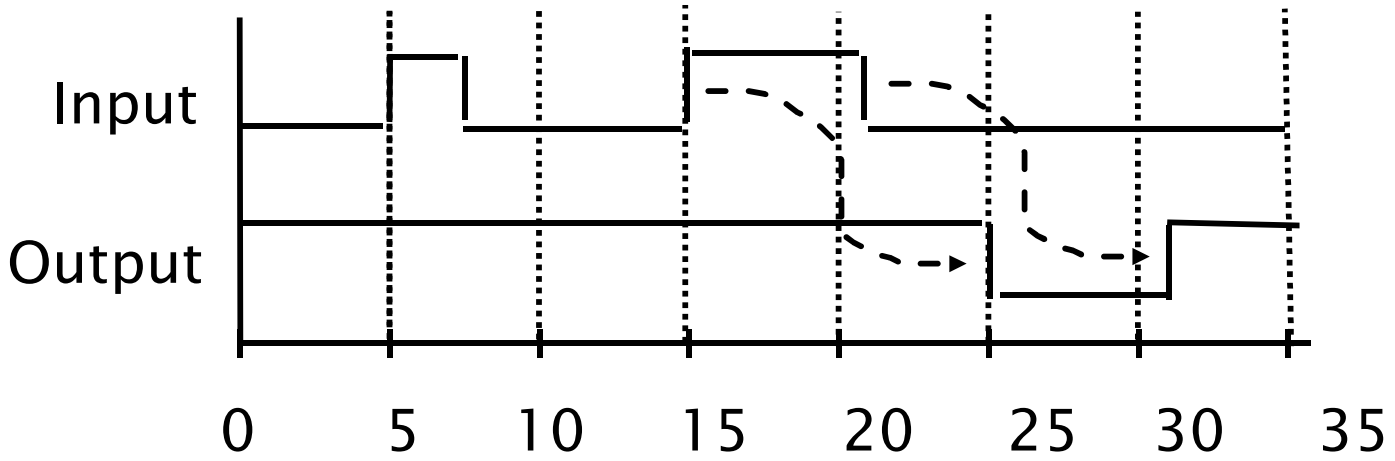
```
Output <= NOT Input AFTER 10 ns;
-- Propagation delay and minimum pulse width are 10ns
```



Inertial Delay (cont.)

- Example of gate with ‘inertia’ smaller than propagation delay
  - e.g. Inverter with propagation delay of 10ns which suppresses pulses shorter than 5ns

```
Output <=  REJECT 5ns INERTIAL NOT Input AFTER 10ns;
```



Delta Delay

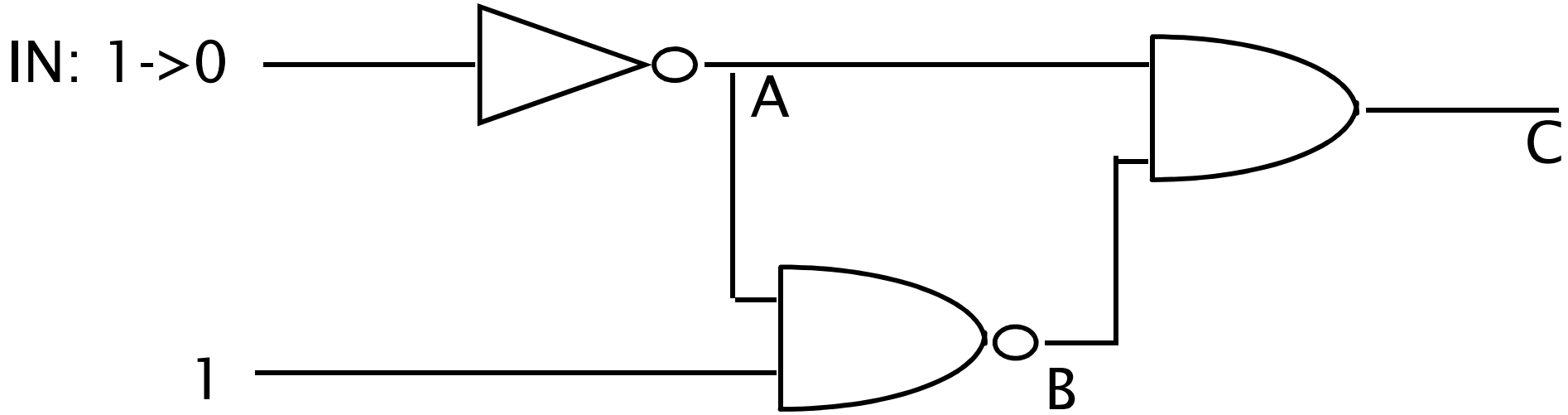
- Default signal assignment propagation delay if no delay is explicitly prescribed
  - VHDL signal assignments do not take place immediately
  - Delta is an infinitesimal VHDL time unit so that all signal assignments can result in signals assuming their values at a future time
  - E.g.

```
Output <= NOT Input;
-- Output assumes new value in one delta cycle
```

- Supports a model of concurrent VHDL process execution
  - Order in which processes are executed by simulator does not affect simulation output

Delta Delay
An Example without Delta Delay

- What is the behavior of C?



NAND gate evaluated first:

IN: 1->0

A: 0->1

B: 1->0

C: 0->0

AND gate evaluated first:

IN: 1->0

A: 0->1

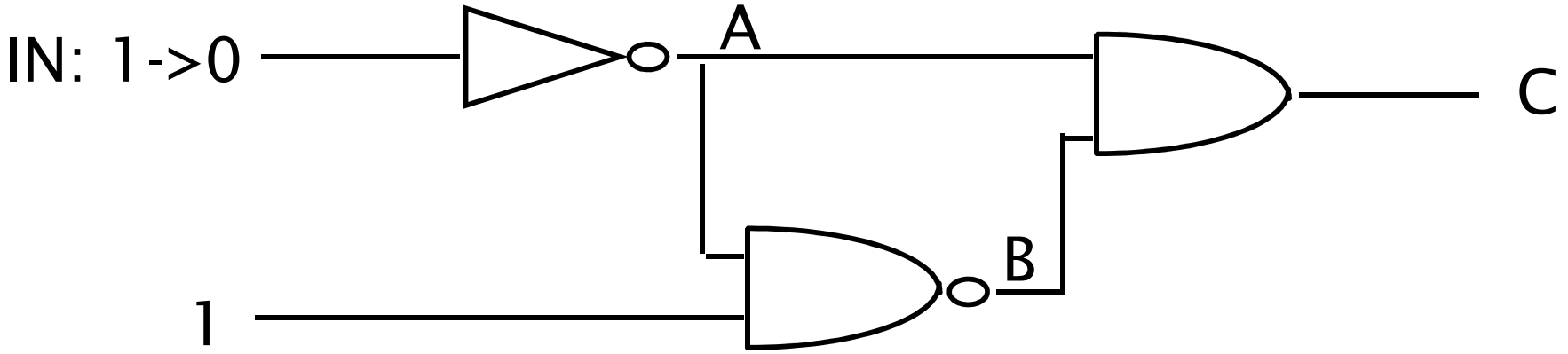
C: 0->1

B: 1->0

C: 1->0

Delta Delay
An Example with Delta Delay

- What is the behavior of C?



Using delta delay scheduling

Time	Delta	Event
0 ns	1	IN: 1->0 eval INVERTER
	2	A: 0->1 eval NAND, AND
	3	B: 1->0 C: 0->1 eval AND
	4	C: 1->0
1 ns		

Signals and Variables

- This example highlights the difference between signals and variables

```
ARCHITECTURE test1 OF mux IS
   SIGNAL x : BIT := '1';
   SIGNAL y : BIT := '0';
BEGIN
   PROCESS (in_sig, x, y)
     BEGIN
       x <= in_sig XOR y;
       y <= in_sig XOR x;
     END PROCESS;
END test1;
```

```
ARCHITECTURE test2 OF mux IS
   SIGNAL y : BIT := '0';
BEGIN
   PROCESS (in_sig, y)
     VARIABLE x : BIT := '1';
   BEGIN
     x := in_sig XOR y;
     y <= in_sig XOR x;
   END PROCESS;
END test2;
```

- Assuming a 1 to 0 transition on *in_sig*, what are the resulting values for *y* in the both cases?

VHDL Objects
Signals vs Variables

- A key difference between variables and signals is the assignment delay

```
ARCHITECTURE sig_ex OF test IS
    PROCESS (a, b, c, out_1)
    BEGIN
      out_1 <= a NAND b;
      out_2 <= out_1 XOR c;
    END PROCESS;
END sig_ex;
```

Time	a	b	c	out_1	out_2
0	0	1	1	1	0
1	1	1	1	1	0
1+d	1	1	1	0	0
1+2d	1	1	1	0	1

VHDL Objects
Signals vs Variables (Cont.)

```
ARCHITECTURE var_ex OF test IS
BEGIN
    PROCESS (a, b, c)
    VARIABLE out_3 : BIT;
    BEGIN
      out_3 := a NAND b;
      out_4 <= out_3 XOR c;
    END PROCESS;
END var_ex;
```

Time	a	b	c	out_3	out_4
0	0	1	1	1	0
1	1	1	1	0	0
1+d	1	1	1	0	1

VHDL para FF tipo D Disparado por Borda de Subida

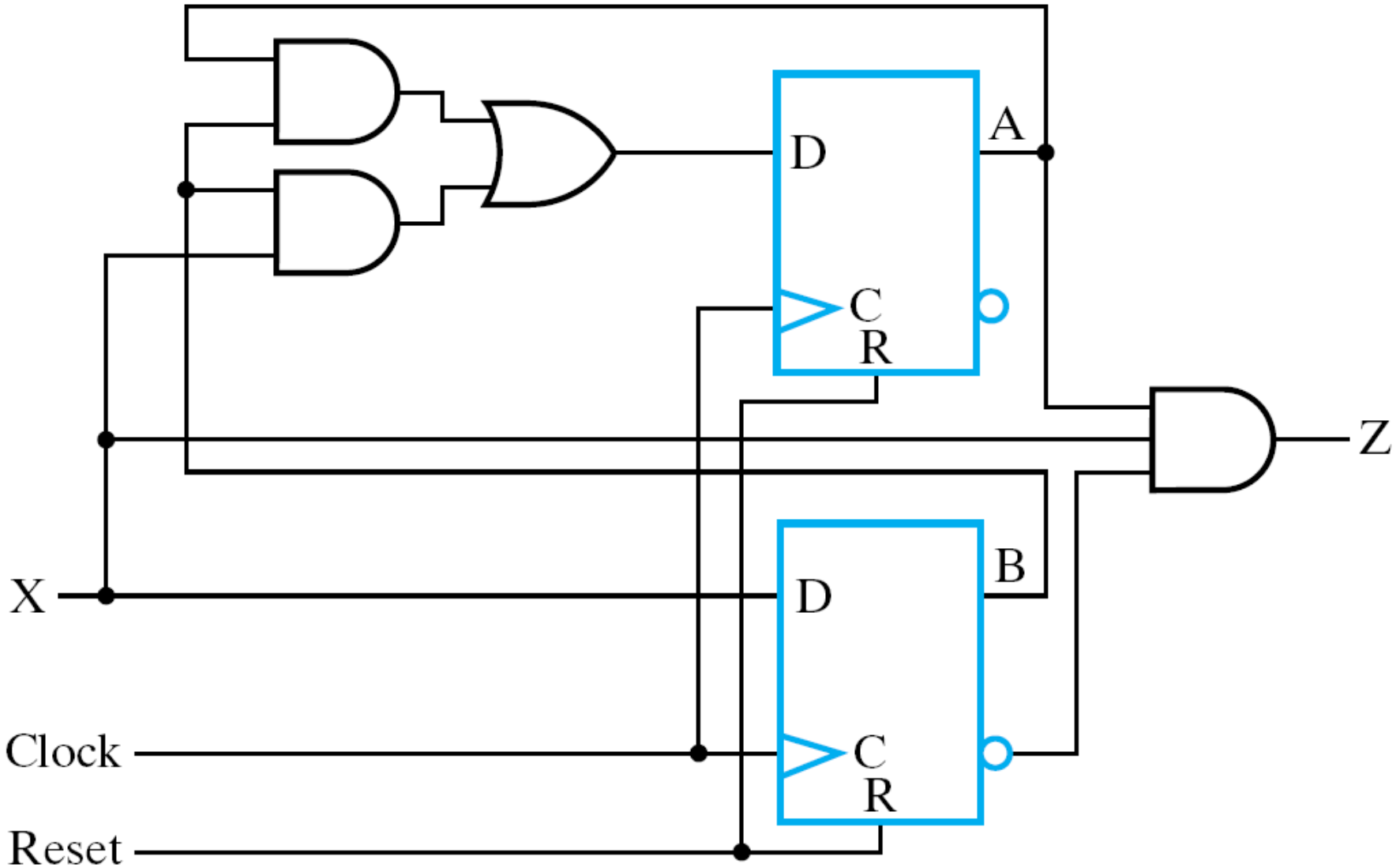
```
-- Positive Edge-Triggered D Flip-Flop with Reset:
-- VHDL Process Description

library ieee;
use ieee.std_logic_1164.all;

entity dff is
   port(CLK, RESET, D: in std_logic;
        Q, Q_n: out std_logic);
end dff;
```

Neste processo, “D” não aparece na lista de sensitividade porque uma mudança no valor de “D” não pode iniciar uma mudança em “Q” !

```
architecture  pet_pr of dff is
-- Implements positive edge-triggered bit state storage
-- with asynchronous reset.
   signal state: std_logic;
begin
   Q <= state;
   Q_n <= not state;
   process (CLK, RESET)
   begin
     if (RESET = '1') then
       state <= '0';
     else
       if (CLK'event and ClK = '1') then
          state <= D;
       end if;
     end if;
   end process;
end;
```

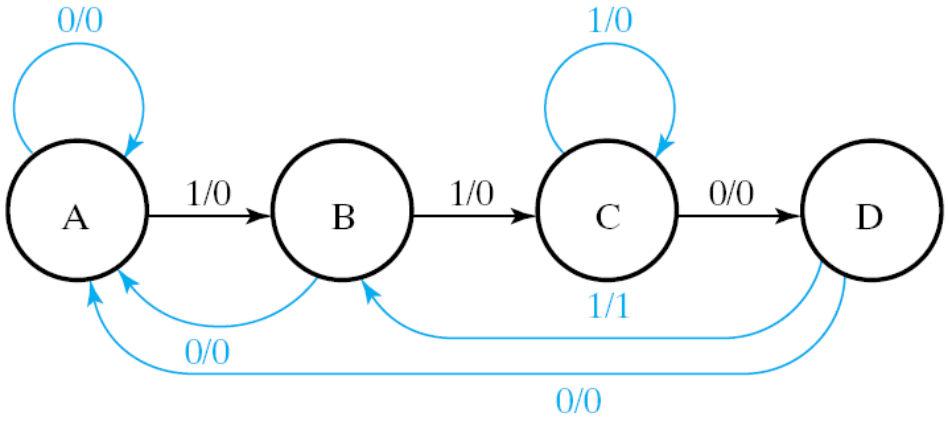
Diagrama Lógico do Reconhecedor 1101

Foram usados FFs tipo D com Reset não-invertido



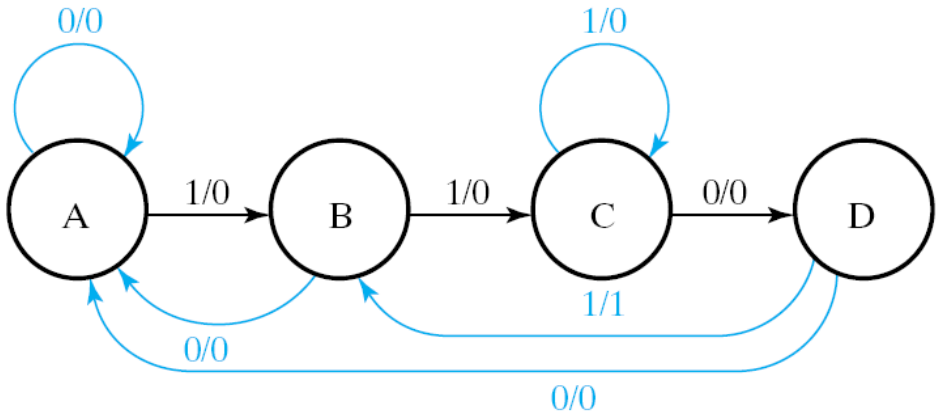
Descrição VHDL do Reconhecedor de Sequência (1101) – Estado Atual

```
-- Sequence Recognizer: VHD Process Description
-- (See Figure 6-24(d) for state diagram)
library ieee;
use ieee.std_logic_1164.all;
entity seq_rec is
   port(CLK, RESET, X: in std_logic;
        Z: out std_logic);
end seq_rec;

architecture process_3 of seq_rec is
   type state_type is (A, B, C, D);
   signal state, next_state : state_type;
begin

-- Process 1 - state_register: implements positive edge-triggered
-- state storage with asynchronous reset.
   state_register: process (CLK, RESET)
   begin
     if (RESET = '1') then
        state <= A;
     elsif (CLK'event  and CLK = '1') then
           state <= next_state;
        end if;
     end if;
end process;
```



Descrição VHDL do Reconhecedor de Sequência (1101) – Próximo Estado

```
-- Process 2 - next_state_function: implements next state as
-- a function of input X and state.
   next_state_func: process (X, state)
   begin
      case state is
         when A =>
     if X = '1' then
        next_state <= B;
     else
        next_state <= A;
           end if;
        when B =>
     if X = '1' then
        next_state <= C;
     else
        next_state <= A;
           end if;
     when C =>
         if X = '1' then
            next_state <= C;
         else
            next_state <= D;
               end if;
            when D =>
         if X = '1' then
            next_state <= B;
         else
            next_state <= A;
               end if;
          end cas
       end process;
```



Descrição VHDL do Reconhecedor de Sequência (1101) – Saída

```
-- Process 3 - output_function: implements output as function
-- of input X and state.
   output_func: process (X, state)
   begin
      case state is
         when A =>
       Z <= '0';
    when B =>
       Z <= '0';
    when C =>
       Z <= '0';
          when D =>
     if X = '1' then
        Z <= '1';
     else
        Z <= '0';
           end if;
      end case;
   end process;
end;
```