

LINGUAGENS DE DESCRIÇÃO DE HARDWARE

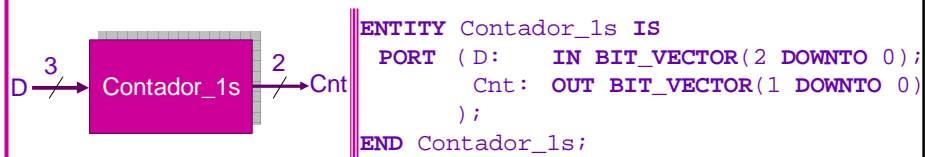
- ✿ *As linguagens de descrição de hardware ou HDL permitem capturar na forma de programas executáveis as características do modelo do sistema embutido;*
- ✿ *As características usadas na descrição de hardware de um sistema embutido são:*
 - *Transições de estados: sistemas embutidos são intrinsecamente baseados em estados e constantemente transitam de um modo para outro dependendo de eventos externos;*
 - *Comportamento hierárquico: sistemas embutidos são geralmente vistos como uma hierarquia de comportamentos, que podem ser seqüenciais ou concorrentes*
 - *Concorrência: sistemas embutidos geralmente têm vários comportamentos concorrentes, que interagem entre eles para implementar a funcionalidade do sistema;*

LINGUAGENS DE DESCRIÇÃO DE HARDWARE

- ✿ *As características usadas na descrição de hardware de um sistema embutido são (Cont.):*
 - *sincronização: sistemas embutidos geralmente consistem de vários processos concorrentes que interagem entre eles através de compartilhamento de objetos. Eles então precisam de sincronizar o uso desses objetos;*
 - *Instruções de alto nível: sistemas embutidos geralmente têm comportamentos que podem facilmente expressos usando instruções de alto-nível;*
- ✿ *VHDL ou Very High Speed Integrated Circuit HDL é uma linguagem popular para descrição de hardware para sistemas embutidos já que tem construções para capturar a maioria das necessidades deles.*
- ✿ *Depois de várias mudanças e revisões, em 1987, foi adotada com HDL padrão da IEEE.*

LINGUAGENS DE DESCRIÇÃO DE HARDWARE

- Em VHDL, qualquer circuito lógico é representado como uma entidade de projeto ou **ENTITY**: o circuito lógico pode ser simples (uma porta lógica) ou complicado (um micro-processador).
- Uma entidade de projeto consiste de uma descrição de interface e um ou mais corpos arquiteturais;
- Exemplo:** Um contador de uns, chamado **Contador_1s** tem como entrada **D** que é um vetor de 3 bits e como saída **CNT** que é um vetor de 2 bits.



LINGUAGENS DE DESCRIÇÃO DE HARDWARE

- Uma arquitetura descreve a funcionalidade de uma entidade de projeto.
- A especificação de uma arquitetura pode ser de diferentes tipos:
 - Comportamental ou algorítmica;
 - Baseada no fluxo de dados;
 - Estrutural;
 - Híbrida.
- Várias especificações arquiteturais com identificadores distintos podem co-existir para uma única entidade de projeto.

LINGUAGENS DE DESCRIÇÃO DE HARDWARE

```
ENTITY Contador_1s IS
  PORT ( D:    IN  BIT_VECTOR(2 DOWNTO 0);
         Cnt:  OUT BIT_VECTOR(1 DOWNTO 0)
        );
END Contador_1s;

ARCHITECTURE Comportamental OF Contador_1s IS
BEGIN
  ...
END Comportamental;

ARCHITECTURE Fluxo_Dados OF Contador_1s IS
BEGIN
  ...
END Fluxo_Dados;

...

ARCHITECTURE Estrutural OF Contador_1s IS
BEGIN
  ...
END Estrutural;

...
```

LINGUAGENS DE DESCRIÇÃO DE HARDWARE

```
ARCHITECTURE Comportamental OF Contador_1s IS
BEGIN
  P: PROCESS (D)
    VARIABLE Num: INTEGER RANGE 0 TO 3;
  BEGIN
    Num := 0;
    FOR I IN 0 TO 2 LOOP
      IF D(I) = '1' THEN Num := Num + 1;
    ENDIF
  END LOOP;
  CASE Num IS
    WHEN 0 => Cnt <= "00";
    WHEN 1 => Cnt <= "01";
    WHEN 2 => Cnt <= "10";
    WHEN 3 => Cnt <= "11";
  END CASE;
  END PROCESS;
END Comportamental;
```

LINGUAGENS DE DESCRIÇÃO DE HARDWARE

🧠 O projeto lógico permite calcular *Cnt* como abaixo:

$$\text{Cnt}(1) = D1.D0 + D2.D0 + D2.D1$$

$$\text{Cnt}(0) = D2.\overline{D1}.\overline{D0} + \overline{D2}.\overline{D1}.D0 + D2.D1.D0 + \overline{D2}.D1.\overline{D0}$$

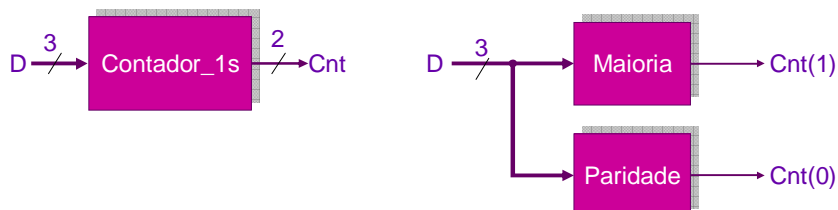
```
ARCHITECTURE Fluxo_Dados OF Contador_1s IS
  SIGNAL T0, T1, T2, T3: BIT;
BEGIN
  T0 <= D(2) AND D(1);
  T1 <= NOT D(0);
  T2 <= NOT D(1);
  T3 <= NOT D(2);
  Cnt(1) <= (D(1) AND D(0)) OR (D(2) AND D(0)) OR T0;
  Cnt(0) <= (D(2) AND T2 AND T1) OR (T3 AND T2 AND D(0))
            OR (T0 AND D(0)) OR (T2 AND D(1) AND T1);
END Fluxo_Dados;
```

LINGUAGENS DE DESCRIÇÃO DE HARDWARE

🧠 O componente *Contador_1s* pode ser visto como dois sub-componentes: um para calcular o bit mais significativo ou MSB do resultado e outro para calcular o bit menos significativo ou do resultado.

🧠 O bit mais significativo pode ser calculado com a função de maioria que retorna 1 se a maioria dos bits na entrada é 1;

🧠 O bit menos significativo pode ser calculado com a função de paridade ímpar que retorna 1 se tiver um número ímpar de 1s na entrada;



LINGUAGENS DE DESCRIÇÃO DE HARDWARE

```
ENTITY Contador_1s IS
  PORT (D:   IN  BIT_VECTOR(2 DOWNTO 0);
        Cnt: OUT BIT_VECTOR(1 DOWNTO 0)
        );
END Contador_1s;

ARCHITECTURE Estrutural OF Contador_1s IS
  COMPONENT Maioria
    PORT (E: IN  BIT_VECTOR(2 DOWNTO 0);
          S: OUT BIT);
  END COMPONENT;
  COMPONENT Paridade
    PORT (E: IN  BIT_VECTOR(2 DOWNTO 0);
          S: OUT BIT);
  END COMPONENT;
BEGIN
  Componente1: Maioria  PORT MAP(D, Cnt(1));
  Componente2: Paridade PORT MAP(D, Cnt(0));
END Estrutural;
```

LINGUAGENS DE DESCRIÇÃO DE HARDWARE

```
ENTITY Maioria IS
  PORT (E: IN  BIT_VECTOR(2 DOWNTO 0);
        S: OUT);
END Maioria;

ARCHITECTURE Estrutural OF Maioria IS
  COMPONENT And2
    PORT (E1, E2: IN  BIT;
          S:      OUT BIT);
  END COMPONENT;
  COMPONENT Or3
    PORT (E1, E2, E3: IN  BIT;
          S:          OUT BIT);
  END COMPONENT;
  SIGNAL T1, T2, T3: BIT;
BEGIN
  Porta1: And2 PORT MAP(E(0), E(1), T1);
  Porta2: And2 PORT MAP(E(0), E(2), T2);
  Porta3: And2 PORT MAP(E(1), E(2), T3);
  Porta4: OR3  PORT MAP(T1, T2, T3, S);
END Estrutural;
```

LINGUAGENS DE DESCRIÇÃO DE HARDWARE

```
ENTITY And2 IS
  PORT ( E1, E2: IN  BIT;
         S:      OUT BIT);
END And2;
ARCHITECTURE Fluxo_Dados OF And2 IS
BEGIN
  S <= E1 AND E2;
END Fluxo_Dados;

ENTITY Or3 IS
  PORT ( E1, E2, E3: IN  BIT;
         S:          OUT BIT);
END Or3;
ARCHITECTURE Fluxo_Dados OF Or3 IS
BEGIN
  S <= E1 OR E2 OR E3;
END Fluxo_Dados;
```

LINGUAGENS DE DESCRIÇÃO DE HARDWARE

```
ENTITY Paridade IS
  PORT ( E: IN  BIT_VECTOR(2 DOWNTO 0);
         S: OUT BIT);
END Paridade;
ARCHITECTURE Estrutural OF Paridade IS
  COMPONENT Xor2
    PORT ( E1, E2: IN  BIT;
           S:      OUT BIT);
  END COMPONENT;
  SIGNAL T: BIT;
BEGIN
  Porta1: Xor2 PORT MAP(E(0), E(1), T);
  Porta2: Xor2 PORT MAP(T, E(2), S);
END Fluxo_Dados;
ENTITY Xor2 IS
  PORT ( E1, E2: IN  BIT;
         S:      OUT BIT);
END Xor2;
ARCHITECTURE Fluxo_Dados OF Xor2 IS
BEGIN
  S <= E1 XOR E2;
END Fluxo_Dados;
```

LINGUAGENS DE DESCRIÇÃO DE HARDWARE

```
ENTITY And2 IS
  PORT ( E1, E2: IN  BIT;
         S:      OUT BIT);
END And2;
ARCHITECTURE Fluxo_Dados OF And2 IS
BEGIN
  S <= E1 AND E2;
END Fluxo_Dados;

ENTITY Or3 IS
  PORT ( E1, E2, E3: IN  BIT;
         S:          OUT BIT);
END Or3;
ARCHITECTURE Fluxo_Dados OF Or3 IS
BEGIN
  S <= E1 OR E2 OR E3;
END Fluxo_Dados;
```

LINGUAGENS DE DESCRIÇÃO DE HARDWARE

- 🔧 *Para simular o hardware descrito num programa VHDL, uma entidade de projeto adicional, geralmente chamada TEST_BENCH ou vetor de teste precisa ser incluída;*
- 🔧 *A entidade TEST_BENCH não tem nem sinais de entrada nem sinais de saída;*
- 🔧 *A entidade TEST_BENCH cria uma instância do componente que está sendo testado mapeando os estímulos externos aos sinais de entrada do componente;*
- 🔧 *Para testar o comportamento do componente Contador_1s, precisa-se criar os 8 estímulos externos do sinal D já que este sinal consiste de três bits.*

LINGUAGENS DE DESCRIÇÃO DE HARDWARE

```
ENTITY TEST_BENCH IS
END TEST_BENCH;

ARCHITECTURE Teste_Contador_1s OF TEST_BENCH IS
  COMPONENT Contador_1s
    PORT ( D:    IN    BIT_VECTOR(2 DOWNTO 0);
           Cnt: OUT  BIT_VECTOR(1 DOWNTO 0));
  END COMPONENT;
  SIGNAL S1: BIT_VECTOR(2 DOWNTO 0);
  SIGNAL S2: BIT_VECTOR(1 DOWNTO 0);
BEGIN
  Count: Contador_1s PORT MAP(S1, S2);
  PROCESS
  BEGIN
    S1 <= "000" AFTER 1 ns, "001" AFTER 2 ns,
          "010" AFTER 3 ns, "011" AFTER 4 ns,
          "100" AFTER 5 ns, "101" AFTER 6 ns,
          "110" AFTER 7 ns, "111" AFTER 8 ns;

    WAIT;
  END PROCESS;
END Estrutural;
```

LINGUAGENS DE DESCRIÇÃO DE HARDWARE

👉 Para os componentes que possuem vários corpos arquiteturais, a arquitetura a ser usada na instânciação é definida:

- ❶ Na parte de instruções da ARCHITECTURE: Geralmente, quando tem uma única instância do componente;

```
Count: ENTITY WORK.Contador_1s(Estrutural) PORT MAP ...
```

- ❷ Na parte declarativa da ARCHITECTURE: Geralmente, quando tem várias instâncias do mesmo componente;

```
COMPONENT A2 PORT(E1,E2: IN BIT; S:OUT BIT); -- Declaração
FOR ALL: A2 USE ENTITY WORK.And2(Comportamental);
...
Porta1: A2PORT MAP(E(0), E(1), T1); -- Instruções
Porta2: A2PORT MAP(E(0), E(2), T2);
```


LINGUAGENS DE DESCRIÇÃO DE HARDWARE

- ③ *Numa parte especial, chamada CONFIGURATION: Geralmente, quando tem várias instâncias de vários componentes;*

```
COMPONENT C                                -- Declaração
  PORT ( A: IN  BIT_VECTOR(2 DOWNT0 0);
        B: OUT  BIT_VECTOR(1 DOWNT0 0));
END COMPONENT;

...
  Count: C  PORT MAP(S1, S2);              -- Instruções
...

CONFIGURATION Config OF Teste_Contador_1s IS
  FOR Teste_Contador_1s
    FOR Count: C
      USE ENTITY WORK.Contador_1s(Comportamental);
    END FOR;
    ...
  END FOR;
  ...
END Config;
```

LINGUAGENS DE DESCRIÇÃO DE HARDWARE

Módulos em VHDL

- 🔧 *Um módulo é chamado PACKAGE que tem uma parte declarativa e outra de detalhamento;*

```
PACKAGE Modulo_Comp IS
  COMPONENT C1 PORT (...); END COMPONENT;
  COMPONENT C2 PORT (...); END COMPONENT;
  ...
  COMPONENT Cn PORT (...); END COMPONENT;
END Modulo_Comp;

PACKAGE Modulo_Util IS
  TYPE Inteiros IS ARRAY(0 TO 10) OF INTEGER;
  ...
  FUNCTION Calcular(a,b: BIT) RETURN BIT;
  ...
  PROCEDURE Aplicar(x,y: INTEGER);
  ...
END Modulo_Util;
```

LINGUAGENS DE DESCRIÇÃO DE HARDWARE

- ✿ *A parte de detalhamento **PACKAGE BODY** especifica a implementação das funções e procedimentos da parte declarativa.*

```
PACKAGE BODY Modulo_Util IS
  FUNCTION Calcular(a,b: BIT) RETURN BIT;
  BEGIN
    ...
    RETURN ...
  END Calcular;
  ...
  PROCEDURE Aplicar(x,y: INTEGER);
  BEGIN
    ...
  END Aplicar;
  ...
END Modulo_Util;
```

LINGUAGENS DE DESCRIÇÃO DE HARDWARE

- ✿ *Para usar um **PACKAGE**, basta incluir uma instrução de uso.*

```
USE WORK.Modulo_Compt;
USE WORK.Modulo_Util;
...
USE WORK.ALL;
```

- ✿ *Um conjunto de **ENTITYs**, **ARCHITECTUREs** e **PACKAGEs** podem ser compilados para formar uma biblioteca ou **LIBRARY**;*

- ✿ *A linguagem **VHDL** não permite a criação de bibliotecas mas permite usá-las;*

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
```

LINGUAGENS DE DESCRIÇÃO DE HARDWARE

- 🔧 A biblioteca **WORK** é disponível implicitamente para todos os projetos;
- 🔧 O módulo **std_logic_1164** da biblioteca **IEEE** é distribuído com qualquer ferramenta baseada em VHDL;
- 🔧 O módulo **std_logic_1164** define um novo sistema lógico que permite usar nove valores lógicos;

Valor	representa
'U'	não inicializado
'X'	não definido forte
'0'	0 forte
'1'	1 forte

'Z'	alta impedância
'W'	não definido fraco
'L'	0 fraco
'H'	1 fraco
'-'	don't care

LINGUAGENS DE DESCRIÇÃO DE HARDWARE

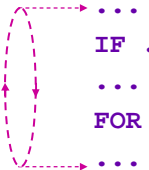
- 🔧 Tabela de verdade para a porta AND no sistema lógico **std_logic**

	'U'	'X'	'0'	'1'	'Z'	'W'	'L'	'H'	'-'
'U'	'U'	'U'	'0'	'U'	'U'	'U'	'0'	'U'	'U'
'X'	'U'	'X'	'0'	'X'	'X'	'X'	'0'	'X'	'X'
'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'
'1'	'U'	'X'	'0'	'1'	'X'	'X'	'0'	'1'	'X'
'Z'	'U'	'X'	'0'	'X'	'X'	'X'	'0'	'X'	'X'
'W'	'U'	'X'	'0'	'X'	'X'	'X'	'0'	'X'	'X'
'L'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'
'H'	'U'	'X'	'0'	'1'	'X'	'X'	'0'	'1'	'X'
'-'	'U'	'X'	'0'	'X'	'X'	'X'	'0'	'X'	'X'

LINGUAGENS DE DESCRIÇÃO DE HARDWARE

Comportamento Seqüencial

```
ARCHITECTURE Sequential
...
BEGIN
  ...
  P: PROCESS ...
    ...
    IF ... THEN ... ELSE ... END IF;
    ...
    FOR ... LOOP
      ...
    END PROCESS
  ...
END Sequential
```



LINGUAGENS DE DESCRIÇÃO DE HARDWARE

Comportamento Concorrente

```
ARCHITECTURE Concorrente
...
BEGIN
  ...
  → atribuição de sinal 1
  ...
  → instância de componente 1
  ...
  → atribuição de sinal 2
  ...
  → instância de componente 2
  ...
  → atribuição de sinal n
  ...
  → instância de componente n
  ...
END Concorrente
```

LINGUAGENS DE DESCRIÇÃO DE HARDWARE

Objetos em VHDL

Utilização dos objetos disponíveis em VHDL		Comportamento					
		Concorrente			Seqüencial		
		declara	modifica	usa	declara	modifica	usa
OBJETOS	Sinal	SIM	SIM	SIM	NÃO	SIM	SIM
	Variáveis	NÃO	NÃO	SIM	SIM	SIM	SIM
	Constantes	SIM	—	SIM	SIM	—	SIM

LINGUAGENS DE DESCRIÇÃO DE HARDWARE

Atribuição de sinais

🔧 Atribuição sem atraso

`S <= waveform;`



🔧 Atribuição com atraso de transporte

`S <= TRANSPORT waveform AFTER 5 ns;`

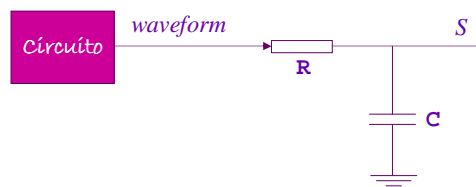


LINGUAGENS DE DESCRIÇÃO DE HARDWARE

🔧 Atribuição com atraso de inércia

```
S <= waveform AFTER 5 ns;
```

```
S <= REJECT 3 ns INERTIAL waveform AFTER 5 ns;
```



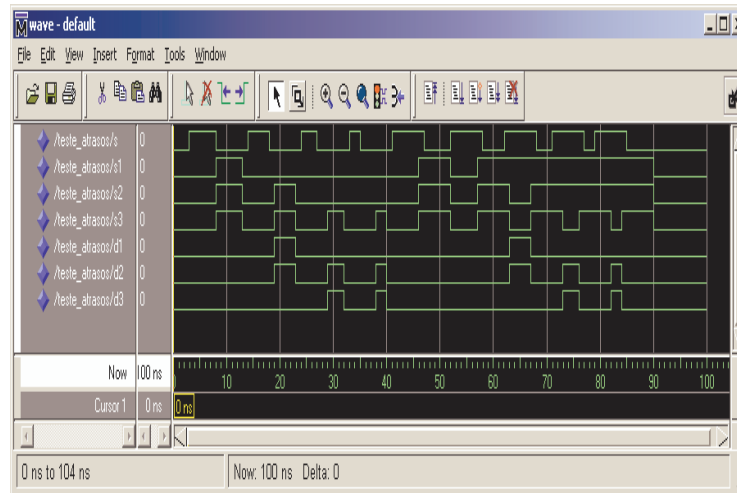
LINGUAGENS DE DESCRIÇÃO DE HARDWARE

```
ENTITY Atrasos IS END Atrasos;

ARCHITECTURE Teste_Atrasos OF Atrasos IS
    SIGNAL S,S1, S2, S3,D1, D2, D3: BIT;
    BEGIN
        S1 <= S AFTER 5 ns;
        S2 <= REJECT 3 ns INERTIAL S AFTER 5 ns;
        S3 <= TRANSPORT S AFTER 5 ns;
        D1 <= S1 XOR S2;
        D2 <= S1 XOR S3;
        D3 <= S2 XOR S3;
        S <= '1' AFTER 3 NS, '0' AFTER 8 NS, '1' AFTER 14 NS,
            '0' AFTER 18 NS, '1' AFTER 24 NS, '0' AFTER 27 NS,
            '1' AFTER 33 NS, '0' AFTER 35 NS, '1' AFTER 41 NS,
            '0' AFTER 47 NS, '1' AFTER 52 NS, '0' AFTER 58 NS,
            '1' AFTER 62 NS, '0' AFTER 68 NS, '1' AFTER 71 NS,
            '0' AFTER 77 NS, '1' AFTER 79 NS, '0' AFTER 85 NS;
    END Teste_Atrasos;
```




LINGUAGENS DE DESCRIÇÃO DE HARDWARE

Resultado da simulação do exemplo Atrasos



LINGUAGENS DE DESCRIÇÃO DE HARDWARE

Parametrização de projetos

-  *Modelos de componentes podem ser parametrizados para uso geral em diferentes projetos;*
-  *O comportamento dos componentes gerados a partir de modelos parametrizados depende dos valores atribuídos aos parâmetros;*
-  *Exemplo de uma porta NOT cujo o tempo de resposta é parametrizado.*

```
...  
COMPONENT Inv                                -- declaração  
    GENERIC (Tph: TIME; Tphl: TIME);  
    PORT (E: IN BIT; S: OUT BIT);  
END COMPONENT;  
SIGNAL i,o: BIT;  
...                                           -- instrução  
porta: Inv GENERIC MAP(2 NS, 4 NS) PORT MAP(i,o);
```

LINGUAGENS DE DESCRIÇÃO DE HARDWARE

- *A composição de um componentes em termos de outros sub-componentes pode ser parametrizada.*

```
ENTITY Registrador IS
    GENERIC (N: NATURAL);
    PORT (Clk: IN BIT; E: IN BIT_VECTOR(N-1 DOWNT0 0);
          S: OUT BIT_VECTOR(N-1 DOWNT0 0));
END Registrador;

...
COMPONENT Flip_Flop                                -- declaração
    PORT (Clk: IN BIT; P: IN BIT; Q: OUT BIT);
END Flip_Flop;
SIGNAL C: BIT;
SIGNAL D,R: BIT_VECTOR(N-1 DOWNT0 0);
...
FOR i IN 0 TO N-1 GENERATE                          -- instrução
    FF: Flip_Flop PORT MAP(C, D(i), R(i));
END GENERATE;
```

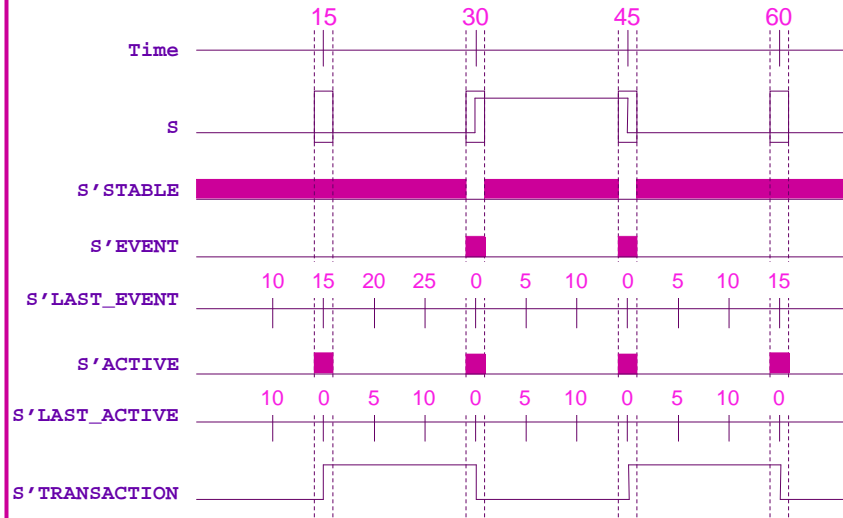
LINGUAGENS DE DESCRIÇÃO DE HARDWARE

Atributos de sinais

- *Permitem monitorar eventos, transações assim como reportar o tempo de acontecimento do eventos e transação. Existem vários.*
- *S' EVENT: retorna TRUE cada vez que ocorre uma mudança de estado no sinal S e o estado de S antes do evento é diferente do estado deste depois do evento; S' STABLE tem significado oposto.*
- *S' ACTIVE: retorna TRUE cada vez que ocorre uma transação no sinal S. O estado de S antes e depois da transação pode ser o mesmo;*
- *S' TRANSACTION: retorna um sinal que muda de estado cada vez que ocorre uma transação no sinal S;*
- *S' LAST_EVENT: retorna o tempo percorrido desde a última mudança de estado do sinal S;*
- *S' LAST_ACTIVE: retorna o tempo percorrido desde o acontecimento da última transação*

LINGUAGENS DE DESCRIÇÃO DE HARDWARE

Exemplo de uso dos atributos de sinais



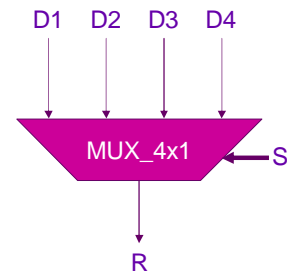
LINGUAGENS DE DESCRIÇÃO DE HARDWARE

Seleção de dados

```

ENTITY Mux_4x1 IS
    PORT(D1,D2,D3,D4: IN  BIT;
          S:           IN  BIT_VECTOR(1 DOWNTO 0);
          R:           OUT BIT);
END ENTITY;

ARCHITECTURE Fluxo_Dados OF Mux_4x1 IS
BEGIN
    WITH S SELECT
        R <= D1 AFTER 3 NS WHEN "00",
              D2 AFTER 3 NS WHEN "01",
              D3 AFTER 3 NS WHEN "10",
              D4 AFTER 3 NS WHEN "11";
END Fluxo_Dados;
    
```



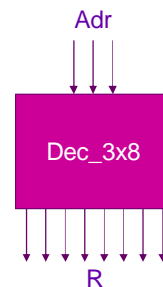
LINGUAGENS DE DESCRIÇÃO DE HARDWARE

```

ENTITY Dec_3x8 IS
    PORT(Adr: IN  STD_LOGIC_VECTOR(2 DOWNTO 0);
          R:  OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
END ENTITY;

ARCHITECTURE Fluxo_Dados OF Dec_3x8 IS
BEGIN
    WITH Adr SELECT
        R <= "00000001" AFTER 2 NS WHEN "000",
              "00000010" AFTER 2 NS WHEN "001",
              "00000100" AFTER 2 NS WHEN "010",
              "00001000" AFTER 2 NS WHEN "011",
              "00010000" AFTER 2 NS WHEN "100",
              "00100000" AFTER 2 NS WHEN "101",
              "01000000" AFTER 2 NS WHEN "110",
              "10000000" AFTER 2 NS WHEN "111",
              "XXXXXXXX" WHEN OTHERS
END Fluxo_Dados;

```



LINGUAGENS DE DESCRIÇÃO DE HARDWARE

🔒 Atribuições protegidas de sinais

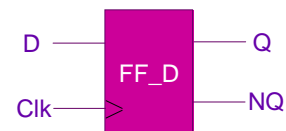
```

ENTITY FF_D IS
    PORT(Clk,D: IN  BIT; Q,NQ: OUT BIT);
END ENTITY;

ARCHITECTURE Fluxo_Dados1 OF FF_D IS
BEGIN
    Q <= D AFTER 2 NS WHEN (Clk = '1' AND Clk'Event)
        ELSE Q;
    NQ <= NOT D AFTER 3 NS WHEN (Clk = '1' AND Clk'Event)
        ELSE Q;
END Fluxo_Dados1;

ARCHITECTURE Fluxo_Dados2 OF FF_D IS
BEGIN
    FF: BLOCK (Clk = '1' AND Clk'Event)
    BEGIN
        Q <= GUARDED D      AFTER 2 NS;
        NQ <= GUARDED NOT D AFTER 3 NS;
    END BLOCK FF;
END Fluxo_Dados2;

```



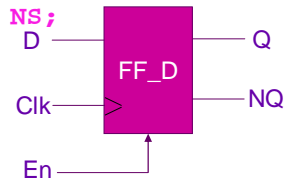
LINGUAGENS DE DESCRIÇÃO DE HARDWARE

```

ENTITY FF_D_Enable IS
    PORT(Clk,E,D: IN BIT; Q,NQ: OUT BIT);
END ENTITY;

ARCHITECTURE Fluxo_Dados OF FF_D_Enable IS
BEGIN
    TR: BLOCK (Clk = '1' AND Clk'Event)
    BEGIN
        EN: BLOCK (E = '1' AND GUARD)
        BEGIN
            Q <= GUARDED D AFTER 2 NS;
            NQ <= GUARDED NOT D AFTER 3 NS;
        END BLOCK EM;
    END BLOCK TR;
END Fluxo_Dados;

```



LINGUAGENS DE DESCRIÇÃO DE HARDWARE

🚧 Sinais resolvidos

- Em VHDL, sinais não pode ter mais de um “driver”, isto é atribuir concorrentemente ao mesmo sinal valores diferentes;
- Quando uma sinal tem mais de um driver na especificação de um componente, é preciso providenciar uma função de resolução que é usada caso haja conflito durante a simulação.

```

ENTITY Circuito IS
    PORT(A, B, C: IN BIT; Z: OUT BIT);
END Circuito;

ARCHITECTURE Com_Fumaca OF Circuito IS
    SIGNAL T: BIT;
BEGIN
    T <= A; T <= B, T <= C; -- T com 3 drivers
    Z <= T;
END Com_Fumaca;

```

LINGUAGENS DE DESCRIÇÃO DE HARDWARE

```
ARCHITECTURE Sem_Fumaca OF Circuito IS

  TYPE vetor_bit IS ARRAY(NATURAL <>) OF BIT;
  FUNCTION Anding(D: vetor_bit) RETURN BIT
  VARIABLE Acc: BIT := '1';
  BEGIN
    FOR I IN D'RANGE LOOP
      Acc := Acc AND D(I);
    END LOOP;
    RETURN Acc;
  END Anding;
  SIGNAL T: Anding BIT;

BEGIN
  T <= A; T <= B;  T <= C;
  Z <= T;
END Sem_Fumaca;
```

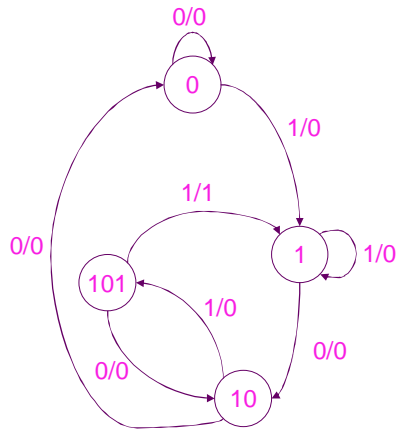
LINGUAGENS DE DESCRIÇÃO DE HARDWARE

Sinais do tipo REGISTER e BUS

- Em VHDL, sinais não podem ficar sem nenhum driver.
- Quando as atribuições de um sinal S são todas protegidas, pode acontecer que todas as condições de proteção sejam falsas e por tanto o pode acontecer S fique sem driver.
- Este tipo de sinal precisar ser declarado como BUS ou REGISTER para resolver a situação quando este fica sem driver.
- Quando sinais do tipo REGISTER ficam sem driver, estes guardam o seu último valor.
- Quando sinais do tipo BUS ficam sem driver, estes são colocados em alta impedância.

LINGUAGENS DE DESCRIÇÃO DE HARDWARE

Descrição de máquinas de estados



Máquina de estados Mealy que detecta a sequência 1011 numa entrada serial de bits

LINGUAGENS DE DESCRIÇÃO DE HARDWARE

Primeira implementação baseada em fluxo de dados

```
ENTITY Detector IS
  PORT (X, Clk: IN BIT; Z: OUT BIT);
END Detector;

ARCHITECTURE Fluxo_Dados1 OF Detector IS

  TYPE estado IS (reset, tem1, tem10, tem101);
  TYPE vetor_estados IS ARRAY (NATURAL <>) OF estado;

  FUNCTION Um_de(D: vetor_estados) RETURN estado IS
  BEGIN
    RETURN D(D'LEFT);
  END Um_de;

  SIGNAL atual: Um_de estado REGISTER := reset;
```

LINGUAGENS DE DESCRIÇÃO DE HARDWARE

```
BEGIN
  Sincro: BLOCK(clk = '1' AND NOT Clk'STABLE)
  BEGIN
    TN: BLOCK(atual = reset AND GUARD)
    BEGIN
      atual <= GUARDED S1 WHEN X = '1' ELSE reset;
    END BLOCK TN;
    T1: BLOCK(atual = tem1 AND GUARD)
    BEGIN
      atual <= GUARDED tem10 WHEN X = '0' ELSE tem1;
    END BLOCK T1;
    T10: BLOCK(atual = tem10 AND GUARD)
    BEGIN
      atual <= GUARDED tem101 WHEN X = '1' ELSE reset;
    END BLOCK T10;
    T101: BLOCK(atual = tem101 AND GUARD)
    BEGIN
      atual <= GUARDED tem1 WHEN X = '1' ELSE tem10;
      Z <= '1' WHEN (atual = tem101 AND X = '1')
        ELSE '0';
    END BLOCK T101;
  END BLOCK Sincro;
END Fluxo_Dados1;
```

LINGUAGENS DE DESCRIÇÃO DE HARDWARE

 *Segunda implementação baseada em fluxo de dados (1 hot bit)*

```
ENTITY Detector IS
  PORT (X, Clk: IN BIT; Z: OUT BIT);
END Detector;
ARCHITECTURE Fluxo_Dados2 OF Detector IS
  FUNCTION Oring(D: BIT_VECTOR) RETURN BIT IS
    VARIABLE acc: BIT := '0';
  BEGIN
    FOR i IN D'RANGE LOOP
      acc := acc OR D(i);
    END LOOP
    RETURN acc;
  END Oring;
  SUBTYPE orbit IS Oring BIT;
  TYPE vetor_orbit IS ARRAY (NATURAL <>) OF orbit;
  SIGNAL atual: vetor_orbit REGISTER := "1000";
```

LINGUAGENS DE DESCRIÇÃO DE HARDWARE

```
BEGIN
  Sincro: BLOCK(clk = '1' AND NOT Clk'STABLE)
  BEGIN
    TN: BLOCK(atual(1) = '1' AND GUARD)
    BEGIN  atual(1) <= GUARDED '1' WHEN X = '0' ELSE '0';
           atual(2) <= GUARDED '1' WHEN X = '1' ELSE '0';
    END BLOCK TN;
    T1: BLOCK(atual(2) = '1' AND GUARD)
    BEGIN  atual(3) <= GUARDED '1' WHEN X = '0' ELSE '0';
           atual(2) <= GUARDED '1' WHEN X = '1' ELSE '0';
    END BLOCK T1;
    T10: BLOCK(atual(3) = '1' AND GUARD)
    BEGIN  atual(1) <= GUARDED '1' WHEN X = '0' ELSE '0';
           atual(4) <= GUARDED '1' WHEN X = '1' ELSE '0';
    END BLOCK T10;
    T101: BLOCK(atual(4) = '1' AND GUARD)
    BEGIN  atual(3) <= GUARDED '1' WHEN X = '0' ELSE '0';
           atual(2) <= GUARDED '1' WHEN X = '1' ELSE '0';
           Z <= '1' WHEN (atual(4) = '1' AND X = '1') ELSE '0';
    END BLOCK T101;
    atual <= GUARDED "0000";
  END BLOCK Sincro;
END Fluxo_Dados2;
```

LINGUAGENS DE DESCRIÇÃO DE HARDWARE

Terceira implementação baseada no comportamento

```
ENTITY Detector IS
  PORT (X, Clk: IN BIT; Z: OUT BIT);
END Detector;

ARCHITECTURE Comportamental OF Detector IS
  TYPE estado IS (reset, tem1, tem10, tem101,
                 tem1011);
  SIGNAL atual: estado := reset;
BEGIN
  Saida: Process(atual)
  BEGIN
    IF atual = tem1011 THEN Z <= '1';
    ELSE Z <= '0';
    END IF;
  END PROCESS;
END PROCESS;
```

LINGUAGENS DE DESCRIÇÃO DE HARDWARE

```

Transicao: PROCESS(Clk)
BEGIN
  IF Clk = '1' THEN
    CASE atual IS
      WHEN reset => IF X = '1' THEN atual <= tem1;
                     ELSE atual <= reset;
                     END IF;

      WHEN tem1 => IF X = '0' THEN atual <= tem10;
                     ELSE atual <= tem1;
                     END IF;

      WHEN tem10 => IF X = '1' THEN atual <= tem101;
                     ELSE atual <= reset;
                     END IF;

      WHEN tem101 => IF X = '1' THEN atual <= tem1011;
                     ELSE atual <= tem10;
                     END IF;

      WHEN tem1011 => IF X = '1' THEN atual <= tem1;
                      ELSE atual <= tem10;
                      END IF;

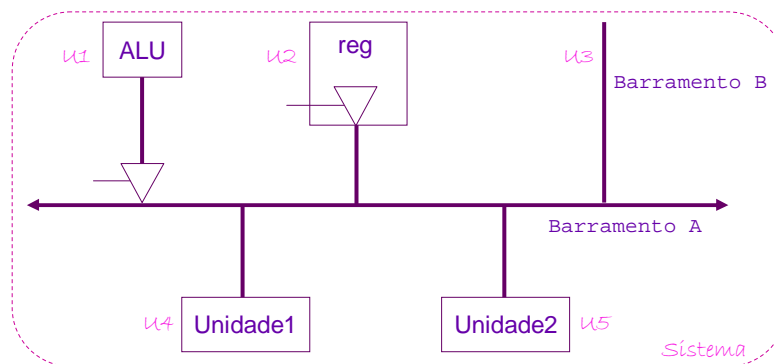
    END CASE;
  END IF;
END PROCESS;
END Comportamental;

```

LINGUAGENS DE DESCRIÇÃO DE HARDWARE

Modelagem de Barramentos

Barramentos de três estados



```

TYPE qit IS ('0', '1', 'X', 'Z');
SUBTYPE wired_qit IS wiring qit;

TYPE wired_qit_vector IS ARRAY(NATURAL <>) OF wired_qit;

```


LINGUAGENS DE DESCRIÇÃO DE HARDWARE

```
ARCHITECTURE Parcial OF Sistema IS
...
COMPONENT Alu
    PORT(...; Zout: OUT qit_vector(7 DOWNT0 0));
END COMPONENT;
COMPONENT Reg IS
    PORT(...; Zout: OUT wired_qit_vector(7 DOWNT0 0));
END COMPONENT;
COMPONENT Unidadel IS
    PORT(Zin: OUT qit_vector(7 DOWNT0 0); ...);
END COMPONENT;
COMPONENT Unidade2 IS
    PORT(Zin: OUT wired_qit_vector(7 DOWNT0 0); ...);
END COMPONENT;
....
```

LINGUAGENS DE DESCRIÇÃO DE HARDWARE

```
SIGNAL Barramentoa: wired_qit_vector(7 DOWNT0 0);
SIGNAL Barramentob: wired_qit_vector(7 DOWNT0 0);
SIGNAL Aluout, unidlin: qit_vector(7 DOWNT0 0);
BEGIN
    ...
    U1: Alu PORT MAP(..., Aluout);
    Barramentoa <= wired_qit_vector) Aluout;
    ...
    U2: Reg PORT MAP(..., Barramentoa);
    ...
    U3: Barramentoa <= Barramentob;
    ...
    unidlin <= qit_vector(Barramentoa);
    U4: Unidadel PORT MAP(unidlin, ...);
    ...
    U5: Unidade2 PORT MAP(Barramentoa, ...);
    ...
END Parcial;
```