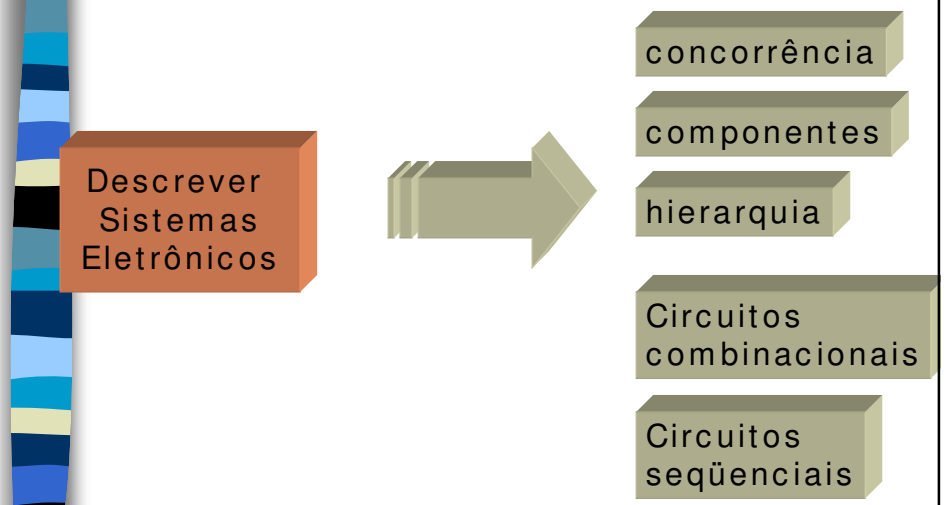
VHDL

Processos
(Combinacional e Seqüêncial)

Objetivo das HDL's



Processos - Lógica Combinacional

- Declarações em processos incluem um conjunto de declarações seqüenciais que atribuem valores à sinais. Estas declarações permitem a execução passo-a-passo da computação. Declarações em processos que descrevem apenas comportamento combinacional podem ser usados para criar lógica combinacional.
- Para assegurar que um processo é combinacional, sua "sensitive list" deve conter todos os sinais que são lidos no processo.
- "Uma sensitive list contém todos os sinais que causam as declarações do processo serem executados se seus valores mudam".

VHDL: Comportamental (processo)

- **Algoritmo (Processo)**
  - seqüencial
- **Lista de sensibilidade**
  - sinais de entrada
- **Região declarativa**
  - Região entre o fim da lista de sensibilidade e a palavra chave begin.
  - Usada para declarar variáveis ou constantes dentro do processo.
- **Campo de atribuições**
  - Campo entre a chave begin e end ALG;


```
architecture ALG of COMPARE is
begin
  process (A,B)
  begin
    if (A = B) then
      C <= '1' after 1ns;
    else
      C <= '0' after 2ns;
    end if;
  end process;
end ALG;
```

Processos - Lógica combinacional (exemplo)

```
ENTITY proc IS       -- Registrador de 1 bit
    PORT
    (
    a,b  : IN BIT;
    c : OUT  BIT
    );
END proc;
ARCHITECTURE maxpld  OF proc IS
BEGIN
    PROCESS(a, b) - - Processo
      BEGIN
         q <= a AND b;
      END PROCESS;
END maxpld
```

Sinais e variáveis

- Existem dois tipos principais dos objetos usados para armazenar dados:
  - **Sinal**
    - **Usado na modelo estrutural e no fluxo de dados.**
    - **Os sinais são usados para conectar sub-módulos em um projeto e, se dentro de um processo, seu valor só muda na saída do processo.**
  - **Variável**
    - **pode somente ser usado dentro de processos. Uma variável comporta-se como você esperaria em uma língua de programação do software.**
    - **Variáveis são modificadas com a atribuição variável.**
    - **Exemplo**

      a := b;

      O valor de b é copiado automaticamente para a dentro do processo.

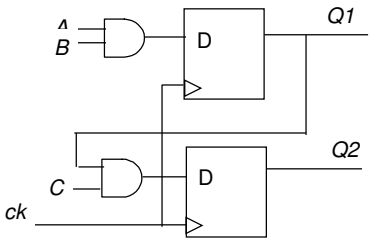
Variáveis vs. Sinais

- Variáveis:
  - target := expression;
  - declaradas no processo
  - valores temporários
  - são locais ao processo
  - valor alterado imediatamente

- Sinais:
  - target <= [expression] [after delay];
  - declarados.
    - entities
    - architectures
    - packages
  - passagem de valores entre processos
  - passagem de valores entre entidades
  - valores alterados ao final da execução do processo (após execução de um wait)

Variável vs. Signal?

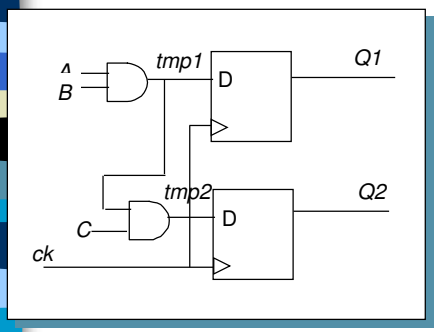
Circuito 1




```
Architecture v1 of circuito1 is
    signal Q1,Q2: bit;
begin
    process (ck)
    begin
        if ck=1 and ck´event then
            Q1 <= A and B;
            Q2 <= Q1 and C;
        end if;
    end process;
end v1;
```

Variável vs. Signal?

Circuito 2




```
Architecture v1 of circuito2 is
    signal Q1,Q2: bit;
begin
    process (ck)
    variable tmp1, tmp2: bit;
    begin
        if ck=1 and ck´event then
            tmp1 := A and B;
            tmp2 := tmp1 and C;
        end if;
        Q1 <= tmp1;
        Q2 <= tmp2;
    end process;
end v1;
```

Exemplo – uso de variáveis

- Implementar um circuito que conta o número de bits no sinal de entrada *d*.
  - Código em VHDL

```
Entity proc is
port ( d : in bit_vector (2 downt 0);
        q : out integer range 0 to 3
            );
    End proc;
```

- Variável num_bits tem seu valor modificado dentro do processo
- O sinal q tem seu o valor atualizado dentro do processo por causa da variável num_bits

```
Architecture maxpldl of proc is
begin
  process(d) – conta o número de bits
    variable num_bits : integer;
  begin
    num_bits := 0;
    for i in d´range loop
      if d(i) = '1' then
        num_bits := num_bits + 1;
      end if;
    end loop;
   q <= num_bits;
  end process;
 end maxpld;
```

Loops

- **VHDL tem uma indicação básica do laço similar àquela usual quando em outras línguas de programação.**
  - **Estrutura do Loop usando for:**

```
for item in 1 to last_item loop
    table(item) := 0;
end loop;
```

Implementação de arquiteturas através de Processos - Lógica seqüencial

- Processos em lógica seqüencial é caracterizado quando saídas de um determinado circuitos em um determinado tempo são função das entradas naquele tempo e de todos os tempos precedentes.
- Todos os circuitos seqüenciais devem incluir um ou mais registradores.
- Exemplos:
  - **Máquinas de estados são especialmente úteis para implementação de lógica seqüencial.**
  - **Contadores, Registradores de deslocamento e controladores**.
- "Lembrar que embora dentro de um processo possamos ter lógica seqüencial, processos em uma arquitetura atuam concorrentemente".

Declarações seqüenciais

- **If Statement**
  - Permite a execução de uma das indicações dependendo do resultado do teste
    - **if_statement then**
      seqüência de declarações
      { elsif condição then
      seqüência de declarações}
      [else
      seqüência de declarações]
      end if ;

- **As condições são expressões resultantes dos valores Booleanos. As condições são avaliadas sucessivamente até que se encontre um valor verdadeiro. Nesse caso a lista correspondente à indicação é executada. Se não e, se a condição else está presente, sua lista de declarações é executada.**

Declarações seqüenciais

- **Case**
  - **A indicação do caso permite que a seleção das indicações execute dependendo do valor de uma expressão da seleção.**
  - **A sintaxe é:**

```
case element of
    when a =>
        statements for a;
    when   b =>
        statements b;
    when c =>
        statements c;
end case;
```

- **A expressão da seleção deve resultar em um tipo discreto, ou em um array one-dimensional de caracteres. A alternativa cuja lista contém o valor da expressão é selecionada e a lista da indicação executada.**

Declarações seqüenciais

- **Case**

```
case opcode of
    when "00" => perform_add;
    when "01" => perform_subtract;
    when others => signal_illegal_opcode;
end case;
```

Exemplo Contador

- **Contador de 8 bits controlado pelos sinais clk, clear, ld, d, enable e up_down;**
- **Entidade**

```
Entity counter is
  port (
    d       :  in integer range 0 to 255;
    clk     :  in bit;
    clear   : in bit;
    ld      : in bit;
    q       : out integer range 0 to 255
      );
end counter;
```

- **Arquitetura do contador**

```
Architecture a of counter is
begin
-- clear, load síncronos
Process(clk)
    variable cnt : integer range 0 to 255;
begin
    if(clk'event and clk = '1') then
      if clear = '0'then
          cnt := '0';
       else
          if ld = '0' then
            cnt := d;
          else
             cnt := cnt+1;
          end if;
       end if;
      end if;
    q <= cnt;
end process;
end a;
```

FSM em VHDL

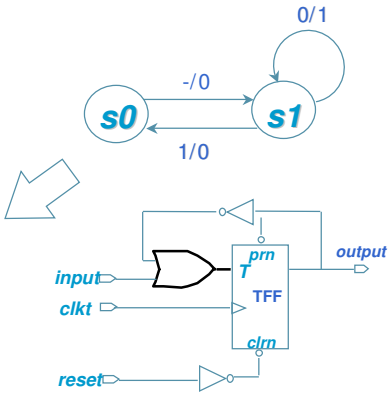
- **Algoritmo (FSM)**
  - Seqüencial
- **Definição de estados**
- **Lista de sensibilidade**
  - sinais de entrada
- **Região declarativa**
  - Região entre o fim da lista de sensibilidade e a palavra chave begin.
  - Usada para declarar variáveis ou constantes dentro do processo.
- **Campo de atribuições**
  - Campo entre a chave begin e end ALG;

```
architecture arc_fsm of fsm is
  type estados is (zero, um, ....);
    signal estado_atual : estados;
begin
  process (clk)
    begin
      if (reset = '1') then
        estado_atual <= zero;
      elsif (clk = '1' and clk'event) then
        case estado_atual is
        .............
          end case;
      end if;
  end process;
end arc_bombons;
```

Estados

Processos - Lógica seqüêncial (FSM)

```
EN TITY state_machine IS
PORT(    clk, input, reset       : IN BIT;
         output       : OUT BIT );
END state machine;
ARCHITECTURE  maxpld OF  state_machine IS
         TYPE STATE_TYPE IS (s0, s1);
         SIGNAL state : STATE_TYPE;
BEGIN
         PROCESS (clk)
         BEGIN
         IF RESET = '1' THEN
                  state <= s0;
         ELSIF (clk'EVENT AND  clk = '1' ) THEN
                  CASE state IS
                     WHEN so =>
                              state <= s1;
                     WHEN s1 =>
                              IF input = '1' THEN
                                state <= s0;
                              ELSE
                                state <= s1;
                              END IF;
                  END CASE;
         END PROCESS;
         output <= '1' WHEN state = s1 ELSE '0';
END maxpld;
```



Máquina de vender bombom
Versão 02

- A máquina libera um pacote de bomboms se o cliente depositar pelo menos 15 centavos. A máquina não dá trôco.
- Moedas aceitas na máquina
  - 5 centavos (C)
  - 10 centavos (D)

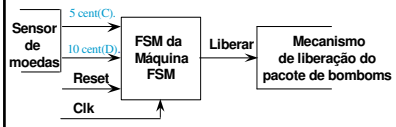
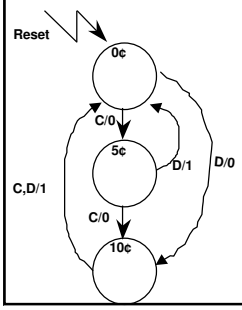


Diagrama de estados



Código VHDL

```
entity bombom01 is
        port        ( reset, clk, c, d : in bit;
                      libera_bombom : out bit );
end bombom01;
architecture arc_bombons of bombom01 is
        type estados is (tem_zero, tem_cinco, tem_dez);
        signal estado_atual : estados;
begin
    process (clk)
        begin
            if (reset = '1') then
               estado_atual <= tem_zero;
               libera_bombom <= '0';
            elsif (clk = '1' and clk'event) then
              case estado_atual is
                  when tem_zero =>
                   libera_bombom <= '0';
                   if (c = '1' ) then
                     estado_atual <= tem_cinco;
                   elsif (d = '1') then
                     estado_atual <= tem_dez;
                   end if;
                  when tem_cinco =>
                   if (c = '1') then
                     estado_atual <= tem_dez;
                   elsif (d = '1') then
                     estado_atual <= tem_zero;
                     libera_bombom <= '1';
                   end if;
                   when tem_dez =>
                   if (c = '1' or d ='1') then
                     estado_atual <= tem_zero;
                     libera_bombom <= '1';
                   end if;
              end case;
            end if;
        end process;
end arc_bombons;
```