Controle de Dispositivos e Linguagem de Descrição de Hardware - VHDL

Prof. Marco Aurélio Benedetti Rodrigues

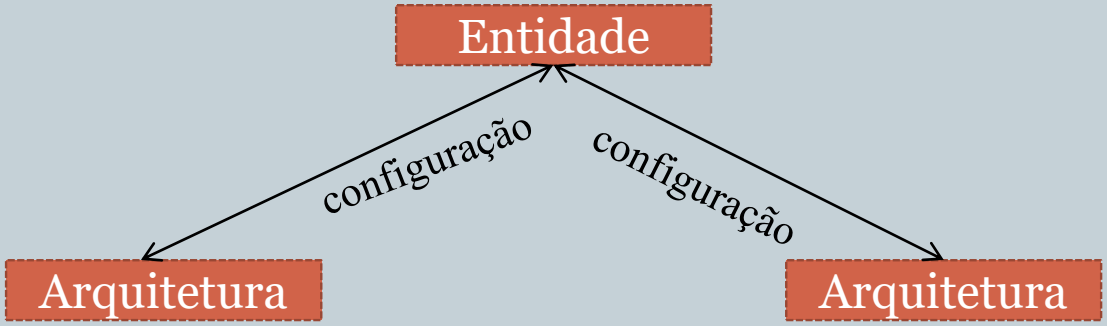
Estagiário em Docência: Naelso Alves Cunha

A linguagem VHDL

- VHDL é uma linguagem utilizada para descrever o comportamento de um projeto de Circuito Digital:
  - V = *Very High Speed Integrated Circuit* (VHSIC)
  - H = *Hardware*
  - D = Descrição
  - L = Linguagem
- O uso da Hierarquia dos projetos de VHDL permite uma criação rápida de projetos complexos de circuitos digitais.

Entidades, Arquiteturas e Configurações

- A estrutura do projeto em VHDL assemelha-se a estrutura de um projeto de software orientado a objeto;
- Todos os projetos em VHDL fornecem uma interface externa e uma implementação interna;
- Um projeto em VHDL consiste de Entidades, arquiteturas e configurações.



Entidades, Arquiteturas e Configurações

- ***Entity***

- A *entity* é a parte principal de qualquer projeto, pois descreve a interface do sistema;
- Tudo que é descrito na *entity* fica automaticamente visível a outras unidades associadas com a *entity*;
- O nome do sistema é o próprio nome da *entity*;
- Deve-se sempre iniciar um projeto em VHDL pela *entity*.

Entidades, Arquiteturas e Configurações

- A entidade é uma especificação de projeto de uma interface externa;
- A declaração de uma entidade contém:
  - O nome da entidade;
  - Um conjunto genérico de declarações especificando instância e parâmetros;
  - Um conjunto de portas de entradas e saídas do projeto de hardware.

```
ENTITY entity_name IS
GENERIC(
generic_1_name : generic_1_type;
generic_2_name : generic_2_type;
generic_n_name : generic_n_type
);
```

```
ENTITY entity_name IS
PORT(
port_1_name : port_1_dir port_1_type;
port_2_name : port_2_dir port_2_type;
port_n_name : port_n_dir port_n_type
);
END entity_name;
```

Entidades, Arquiteturas e Configurações

- Exemplo de declaração AND:

```
ENTITY and IS
PORT ( a : IN std_logic;
       b : IN std_logic;
       c : OUT std_logic );
END and;
```

A declaração de PORT pode ser:

IN : porta de entrada

OUT : porta de saída

INOUT : porta Bidirecional

BUFFER : porta de saída Bufferizada

Entidades, Arquiteturas e Configurações

- Arquiteturas
  - A arquitetura é uma especificação da implementação interna de um projeto;
  - Múltiplas ARQUITETURAS podem ser criadas em uma entidade particular, melhorando:
    - A performance do circuito;
    - A área do chip;
    - O consumo de energia;
    - A facilidade de Simulação.

```
ARCHITECTURE architecture_name OF entity_name IS
BEGIN
-- Definição do código da Arquitetura.
END architecture_name;
```

Eletrônica Digital – VHDL

- ***Architecture***

  A *entity* de um sistema é tão importante que a *architecture* é especificada na forma de *architecture of entity*.

  Um sistema pode ser descrito em termos de funcionalidade, isto é, o que o sistema faz, ou em termos de estrutura, isto é, como o sistema é composto.

  A *Architecture* define as funcionalidades do sistema.

Entidades, Arquiteturas e Configurações

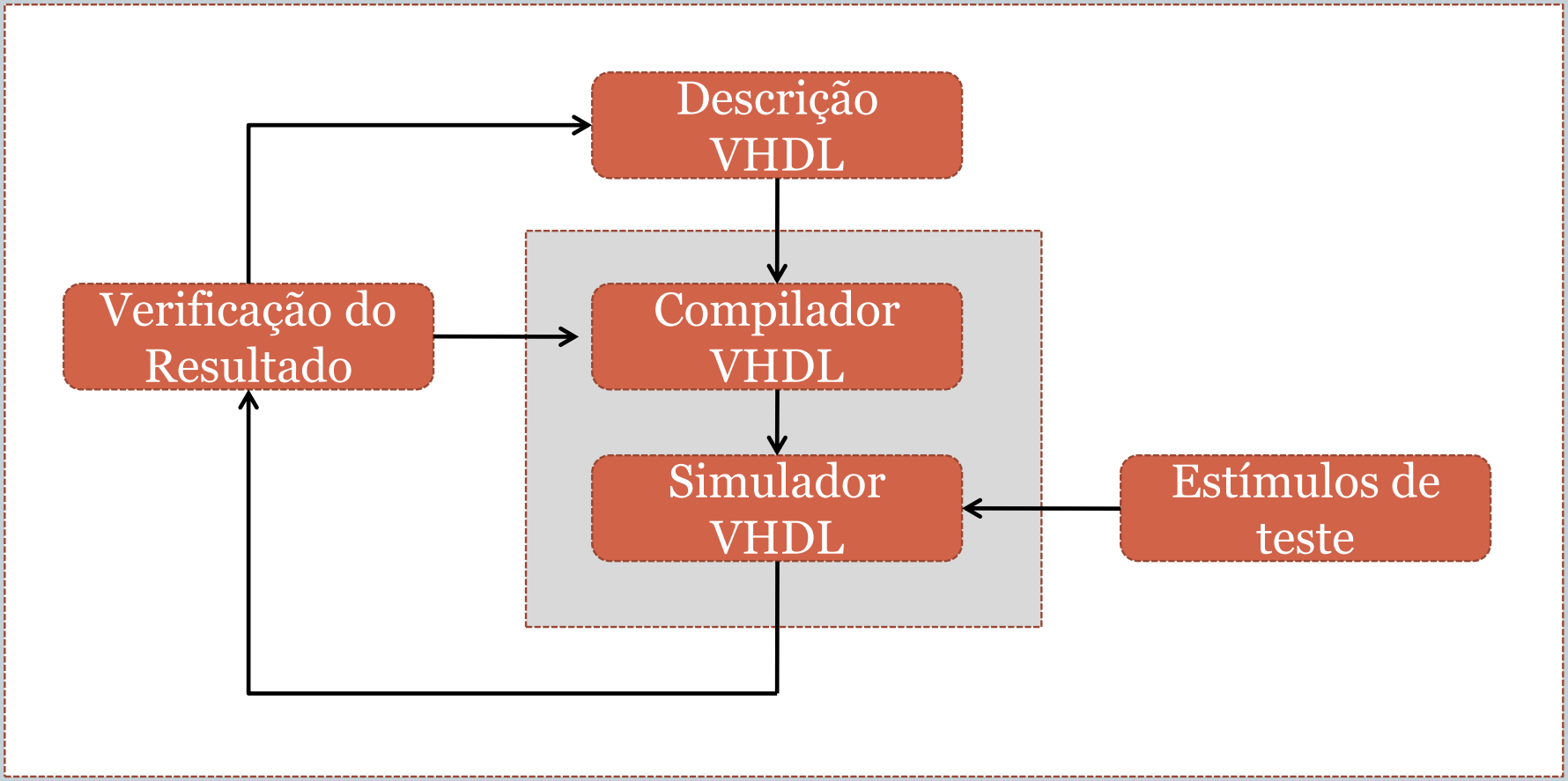
- Exemplo de Arquitetura

```
ARCHITECTURE synthesis1 OF and IS
BEGIN
  c <= a AND b;
END synthesis1;
```

- Uma **configuração** é uma especificação do mapeamento entre uma arquitetura e uma instância particular de uma entidade.
- Por *default* existe uma configuração para cada entidade.

Eletrônica Digital - VHDL

- Descrição de um código em VHDL



Bibliotecas e Pacotes

- As bibliotecas (LIBRARY) fornecem um conjunto de pacotes, componentes e funções que simplificam a tarefa do projeto de Hardware.
- Os pacotes fornecem uma coleção de tipos de dados e subprogramas.
- Exemplo:

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
```

Bibliotecas e Pacotes

- O padrão IEEE 1164-1993 define uma pacote que fornece um conjunto de tipos de dados que podem ser utilizados na síntese lógica.
  - Os pinos externos de um projeto sintetizável devem usar os tipos especificados no pacote *std_logic_1164*
  - IEEE recomenda o uso dos seguintes tipos de dados para representar sinais em um sistema sintetizável:

```
std_logic
std_logic_vector(<max> DOWNTO <min>)
```

Sinais

- Sinais representam os fios e os elementos de armazenamento;
- Sinais somente são definidos dentro de arquiteturas e associados com um tipo de dado;
- Representações binárias são suficientes para linguagem de programação; porém, fios não podem ser modelados corretamente usando uma representação binária;
- Valores adicionais são necessários para representar o estado dos fios;
- A representação MVL (*Multi-Value Logic*) fornece valores de representação de sinais mais complexos.

Sinais

- MVL-4
  - Representa 4 valores
    - Logico - 1
    - Lógico - 0
    - Desconhecido - X
    - Alta impedância - Z
- MVL-9
  - Representa 9 valores
    - Não incializado -`U`
    - *Don't care* - `-`
    - Lógico - 1
    - Lógico - 0
    - Desconhecido - `X`
    - fraco - 1
    - fraco - 0
    - desconhecido fraco - `W`
    - Alta impedância - `Z`

Tipo de Dados

- VHDL suporta um conjunto de tipos definidos como:

Data Type	Characteristics
BIT	Binary, Unresolved
BIT_VECTOR	Binary, Unresolved, Array
INTEGER	Binary, Unresolved, Array
REAL	Floating Point

- VHDL suporta os seguintes operadores lógicos:

AND	NAND	NOT
OR	NOR	
XOR	XNOR	

Tipo de operadores

- VHDL suporta os seguintes operadores relacionais:

=	Igual
/=	Diferente
>	Maior que
<	Menor que
>=	Maior ou igual que
<=	Menor ou igual que

Tipo de operadores

- VHDL suporta os seguintes operadores matemáticos:

Operador	Operação	Exemplo
+	Adição	I := i + 2;
-	Subtração	J <= j -10;
*	Multiplicação	M := fator * n;
/	Divisão	K := i / 2;
**	Potenciação	I := i ** 3;
ABS	Valor absoluto	Y <= ABS(tmp)
MOD	Módulo	Z <= MOD(t);
REM	resto	R <= REM(tot);

Exemplo

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

entity digi_clk is
port (
  clk1    : in std_logic;
  seconds : out std_logic_vector(5 downto 0);
  minutes : out std_logic_vector(5 downto 0);
  hours   : out std_logic_vector(4 downto 0)
);
end digi_clk;
```

Exemplo

```
architecture Behavioral of digi_clk is
  signal sec,min : integer range 0 to 60 :=0;
  signal hour : integer range 0 to 24 :=0;
  signal count : integer :=1;
  signal clk : std_logic :='0';
begin
  seconds <= conv_std_logic_vector(sec,6);
  minutes <= conv_std_logic_vector(min,6);
  hours   <= conv_std_logic_vector(hour,5);
```

Exemplo

```
-- clk generation. For 100 MHz clock this generates 1 Hz clock.
   process(clk1)
   begin
      if(clk1'event and clk1='1') then
         count <=count+1;
         if(count = 50000000) then
            clk <= not clk;
            count <=1;
         end if;
      end if;
   end process;
```

Exemplo

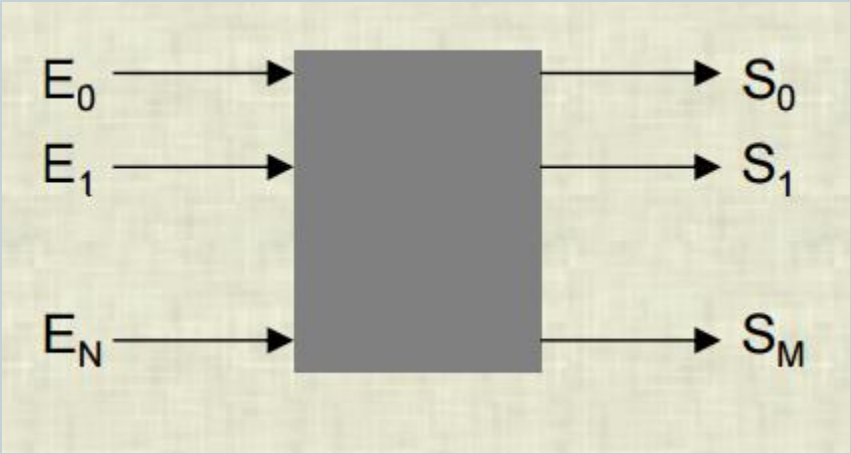
```
 process(clk)   --period of clk is 1 second.
  begin
    if(clk'event and clk='1') then
    sec <= sec+ 1;
      if(sec = 59) then
        sec<=0;
        min <= min + 1;
        if(min = 59) then
          hour <= hour + 1;
          min <= 0;
          if(hour = 23) then
            hour <= 0;
          end if;
        end if;
      end if;
    end if;
  end process;
end Behavioral;
```

Codificador

- Codificador é um circuito que mapeia um conjunto de entradas em um conjunto de saídas, segundo uma função de codificação;
- Em outras palavras, é um circuito que transforma uma informação de um formato para outro;
- Normalmente os codificadores apresentam mais portas de entrada do que portas de saída.

Codificador

- Um codificador é normalmente implementado de forma Combinacional.
- Implementação deste em VHDL pode ser realizada com o comando *with select*.



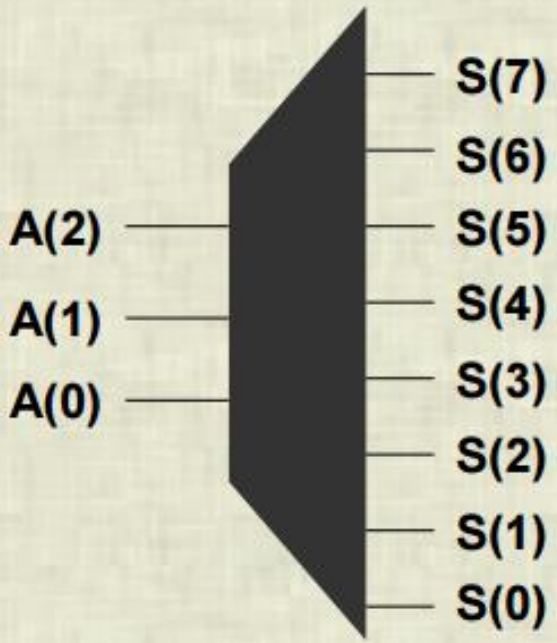
Decodificador

- Decodificador é equivalente a um codificador, um circuito que mapeia entradas em saídas segundo uma função de codificação;
- Genericamente, um circuito é chamado de decodificador, quando a função de transformação que ele realiza é inversa a de um circuito codificador;
- Normalmente os decodificadores apresentam mais portas de saída do que entrada.

Decodificador

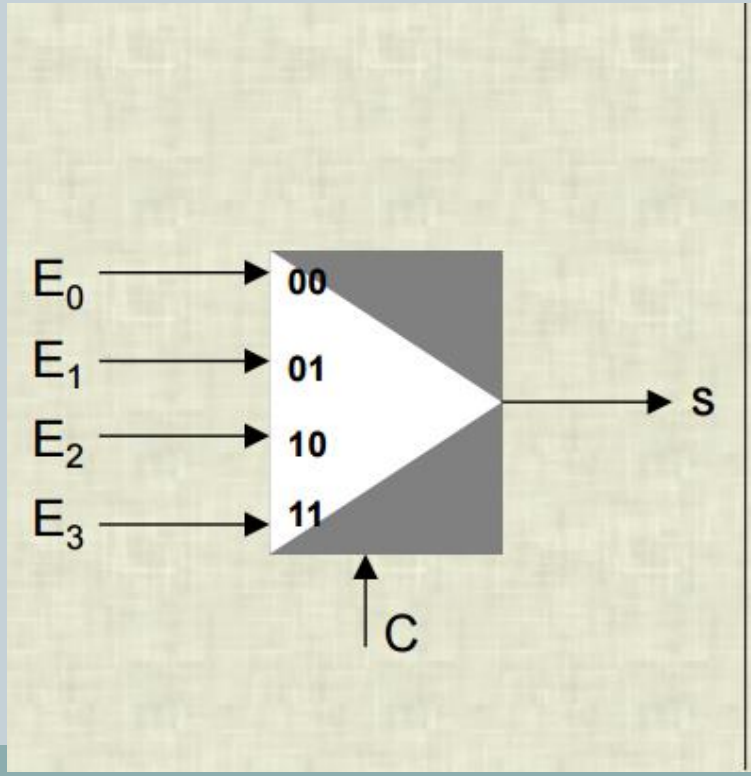
- Um caso especial de decodificador é o binário. Este relaciona n entradas em 2^n saídas; normalmente utilizado para endereçamento de memórias RAM.

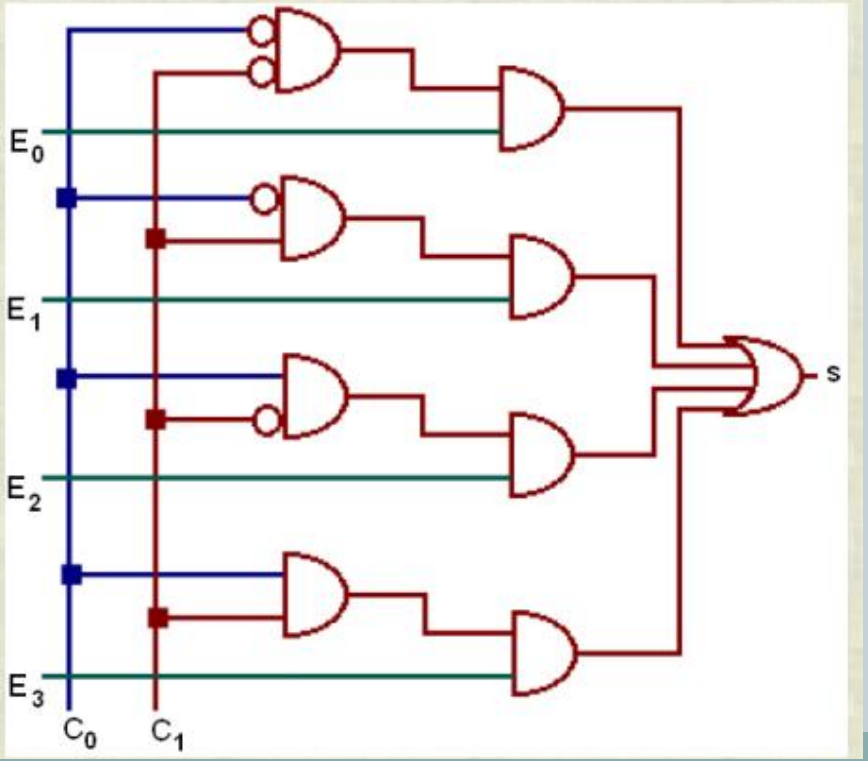
```
with A select
   S <= "00000001" when "000",
        "00000010" when "001",
        "00000100" when "010",
        "00001000" when "011",
        "00010000" when "100",
        "00100000" when "101",
        "01000000" when "110",
        "10000000" when "111";
```



Multiplexador (Mux)

- É um circuito que permite selecionar dentre varias entradas, uma delas para enviar o sinal para uma saída, conforme um sinal de seleção.





Multiplexador (Mux)

Multiplexadores podem ser implementados com diversos comandos:

```
s <= (E(0) and (not C(0) and not C(1) )) or
     (E(1) and (not C(0) and C(1) ))     or
     (E(2) and (C(0) and not C(1) ))     or
     (E(3) and (C(0) and C(1) ));
```

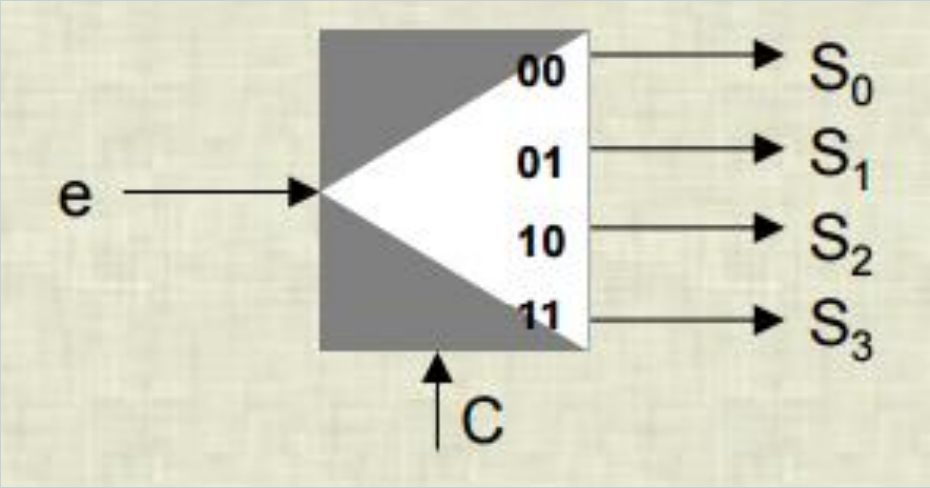
```
with C select
     s <= E(0) when "00",
          E(1) when "01",
          E(2) when "10",
          E(3) when others;
```

```
s <= E(0) when C(0) = 0 and C(1) = 0 else 'Z';
s <= E(1) when C(0) = 0 and C(1) = 1 else 'Z';
s <= E(2) when C(0) = 1 and C(1) = 0 else 'Z';
s <= E(3) when C(0) = 1 and C(1) = 1 else 'Z';
```

```
process(E, C)
begin
     case sel is
          when "00" =>
               s <= E(0);
          when "01" =>
               s <= E(1);
          when "10" =>
               s <= E(2);
          when others =>
               s <= E(3);
     end case;
end process;
```

Demultiplexador (Demux)

- É um circuito que opera de forma inversa ao multiplexador. Ou seja, recebe uma entrada e distribui esta em uma de várias saídas conforme um sinal de seleção



DESCRIÇÃO DE COMPORTAMENTO

- O comportamento, ou funcionalidade, de um sistema corresponde a uma lista de operações a serem executadas para se obter um determinado resultado;

- *Process* é o modo formal de se fazer uma lista sequencial dessas operações; esse comando se assemelha-se ao *.CLK do AHDL, isto é, é o sincronismo do sistema.

DESCRIÇÃO DE COMPORTAMENTO

- Existem algumas regras para se fazer uma descrição *process;* Inicialmente, deve-se especificar que a listagem corresponde a um *process;* Pode-se dar um nome ao process e colocá-lo no inicio da linha seguindo de dois pontos e, após, a declaração *end of process*.

- Para separar as operações sequenciais de comandos, tais como v*ariable* ou *constant*, a declaração *begin é* usada para marcar o inicio da listagem das operações sequenciais.

DESCRIÇÃO DE COMPORTAMENTO

- A listagem a seguir ilustra o uso de *process* para implementar um multiplexador 2x1.

```
MUX2x1: process (A, B, Sel)
constant High: Bit := '1';
       begin
              y <= A;
              if (Sel = '1') then Y <= B;
       end if;
end process MUX2x1;
```

Exemplo

```
SIGNAL reset, clock, d, q          :std_logic;

PROCESS (reset, clock)
-- reset and clock are in the sensitivity list to
-- indicate that they are important inputs to the process
BEGIN
    -- IF keyword is only valid in a process
    IF (reset = '0') THEN
          q <= 0;
    ELSIF (clock'EVENT AND clock = '1') THEN
          q <= d;
    END IF;
END PROCESS;
```

The EVENT attribute is true if an edge has been detected on the corresponding signal.

Exemplo

```
SIGNAL a, b, c, d          :std_logic;

PROCESS (a, b, d)
-- a, b, and d are in the sensitivity list to indicate that
-- the outputs of the process are sensitive to changes in them
BEGIN
    -- CASE keyword is only valid in a process
    CASE d IS
          WHEN '0' =>
                    c <= a AND b;
          WHEN OTHERS =>
                    c <= '1';
    END CASE;
END PROCESS;
```

Especificação de um Flip Flop D

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY dffe IS
    PORT( rst, clk, ena, d : IN     std_logic;
          q                : OUT    std_logic );
END dffe;

ARCHITECTURE synthesis1 OF dffe IS
BEGIN
    PROCESS (rst, clk)
    BEGIN
          IF (rst = '1') THEN
                q <= '0';
          ELSIF (clk'EVENT) AND (clk = '1') THEN
                IF (ena = '1') THEN
                      q <= d;
                END IF;
          END IF;
    END PROCESS;
END synthesis1;
```

Especificações Básicas do Projeto

Você entende completamente como o projeto deve operar?

Quantos Bits de dados são necessários?

Quantos acionamentos são necessários no projeto (saídas)?

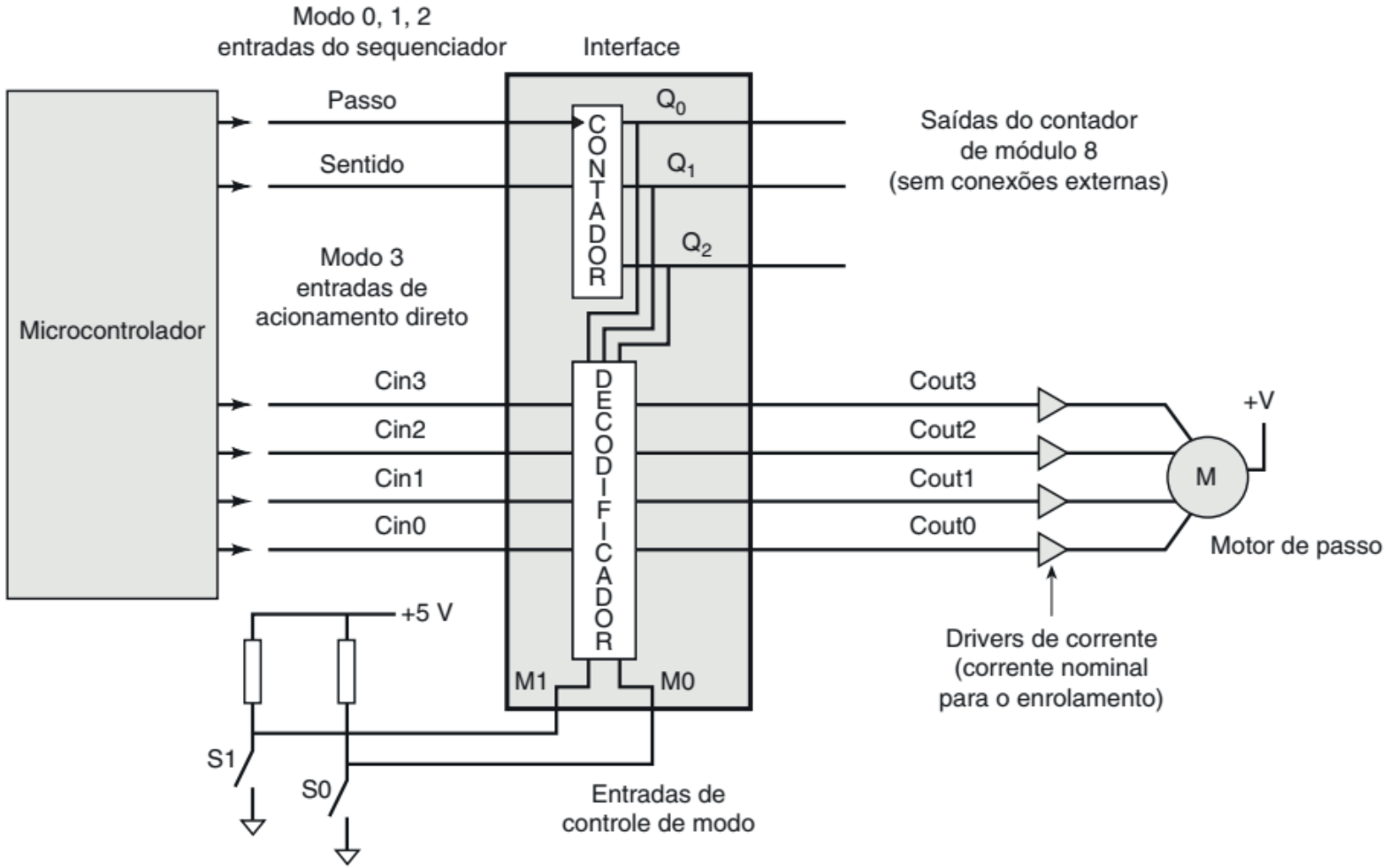
Quais os nomes das entradas e saídas?

Quais as entradas e saídas são ativas em Alto e Baixo (default)?

Quais as necessidades de Velocidade (escolha do chip)?

O que definiria o completo sucesso do seu projeto?

Exemplo - Controlador de Motor de Passo



Controlador de Motor de Passo

- Exemplo de um Controlador de Motor de Passo

Passo completo	Meio passo	Wave-Drive
Enrolamento 3210	Enrolamento 3210	Enrolamento 3210
1010	1010	
	1000	1000
1001	1001	
	0001	0001
0101	0101	
	0100	0100
0110	0110	
	0010	0010

Modo	M1	M0	Sinais de entrada	Saída
0	0	0	Passo, sentido	Sequência de contagem de passo completo
1	0	1	Passo, sentido	Sequência de contagem wave-drive
2	1	0	Passo, sentido	Sequência de contagem de meio passo
3	1	1	Quatro entradas de controle	Acionamento direto a partir de entradas de controle

Controlador de Motor de Passo

```
SUBDESIGN  fig10_2
(
  step, dir      :INPUT;
  q[2..0]        :OUTPUT;
)
VARIABLE
count[2..0]      : DFF;

BEGIN
  count[].clk = step;
  IF dir THEN count[].d = count[].q + 1;
  ELSE        count[].d = count[].q - 1;
  END IF;
  q[] = count[].q;
END;
```

```
SUBDESIGN  fig10_5
(
   step, dir       :INPUT;
   q[2..0]         :OUTPUT;
   cout[3..0]      :OUTPUT;
)
VARIABLE
   count[2..0]     : DFF;

BEGIN
   count[].clk = step;
   IF dir THEN count[].d = count[].q + 1;
   ELSE        count[].d = count[].q - 1;
   END IF;
   q[] = count[].q;
   CASE count[] IS
      WHEN  B"000"    => cout[] = B"1010";
      WHEN  B"001"    => cout[] = B"1001";
      WHEN  B"010"    => cout[] = B"0101";
      WHEN  B"011"    => cout[] = B"0110";
      WHEN  B"100"    => cout[] = B"1010";
      WHEN  B"101"    => cout[] = B"1001";
      WHEN  B"110"    => cout[] = B"0101";
      WHEN  B"111"    => cout[] = B"0110";
   END CASE;
END;
```

Controlador de Motor de Passo

```
SUBDESIGN  fig10_8
(
   step, dir               :INPUT;
   m[1..0], cin[3..0]      :INPUT;
   cout[3..0], q[2..0]     :OUTPUT;
)
VARIABLE
   count[2..0]    : DFF;
BEGIN
   count[].clk = step;
   IF dir THEN count[].d = count[].q + 1;
   ELSE        count[].d = count[].q - 1;
   END IF;
   q[] = count[].q;
   CASE m[] IS
   WHEN 0 =>
         CASE count[] IS          -- FULL STEP
         WHEN  B"000"   => cout[] = B"1010";
         WHEN  B"001"   => cout[] = B"1001";
         WHEN  B"010"   => cout[] = B"0101";
         WHEN  B"011"   => cout[] = B"0110";
         WHEN  B"100"   => cout[] = B"1010";
         WHEN  B"101"   => cout[] = B"1001";
         WHEN  B"110"   => cout[] = B"0101";
         WHEN  B"111"   => cout[] = B"0110";
         END CASE;
   WHEN 1 =>
         CASE count[] IS          -- WAVE DRIVE
         WHEN  B"000"   => cout[] = B"1000";
         WHEN  B"001"   => cout[] = B"0001";
         WHEN  B"010"   => cout[] = B"0100";
         WHEN  B"011"   => cout[] = B"0010";
         WHEN  B"100"   => cout[] = B"1000";
         WHEN  B"101"   => cout[] = B"0001";
         WHEN  B"110"   => cout[] = B"0100";
         WHEN  B"111"   => cout[] = B"0010";
         END CASE;
   WHEN 2 =>
         CASE count[] IS          -- HALF STEP
         WHEN  B"000"   => cout[] = B"1010";
         WHEN  B"001"   => cout[] = B"1000";
         WHEN  B"010"   => cout[] = B"1001";
         WHEN  B"011"   => cout[] = B"0001";
         WHEN  B"100"   => cout[] = B"0101";
         WHEN  B"101"   => cout[] = B"0100";
         WHEN  B"110"   => cout[] = B"0110";
         WHEN  B"111"   => cout[] = B"0010";
         END CASE;
   WHEN 3 =>   cout[] = cin[];   -- Direct Drive
   END CASE;
END;
```

Controlador de Motor de Passo

```
ENTITY fig10_3 IS
PORT( step, dir     :IN BIT;
      q             :OUT INTEGER RANGE 0 TO 7);
END fig10_3;

ARCHITECTURE vhdl OF fig10_3 IS
BEGIN
   PROCESS (step)
   VARIABLE count    :INTEGER RANGE 0 TO 7;
   BEGIN
      IF (step'EVENT AND step = '1') THEN
         IF dir = '1' THEN count := count + 1;
         ELSE              count := count - 1;
         END IF;
      END IF;
      q <= count;
   END PROCESS;
END vhdl;
```

```
ENTITY fig10_6 IS
PORT (   step, dir :IN BIT;
         q         :OUT INTEGER RANGE 0 TO 7;
         cout      :OUT BIT_VECTOR (3 downto 0));
END fig10_6;

ARCHITECTURE vhdl OF fig10_6 IS
BEGIN
   PROCESS (step)
   VARIABLE count   :INTEGER RANGE 0 TO 7;
   BEGIN
      IF (step'EVENT AND step = '1') THEN
         IF dir = '1' THEN count := count + 1;
         ELSE              count := count - 1;
         END IF;
         q <= count;
      END IF;
         CASE count IS
            WHEN 0   => cout <= B"1010";
            WHEN 1   => cout <= B"1001";
            WHEN 2   => cout <= B"0101";
            WHEN 3   => cout <= B"0110";
            WHEN 4   => cout <= B"1010";
            WHEN 5   => cout <= B"1001";
            WHEN 6   => cout <= B"0101";
            WHEN 7   => cout <= B"0110";
         END CASE;
       END PROCESS;
END vhdl;
```

Controlador de Motor de Passo

```
ENTITY fig10_9 IS
PORT (   step, dir     :IN BIT;
         m             :IN BIT_VECTOR (1 DOWNTO 0);
         cin           :IN BIT_VECTOR (3 DOWNTO 0);
         q             :OUT INTEGER RANGE 0 TO 7;
         cout          :OUT BIT_VECTOR (3 DOWNTO 0));
END fig10_9;

ARCHITECTURE vhdl OF fig10_9 IS
BEGIN
   PROCESS (step)
   VARIABLE count    :INTEGER RANGE 0 TO 7;
   BEGIN
      IF (step'EVENT AND step = '1') THEN
         IF dir = '1' THEN count := count + 1;
         ELSE              count := count - 1;
         END IF;
      END IF;
      q <= count;
   CASE m IS
      WHEN "00" =>               -- FULL STEP
         CASE count IS
            WHEN 0   => cout <= "1010";
            WHEN 1   => cout <= "1001";
            WHEN 2   => cout <= "0101";
            WHEN 3   => cout <= "0110";
            WHEN 4   => cout <= "1010";
            WHEN 5   => cout <= "1001";
            WHEN 6   => cout <= "0101";
            WHEN 7   => cout <= "0110";
         END CASE;
      WHEN "01" =>               -- WAVE DRIVE
         CASE count IS
            WHEN 0   => cout <= "1000";
            WHEN 1   => cout <= "0001";
            WHEN 2   => cout <= "0100";
            WHEN 3   => cout <= "0010";
            WHEN 4   => cout <= "1000";
            WHEN 5   => cout <= "0001";
            WHEN 6   => cout <= "0100";
            WHEN 7   => cout <= "0010";
         END CASE;
      WHEN "10" =>               -- HALF STEP
         CASE count IS
            WHEN 0   => cout <= "1010";
            WHEN 1   => cout <= "1000";
            WHEN 2   => cout <= "1001";
            WHEN 3   => cout <= "0001";
            WHEN 4   => cout <= "0101";
            WHEN 5   => cout <= "0100";
            WHEN 6   => cout <= "0110";
            WHEN 7   => cout <= "0010";
         END CASE;
      WHEN "11" =>   cout <= cin;--Direct Drive
   END CASE;
   END PROCESS;
END vhdl;;
```

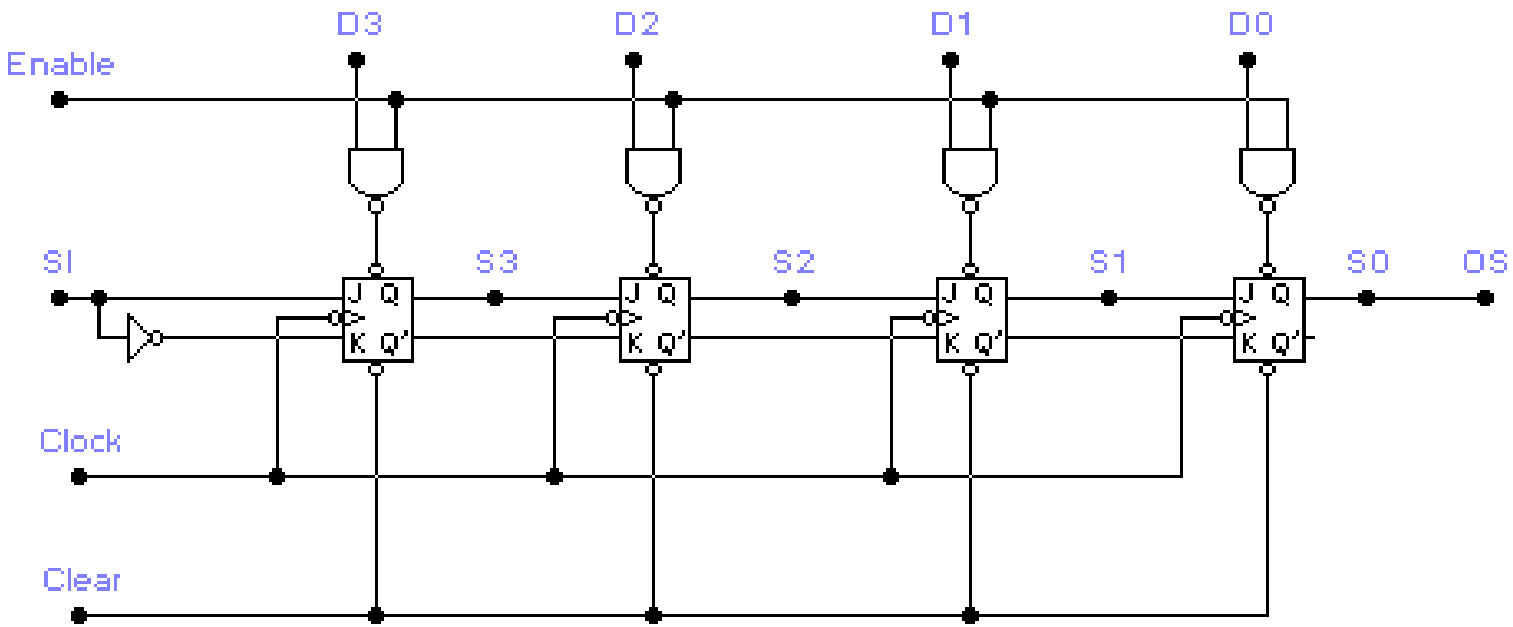
Registrador de 8 bits com Clear

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
ENTITY reg8 IS
   PORT ( D : IN STD_LOGIC_VECTOR(7 DOWNTO 0) ;
          Resetn, Clock: IN STD_LOGIC ;
          Q : OUT STD_LOGIC_VECTOR(7 DOWNTO 0) ) ;
END reg8 ;
ARCHITECTURE Behavior OF reg8 IS
BEGIN
   PROCESS ( Resetn, Clock )
   BEGIN
   IF Resetn = '0' THEN
          Q <= "00000000" ;
   ELSIF Clock'EVENT AND Clock = '1' THEN Q <= D ;
   END IF ;
   END PROCESS ;
END Behavior ;
```

Registrador de N bits

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY reg IS
   GENERIC ( N : INTEGER := 8 ) ;
   PORT (R : IN STD_LOGIC_VECTOR(N-1 DOWNTO 0) ;
   Rin, Clock: IN STD_LOGIC ;
   Q : OUT STD_LOGIC_VECTOR(N-1 DOWNTO 0) ) ;
END reg ;
ARCHITECTURE Behavior OF reg IS
BEGIN
   PROCESS
   BEGIN
         WAIT UNTIL Clock'EVENT AND Clock = '1' ;
         IF Rin = '1' THEN Q <= R ;
         END IF ;
   END PROCESS ;
END Behavior ;
```

Registrador de deslocamento



entrada série	saída paralelo
entrada paralelo	saída série
entrada série	saída série
entrada paralelo	saída paralelo

Registrador de deslocamento de 4 bits

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
ENTITY  shift4 IS
PORT ( R : IN STD_LOGIC_VECTOR(3 DOWNTO 0) ;
   Clock : IN STD_LOGIC ;
   L, w : IN STD_LOGIC ;
   Q : BUFFER STD_LOGIC_VECTOR(3 DOWNTO 0) )
   ;
END shift4 ;
ARCHITECTURE Behavior OF shift4 IS
BEGIN
PROCESS
```

```
BEGIN
   WAIT UNTIL Clock'EVENT AND Clock
   = '1' ;
   IF L = '1' THEN Q <= R ;
   ELSE
         Q(0) <= Q(1) ;
         Q(1) <= Q(2);
         Q(2) <= Q(3) ;
         Q(3) <= w ;
   END IF ;
END PROCESS ;
END Behavior ;
```

Registrador de deslocamento de N bits

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
ENTITY shift IS
   GENERIC ( N : INTEGER := 8 ) ;
   PORT ( R : IN STD_LOGIC_VECTOR(N-1 DOWNTO 0) ;
   Clock : IN STD_LOGIC ;
   L, w: IN STD_LOGIC ;
   Q : BUFFER STD_LOGIC_VECTOR(N-1 DOWNTO 0) ) ;
END shift ;
ARCHITECTURE Behavior OF shift IS
BEGIN
```

```
PROCESS
   BEGIN
   WAIT UNTIL Clock'EVENT AND Clock = '1' ;
   IF L = '1' THEN Q <= R ;
   ELSE
          Genbits: FOR i IN 0 TO N-2 LOOP
          Q(i) <= Q(i+1) ;
          END LOOP ;
          Q(N-1) <= w ;
   END IF ;
END PROCESS ;
END Behavior ;
```

Registrador de Deslocamento

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
ENTITY shift4 IS
   PORT ( R : IN STD_LOGIC_VECTOR(3 DOWNTO 0) ;
   L, w, Clock : IN STD_LOGIC ;
   Q : BUFFER STD_LOGIC_VECTOR(3 DOWNTO 0) ) ;
END shift4 ;
ARCHITECTURE   Structure  OF  shift4  IS
   COMPONENT muxdff
   PORT ( D0, D1, Sel, Clock : IN STD_LOGIC ;
          Q : OUT STD_LOGIC ) ;
   END COMPONENT ;
BEGIN
   Stage3: muxdff PORT MAP ( w, R(3), L, Clock,
   Q(3) ) ;
   Stage2: muxdff PORT MAP ( Q(3), R(2), L, Clock,
   Q(2) ) ;
   Stage1: muxdff PORT MAP ( Q(2), R(1), L, Clock,
   Q(1) ) ;
   Stage0: muxdff PORT MAP ( Q(1), R(0), L, Clock,
   Q(0) ) ;
END Structure ;
```

Contador Crescente Módulo 6 - AHDL

```
SUBDESIGN fig10_23
(
   clock, enable      :INPUT;     -- synch clock and enable.
   q[2..0], tc        :OUTPUT;    -- 3-bit counter
)
VARIABLE
   count[2..0]   :DFF;   -- declare a register of D flip-flops.

BEGIN
   count[].clk = clock;     -- connect all clocks to synchronous source
   IF enable THEN
      IF count[].q < 5 THEN
         count[].d = count[].q + 1;  -- increment current value by one
      ELSE count[].d = 0;            -- recycle,force unused states to 0
      END IF;
   ELSE count[].d = count[].q;       -- not enabled: hold at this count
   END IF;
   tc = enable & count[].q == 5;     -- detect maximum count if enabled
   q[] = count[].q;                  -- connect register to outputs
END;
```

Contador Crescente Módulo 6 - VHDL

```
ENTITY fig10_24 IS
PORT( clock, enable       :IN BIT ;
      q                   :OUT INTEGER RANGE 0 TO 5;
      tc                  :OUT BIT
    );
END fig10_24;

ARCHITECTURE a OF fig10_24 IS
BEGIN
   PROCESS (clock)                          -- respond to clock
   VARIABLE count   :INTEGER RANGE 0 TO 5;
```

Contador Crescente Módulo 6 - VHDL

```
    BEGIN
       IF (clock = '1' AND clock'event) THEN
          IF enable = '1' THEN              -- synchronous cascade input
             IF count < 5 THEN              -- < max (terminal) count?
                count := count + 1;
             ELSE
                count := 0;
             END IF;
          END IF;
       END IF;
       IF (count = 5) AND (enable = '1') THEN      -- synch cascade output
             tc <= '1';                            -- indicate terminal ct
       ELSE  tc <= '0';
       END IF;
       q <= count;                                 -- update outputs
    END PROCESS;
END a;
```

Somador / Subtrator

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

PACKAGE my_package IS
  CONSTANT ADDER_WIDTH : integer := 5;
  CONSTANT RESULT_WIDTH : integer := 6;

  SUBTYPE ADDER_VALUE IS integer RANGE 0 TO 2
  ** ADDER_WIDTH - 1;
  SUBTYPE RESULT_VALUE IS integer RANGE 0 TO 2
  ** RESULT_WIDTH - 1;
END my_package;
```

Somador / Subtrator

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE work.my_package.ALL;

ENTITY addsub IS
  PORT
  (
      a:        IN ADDER_VALUE;
      b:        IN ADDER_VALUE;
      addnsub:  IN STD_LOGIC;
      result:          OUT
  RESULT_VALUE
  );
END addsub;
```

```
ARCHITECTURE rtl OF addsub IS
BEGIN
  PROCESS (a, b, addnsub)
  BEGIN
      IF (addnsub = '1') THEN
            result <= a + b;
      ELSE
            result <= a - b;
      END IF;
  END PROCESS;
END rtl;
```

Contador Crescente Saída Binária

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE ieee.std_logic_unsigned.all ;
ENTITY upcount IS
   PORT ( Clock, Resetn, E : IN STD_LOGIC ;
   Q : OUT STD_LOGIC_VECTOR (3 DOWNTO 0)) ;
ARCHITECTURE Behavior OF upcount IS
SIGNAL Count : STD_LOGIC_VECTOR (3 DOWNTO 0) ;
BEGIN
PROCESS ( Clock, Resetn )
   BEGIN
   IF Resetn = '0' THEN
   Count <= "0000" ;
   ELSIF (Clock'EVENT AND Clock = '1');
THEN
          IF E = '1' THEN
         Count <= Count + 1 ;
           ELSE
         Count <= Count ;
           END IF ;
END IF ;
END PROCESS ;


Q <= Count ;
END Behavior ;
```

Contador com Carga

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
ENTITY upcount IS
   PORT ( R : IN INTEGER RANGE 0 TO 15 ;
   Clock, Resetn, L : IN STD_LOGIC ;
   Q :BUFFER INTEGER RANGE 0 TO 15 ) ;
END upcount ;

ARCHITECTURE Behavior OF upcount IS
BEGIN
PROCESS ( Clock, Resetn )
BEGIN
   IF Resetn = '0' THEN
   Q <= 0 ;
```

```
ELSIF (Clock'EVENT AND Clock = '1') THEN
         IF L = '1' THEN
                  Q <= R ;
         ELSE
                  Q <= Q + 1 ;
         END IF;
END IF;
END PROCESS;
END Behavior;
```

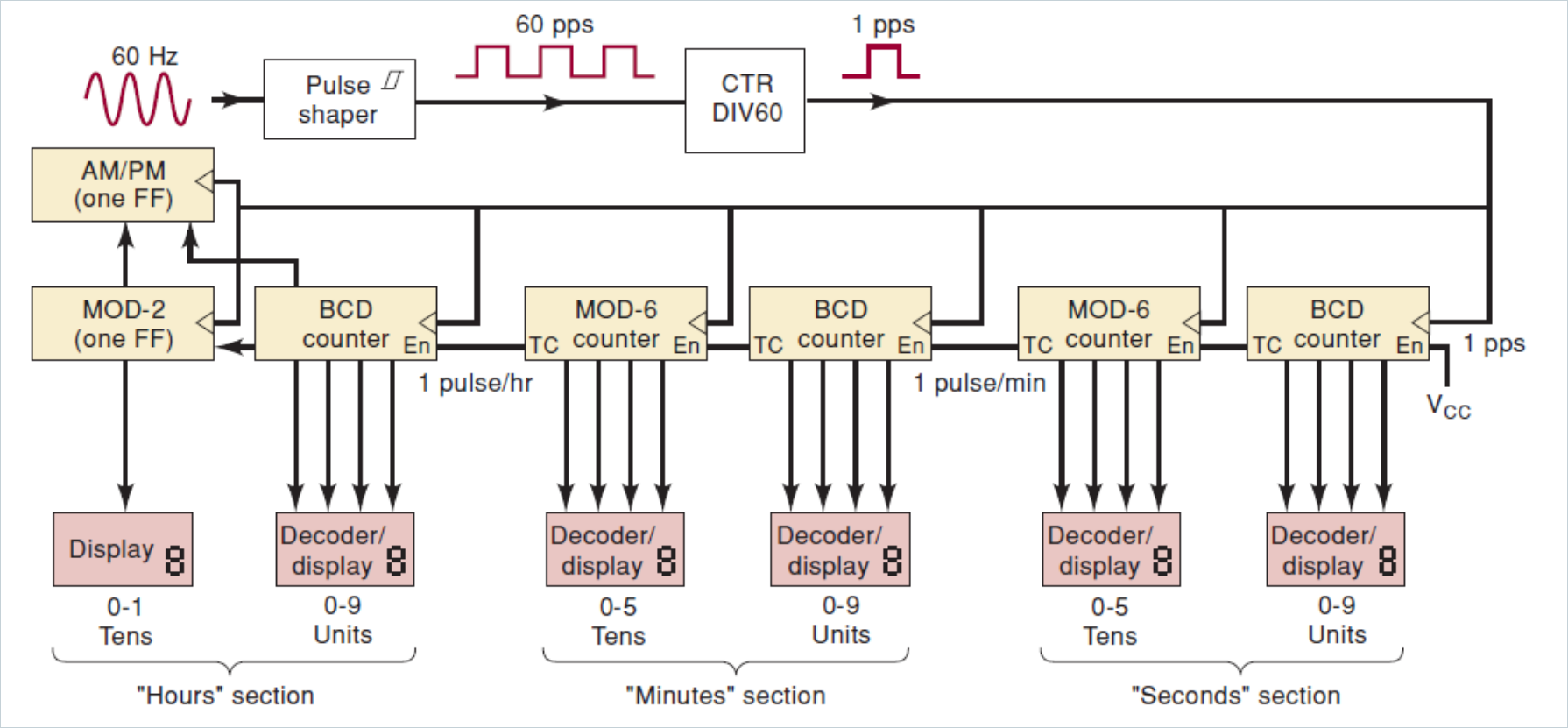
Obs: com o uso do tipo BUFFER,
o sinal count não é necessário

Contador Decrescente

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
ENTITY downcnt IS
   GENERIC ( modulus : INTEGER := 8 ) ;
   PORT ( Clock, L, E : IN STD_LOGIC ;
   Q : OUT INTEGER RANGE 0 TO modulus-1
   ) ;
END downcnt ;

ARCHITECTURE Behavior OF downcnt IS
SIGNAL Count : INTEGER RANGE 0 TO
   modulus-1 ;
BEGIN
```

```
PROCESS
BEGIN
   WAIT UNTIL (Clock'EVENT AND Clock = '1');
      IF E = '1' THEN
         IF L = '1' THEN
- carrega o módulo
            Count <= modulus-1;
         ELSE
            Count <= Count-1;
         END IF;
       END IF;
END PROCESS;

Q <= Count;
END Behavior;
```

Projeto Relógio Digital



Contador de Módulo 10 - AHDL

```
SUBDESIGN fig10_26
(
   clock, enable      :INPUT;      -- synch clock and enable.
   q[3..0], tc        :OUTPUT;     -- 4-bit Decade counter
)
VARIABLE
   count[3..0] :DFF;          -- declare a register of D flip flops.

BEGIN
   count[].clk = clock;       -- connect all clocks to synchronous source
   IF enable THEN
      IF count[].q < 9 THEN
         count[].d = count[].q + 1;    -- increment current value by one
      ELSE count[].d = 0;              -- recycle,force unused states to 0
      END IF;
   ELSE count[].d = count[].q;         -- not enabled: hold at this count
   END IF;
   tc = enable & count[].q == 9;       -- detect maximum count
   q[] = count[].q;                    -- connect register to outputs
END;
```

Contador de Módulo 10 - VHDL

```
ENTITY fig10_27 IS
PORT( clock, enable     :IN BIT ;
      q                 :OUT INTEGER RANGE 0 TO 9;
      tc                :OUT BIT
    );
END fig10_27;

ARCHITECTURE a OF fig10_27 IS
BEGIN
   PROCESS (clock)                          -- respond to clock
   VARIABLE count   :INTEGER RANGE 0 TO 9;

   BEGIN
      IF (clock = '1' AND clock'event) THEN
         IF enable = '1' THEN               -- synchronous cascade input
            IF count < 9 THEN               -- decade counter
               count := count + 1;
            ELSE
               count := 0;
            END IF;
         END IF;
      END IF;
      IF (count = 9) AND (enable = '1') THEN    -- synch cascade output
            tc <= '1';
      ELSE  tc <= '0';
      END IF;
      q <= count;                                -- update outputs
   END PROCESS;
END a;
```

Contador de Módulo 12 - AHDL

```
SUBDESIGN fig10_28
(
   clk, ena           :INPUT;
   low[3..0], hi, pm :OUTPUT;
)
VARIABLE
   low[3..0]     :DFF;
   hi            :DFF;
   am_pm         :JKFF;
   time          :NODE;
BEGIN
   low[].clk = clk;          -- synchronous clocking
   hi.clk = clk;
   am_pm.clk = clk;
   IF ena THEN               -- use enable to count
```

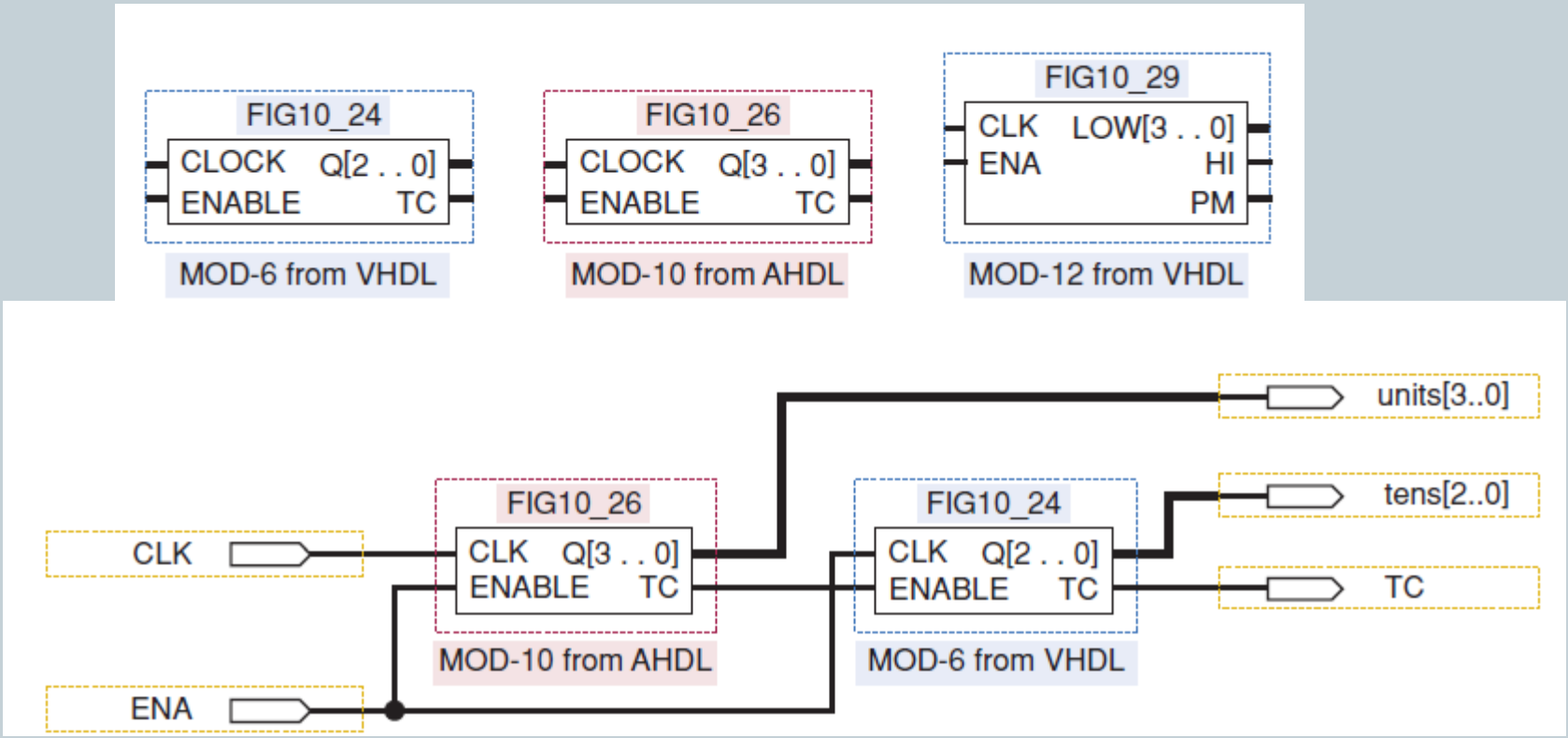
Contador de Módulo 12 - AHDL

```
BEGIN
   low[].clk = clk;          -- synchronous clocking
   hi.clk = clk;
   am_pm.clk = clk;
   IF ena THEN               -- use enable to count
       IF low[].q < 9 & hi.q == 0   THEN
         low[].d = low[].q + 1; --inc lo digit
         hi.d = hi.q;    -- hold hi digit
      ELSIF low[].q == 9 THEN
         low[].d = 0;
         hi.d = VCC;
      ELSIF hi.q == 1 & low[].q < 2 THEN
               low[].d = low[].q + 1;
               hi.d = hi.q;
      ELSIF hi.q == 1 & low[].q == 2 THEN
               low[].d = 1;
               hi.d = GND;
      END IF;
   ELSE
      low[].d = low[].q;
      hi.d = hi.q;
   END IF;
   time = hi.q == 1 & low[3..0].q == 1 & ena;    -- detect 11:59:59
   am_pm.j = time;       -- toggle am/pm at noon and midnight
   am_pm.k = time;
   pm = am_pm.q;
END;
```

Contador de Módulo 12 - VHDL

```
ENTITY fig10_29 IS
PORT( clk, ena       :IN BIT ;
      low            :OUT INTEGER RANGE 0 TO 9;
      hi             :OUT INTEGER RANGE 0 TO 1;
      pm             :OUT BIT            );
END fig10_29;

ARCHITECTURE a OF fig10_29 IS
BEGIN
   PROCESS (clk)                          -- respond to clock
   VARIABLE am_pm :BIT;
   VARIABLE ones  :INTEGER RANGE 0 TO 9;  -- 4-bit units signal
   VARIABLE tens  :INTEGER RANGE 0 TO 1;  -- 1-bit tens signal
   BEGIN
      IF (clk = '1' AND clk'EVENT) THEN
         IF ena = '1' THEN                -- synchronous cascade input
            IF (ones = 1) AND (tens = 1) THEN   -- at 11:59:59
               am_pm := NOT am_pm;              -- toggle am/pm
            END IF;
            IF (ones < 9) AND  (tens = 0) THEN  -- states 00-08
               ones := ones + 1;          -- increment units
            ELSIF ones = 9 THEN           -- state 09...set to 10:00
               ones := 0;                 -- units reset to zero
               tens := 1;                 -- tens bump up to 1
            ELSIF (tens = 1) AND (ones < 2) THEN-- states 10, 11
               ones := ones + 1;          -- increment units
            ELSIF (tens = 1) AND (ones = 2) THEN -- state 12
               ones := 1;
               tens := 0;
            END IF;
         END IF;
      END IF;
      pm <= am_pm;
      low <= ones;
      hi <= tens;
   END PROCESS;
END a;
```

Combinando módulos



Combinando módulos em AHDL

```
INCLUDE "fig10_26.inc";       -- mod-10 counter module
INCLUDE "fig10_23.inc";       -- mod-6 counter module

SUBDESIGN fig10_36
(
   clk, ena                          :INPUT;
   ones[3..0], tens[2..0], tc        :OUTPUT;
)
VARIABLE
   mod10           :fig10_26;        -- mod-10 for units
   mod6            :fig10_23;        -- mod-6 for tens
BEGIN
   mod10.clock = clk;                -- synchronous clocking
   mod6.clock = clk;
   mod10.enable = ena;
   mod6.enable = mod10.tc;           -- cascade
   ones[3..0] = mod10.q[3..0];       -- 1s
   tens[2..0] = mod6.q[2..0];        -- 10s
   tc = mod6.tc;                     -- Make terminal count at 59
END;
```

Combinando módulos em VHDL

```
ENTITY fig10_37 IS
PORT( clk, ena       :IN BIT ;
      tens           :OUT INTEGER RANGE 0 TO 5;
      ones           :OUT INTEGER RANGE 0 TO 9;
      tc             :OUT BIT            );
END fig10_37;

ARCHITECTURE a OF fig10_37 IS
SIGNAL cascade_wire  :BIT;
COMPONENT fig10_24                          -- MOD-6 module
PORT( clock, enable  :IN BIT ;
      q              :OUT INTEGER RANGE 0 TO 5;
      tc             :OUT BIT);
END COMPONENT;
COMPONENT fig10_27                          -- MOD-10 module
PORT( clock, enable  :IN BIT ;
      q              :OUT INTEGER RANGE 0 TO 9;
      tc             :OUT BIT);
END COMPONENT;

BEGIN
   mod10:fig10_27
      PORT MAP(   clock => clk,
                  enable => ena,
                  q  => ones,
                  tc => cascade_wire);

   mod6:fig10_24
      PORT MAP(   clock => clk,
                  enable => cascade_wire,
                  q  => tens,
                  tc => tc);
END a;
```

Frequencímetro

- Dispositivo capaz de contar um número de pulsos em um determinado espaço de tempo.

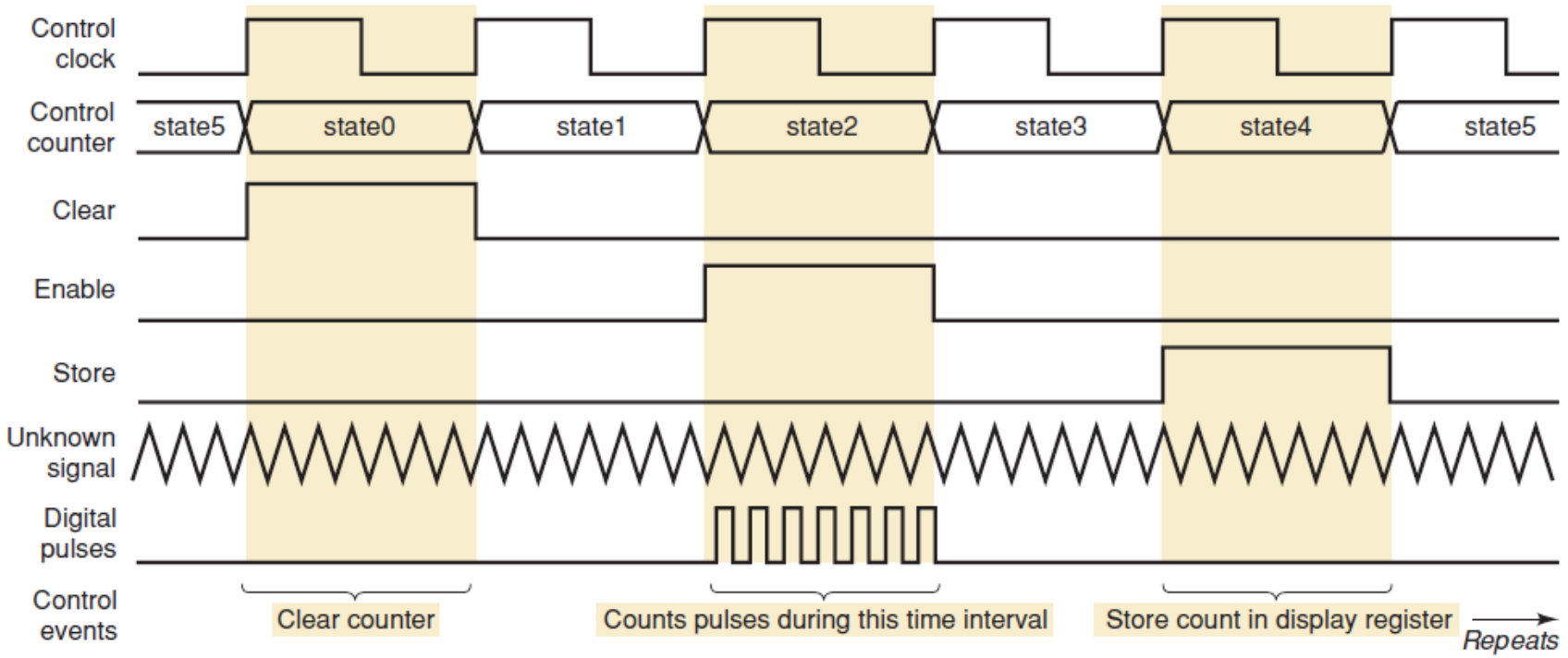
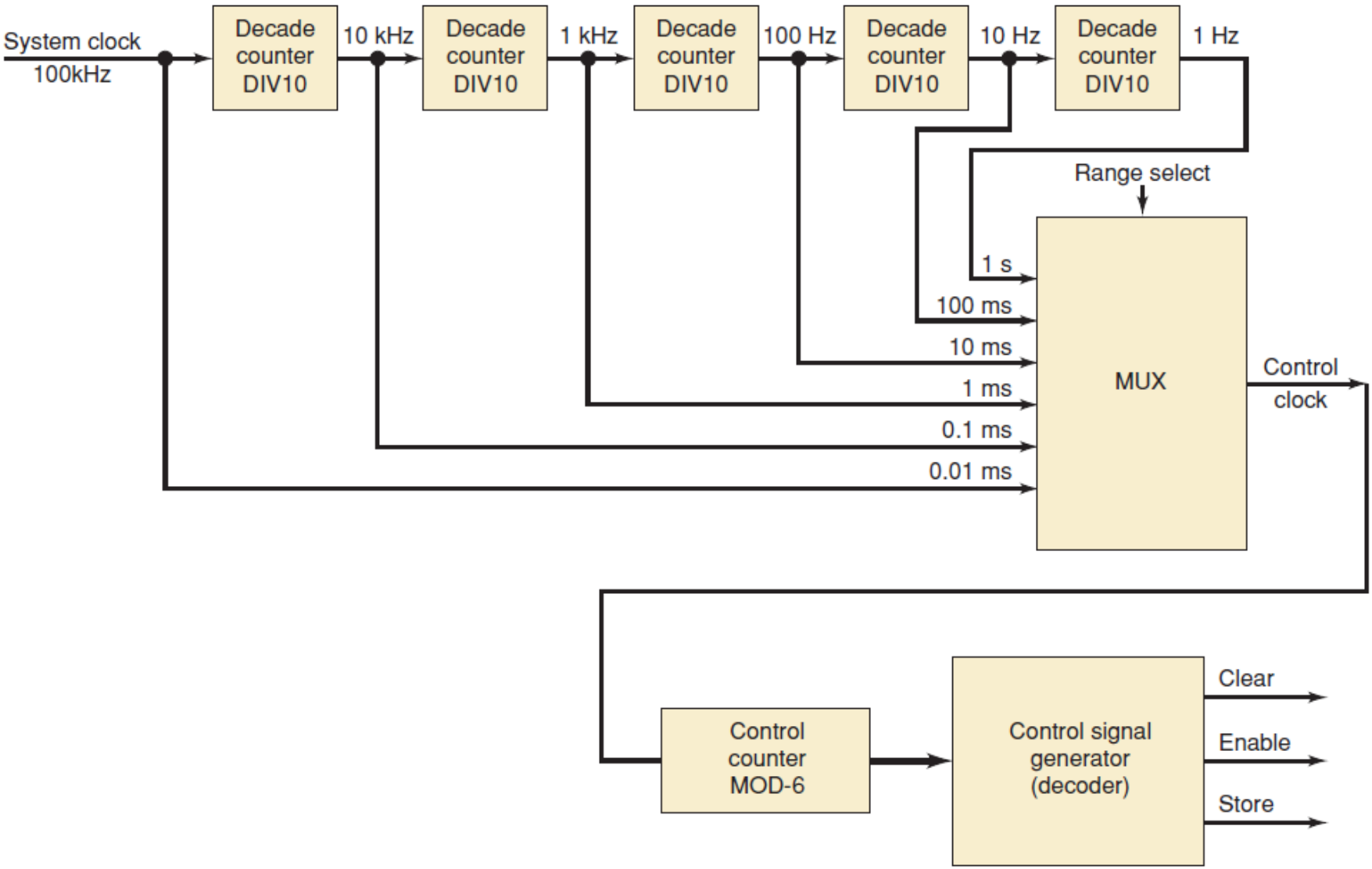


Diagrama em Bloco – do frequencímetro



Controle de Dispositivos

- Existe três tipos de controles relacionandos ao dispositivos de controle de hardware.
  - **Controle Simples**
  - **Controle Buferizado**
  - **Controle de Barramentos**

Controle Simples

- Circuito que utiliza o acesso direto a um determinado disposito (pinagem). O acionamento pode ser realizado de entrada ou saída, desde que sejam utilizados dispositivos do tipo pull-up ou pull-down com a mesma tensão do dispositivo a utilizar esse tipo de controle.

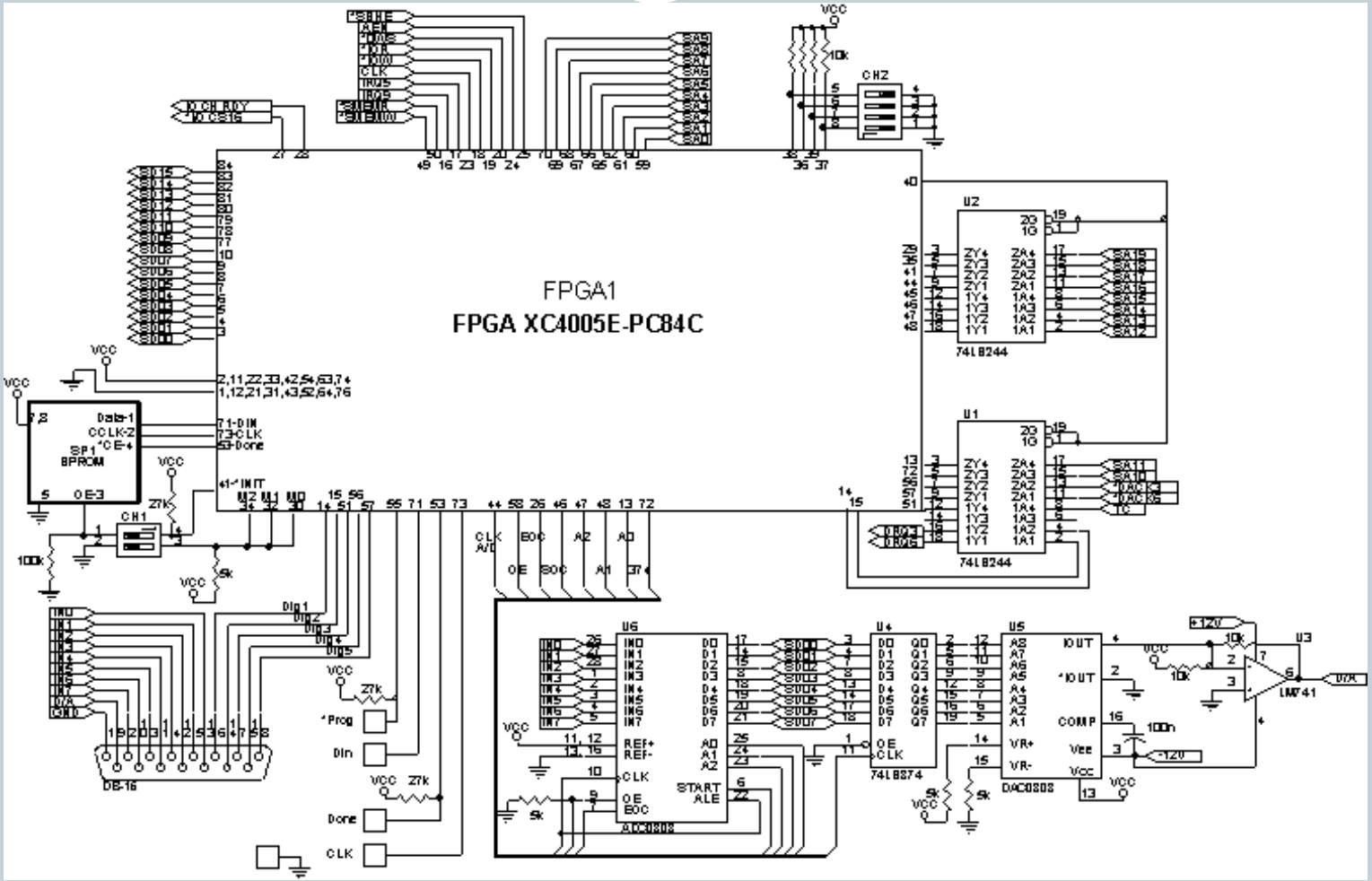
Controle Buferizado

- Esse controle de I/O utiliza um dispositivo do tipo latch ou buffer para controlar um determinado dispositivo.
- Esse tipo de controle é util para realizar a interface com o mundo real, ou onde pode não haver compatibilidade de níveis de energia.
- O controle bufferizado proteje o dispositivo a ser controlado de problemas de curto circuito, isto é, um barramento ou acionamento de entrada ligado a outro de saída.

Controle de Barramentos

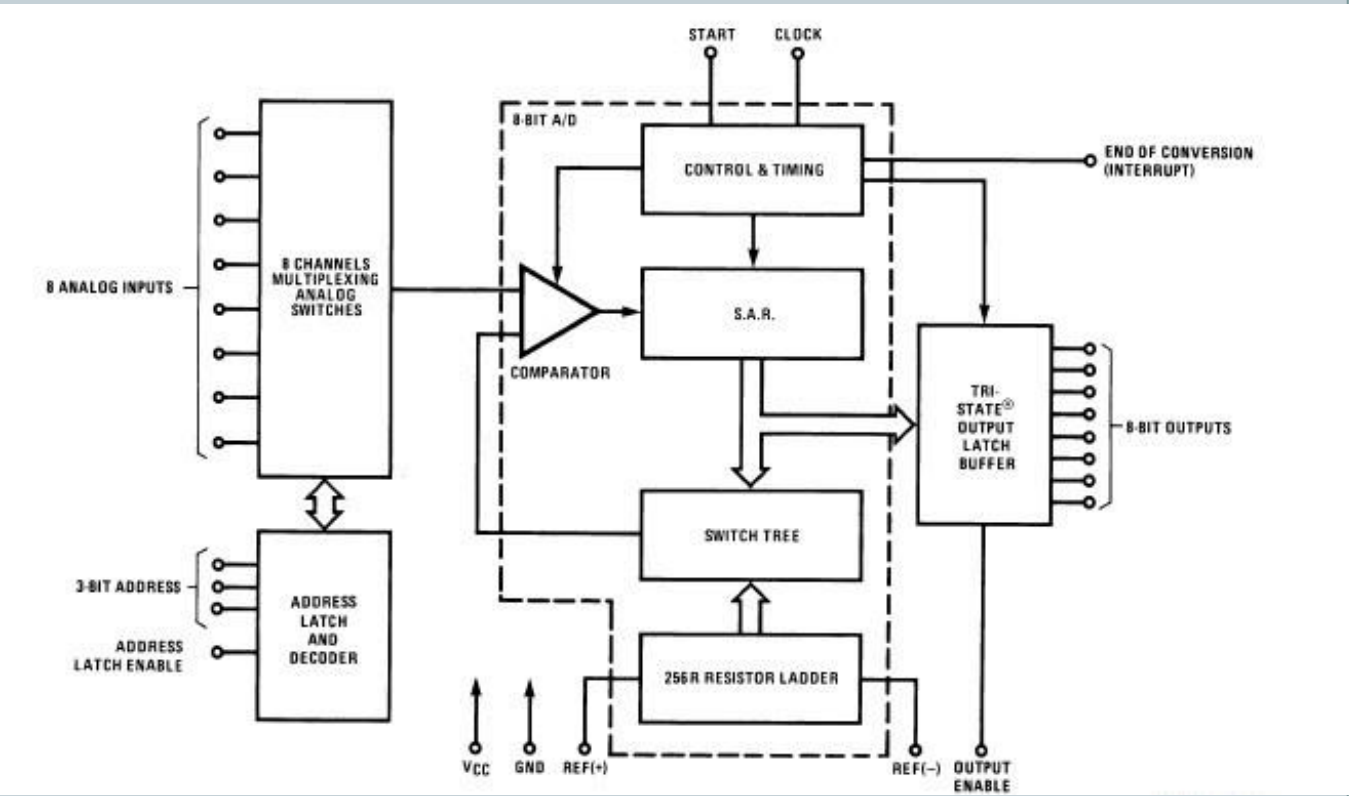
- O controle de barramentos é identico ao controle simples porém utiliza um acionamento (pino) que indica se o processo de controle é relacionado a uma atividade de entrada ou de saída do sinal.
- Esse controle so pode existir entre dispositivos que utilizam o mesmo nível de tensão e normalmente não utiliza dispositivos do tipo buffer, pois quanto maior a velocidade do controle melhor.
- O controle de barramento é utilizado por exemplo para o controle de memórias e dispositivos de troca de informações.

Controle de Dispositivos



Controle simples

- Como controlar os dispositivos de I/O do conversor ADC 0808 ?



Exemplo: **Controle Simples**

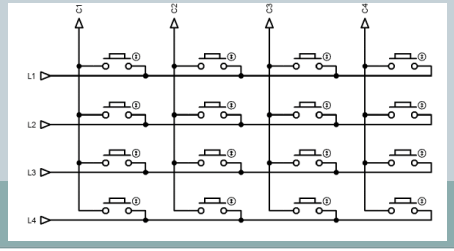
- Crie um diagrama em blocos e posteriormente escolha uma linguagem de descrição de hardware para gerar o o código do controle de um conversor ADC0808
- Especificações:
  - Existe um clock na placa de 5Mhz.
  - Definir a taxa de amostragem do conversor A/D em 2500 amostras por segundo.

Exemplo: **Controle de barramento**

- Utilizando o mesmo exemplo do conversor A/D, utilize os sinais RD e IO de um microprocessador para a transmissão dos dados convertidos para um barramento.
  - RD – indica uma operação de leitura.
  - IO – indica uma operação de acesso a leitura de um periférico.
- O dado somente deve ser liberado para o barramento após ter sido completo o ciclo de conversão do A/D e os pedidos de leitura sejam realizado pelo microcontrolador.

Exemplo: **Controle Bufferizado**

- Existe um teclado com 16 teclas porém um microcontrolador somente possui 4 pinos para leituras de teclas. Crie um dispositivo encoder para a leitura das teclas e envio dos referidos códigos em para um microcontrolador.
  - Utilizar o método de varredura das teclas através de uma matriz de acionamento.
    - O circuito gera um acionamento para uma linha e verifica se teve o retorno em alguma coluna. Caso exista um retorno significa que houve uma tecla pressionada, onde esta deve ser codificada e enviada para o microcontrolador.

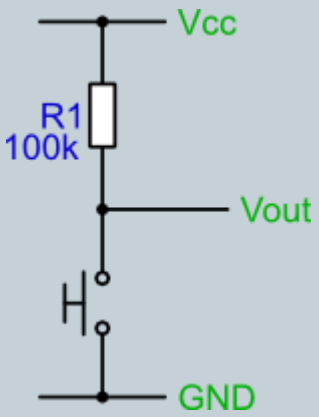


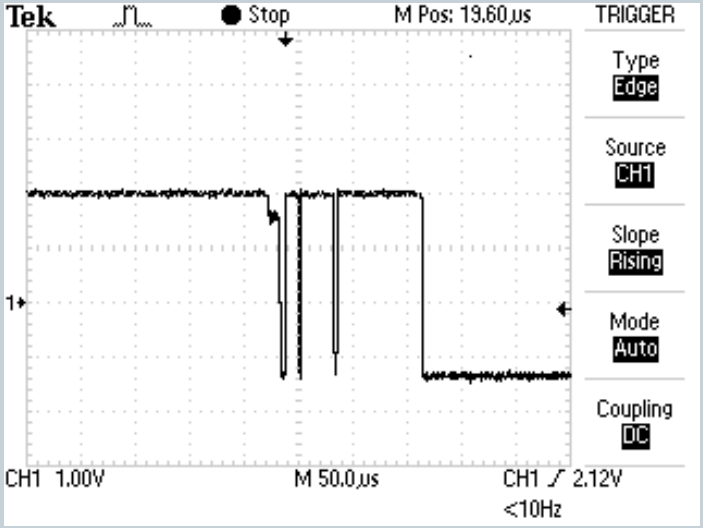
Exemplos de controles

- Fazer o diagrama em bloco do circuito para o encoder de teclado.
- Escrever o código dos blocos do encoder
- Existe um clock na máquina de 50Mhz que pode ser utilizado no circuito.

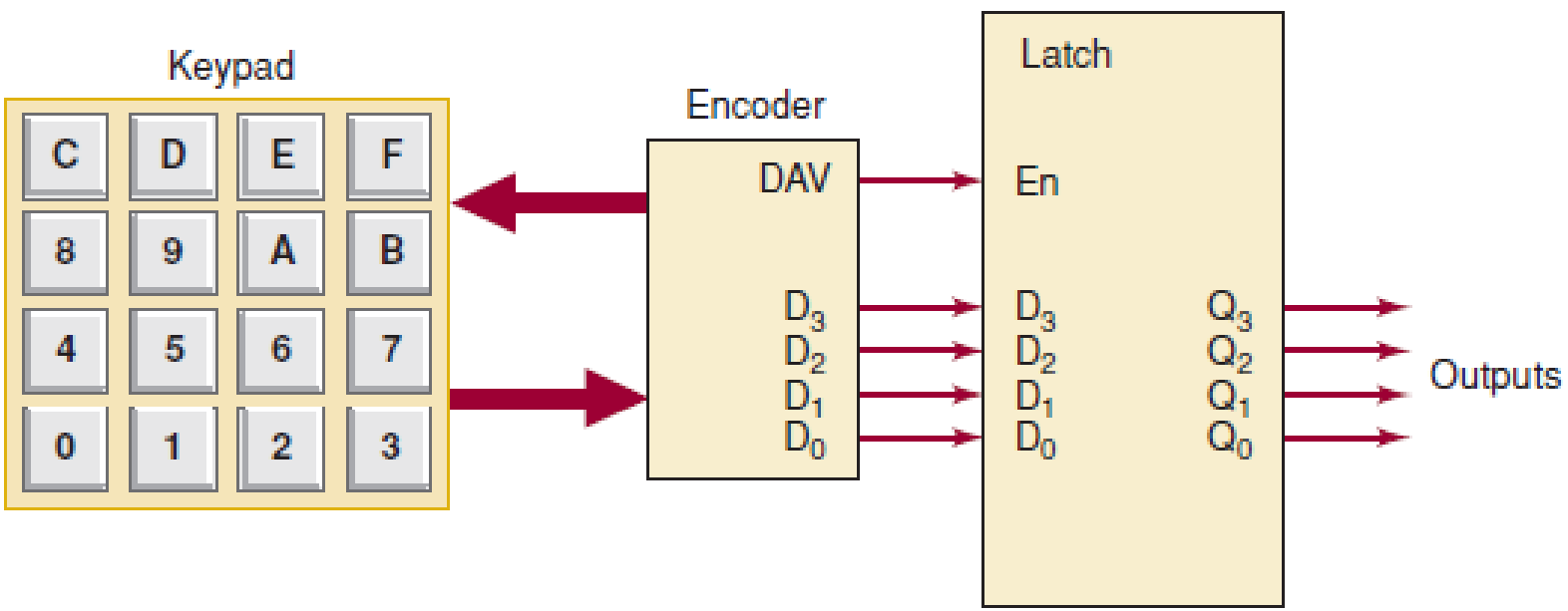
Exemplos de controles

- Projetar uma rotina de espera, isto é, um circuito capaz de fazer um delay, por um determinado tempo, enquanto a tecla for pressionada para a identificar o acionamento.



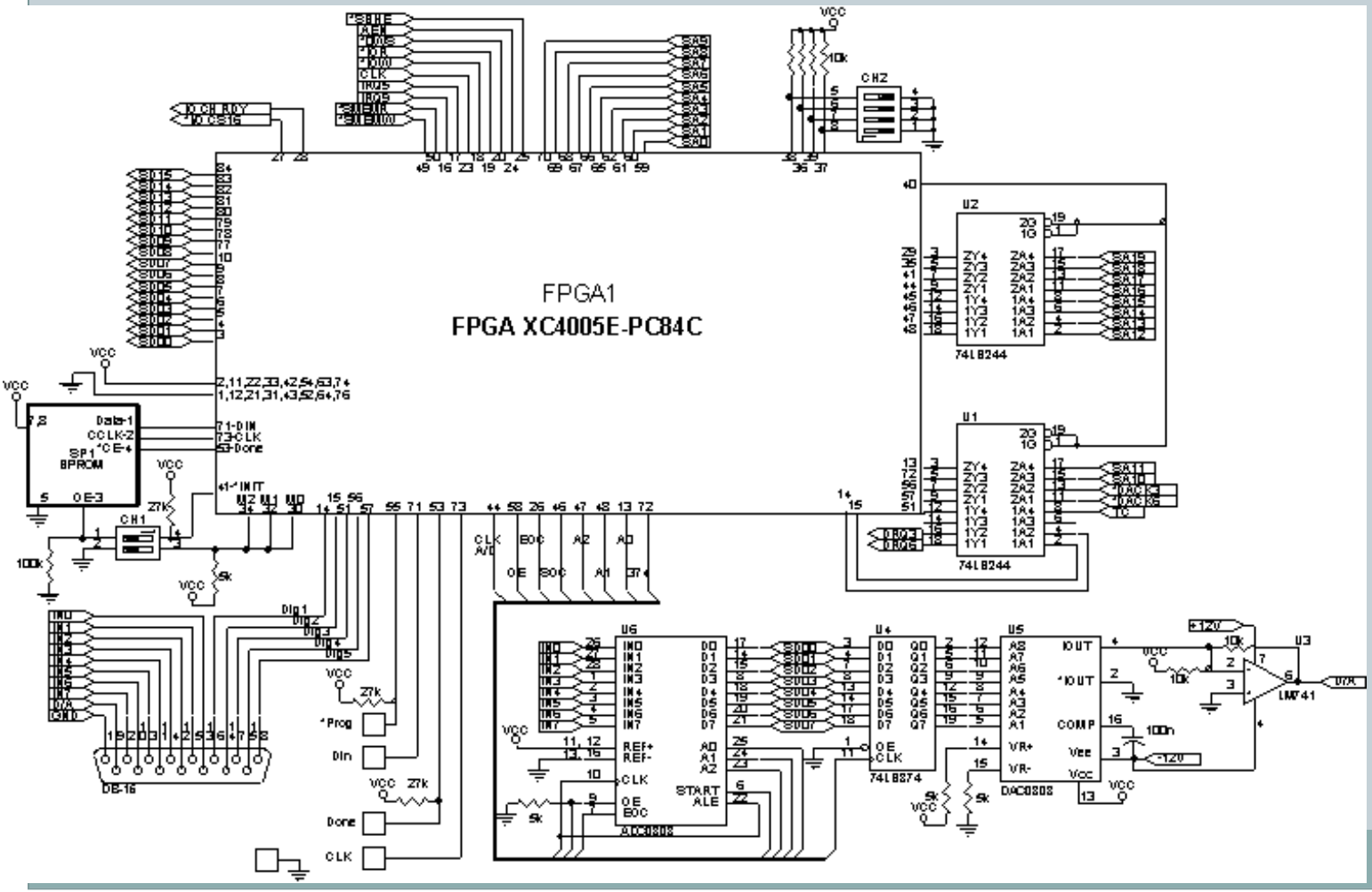


Controle de Dispositivo Bufferizado



Exercícios

Baseado na figura abaixo, fazer o que se pede:



Projetar um circuito para controlar os conversores A/D e D/A

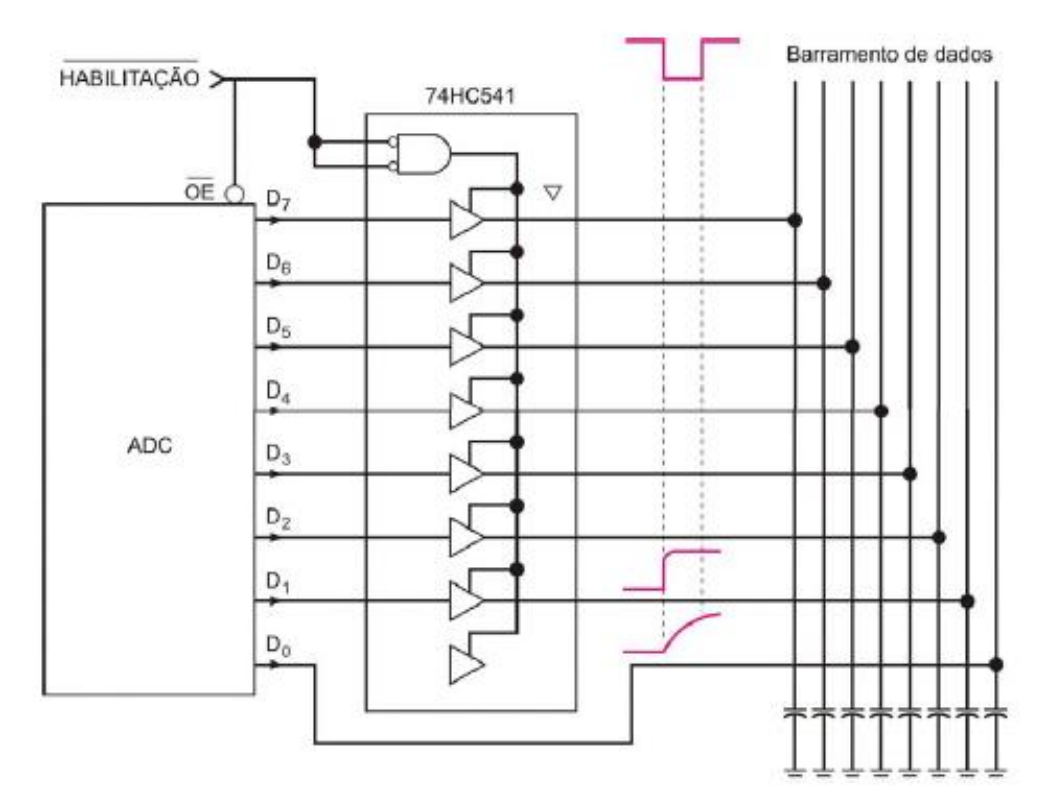
Exercícios

- Projetar um circuito para acionamento de periféricos via conector DB25, de acordo com o valor do conversor A/D. Seguindo a tabela:

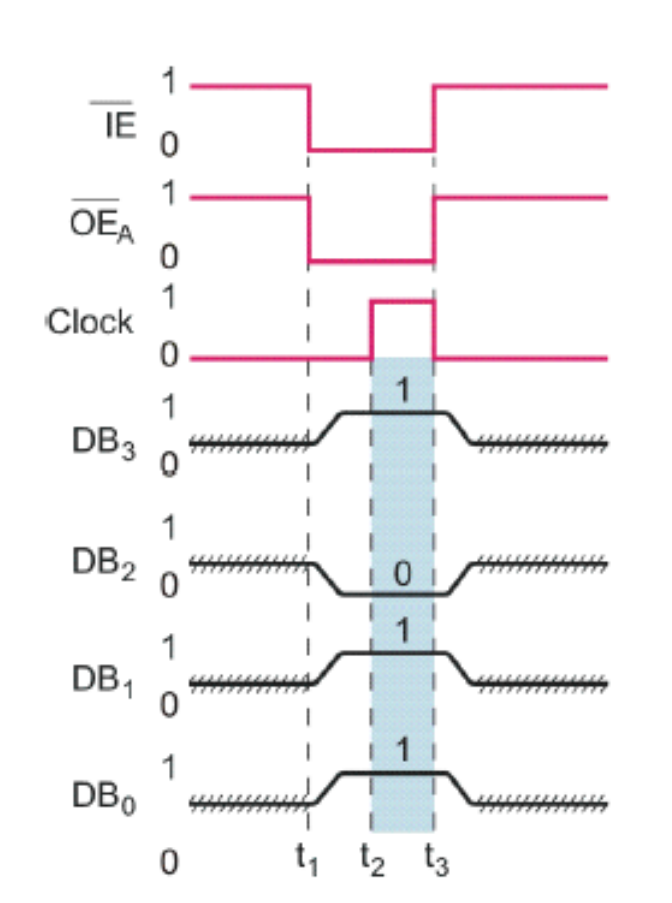
Valor do Conversor A/D	Acionamento
0<V<10	D0
10<V<20	D0 e D1
20<V<30	D0,D1 e D2
V>30	D3

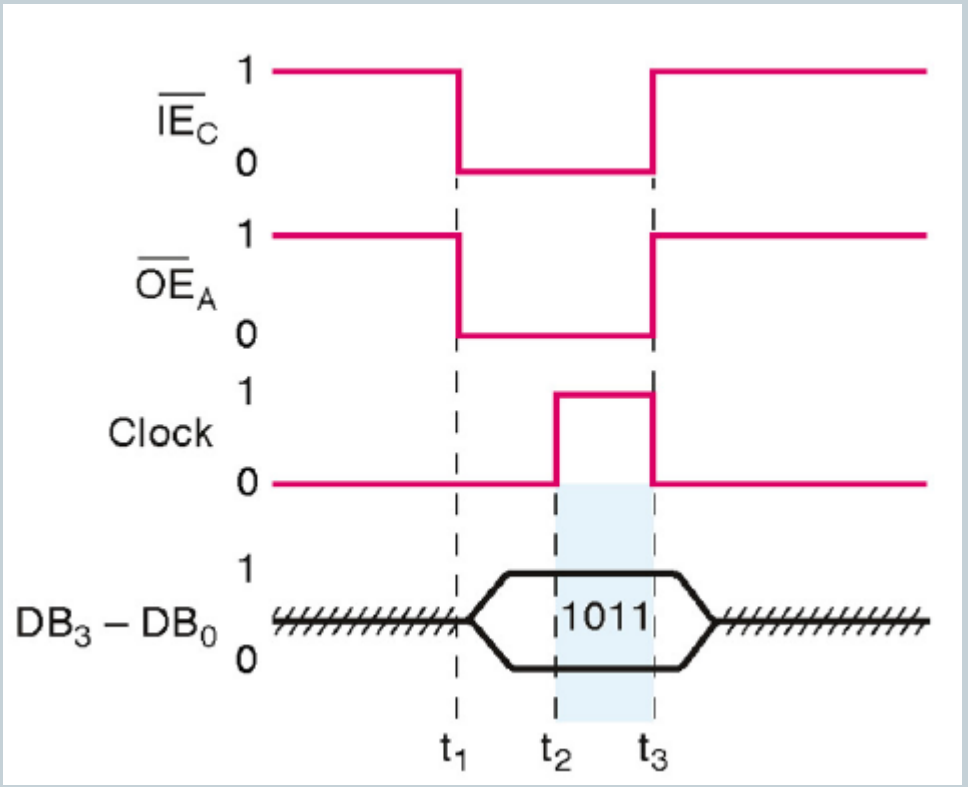
Controle de Barramento

- Atrasos devido ao efeito capacitivo

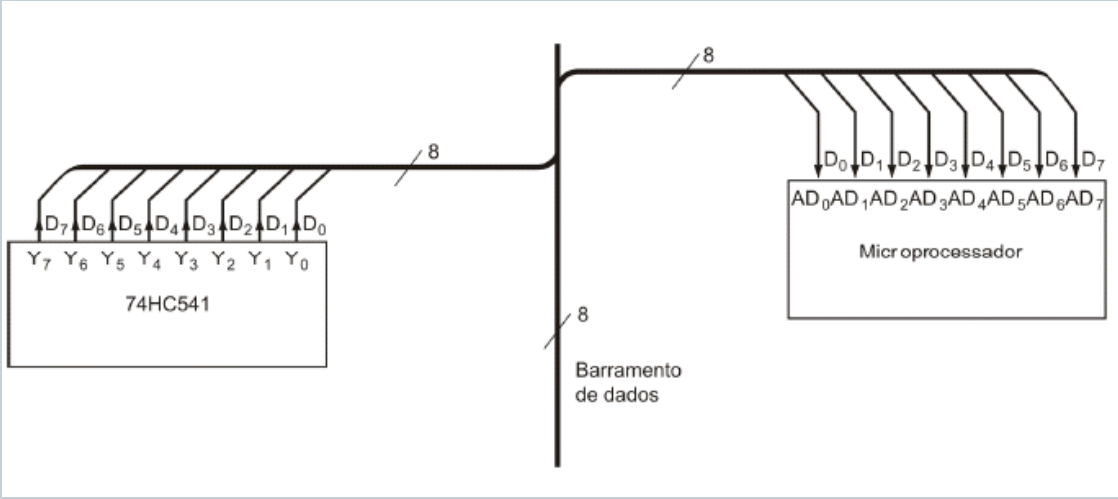


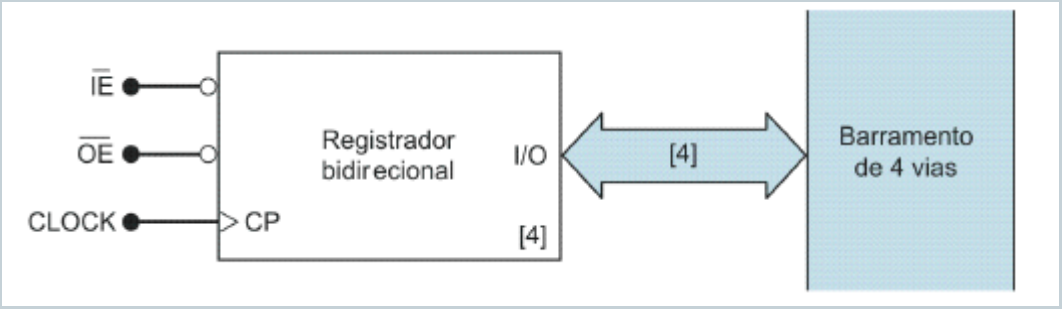
Controle de Barramento

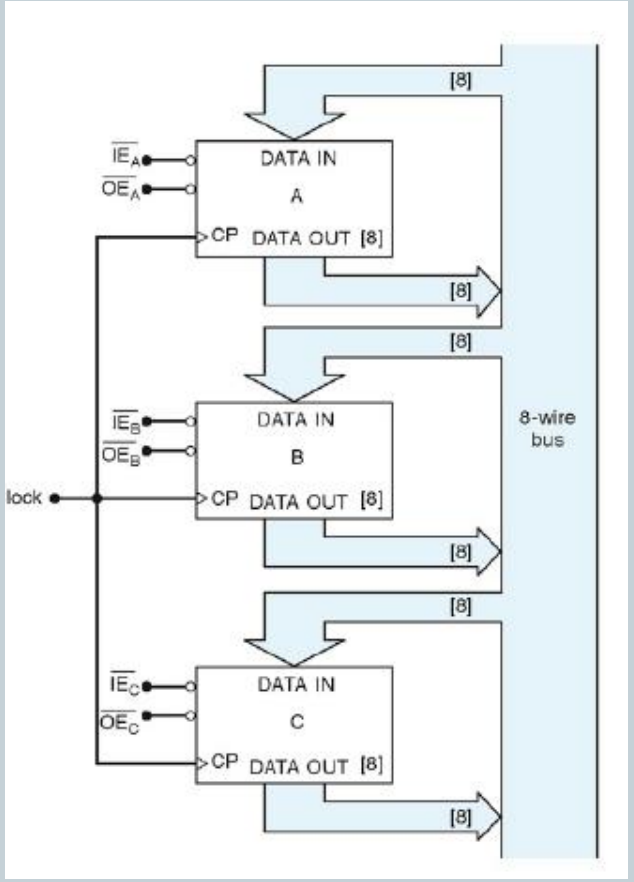




Controle de Barramento



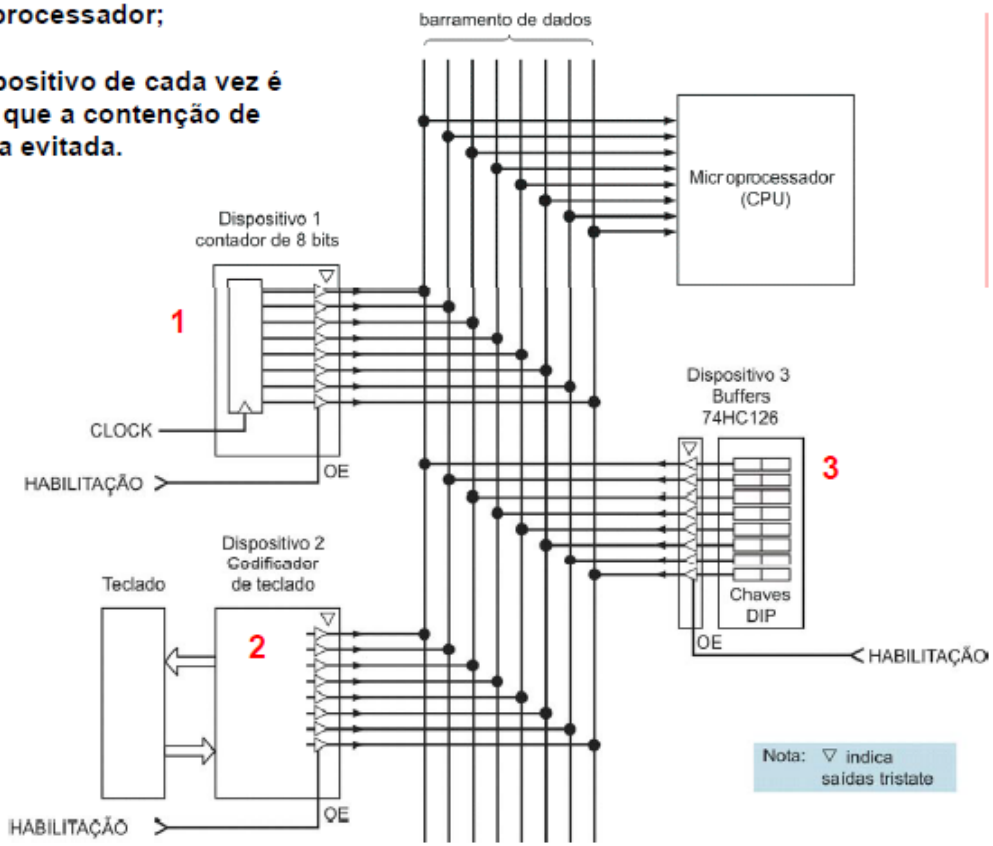




Controle de Barramento

Três dispositivos diferentes podem transmitir oito bits de dados por meio de um barramento de dados de oito linhas, para um microprocessador;

apenas um dispositivo de cada vez é habilitado para que a contenção de barramento seja evitada.



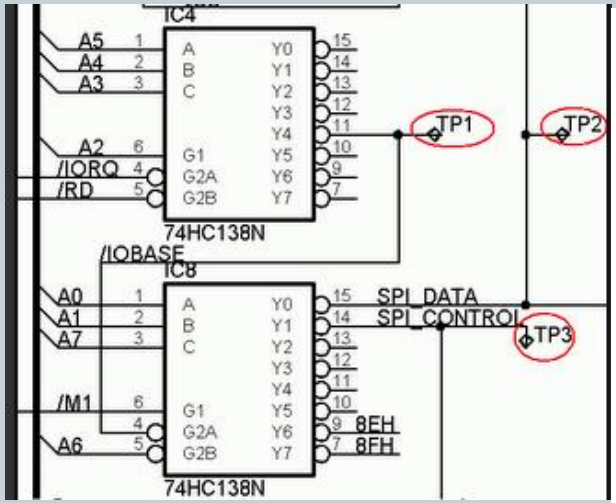
Controle de Barramento

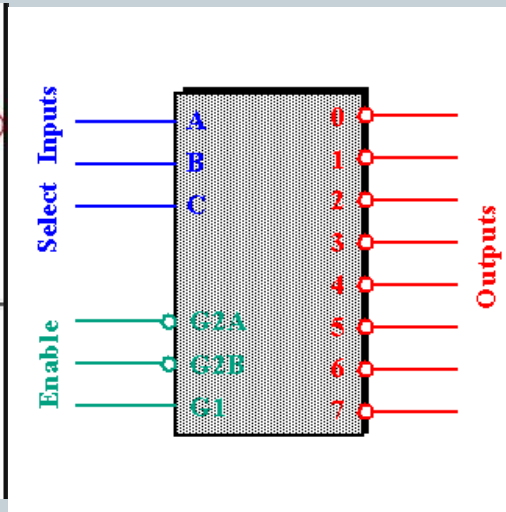
- Existe um endereço específico para o acionamento de cada periférico
- Exemplo: Periféricos do PC.

Component	PC/XT	AT
DMA controller (8237A-5)	000-00F	000-01F
Interrupt controller (8259A)	020-021	020-03F
Timer	040-043	040-05F
Programmable Peripheral Interface (PPI 8255A-5)	060-063	none
Keyboard (8042)	none	060-06F
Realtime clock (MC146818)	none	070-07F
DMA page register	080-083	080-09F
Interrupt controller 2 (8259A)	none	0A0-0BF
DMA controller 2 (8237A-5)	none	0C0-0DF
Math coprocessor	none	0F0-0F1
Math coprocessor	none	0F8-0FF
Hard drive controller	320-32F	1F0-1F8
Game port (joysticks)	200-20F	200-207
Expansion unit	210-217	none
Interface for second parallel printer	none	278-27F
Second serial interface	2F8-2FF	2F8-2FF
Prototype card	300-31F	300-31F
Network card	none	360-36F
Interface for first parallel printer	378-37F	378-37F
Monochrome Display Adapter and parallel interface	3B0-3BE	3B0-3BF
Color/Graphics Adapter	3D0-3DF	3D0-3DF
Disk controller	3F0-3F7	3F0-3F7
First serial interface	3F8-3FF	3F8-3FF

Controle de Barramento

- Como fazer o controle para o acionamento dos periféricos.
  - Definição do endereço
  - Criação da lógica de decodificação para o endereço

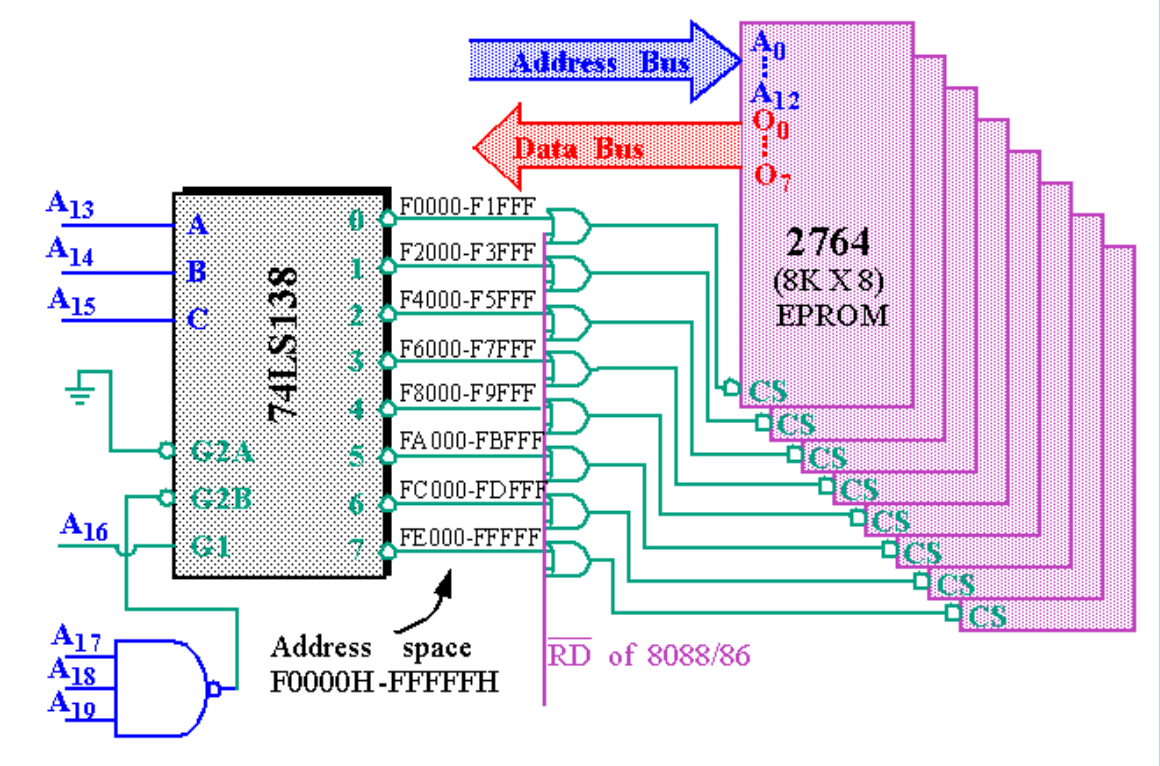




Inputs						Output							
Enable			Select										
G2A	G2B	G1	C	B	A	0	1	2	3	4	5	6	7
1	X	X	X	X	X	1	1	1	1	1	1	1	1
X	1	X	X	X	X	1	1	1	1	1	1	1	1
X	X	0	X	X	X	1	1	1	1	1	1	1	1
0	0	1	0	0	0	0	1	1	1	1	1	1	1
0	0	1	0	0	1	1	0	1	1	1	1	1	1
0	0	1	0	1	0	1	1	0	1	1	1	1	1
0	0	1	0	1	1	1	1	1	0	1	1	1	1
0	0	1	1	0	0	1	1	1	1	0	1	1	1
0	0	1	1	0	1	1	1	1	1	1	0	1	1
0	0	1	1	1	0	1	1	1	1	1	1	0	1
0	0	1	1	1	1	1	1	1	1	1	1	1	0

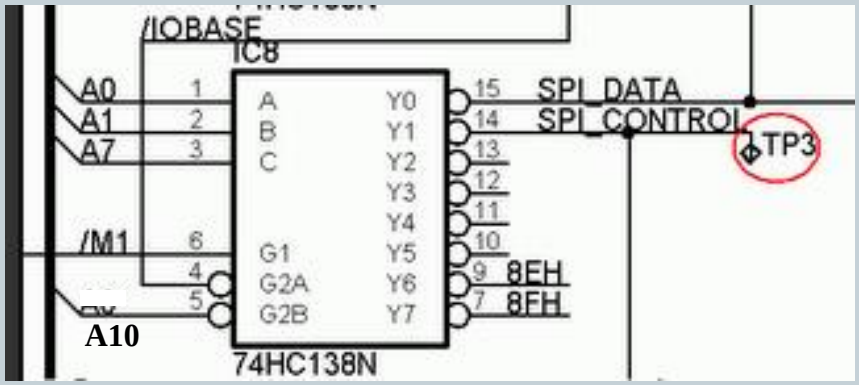
Controle de Barramento

- Controle de endereçamento de memória



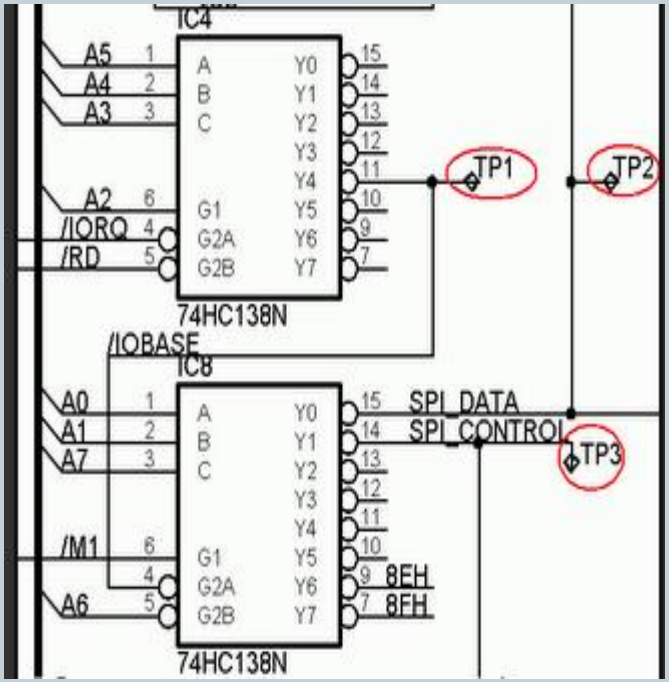
Controle de Barramento

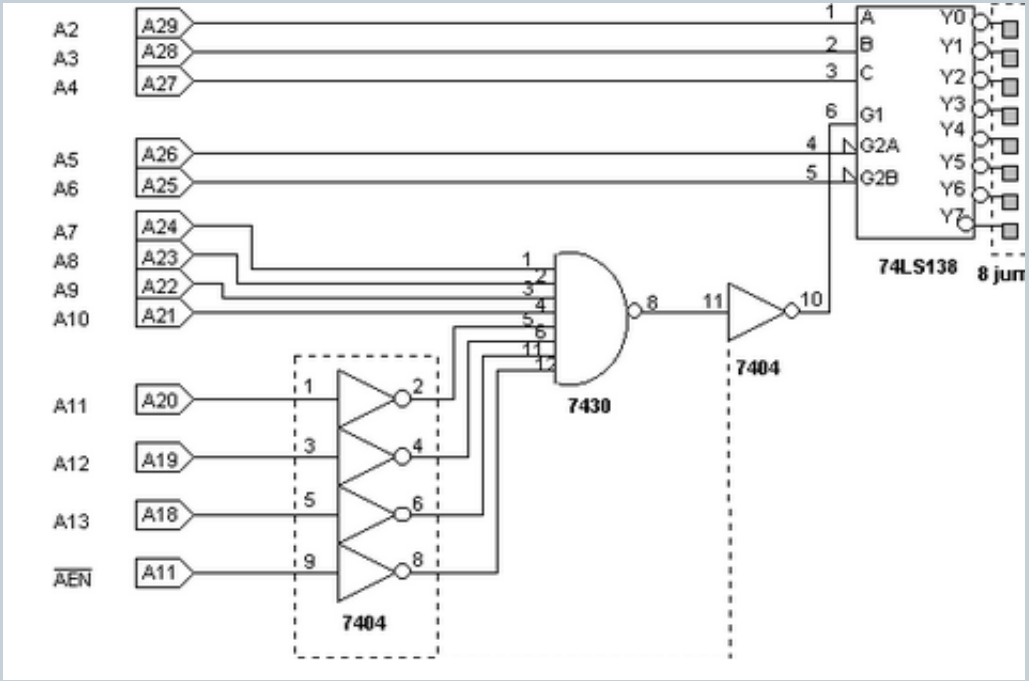
- Áreas de Sombra
  - Quando um dispositivo é mapeado (projeto de um endereço de acionamento) este deve possuir apenas um endereço físico capaz de acioná-lo, salvo se este endereço não for utilizado pelo circuito. Este endereço que não é utilizado pelo circuito e aciona o dispositivo é chamado de endereço de sombra ou área de sombra.



Os endereços A2, A3,A4,A5, A6,A8,A9 não são utilizados Logo qualquer valor aciona o Dispositivo.

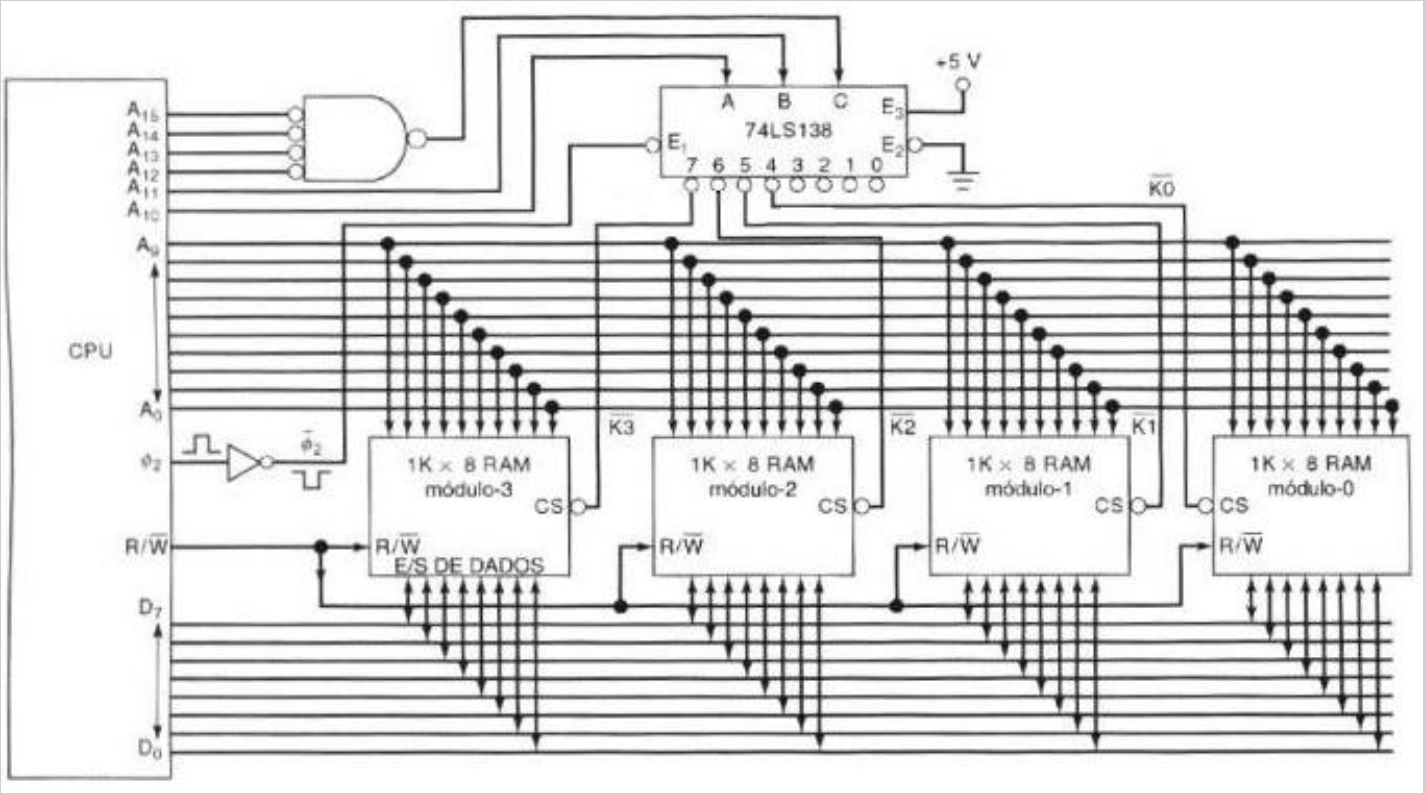
Controle de Barramento



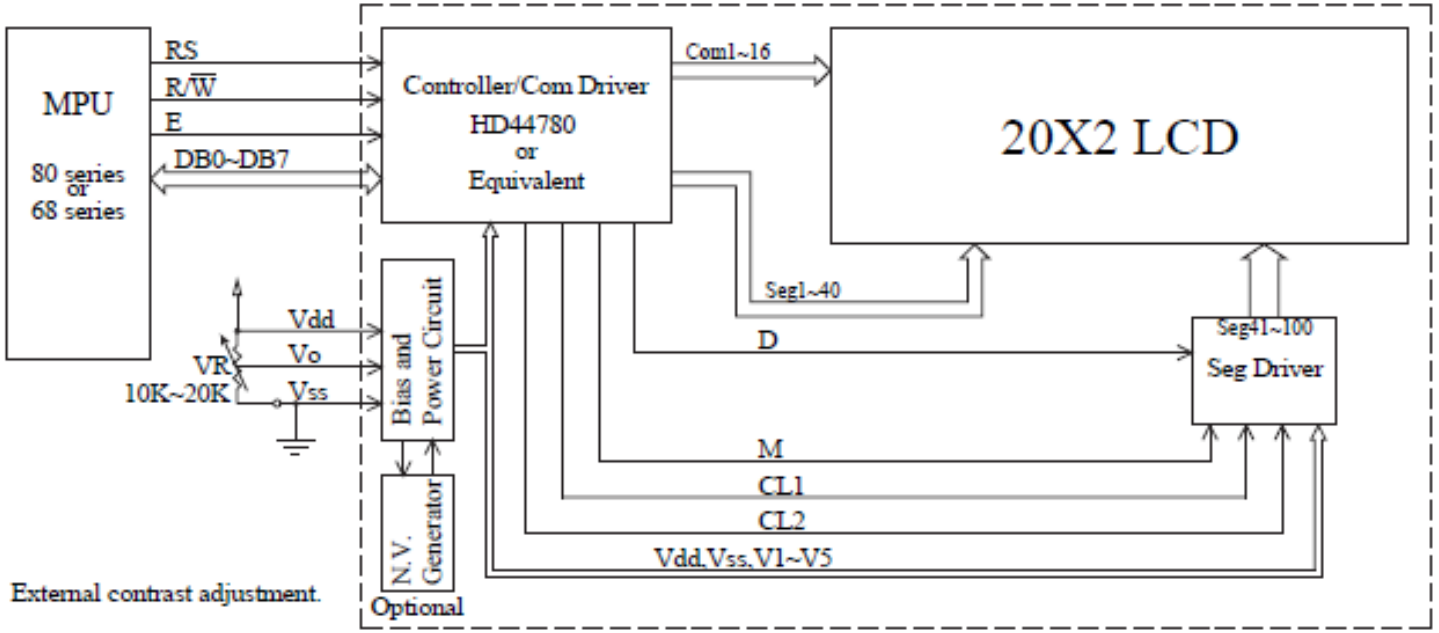


Controle de Barramento

- Identifique o endereço de acionamento de cada memória e os endereços de sombra de todo o circuito.



Controle de LCD

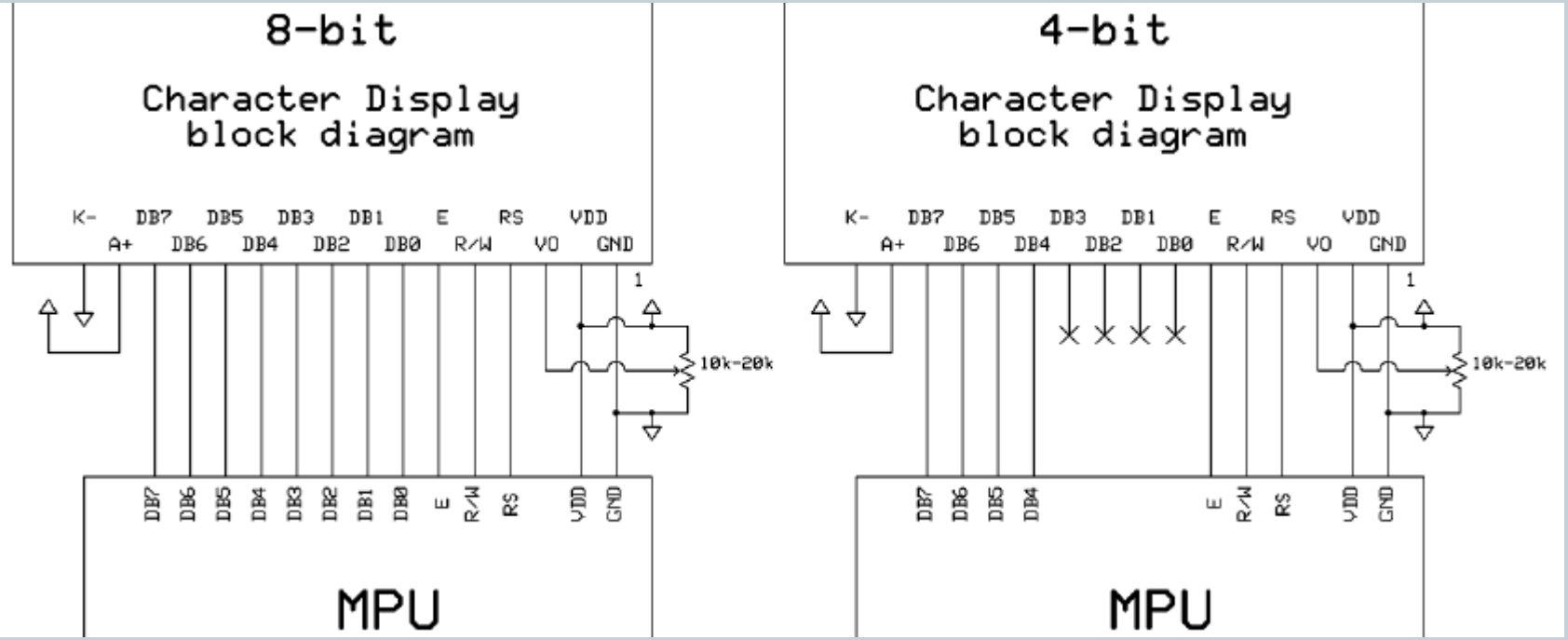


PIN NO.	SYMBOL
1	Vss
2	Vdd
3	Vo
4	RS
5	R/W
6	E
7	DB0
8	DB1
9	DB2
10	DB3
11	DB4
12	DB5
13	DB6
14	DB7
15	LED+
16	LED-

Controle de LCD: Especificação

Pin No.	Symbol	External Connection	Function Description
1	VSS	Power Supply	Ground
2	VDD	Power Supply	Supply Voltage for logic (+5.0V)
3	V0	Adj Power Supply	Power supply for contrast (approx. 0.5V)
4	RS	MPU	Register select signal. RS=0: Command, RS=1: Data
5	R/W	MPU	Read/Write select signal, R/W=1: Read R/W: =0: Write
6	E	MPU	Operation enable signal. Falling edge triggered.
7-10	DB0 – DB3	MPU	Four low order bi-directional three-state data bus lines. These four are not used during 4-bit operation.
11-14	DB4 – DB7	MPU	Four high order bi-directional three-state data bus lines.
15	LED+	Power Supply	Power supply for LED Backlight (+3.5V)
16	LED-	Power Supply	Ground for Backlight

Controle de LCD: Especificação



Controle de LCD

Instruction	Instruction Code										Description	Execution time (fosc=270KHz)
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0		
Clear Display	0	0	0	0	0	0	0	0	0	1	Write "20H" to DDRAM and set DDRAM address to "00H" from AC	1.52ms
Return Home	0	0	0	0	0	0	0	0	1	-	Set DDRAM address to "00H" from AC and return cursor to its original position if shifted. The contents of DDRAM are not changed.	1.52ms
Entry Mode Set	0	0	0	0	0	0	0	1	I/D	S	Assign cursor moving direction and enable the shift of entire display	38μs
Display ON/ OFF Control	0	0	0	0	0	0	1	D	C	B	Set display(D), cursor(C), and blinking of cursor(B) on/off control bit.	38μs
Cursor or Display Shift	0	0	0	0	0	1	S/C	R/L	-	-	Set cursor moving and display shift control bit, and the direction, without changing of DDRAM data.	38μs
Function Set	0	0	0	0	1	DL	N	F	-	-	Set interface data length (DL: 8-bit/4-bit), numbers of display line (N: 2-line/1-line) and, display font type (F:5x10 dots/5x8 dots)	38μs

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	20	21	22	23
40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F	50	51	52	53

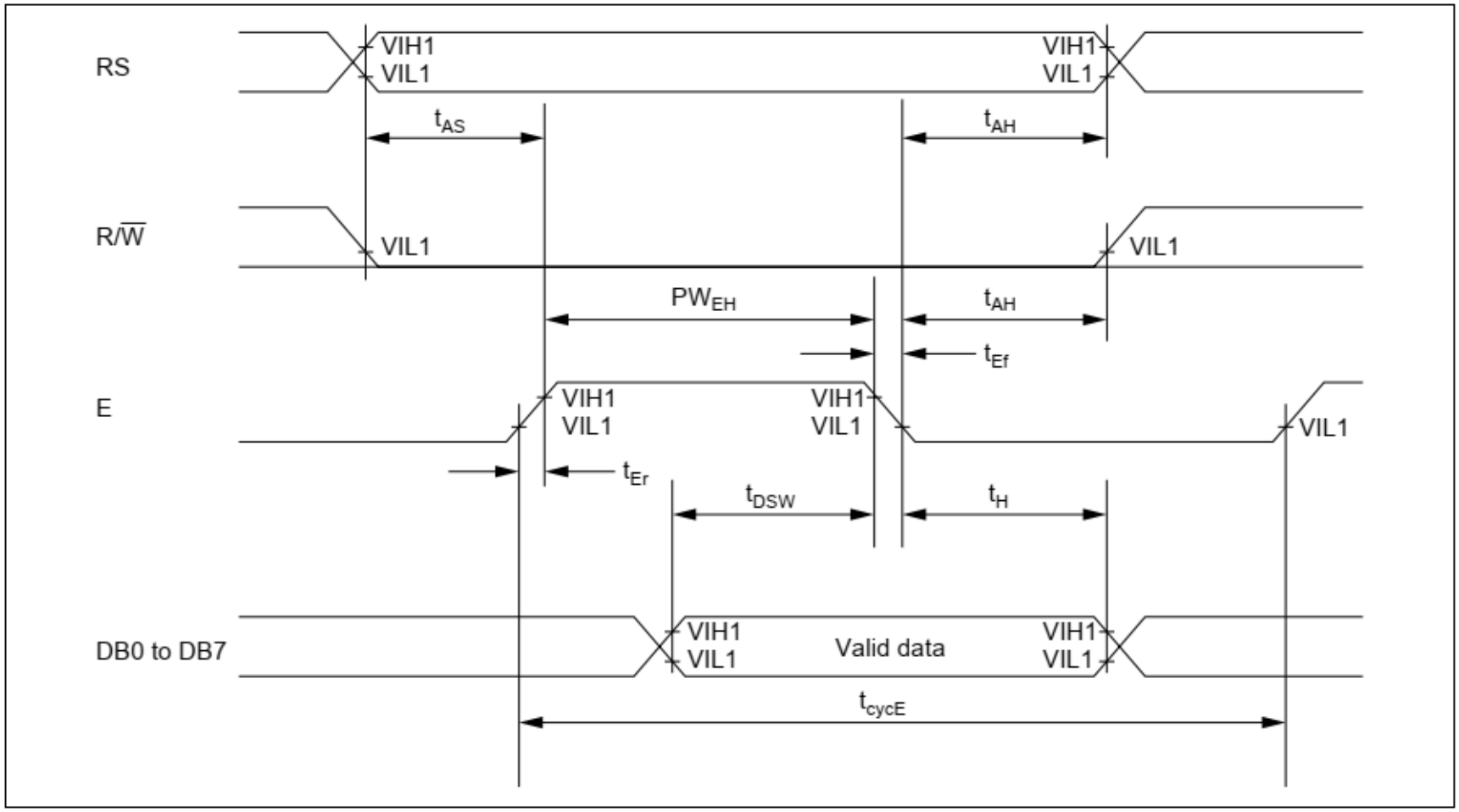
Controle de LCD

Set CGRAM Address	0	0	0	1	AC5	AC4	AC3	AC2	AC1	AC0	Set CGRAM address in address counter.	38μs
Set DDRAM Address	0	0	1	AC6	AC5	AC4	AC3	AC2	AC1	AC0	Set DDRAM address in counter	38μs
Read Busy Flag and Address Counter	0	1	BF	AC6	AC5	AC4	AC3	AC2	AC1	AC0	Whether during internal operation or not can be known by reading BF. The contents of address counter can also be read.	
Write Data to RAM	1	0	D7	D6	D5	D4	D3	D2	D1	D0	Write data into internal RAM (DDRAM/CGRAM).	38μs
Read Data from RAM	1	1	D7	D6	D5	D4	D3	D2	D1	D0	Read data from internal RAM (DDRAM/CGRAM).	38μs

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	20	21	22	23
40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F	50	51	52	53

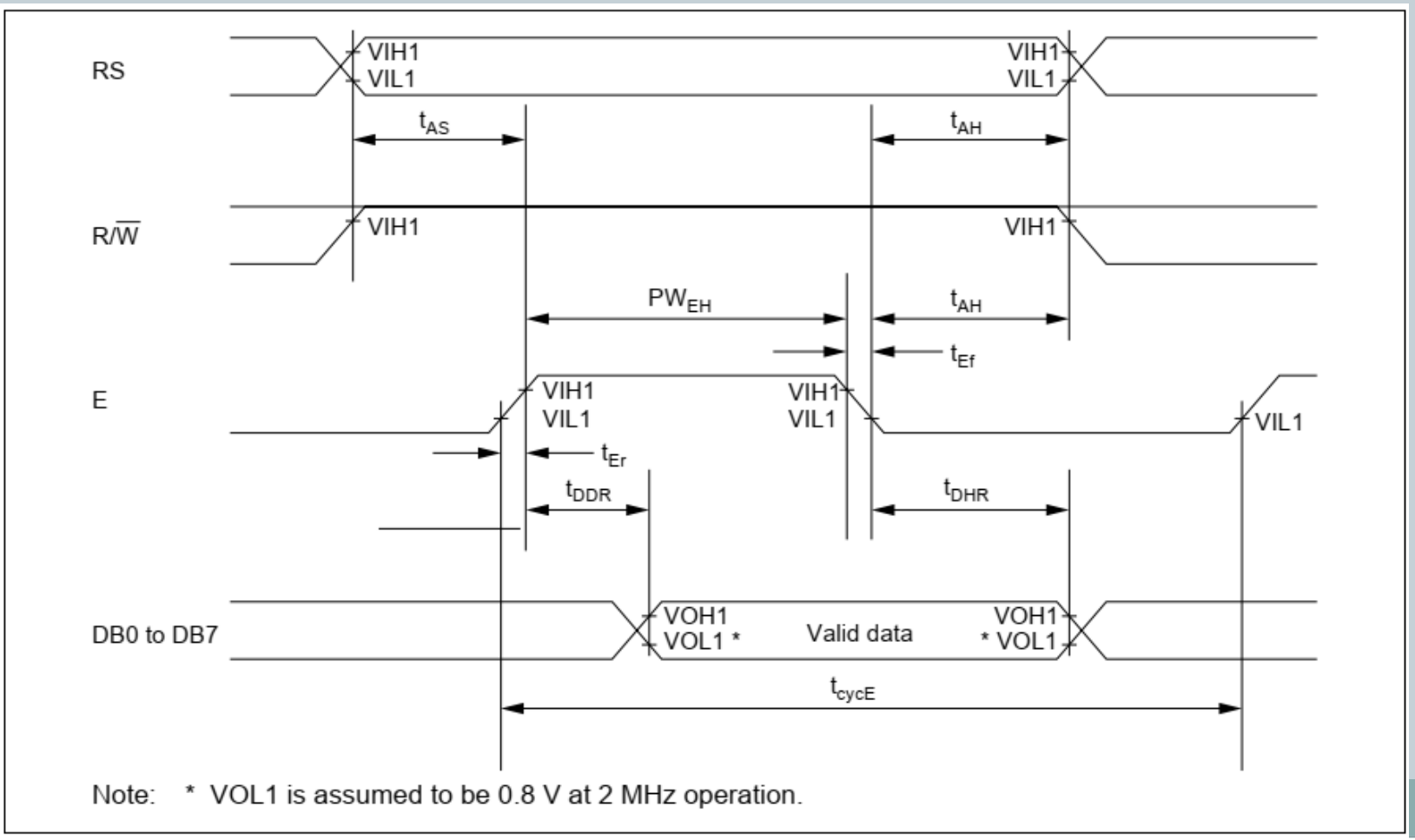
Controle de LCD

- Diagrama de Tempo de Escrita



Controle de LCD

- Diagrama de Tempo de Leitura



Controle de LCD

- Tempos para o Acionamento

Write Operation

Item	Symbol	Min	Typ	Max	Unit
Enable cycle time	t_{cycE}	500	—	—	ns
Enable pulse width (high level)	PW_{EH}	230	—	—	
Enable rise/fall time	t_{Er}, t_{Ef}	—	—	20	
Address set-up time (RS, R/$\overline{W}$ to E)	t_{AS}	40	—	—	
Address hold time	t_{AH}	10	—	—	
Data set-up time	t_{DSW}	80	—	—	
Data hold time	t_{H}	10	—	—	

Read Operation

Item	Symbol	Min	Typ	Max	Unit
Enable cycle time	t_{cycE}	500	—	—	ns
Enable pulse width (high level)	PW_{EH}	230	—	—	
Enable rise/fall time	t_{Er}, t_{Ef}	—	—	20	
Address set-up time (RS, R/$\overline{W}$ to E)	t_{AS}	40	—	—	
Address hold time	t_{AH}	10	—	—	
Data delay time	t_{DDR}	—	—	160	
Data hold time	t_{DHR}	5	—	—	

Controle de LCD

- Algoritmo para Inicialização do LCD

```
/*****************************************************************/
void init()
{
      E = 0;
      Delay(100);                    //Wait >15 msec after power is applied
      command(0x30);                 //command 0x30 = Wake up
      Delay(30);                     //must wait 5ms, busy flag not available
      command(0x30);                 //command 0x30 = Wake up #2
      Delay(10);                     //must wait 160us, busy flag not available
      command(0x30);                 //command 0x30 = Wake up #3
      Delay(10);                     //must wait 160us, busy flag not available
      command(0x38);                 //Function set: 8-bit/2-line
      command(0x10);                 //Set cursor
      command(0x0c);                 //Display ON; Cursor ON
      command(0x06);                 //Entry mode set
}
/*****************************************************************/
```

Controle de LCD

- Algoritmo para Envio de Mensagem para o LCD

```
8-bit Initialization:
/*************************************************************/
void command(char i)
{
      P1 = i;                          //put data on output Port
      D_I =0;                          //D/I=LOW : send instruction
      R_W =0;                          //R/W=LOW : Write
      E  = 1;
      Delay(1);                        //enable pulse width  >= 300ns
        E  = 0;                        //Clock enable: falling edge
}
/*************************************************************/
void write(char i)
{
      P1 = i;                          //put data on output Port
      D_I =1;                          //D/I=LOW : send data
      R_W =0;                          //R/W=LOW : Write
      E  = 1;
      Delay(1);                        //enable pulse width  >= 300ns
      E  = 0;                          //Clock enable: falling edge
}
```

Memória RAM

```
library ieee;
use ieee.std_logic_1164.all;
entity ram_dual is
  port
  (
       data    : in std_logic_vector(7 downto 0);
       raddr   : in natural range 0 to 63;
       waddr   : in natural range 0 to 63;
       we      : in std_logic := '1';
       rclk    : in std_logic;
       wclk    : in std_logic;
       q       : out std_logic_vector(7 downto 0)
  );
end ram_dual;
```

Memória RAM

```
architecture rtl of ram_dual is -- Declare the RAM signal.
  signal ram : memory_t;
Begin
  process(wclk)
  begin
       if(rising_edge(wclk)) then
       if(we = '1') then
          ram(waddr) <= data;
       end if;
       end if;
  end process;
  process(rclk)
   begin
      if(rising_edge(rclk)) then
        q <= ram(raddr);
      end if;
   end process;
end rtl;
```

Trabalho em Sala de Aula

- Criar grupo de até 5 pessoas
  - Projetar o circuito de um frequencímetro conforme o exposto anteriormente;
  - Criar o diagrama de em bloco de cada contador;
  - Criar o diagrama de bloco geral do Sistema;
  - Entregar as funções e desenho dos diagramas em bloco comentados.

  Obs.: O trabalho deve ser realizado em sala de aula e não há a necessidade de implementar no Quartus.

Exercício

- A partir das informações anteriores, sobre o Display de Cristal Líquido (LCD), fazer um conjunto de rotinas em VHDL para inicializar o LCD e enviar os dados para o display (trabalho III).

Exercício

- Fazer um projeto em HDL para realizar o interfaceamento dos dispositivos abaixo, para dispositivos de I/O utilizar o pino WR/RD para indentificar uma operação de leitura/escrita.
- Apresentar o diagrama em bloco do sistema e código em HDL

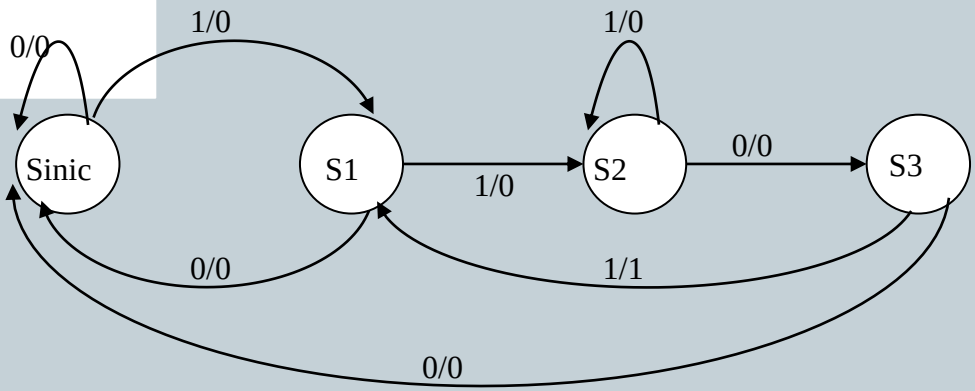
Dispositivo	Endereço	Tipo
Teclado	0x600	Entrada (8bits)
Display	0x3d0	Saida (8 bits)
Microcontrolador	0x360	Entrada e Saída (16 bits)
Priférico (placa A/D)	0x362	I/O (32 bits)

Máquina de Estados

- **Descrição de máquinas de estados usando VHDL**
- A função de transição de estado e a função de saída são descritas como processos separados.
- O primeiro processo descreve a **função de transição de estado** e é ativado sempre que há um evento no sinal de **clock**.
- O segundo processo descreve a **função de saída** e é ativado sempre que há uma transição de estado

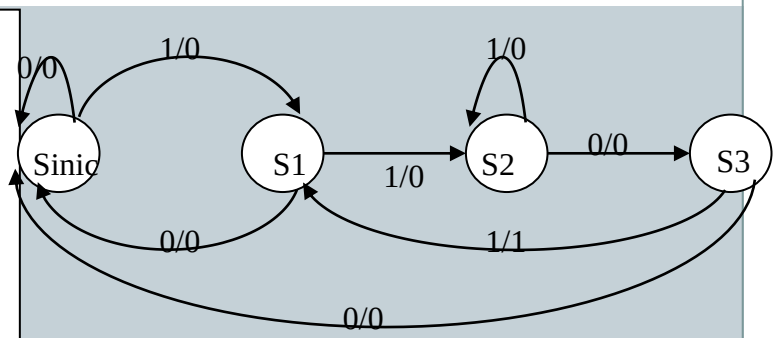
Descrição Em VHDL

```
library IEEE;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
ENTITY detector IS
 PORT
  ( x : IN STD_LOGIC;
    y : OUT STD_LOGIC;
    clk : IN STD_LOGIC
  );
 END detetor;
```



Descrição Em VHDL

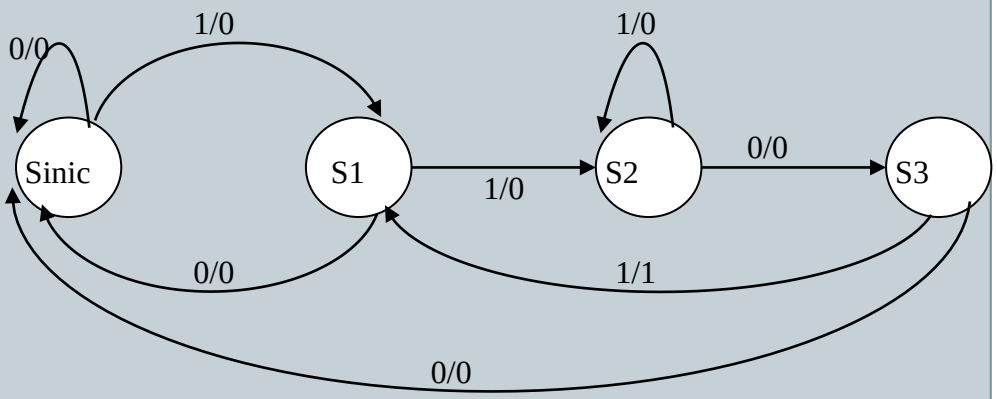
```
ARCHITECTURE comportamental OF detetor IS
  TYPE estados IS (Sinic, S1, S2, S3);
  SIGNAL estado_atual: estados;
  SIGNAL estado_anterior: estados;

BEGIN
   PROCESS (clk)
      BEGIN
         IF RISING_EDGE (clk) THEN
            CASE estado_atual IS
               WHEN Sinic =>
                  IF x = '0' THEN
                     estado_atual <= sinic;
                  ELSE
                     estado_atual <= S1;
                     estado_anterior <= Sinic;
                  END IF;
               WHEN S1 =>
                  IF x = '0' THEN
                     estado_atual <= Sinic;
                     estado_anterior <= S1;
                  ELSE
                     estado_atual <= S2;
                     estado_anterior <= S1;
                  END IF;
```



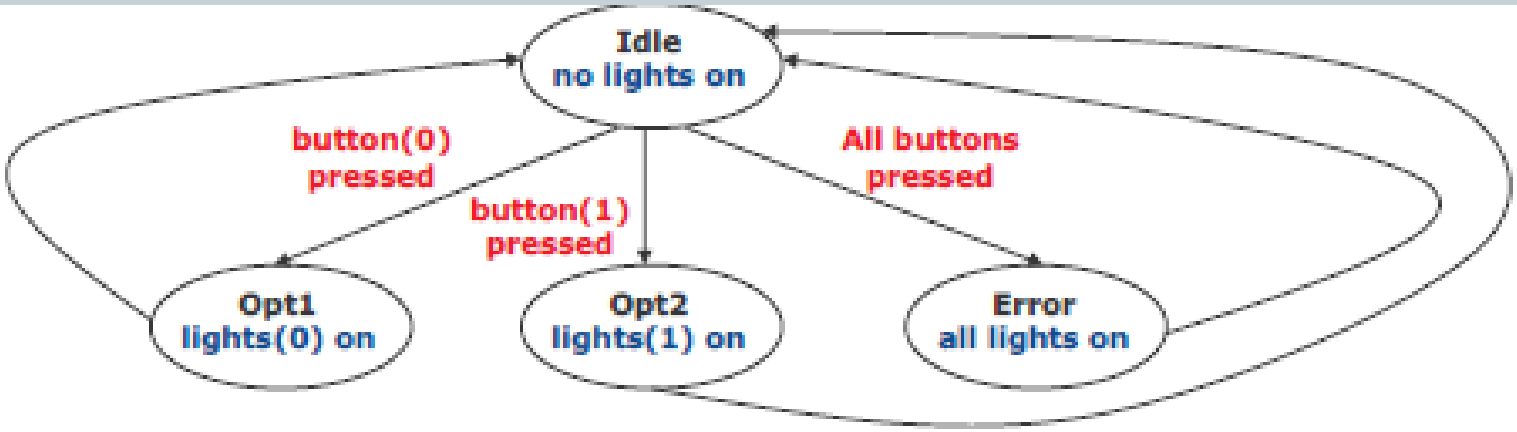

```
               WHEN S2 =>
                  IF x = '0' THEN
                     estado_atual <= S3;
                     estado_anterior <= S2
                  ELSE
                     estado_atual <= S2;
                     estado_anterior <= S2;
                  END IF;
               WHEN S3 =>
                  IF x = '0' THEN
                     estado_atual <= Sinic;
                     estado_anterior <= S3;
                  ELSE estado_atual <= S1;
                     estado_anterior <= S3;
                  END IF;
            END CASE;
         END IF;
   END PROCESS;
```

Descrição VHDL de máquina de Estado

```
PROCESS (estado_atual, estado_anterior)
BEGIN
   CASE estado_atual IS
      WHEN Sinic =>
         y <= '0';
      WHEN S1 =>
         IF estado_anterior = S3 THEN
            y <= '1';
         ELSE
            y <= '0';
         END IF;
      WHEN S2 =>
         y <= '0';
      WHEN S3  =>
         y <= '0';
   END CASE;
END PROCESS;
END comportamental
```



Máquina de Estados Finitos




```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY vending IS
    PORT(
          reset             : IN     std_logic;
          clock             : IN     std_logic;
          buttons           : IN     std_logic_vector(1 DOWNTO 0);
          lights            : OUT    std_logic_vector(1 DOWNTO 0)
    );
END vending;
```

Máquina de Estados Finitos

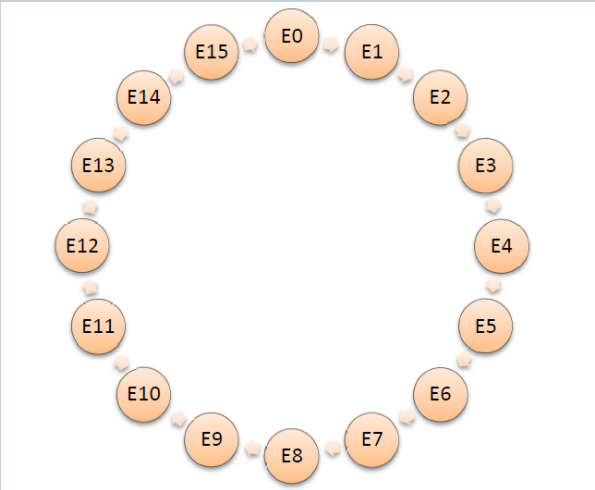
```
ARCHITECTURE synthesis1 OF vending IS
    TYPE            statetype IS (Idle, Opt1, Opt2, Error);
    SIGNAL          currentstate, nextstate   : statetype;
BEGIN
    fsm1: PROCESS( buttons, currentstate )
    BEGIN
          CASE currentstate IS
                    WHEN Idle =>
                              lights <= "00";
                              CASE buttons IS
                                        WHEN "00" =>
                                                  nextstate <= Idle;
                                        WHEN "01" =>
                                                  nextstate <= Opt1;
                                        WHEN "10" =>
                                                  nextstate <= Opt2;
                                        WHEN OTHERS =>
                                                  nextstate <= Error;
                              END CASE;
                    WHEN Opt1 =>
                              lights <= "01";
                              IF buttons /= "01" THEN
                                        nextstate <= Idle;
                              END IF;
```

Máquina de Estados Finitos

```
                WHEN Opt2 =>
                        lights <= "10";
                        IF buttons /= "10" THEN
                                nextstate <= Idle;
                        END IF;
                WHEN Error =>
                        lights <= "11";
                        IF buttons = "00" THEN
                                nextstate <= Idle;
                        END IF;
        END CASE;
    END PROCESS;

    fsm2: PROCESS( reset, clock )
    BEGIN
        IF (reset = '0') THEN
                currentstate <= Idle;
        ELSIF (clock'EVENT) AND (clock = '1') THEN
                currentstate <= nextstate;
        END IF;
    END PROCESS;
END synthesis1;
```

Contador com Máquinas de Estados

- Contador de 4 bits (modulo 16) - contador binário com uma máquina de estados, usando três processos: o primeiro processo verifica o sinal reset e muda o estado na borda de subida do clock. O segundo processo identifica o próximo estado e o terceiro especifica a saída do circuito para cada estado.



Contador com Máquina de estados

```
-- contador binario com ME (máquina de estados)
library ieee;
use ieee.std_logic_1164.all;

entity contbinME is
  port (clock, reset, enable : in std_logic;
        saida : out std_logic_vector (3 downto 0));
end contbinME;

architecture contbinME_arch of contbinME is
  type tipo_estado is (E0, E1, E2, E3, E4, E5, E6, E7,
    E8, E9, E10, E11, E12, E13, E14, E15);
  signal estado, prox_estado: tipo_estado;

begin
  -- processo mudaestado
  mudaestado: process (clock, reset)
  begin
    if reset = '1' then
      estado <= E0;
    elsif clock'event and clock='1' then
      if enable = '1' then
        estado <= prox_estado;
      end if;
    end if;
  end process;
```

```
transicao: process (estado, enable)
begin
  if enable = '1' then
    case estado is
      when E0  => prox_estado <= E1;
      when E1  => prox_estado <= E2;
      when E2  => prox_estado <= E3;
      when E3  => prox_estado <= E4;
      when E4  => prox_estado <= E5;
      when E5  => prox_estado <= E6;
      when E6  => prox_estado <= E7;
      when E7  => prox_estado <= E8;
      when E8  => prox_estado <= E9;
      when E9  => prox_estado <= E10;
      when E10 => prox_estado <= E11;
      when E11 => prox_estado <= E12;
      when E12 => prox_estado <= E13;
      when E13 => prox_estado <= E14;
      when E14 => prox_estado <= E15;
      when E15 => prox_estado <= E0;
    end case;
  end if;
end process;
```

Contador com Máquina de estados

```
  saidas: process (estado)
  begin
    case estado is
      when E0  => saida <= "0000";
      when E1  => saida <= "0001";
      when E2  => saida <= "0010";
      when E3  => saida <= "0011";
      when E4  => saida <= "0100";
      when E5  => saida <= "0101";
      when E6  => saida <= "0110";
      when E7  => saida <= "0111";
      when E8  => saida <= "1000";
      when E9  => saida <= "1001";
      when E10 => saida <= "1010";
      when E11 => saida <= "1011";
      when E12 => saida <= "1100";
      when E13 => saida <= "1101";
      when E14 => saida <= "1110";
      when E15 => saida <= "1111";
    end case;
  end process;
end contbinME_arch;
```

Funções

- Função é uma seção de código que tem o propósito de criar novos processos
- Esses novos processos são normalmente conversão de tipos, operações lógicas, operadores aritméticos, etc.

```
FUNCTION function_name [<parameter list>]
  RETURN data_type IS [declarations]
BEGIN
(sequential statements)
END function_name;
```

Exemplo de Função

```
FUNCTION positive_edge(SIGNAL s: STD_LOGIC)
RETURN BOOLEAN IS
BEGIN
RETURN (s'EVENT AND s='1');
END positive_edge;
------ Chamada de Função -------------
...
IF positive_edge(clk) THEN...
```

Exemplo de Função

```
------ Corpo da Função: -------------------------------
FUNCTION conv_integer (SIGNAL vector: STD_LOGIC_VECTOR)
   RETURN INTEGER IS
VARIABLE result: INTEGER RANGE 0 TO 2**vector'LENGTH-1;
BEGIN
  IF (vector(vector'HIGH)='1') THEN
     result:=1;
  ELSE
     result:=0;
  END IF;
  FOR i IN (vector'HIGH-1) DOWNTO (vector'LOW) LOOP
     result:=result*2;
     IF(vector(i)='1') THEN
        result:=result+1;
```

Exemplo de Função

```
END IF;
   END LOOP;
   RETURN result;
END conv_integer;

------ Chamada da Função----------------------------
y <= conv_integer(a);
```

Função no Programa Principal

```
----------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
----------------------------------------
ENTITY dff IS
PORT (d, clk, rst: IN STD_LOGIC;
      q: OUT STD_LOGIC);
END dff;
---------------
ARCHITECTURE my_arch OF dff IS
----------------------------------------
FUNCTION positive_edge(SIGNAL s: STD_LOGIC)
 RETURN BOOLEAN IS
 BEGIN
 RETURN s'EVENT AND s='1';
 END positive_edge;
----------------------------------------
```

```
BEGIN
 PROCESS (clk, rst)
 BEGIN
 IF (rst='1') THEN q <= '0';
ELSIF positive_edge(clk) THEN q <= d;
 END IF;
END PROCESS;
 END my_arch;
-----------------------------------
```

Função em um pacote

```
------- Package: -------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
----------------------------------------
PACKAGE my_package IS
FUNCTION positive_edge(SIGNAL s:
 STD_LOGIC)
 RETURN BOOLEAN;
END my_package;
--------------------------------
```

```
PACKAGE BODY my_package IS
FUNCTION positive_edge(SIGNAL s: STD_LOGIC)
RETURN BOOLEAN IS
BEGIN
      RETURN s'EVENT AND s='1';
END positive_edge;
END my_package;
```

Função em um pacote

```
------ Main code: ----------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE work.my_package.all;
----------------------------
ENTITY dff IS
PORT ( d, clk, rst: IN
  STD_LOGIC;
q: OUT STD_LOGIC);
END dff;
----------------------------
```

```
ARCHITECTURE my_arch OF dff IS
 BEGIN
 PROCESS (clk, rst)
 BEGIN
 IF (rst='1') THEN q <= '0';
 ELSIF positive_edge(clk) THEN q <= d;
 END IF;
 END PROCESS;
 END my_arch;
```

Procedimentos em VHDL

- Um procedimento (procedure) em VHDL é um conjunto de operações que executa uma determinada ação
- Procedimentos podem ser chamados de qualquer parte do programa VHDL, de forma similar a chamada de funções
- Um procedimento escrito em VHDL também pode ser chamado em varias outras linguagens
- Para a chamada de um procedimento em VHDL, basta escrever-se o nome do procedimento seguido de ponto e virgula (;)

Procedimentos em VHDL

```
procedure contagem is
begin
  if(contador<=100)then
  contador <= contador +1;
else
  contador<=0;
end if;
end contagem;
```

```
...
contagem;

{ chamada do procedimento de atraso}
...
```

Implementa um procedimento de contagem ate 100. Note que contador deve ser uma variável global (**SIGNAL**).

Procedimentos em VHDL

- Procedimentos com Parâmetros em VHDL
  - Para ser mais completa, a utilização de procedimentos em VHDL permite a passagem de parâmetros quando for realizada a sua chamada.
  - Para tanto deve-se inicialmente prever na declaração do procedimento os parâmetros a serem enviados, descrevendo-se o mesmo entre parênteses.
  - Na sua descrição, o parâmetro é composto por um identificador, um modo (**in**, **out** ou **inout**) e o tipo.
  - A palavra-chave **func_code** é utilizada quando deseja-se que o tipo do parâmetro seja dado pelo programa que chama o procedimento.

Procedimentos em VHDL

```
procedure envia_serial (dado : in func_code) is
variable indice : dado_serial := 0;
variable teste : positive := 0;
variable temp : bit := start_bit ;
begin
   if (clock='1') then
      if teste >1 and teste<(tamanho_dado_serial + 1) then
          teste := teste +1;
          if indice <= tamanho_dado_serial then
            temp := dado(indice);
            indice := indice +1;
          end if;
      end if;
   end if;
```

```
if teste = (tamanho_dado_serial + 1) then
   temp := stop_bit;
   dout <= temp;
      return;
end if

dout <= temp;
end procedure envia_serial;

Ex: de Chamada da procedure:
dado := 34;
Envia_serial(dado);
...
```

Procedimentos em VHDL

- Procedimentos com Parâmetros de Retorno em VHDL
  - Para ilustrar que procedimentos também podem retornar valores, a seguir apresenta-se um exemplo de um algoritmo para conversão de valor decimal para BCD:

Procedimentos em VHDL

```
procedure converte_para_BCD (numero: in integer range 0 to 39;
  mais_sig,  menos_sig : out integer range 0 to 9) is
begin
  if numero > 39 then return;
  elsif numero >= 30 then
       mais_sig := 3;
       menos_sig := numero - 30;
  elsif numero >= 20 then
       mais_sig := 2;
       menos_sig := numero - 20;
  elsif numero >= 10 then
       mais_sig := 1;
       menos_sig := numero - 10;
       else mais_sig := 0;
       menos_sig := numero;
       end if;
end procedure converte_para_BCD;

Ex: de Chamada da procedure:
converte para BCD (dado lido,
dezena, unidade);
```

Funções em VHDL

Características	Função	Procedimento
retorno de valor	retorna um valor através do comando `return`	pode retornar vários valores através dos parâmetros passados ao procedimento
parâmetros	lista de parâmetros deve usar modo `in` obrigatoriamente	parâmetros podem possuir modos `in`, `inout` ou `out` (valores de retorno usam modo `out` ou `inout`)
comandos	contém comandos sequenciais	
conceito	generalização de uma expressão (pode ser usado dentro de um comando)	generalização de um comando (pode ser considerado como um bloco de comandos)
sobrecarga	é possível criar vários subprogramas com mesmo nome, mas com lista de parâmetros e valores de retorno diferentes	

Comando WHILE-LOOP

- A estrutura while-loop é um caso especial da estrutura de loop, onde uma condição é testada inicialmente antes de qualquer operação do loop e caso seja verdadeira, o loop é executado. Se a condição não for satisfeita, o loop para de ser executado.
- Exemplo:

```
numero := valor_entrada;
raiz_quadrada := 0;
while raiz_quadrada*raiz_quadrada < numero loop
raiz_quadrada := raiz_quadrada + 0.1;
end loop;
```

Comando FOR-LOOP

- A estrutura for-loop é utilizada principalmente para execução de operações a serem repetidas por um numero determinado de vezes.
- A sequência abaixo descreve o cálculo do fatorial do numero 10.

```
resultado := 1;
for n in 1 to 10 loop
      resultado := resultado *n;
end loop;
```

Comando FOR-LOOP

- A estrutura for-loop permite também a realização de iterações em ordem decrescente. Neste caso basta utilizar-se a palavra-chave downto no lugar de to:

```
contador := 5;
resultado := 0;
for contador in 10 downto 1 loop
  resultado := contador;
end loop;
```

Comando EXIT

- O comando EXIT serve para forçar a saída de um laço conforme as seguintes formas:

```
exit; => utilizado para terminar um while, for;

exit <label loop>; => termina o laço e desloca para
o label;

exit <label loop> when <condição>; => termina o
laço quando (when) condição é satisfeita;
```

Comando EXIT

- Estrutura Loop com Saída Forçada por Exit:
  - Um estrutura LOOP pode entretanto precisar de uma condição de saída. A palavra-chave exit, força a saída do loop mais interno quando uma determinada condição for atendida.

Por exemplo:

```
loop
  loop
       exit when reset_contador = '1';
       contador := contador +1;
       saida := contador;
  end loop;
  contador := 0;
end loop;
```

Comando EXIT

- Vários Loops, Loops encadeados:

```
loop_externo : loop
  contador := 0;
  loop_intermediario : loop
       loop_interno : loop
              saida := contador;
              exit when contagem_habilitada = '0';
              exit loop_intermediario when reset_contador ='1';
       end loop loop_interno;
       contador := contador +1;
  end loop loop_intermediario;
end loop loop_externo;
```

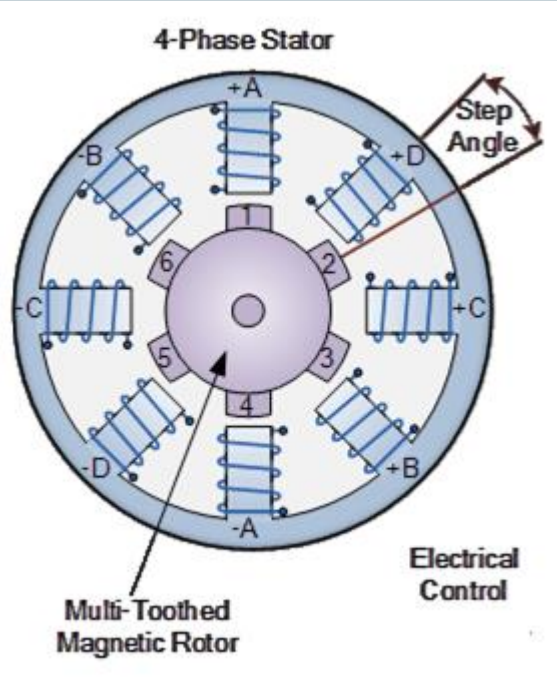
Comando NEXT

- O comando NEXT é utilizado para terminar prematuramente a execução de uma iteração do tipo while, for ou um loop infinito.
- Esta estrutura é similar ao caso de saída por EXIT, só que a palavra-chave NEXT serve para "pular" uma sequência de operações do loop.
- No exemplo abaixo, a condição verdadeira do NEXT faz com que a execução dos comandos logo abaixo não sejam executados, mas o loop continua:

```
loop
    medida_temperatura = entrada_medidor;
    next when medida_temperatura < temperatura_maxima;
    aquecedor := OFF;
    alarme := ON;
end loop;
```

Exercício

- Projetar um circuito em VHDL controlador de motor de passo com 8 enrolamentos, com seleção de passo completo, meio passo, sentido (horário e anti-horário), reset, enable e clock. conforme as tabelas do próximo slide.



Exercício

Passo completo

Nº	B7	B6	B5	B4	B3	B2	B1	B0	D
1	1	0	0	0	0	0	0	0	128
2	0	1	0	0	0	0	0	0	64
3	0	0	1	0	0	0	0	0	32
4	0	0	0	1	0	0	0	0	16
5	0	0	0	0	1	0	0	0	8
6	0	0	0	0	0	1	0	0	4
7	0	0	0	0	0	0	1	0	2
8	0	0	0	0	0	0	0	1	1

Meio Passo

Nº	B7	B6	B5	B4	B3	B2	B1	B0	D
1	1	0	0	0	0	0	0	0	128
2	1	1	0	0	0	0	0	0	192
3	0	1	0	0	0	0	0	0	64
4	0	1	1	0	0	0	0	0	96
5	0	0	1	0	0	0	0	0	32
6	0	0	1	1	0	0	0	0	48
7	0	0	0	1	0	0	0	0	16
8	0	0	0	1	1	0	0	0	24
9	0	0	0	0	1	0	0	0	8
10	0	0	0	0	1	1	0	0	12
11	0	0	0	0	0	1	0	0	4
12	0	0	0	0	0	1	1	0	6
13	0	0	0	0	0	0	1	0	2
14	0	0	0	0	0	0	1	1	3
15	0	0	0	0	0	0	0	1	1
16	1	0	0	0	0	0	0	1	129