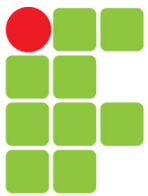
VHDL: Modelagem RTL

Engenharia Eletrônica

Prof. Renan Augusto Starke

Instituto Federal de Santa Catarina – IFSC
Campus Florianópolis
renan.starke@ifsc.edu.br

30 de março de 2020



INSTITUTO FEDERAL
SANTA CATARINA

Ministério da Educação
Secretaria de Educação Profissional e Tecnológica
INSTITUTO FEDERAL DE SANTA CATARINA

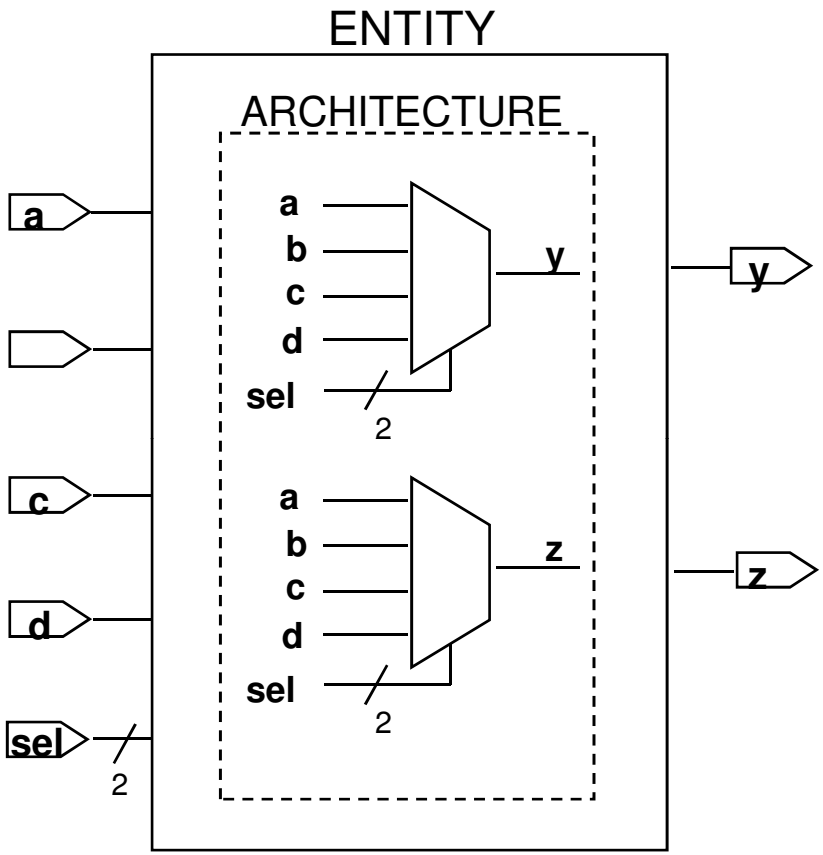
Tópico

Objetivos

- Tópicos da aula de hoje:
  - Recapitulação modelagem RTL
  - Processos síncronos
  - Latches
  - Registradores
  - Máquinas de estado

Modelagem RTL

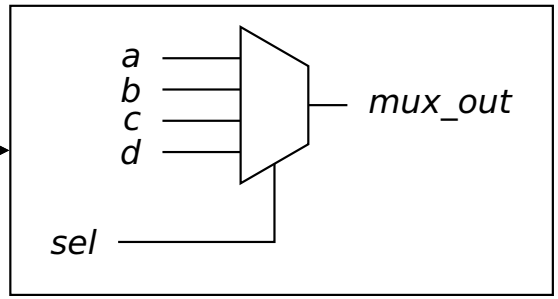
Modelagem comportamental que implica ou infere hardware. Descreve-se a funcionalidade e implica-se na estrutura do circuito.



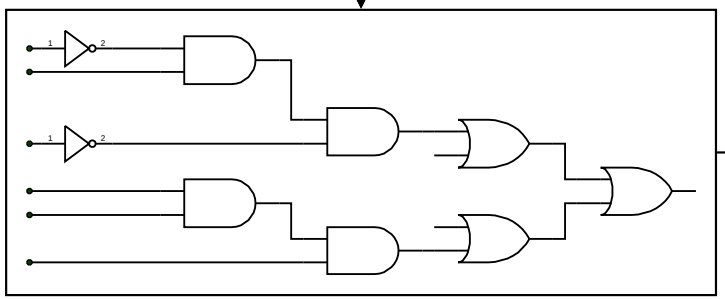
Síntese RTL

```
process (a,b,c,d,sel)
begin
  case (sel) is
    when "00" => mux_out <= a;
    when "01" => mux_out <= b;
    when "10" => mux_out <= c;
    when "11" => mux_out <= d;
  end case
end
```

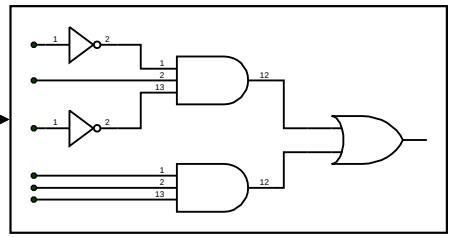
inferido



tradução



otimiz.

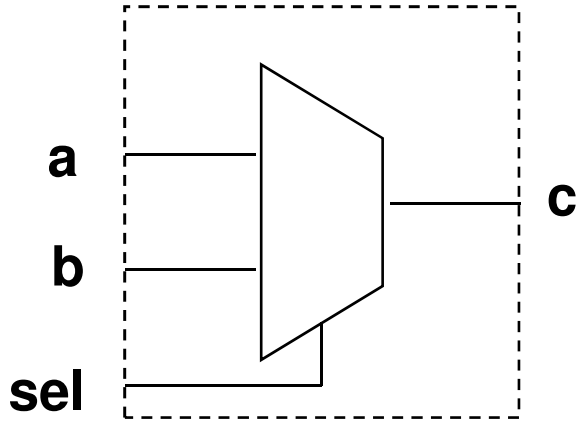


Tópico

Tipos de processos

- **Processos combinacionais**
  - Lista de sensibilidade apresenta todas as entradas
- Exemplo:

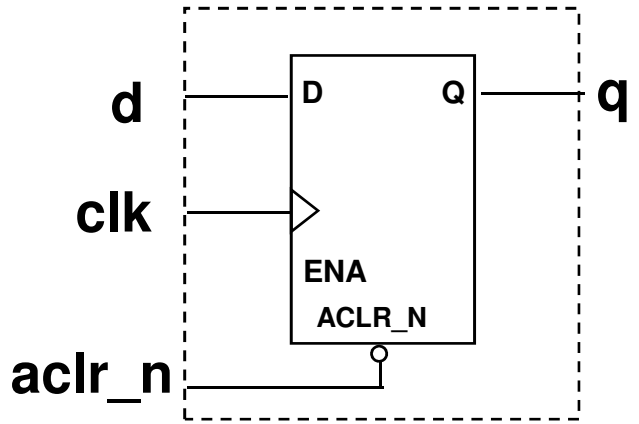
```
process (a, b, sel)
```



- **Processos sequenciais**
  - Sensíveis a um clock e sinais assíncronos de controle
- Exemplo:

```
process (aclr_n, clk)
```

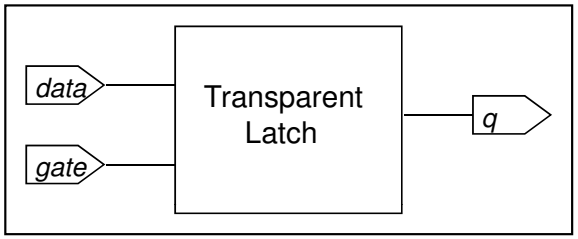
  - A lista de sensibilidade não inclui a entrada “d”



Latch

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

entity latch1 is
  port (
    data : in std_logic;
    gate : in std_logic;
    q : out std_logic
  );
end entity latch1;

architecture logic of latch1 is
begin
  label_1: process (data, gate)
  begin
    if gate = '1' then
      q <= data;
    end if;
  end process label_1;
end architecture behavior;
```

inferido



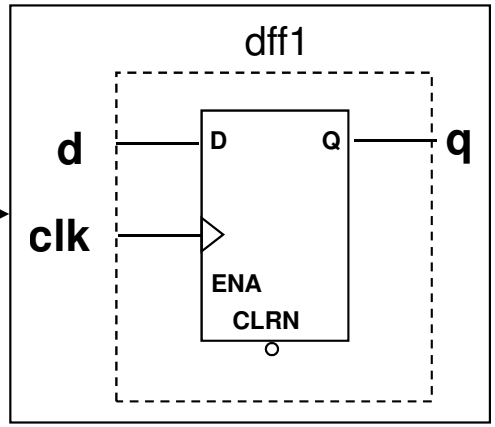
Lista de sensibilidade inclui ambas entradas

O que acontece quando gate = '0'? **Memória**

Flip FLop D

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

entity dff1 is
  port (
    d : in std_logic;
    clk : in std_logic;
    q : out std_logic
  );
end entity dff1;

architecture logic of dff1 is
begin
  process (clk)
  begin
    if clk'event and clk = '1' then
      q <= d;
    end if;
  end process;
end architecture behavior;
```

inferido

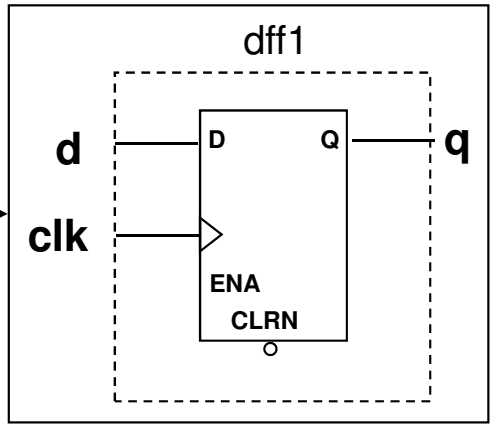


- **clk** é o nome do sinal
- **'EVENT** é um atributo VHDL (mudança do estado)
- **clk=1** significa borda de subida

Flip FLop D

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

entity dff1 is
  port (
    d : in std_logic;
    clk : in std_logic;
    q : out std_logic
  );
end entity dff1;

architecture logic of dff1 is
begin
  process (clk)
  begin
    if rising_edge(clk) then
      q <= d;
    end if;
  end process;
end architecture logic;
```

inferido



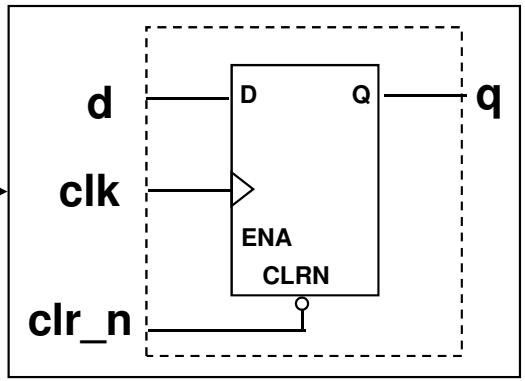
rising_edge():

- Função IEEE definida em no pacote STD_LOGIC_1164
- Especifica que o sinal **deve** variar de 0 a 1
- X, Z, ... para 1 não é considerado

Flip FLop D – clear assíncrono

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

entity dff_clrn is
  port (
    d : in std_logic;
    clr_n : in std_logic;
    clk : in std_logic;
    q : out std_logic
  );
end entity dff_clrn;

architecture logic of dff_clrn is
begin
  process (clk, clr_n)
  begin
    if clr_n = '0' then
      q <= '0';
    elsif rising_edge(clk) then
      q <= d;
    end if;
  end process;
end architecture logic;
```

inferido

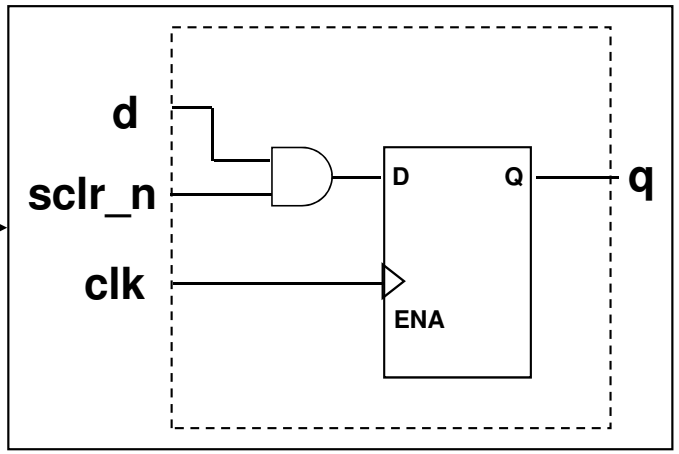


- Implementação correta de sinais de controle assíncronos
- Sinal assíncrono está na lista de sensibilidade e é verificado antes do **clock**
- **clrn_n='0'** não depende do clock

Flip FLop D – clear síncrono

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

entity dff_sclrn is
  port (
    sclr_n : in std_logic;
    d, clk : in std_logic;
    q : out std_logic
  );
end entity dff_sclrn;

architecture logic of dff_sclrn is
begin
  process (clk)
  begin
    if rising_edge (clk) then
      if sclr_n = '0' then
        q <= '0';
      else
        q <= d;
      end if;
    end if;
  end process;
end architecture logic;
```

inferido

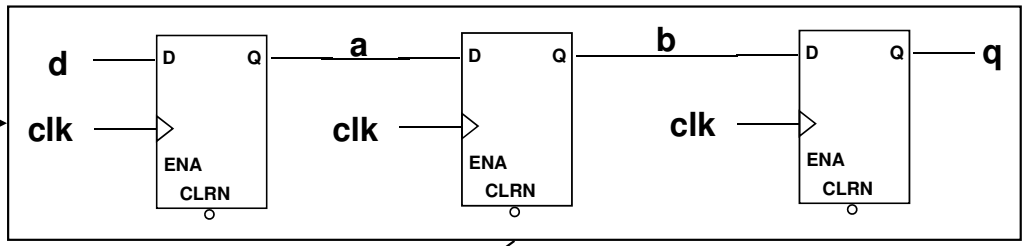


- Implementação correta de sinais de controle síncronos
- Sinais síncronos não são incluídos na lista de sensibilidade e são verificados conforme a borda do **clock**
- **clrn_n='0'** depende do clock

Número de registradores

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

entity reg1 is
  port (
    d : in std_logic;
    clk : in std_logic;
    q : out std_logic
  );
end entity reg1;

architecture logic of reg1 is
  signal a, b : std_logic;
begin
  process (clk)
  begin
    if rising_edge (clk) then
      a <= d;
      b <= a;
      q <= b;
    end if;
  end process;
end architecture logic;
```

inferido

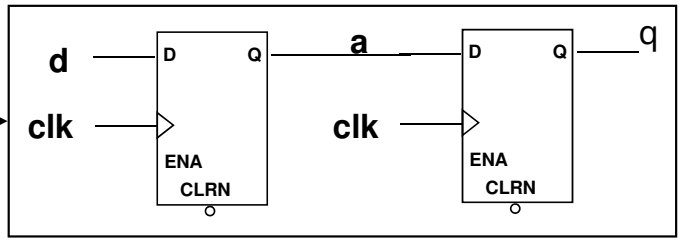


Atribuição de sinais dentro de um bloco if-then síncrono infere registradores

Número de registradores

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

entity reg2 is
  port (
    d : in std_logic;
    clk : in std_logic;
    q : out std_logic
);
end entity reg2;

architecture logic of reg2 is
  signal a, b : std_logic;
begin
  process (clk)
  begin
    if rising_edge (clk) then
      a <= d;
      b <= a;
    end if;
  end process;
  q <= b;
end architecture logic;
```

inferido

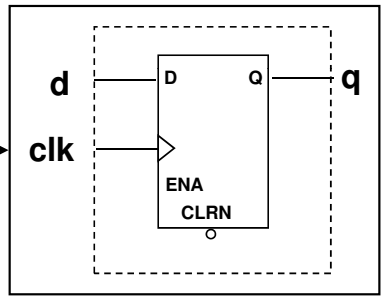


Atribuição de b para q não é mais sensível ao clock: **um registrador a menos**

Número de registradores

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

entity reg3 is
  port (
  d : in std_logic;
  clk : in std_logic;
  q : out std_logic
  );
end entity reg3;

architecture logic of reg3 is
begin
  process (clk)
    variable a, b : std_logic;
  begin
    if rising_edge (clk) then
      a := d;
      b := a;
      q <= b;
    end if;
  end process;
end architecture logic;
```

inferido



Variáveis são atribuídas imediatamente: **apenas um registrador**

Atribuição de variáveis em lógica sequencial

- Variáveis são armazenamentos temporários que normalmente não geram hardware

- Atribuição de variáveis podem ser utilizadas em expressões para atualizar um valor imediatamente
  - Uma variável pode ser atribuída a um sinal para inferir hardware

Exemplo: contador

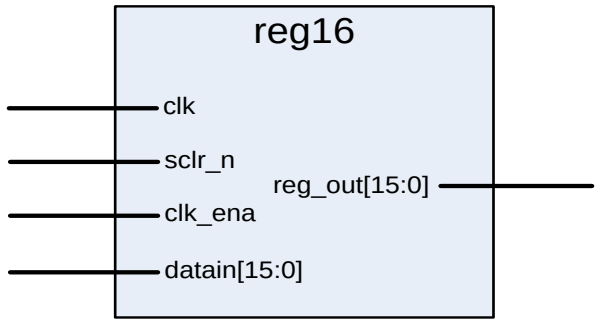
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity counter is
  port (
    clk, aclr, clk_ena : in std_logic;
    q : out std_logic_vector (15 dowto 0);
  );
end entity counter;

architecture logic of counter is
begin
  process (clk, aclr)
    variable q_var : std_logic_vector(15 dowto 0);
  begin
    if aclr = '1' then
      q_var := (others => '0');
    elsif rising_edge (clk) then
      if clk_ena = '1' then
         q_var := q_var + 1;
      end if;
    end if;
    q <= q_var;
  end process;
end architecture logic;
```

Expressão aritmética atribuída à variável antes da escrita do valor conhecido

Variável atribuída à um sinal para geração de hardware

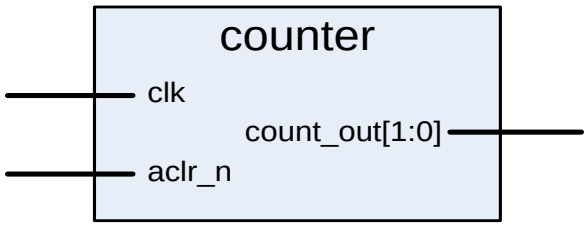
Exercício

- Escreva o código para um registrador de n-bits (GENERICS).
  - Todas as operações ocorrem na subida de clk, com exceção de **sclr_n**.
  - Se **sclr_n** é baixo, limpe a saída (prioridade mais alta).
  - Na subida do clock, verifique se **clk_ena** é alto.
  - Se **clk_ena** é alto, as saídas são atribuídas às entradas
  - Se **clk_ena** é baixo, nada é feito.
  - Simule.



Exercício

- Escreva o código para um contador de 2 bits com controle assíncrono
  - A saída vai para “00” **imediatamente** quando **aclr_n** é baixo
  - Se **aclr_n** não é baixo, o contador é incrementado em 1 na subida do clock
  - Use uma variável
  - **aclr_n** tem prioridade sobre a verificação da subida do clk
- Use pacotes **STD_LOGIC_1164** e **STD_LOGIC_UNSIGNED**
- Simule e sintetize. Verifique o hardware RTL sintetizado na ferramenta de síntese

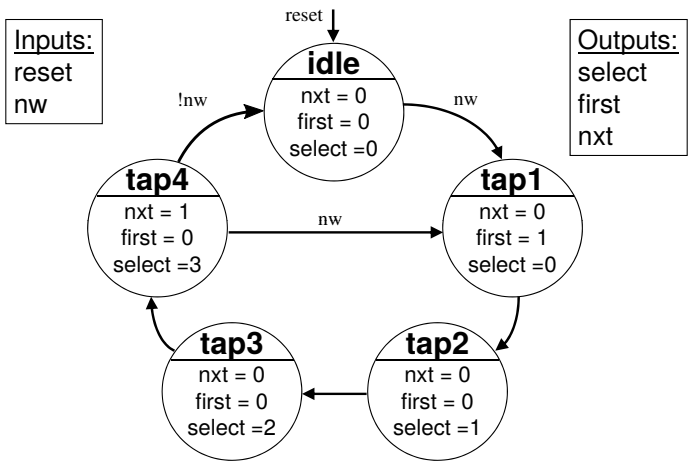


Tópico

Máquinas de estados

É um circuito sequencial que transita em uma sequência conhecida de estados.

- A transição de entre estados é comandada por um sinal de controle.
- O estado atual pode ser definido por registradores.
- Estados futuros são determinados com base no estado atual e pela situação das entradas.



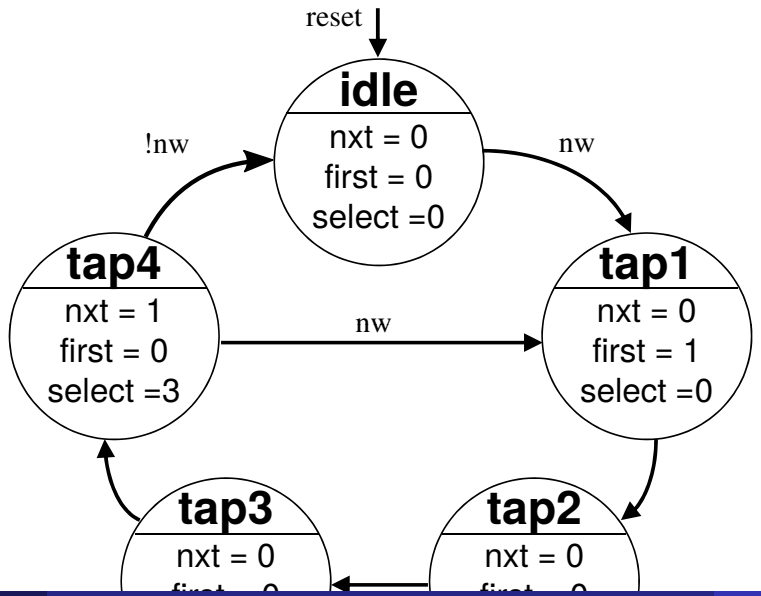
Máquinas de estados em VHDL

- Estados de uma máquina de estados **devem** ser declarados um tipo enumerados.

```
TYPE state_type IS (IDLE, TAP1, TAP2, TAP3, TAP4 );
```

- Após a declaração, um sinal deve ser instanciado utilizando o tipo criado:

```
SIGNAL filter : state_type;
```



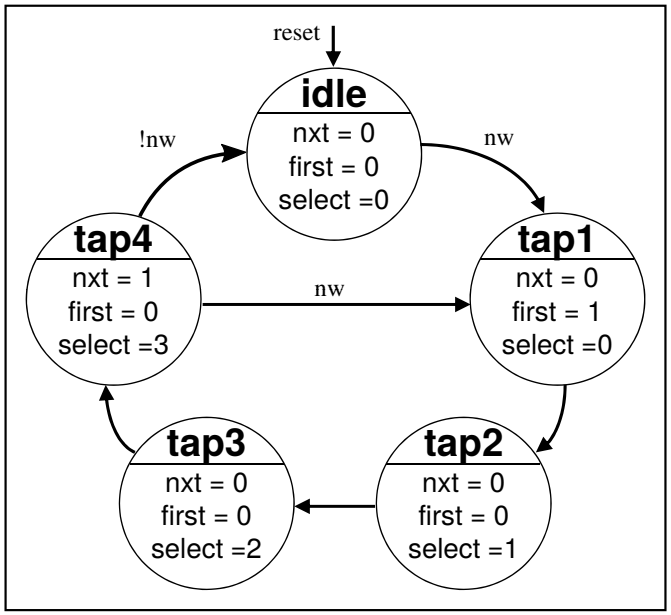
Máquinas de estados em VHDL

- Lógica de transição dos estados:
  - Construção **CASE** dentro de um processo sequencial (clk)

- Há dois métodos para determinar as saídas de uma máquina:
  - Processo combinacional com **CASE**
  - Atribuição selecional e/ou atribuição condicional para cada saída

Máquinas de estados em VHDL

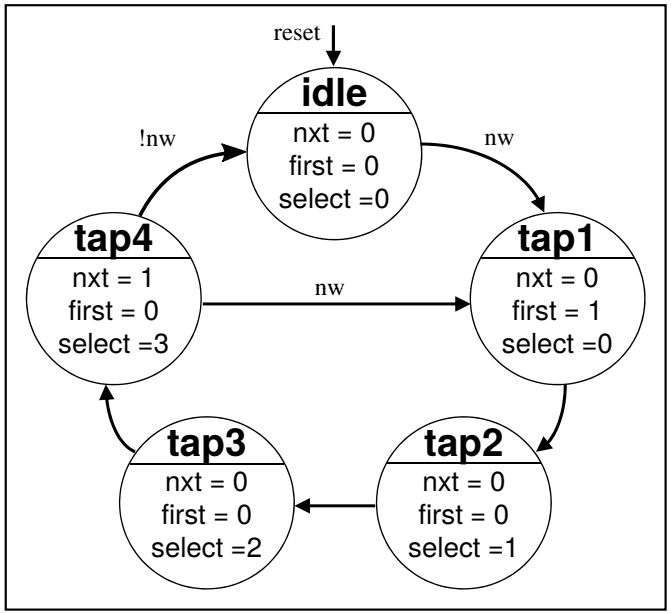
```
library ieee;
use ieee. std_logic_1164.all;

entity filter_sm is
  port (
    clk, reset, nw : in std_logic;
    select_ : out std_logic_vector
      (1 downto 0);
    nxt, first : out std_logic
);
end entity filter_sm;

architecture logic of filter_sm is

  type state_type is (IDLE, TAP1, TAP2,
    TAP3, TAP4);
  signal filter : state_type;

begin
  (...)
```



- Declaração de tipo
- Instância do sinal com o novo tipo

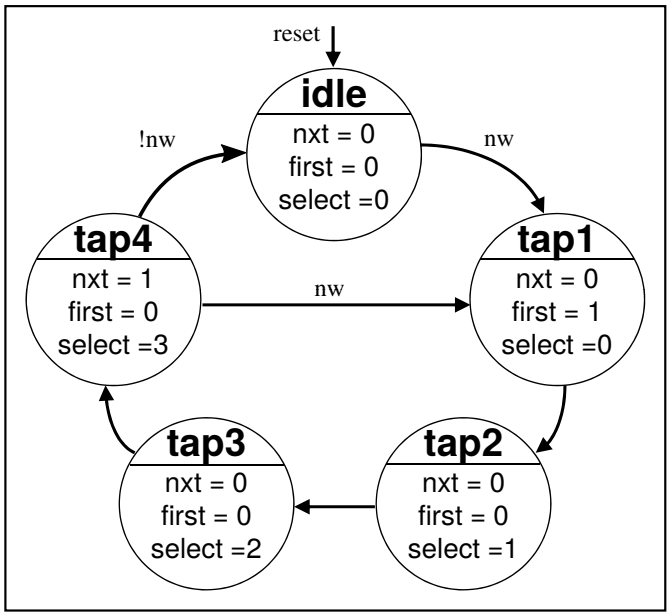
Máquinas de estados: transição dos estados

```
process (reset, clk)
begin
  if reset = '1' then
    filter <= idle;
  elsif clk' event and clk = '1' then
    case filter is
      when idle =>
        if nw = '1' then
          filter <= TAP1;
        end if;
      when TAP1 =>
         filter <= TAP2;
      when TAP2 =>
         filter <= TAP3;
      when TAP3 =>
         filter <= TAP4;
      when TAP4 =>
         if nw = '1' then
            filter <= TAP1;
         else
            filter <= idle;
         end if;
    end case;
  end if;
end process;
```



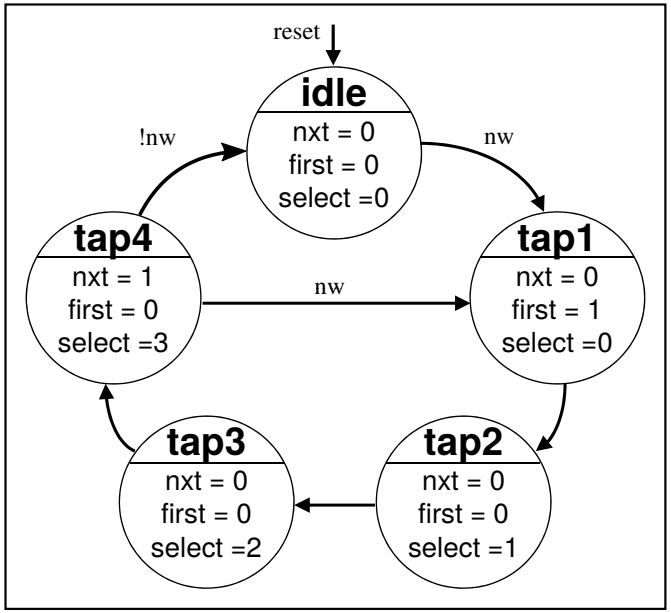
Máquinas de estados: saídas (CASE)

```
output: process (filter)
  begin
    nxt <= '0';
    first <= '0';
    select_ <= "00";

    case filter is
    when idle =>
    when TAP1 =>
      first <= '1';
    when TAP2 =>
      select_ <= "01";
    when TAP3 =>
      select_ <= "10";
    when TAP4 =>
      select_ <= "11";
      nxt <= '1';
    end case;
  end process output;
```



Máquinas de estados: saídas (atribuição de sinais)

```
nxt <= '1' when filter=TAP4 else '0';

first <= '1' when filter=TAP1 else '0';

with filter select
  select_ <= "00" when TAP1,
             "01" when TAP2,
             "10" when TAP3,
             "11" when TAP4,
             "00" when others;
```




DANGER
MAN AT WORK

Tópico

Referências

- *Introduction to Altera Devices and Design Software*

- D'AMORE, Roberto. VHDL: descrição e síntese de circuitos digitais. 2. ed. Rio de Janeiro: Livros Técnicos e Científicos, 2012. 292 p., il. ISBN 9788521620549.

- PEDRONI, Volnei A. Eletrônica digital moderna e VHDL. Rio de Janeiro: Elsevier, 2010. 619 p., ISBN 9788535234657.