UFRGS
UNIVERSIDADE FEDERAL
DO RIO GRANDE DO SUL

# CMP238
# Projeto e Teste de Sistemas VLSI

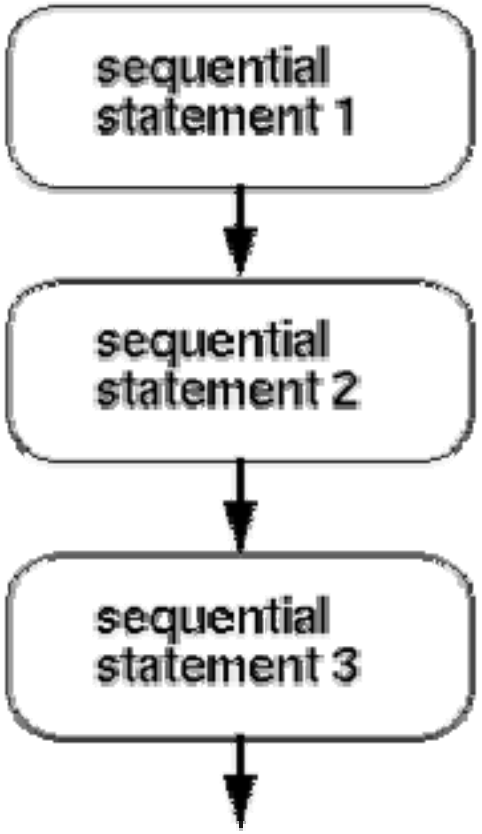
Linguagem de Descrição de Hardware
VHDL

Prof. Fernanda Gusmão de Lima Kastensmidt
fglima@inf.ufrgs.br

# 7 e 8
# Comandos Sequencias

# Comandos Sequenciais

- Execução de acordo com a ordem com que os comandos sequenciais aparecem.
- Permitido apenas dentro da estrutura **process**
- Usado para representar algoritmos.

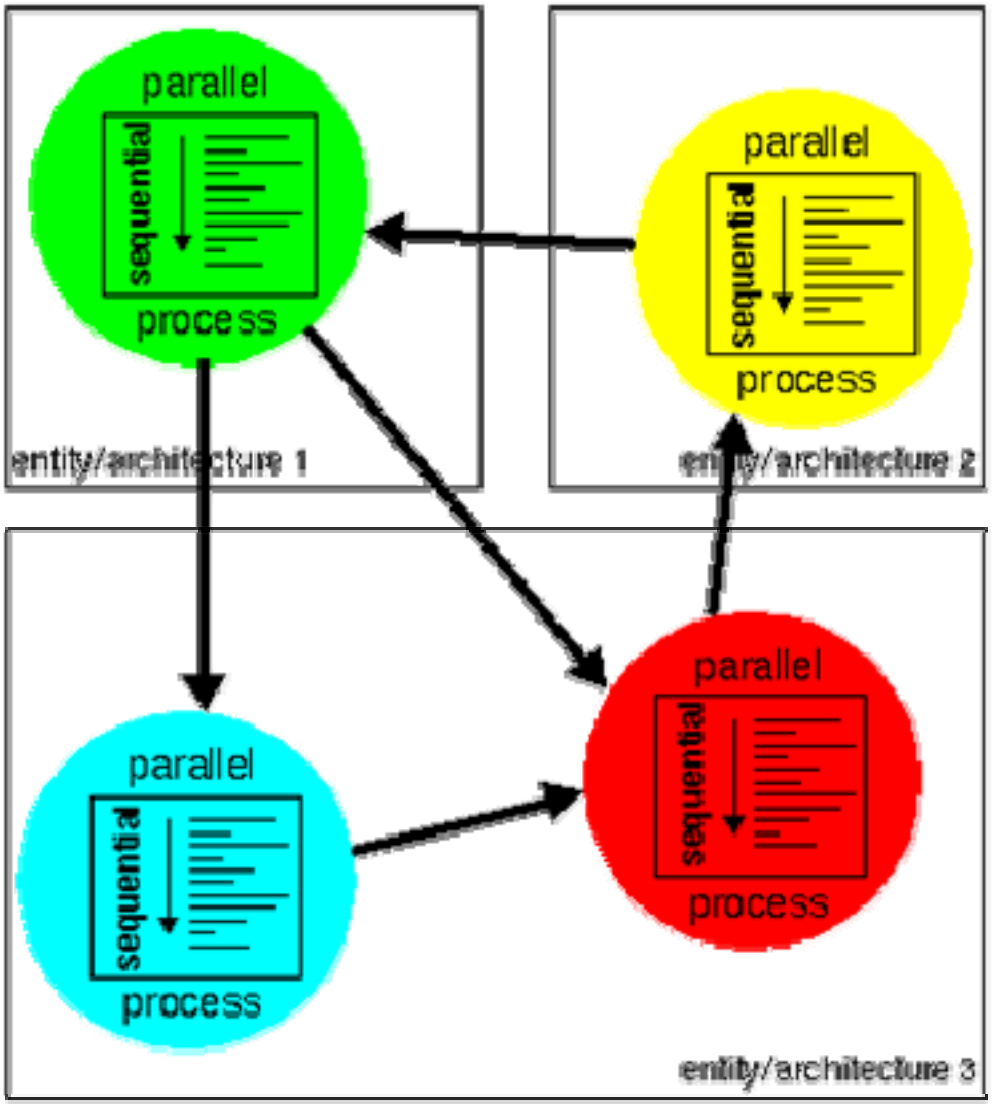


# Process

- Contem comandos sequenciais
- Existe apenas dentro da arquitetura
- Todos os **process** rodam ao mesmo tempo de maneira concorrente.
- **A execução dos process são controladas por:**
  - Lista de sensibilidade (sinais de trigger para a execução do **process**), **ou**
  - Comandos de **wait**
- O label do process é opcional

```
<label>: process (lista)
begin

<comandos>


end process;
```

# VHDL Communication Model



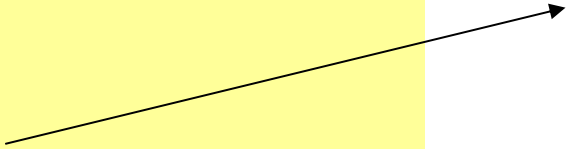
# Exemplo de Process

```
entity AND_OR_XOR is
  port (A,B :                          in   std_logic;
          Z_OR, Z_AND, Z_XOR : out  std_logic);
end AND_OR_XOR;

architecture RTL of AND_OR_XOR is
begin

   A_O_X:  process  (A, B)
    begin
      Z_OR   <= A or    B;
      Z_AND <= A and B;
      Z_XOR <= A xor  B;
     end process  A_O_X ;

end RTL;
```

# Exemplo de Process

```
entity AND_OR_XOR is
  port (A,B                          :  in   std_logic;
        Z_OR                          : out  std_logic);
end AND_OR_XOR;

architecture RTL of AND_OR_XOR is
begin

   A_O_X:  process  (A, B)
    begin
      Z_OR   <= A or    B;
      Z_OR <= A and B;
      Z_OR <= A xor  B;
     end process  A_O_X ;

end RTL;
```

Qual hardware é gerado na síntese?



# Comando: IF… then … else ….

```
if CONDITION then
   -- sequential statements
end if;


if CONDITION then
   -- sequential statements
else
   -- sequential statements
end if;


if CONDITION then
   -- sequential statements
elsif CONDITION then
   -- sequential statements
. . .
else
   -- sequential statements
end if;
```

**A condição é uma expressão booleana**

**Opcional:**

**-elsif**

**-else**

**Cuidado**: todas as opções de ifs devem estar cobertas no comando, se não, latches e flip-flops indesejados surgirão na sintese.

# IF Statement: Examplo

```
entity IF_STATEMENT is
    port (A, B, C, X : in   std_logic_vector (3 downto 0);
          Z          : out std_logic_vector (3 downto 0);
end IF_STATEMENT;
```

```
architecture EXAMPLE1 of IF_STATEMENT is
begin
  process (A, B, C, X)
  begin
    Z <= A;
    if  (X = "1111")  then
      Z <= B;
    elsif  (X > "1000")  then
      Z <= C;
    end if;
  end process;
end EXAMPLE1;
```

Todos os sinais usados no process estão na lista de sensibilidade

```
architecture EXAMPLE2 of IF_STATEMENT is
begin
  process (A, B, C, X)
  begin

    if  (X = "1111")  then
      Z <= B;
    elsif  (X > "1000")  then
      Z <= C;
    else
      Z <= a;
    end if;
  end process;
end EXAMPLE2;
```

# Tente não cobrir todas as opções para ver o que acontece….

```
entity IF_STATEMENT is
     port (A, B, C, X : in   std_logic_vector (3 downto 0);
           Z          : out std_logic_vector (3 downto 0);
 end IF_STATEMENT;
```

```
architecture EXAMPLE1 of IF_STATEMENT is
begin
  process (A, B, C, X)
  begin
    -- Z <= A;
    if  (X = "1111")  then
      Z <= B;
    elsif  (X > "1000")  then
      Z <= C;
    end if;
  end process;
end EXAMPLE1;
```

Todos os sinais usados no process estão na lista de sensibilidade

```
architecture EXAMPLE2 of IF_STATEMENT is
begin
  process (A, B, C, X)
  begin

    if  (X = "1111")  then
      Z <= B;
    elsif  (X > "1000")  then
      Z <= C;
    --else
     -- Z <= a;
    end if;
  end process;
end EXAMPLE2;
```

# Comando: CASE … is… WHEN ….

```
case  EXPRESSION  is

   when  VALUE_1  =>
     -- sequential statements

   when  VALUE_2 | VALUE_3   =>
     -- sequential statements

   when  VALUE_4 to VALUE_N  =>
     -- sequential statements

   when   others =>
     -- sequential statements

end   case ;
```

- Opções não podem ser coincidentes.

- Todas as opções devem ser cobertas:
  - valores simples
  - intervalo de valores
  - seleção de valores por ("|" que significa "or")
  - uso obrigatorio de "when others" para cobrir a(s) ultima(s) opções.

```
entity CASE_STATEMENT is
   port (A, B, C : in   std_logic_vector(3 downto 0);
           X          : in std_logic_vector(3 downto 0);
           Z            :  std_logic_vector(3 downto 0));
end CASE_STATEMENT;

architecture EXAMPLE of CASE_STATEMENT is
signal X_int : integer range from 0 to 15;
begin
X_int <= conv_integer(X);
   process (A, B, C, X_int)
   begin
      case  X_int  is
        when  0  =>
          Z <= A;
        when  7  |  9  =>
          Z <= B;
        when  1 to 5  =>
          Z <= C;
        when others =>
          Z <= 0;
      end case;
   end process;
end EXAMPLE;
```

```
entity RANGE_2 is
port (A, B, C, X : in    std_logic_vector(3 downto 0);
        Z        : out   std_logic_vector(3 downto 0);
end RANGE_2;

architecture EXAMPLE of RANGE_2 is
begin
   process (A, B, C, X)
   begin
      case X is
         when "0000" =>
            Z <= A;
         when "0111" | "1001" =>
            Z <= B;
          when "0001" to "0101" =>        -- wrong
            Z <= C;
         when others =>
            Z <= 0;
      end case;
   end process;
end EXAMPLE;
```

**A sequencia de valores é indefinida para arrays.**

```
entity CONDITIONAL_ASSIGNMENT is
   port (A, B, C, X : in   std_logic_vector (3 downto 0);
         Z_CONC : out std_logic_vector (3 downto 0);
         Z_SEQ    : out std_logic_vector (3 downto 0));
end CONDITIONAL_ASSIGNMENT;

architecture EXAMPLE of CONDITIONAL_ASSIGNMENT is
begin
   -- Concurrent version of conditional signal assignment
   Z_CONC <= B when X = "1111" else
             C when X > "1000" else
             A;

   -- Equivalent sequential statements
   process (A, B, C, X)
   begin
      if  (X = "1111")  then
        Z_SEQ <= B
      elsif  (X > "1000")  then
        Z_SEQ <= C;
      else
        Z_SEQ <= A;
      end if;
   end process;
end EXAMPLE;
```

Comando concorrente

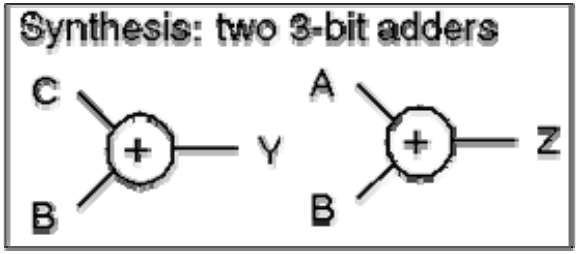
Comando sequencial no process

**IMP: ambos comandos são concorrentes entre si e geram o mesmo circuito combinacional.**

# Variaveis

- **Variaveis são usadas apenas em processes**
  - **São declaradas antes do begin do process**
  - **Conhecidas apenas localmente no process onde foram declaradas**
- **VHDL 93: variaveis globais**
  - Não sintetizavel
- **Assinalamento global**
  - **signal to variable**
  - **variable to signal**
  - **types have to match**

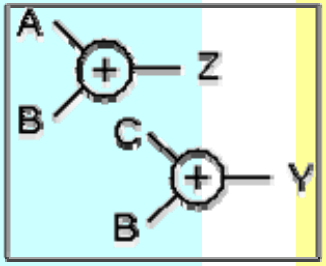
```
architecture RTL of XYZ is
      signal A, B, C : std_logic_vector(7 downto 0);
      signal Y, Z :   std_logic_vector(7 downto 0);
begin
      process (A, B, C)
        variable M, N :  std_logic_vector(7 downto 0);
      begin
         M := A;
         N := B;
         Z <= M + N;
         M := C;
         Y <= M + N;
      end process;
end RTL;
```



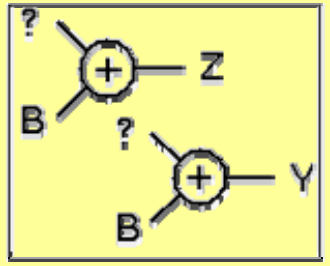
# Variaveis vs. Sinais

- **Valores de sinais são assinalados depois da execução do process.**
- **Apenas o ultimo assinalamento é levado em consideração**
- **M <= A;**
  **é sobre escrito por M <= C;**
- **A segunda entrada do somador é conectado a C.**

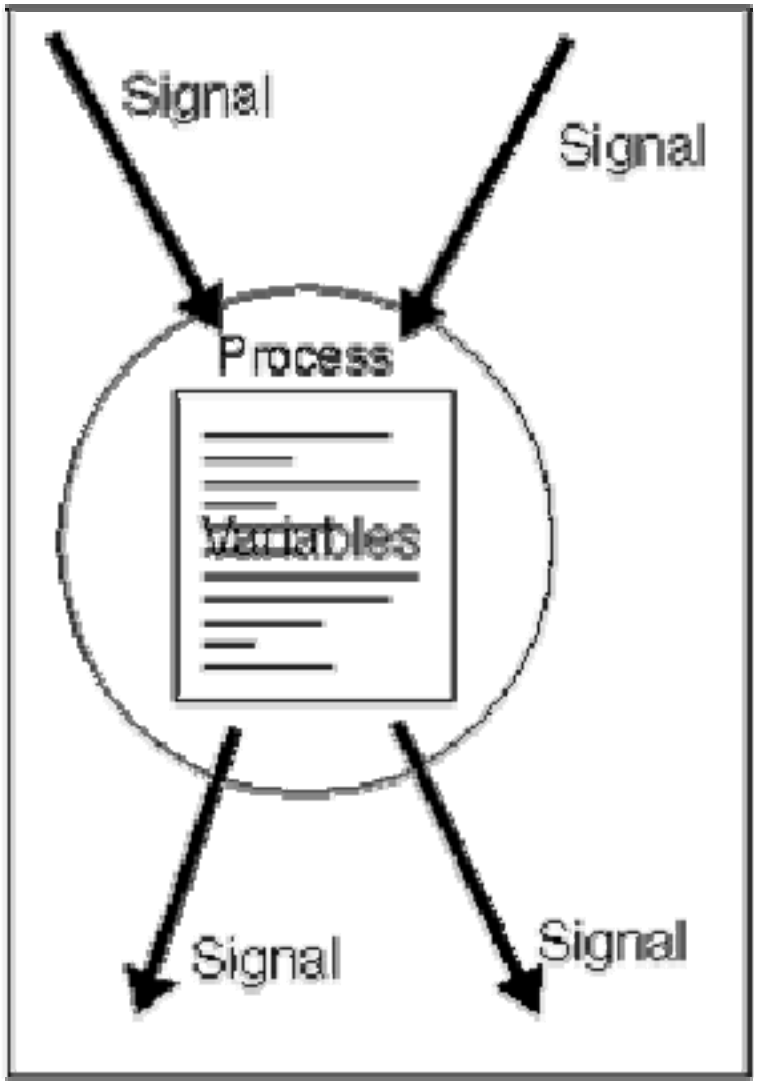
```
signal A, B, C, Y, Z : std_logic_vector(7 downto 0);
begin
    process (A, B, C)
    variable M, N :  std_logic_vector(7 downto 0);
    begin
        M  :=  A;
        N  :=  B;
        Z <= M + N;
        M  :=  C;
        Y <= M + N;
    end process;
```




```
signal A, B, C, Y, Z :  std_logic_vector(7 downto 0);
signal M, N          :  std_logic_vector(7 downto 0);
begin
    process (A, B, C, M, N)

    begin
        M  <=  A;
        N  <=  B;
        Z <= M + N;
        M  <=  C;
        Y <= M + N;
    end process;
```



# Uso de Variaveis



# Global Variables (VHDL'93)

```
architecture BEHAVE of SHARED is
        shared variable S  : integer;
begin
        process (A, B)
        begin
                S :=  A + B;
        end process;

        process (A, B)
        begin
                S :=  A - B;
        end process;
end BEHAVE;
```

**Acessivel por todos os processos.**

**Não é sintetizavel**

**Pode ocasionar erros muito comuns de valores de saida indeterminado.**

# Comando: FOR Loops

```
entity FOR_LOOP is
   port (A : in    integer range 0 to 3;
         Z  : out std_logic_vector (3 downto 0));
end FOR_LOOP;

architecture EXAMPLE of FOR_LOOP is
begin
   process (A)
   begin
      Z <= "0000";
       for  I  in  0 to 3  loop
         if (A = I) then
            Z(I) <= `1`;
         end if;
       end loop;
   end process;
end EXAMPLE;
```

**Se o LOOP é para ser sintetizado, o intervalo do loop não pode depender do valor de um sinal ou variavel, ou seja, deve ser totalmente estático o intervalo.**

# Loop Sintaxe

```
[LOOP_LABEL :]
for IDENTIFIER in DISCRETE_RANGE loop
   -- sequential statements
end loop [LOOP_LABEL] ;
```

```
[LOOP_LABEL :]
while CONDITION loop
   -- sequential statements
end loop [LOOP_LABEL] ;
```

```
entity CONV_INT is
    port (VECTOR: in   bit_vector(7 downto 0);
          RESULT:  out integer);
end CONV_INT;
```

```
architecture A of CONV_INT is
begin
  process(VECTOR)
    variable TMP: integer;

  begin
    TMP := 0;

    for I in 7 downto 0 loop
      if (VECTOR(I)='1') then
        TMP := TMP + 2**I;
      end if;
    end loop;

    RESULT <= TMP;
  end process;
end A;
```

```
architecture B of CONV_INT is
begin
  process(VECTOR)
    variable TMP: integer;

  begin
    TMP := 0;

    for I in VECTOR'range loop
      if (VECTOR(I)='1') then
        TMP := TMP + 2**I;
      end if;
    end loop;

    RESULT <= TMP;
  end process;
end B;
```

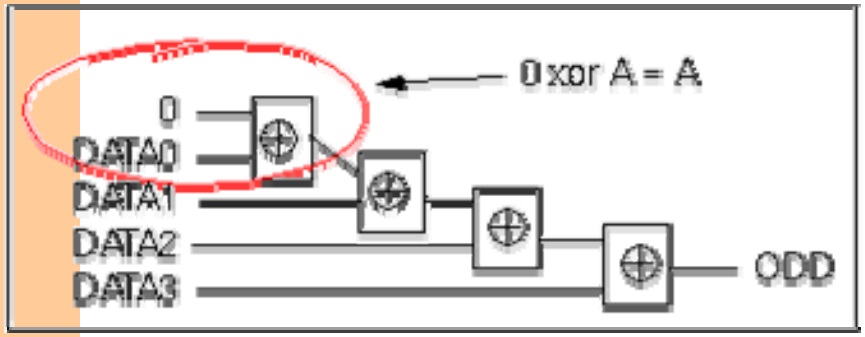
```
architecture C of CONV_INT is
begin
  process(VECTOR)
    variable TMP: integer;
    variable I       : integer;
  begin
    TMP := 0;
    I := VECTOR'high;
    while (I >= VECTOR'low) loop
      if (VECTOR(I)='1') then
        TMP := TMP + 2**I;
      end if;
      I := I - 1;
    end loop;
    RESULT <= TMP;
  end process;
end C;
```

# Geração de código

```
[LOOP_LABEL :]
for IDENTIFIER in DISCRETE_RANGE generate
   -- statements
end generate [LOOP_LABEL] ;
```

```
signal A, B : std_logic_vector(8 downto 0);
signal F : std_logic_vector(7 downto 0);

architecture B of CONV_INT is
begin
t1:   for I in 1 to 7 generate
   F(i) <= A(I+1) or B(I-1);
   end generate;
end B;
```

```
entity PARITY is
      port (DATA: in   std_logic_vector (3 downto 0);
            ODD : out std_logic);
end PARITY;


architecture RTL of PARITY is
begin
     process (DATA)
        variable TMP : bit;
     begin
        TMP := `0`;

        for I in  DATA`low to DATA`high  loop
           TMP := TMP xor DATA(I);
        end loop;

        ODD <= TMP;
     end process;
end RTL;
```



# Comando: WAIT

**O comando wait' para a execução do process**

- **O process é continuado quando a instrução é completada.**

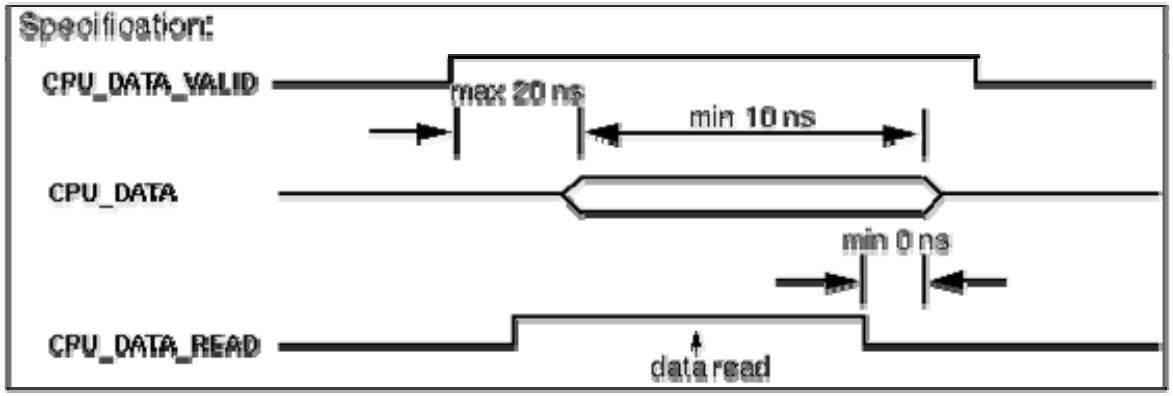
- **wait para um especifico tempo**
- **wait por um evento do sinal**
- **wait por uma condição verdadeira**

**(necessita de um evento do sinal)**

- **indefinido (process não é mais ativado)**

# WAIT Statement: Exemplos

```
STIMULUS: process
begin
    SEL     <= `0`;
    BUS_B <= "0000";
    BUS_A <= "1111";
    wait for 10 ns;

    SEL <= `1`;
    wait for 10 ns;

    SEL <= `0`;
    wait for 10 ns;

    wait;
end process STIMULUS;
```




```
READ_CPU : process
begin
      wait until CPU_DATA_VALID = `1`;
      CPU_DATA_READ <= `1`;
      wait for 20 ns;
      LOCAL_BUFFER <= CPU_DATA;
      wait for 10 ns;
      CPU_DATA_READ <= `0`;
end process READ_CPU;
```

# WAIT Statement: Exemplos

```
entity FF is
     port (D, CLK : in   bit;
           Q      : out bit);
end FF;

architecture BEH_2 of FF is
begin
   process
   begin
      wait until CLK=`1`;

      Q <= D;

   end process;
end BEH_2;
```

# Aprendendo por exemplos

Recapitulando VHDL de comandos que
geram circuitos combinacionais

# Somador 1 bit

Comandos concorrentes de atribuição simples

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity somador1bit is
    Port ( a : in  STD_LOGIC;
           b : in  STD_LOGIC;
           cin : in  STD_LOGIC;
           sum : out  STD_LOGIC;
           cout : out  STD_LOGIC);
end somador1bit;

architecture Behavioral of somador1bit is

begin

sum <= a xor b xor cin;
cout <= (a and b) or (a and cin) or (b and cin);

end Behavioral;
```

# Somador 8 bits

Comandos concorrentes de atribuição simples e instanciação hierarquica de Componentes.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity somador8bits is
    Port ( dadoa : in  STD_LOGIC_VECTOR (7 downto 0);
           dadob : in  STD_LOGIC_VECTOR (7 downto 0);
           carry_in : in  STD_LOGIC;
           dado_soma : out  STD_LOGIC_VECTOR (7 downto 0);
           carry_out : out  STD_LOGIC);
end somador8bits;

architecture Behavioral of somador8bits is

component somador1bit is
    Port ( a : in  STD_LOGIC;
           b : in  STD_LOGIC;
           cin : in  STD_LOGIC;
           sum : out  STD_LOGIC;
           cout : out  STD_LOGIC);
end component;

signal carry_int : std_logic_vector(6 downto 0);

begin

bit0: somador1bit
port map(dadoa(0), dadob(0), carry_in, dado_soma(0), carry_int(0));
bit1: somador1bit
port map(dadoa(1), dadob(1), carry_int(0), dado_soma(1), carry_int(1));
bit2: somador1bit
port map(dadoa(2), dadob(2), carry_int(1), dado_soma(2), carry_int(2));
bit3: somador1bit
port map(dadoa(3), dadob(3), carry_int(2), dado_soma(3), carry_int(3));
bit4: somador1bit
port map(dadoa(4), dadob(4), carry_int(3), dado_soma(4), carry_int(4));
bit5: somador1bit
port map(dadoa(5), dadob(5), carry_int(4), dado_soma(5), carry_int(5));
bit6: somador1bit
port map(dadoa(6), dadob(6), carry_int(5), dado_soma(6), carry_int(6));
bit7: somador1bit
port map(dadoa(7), dadob(7), carry_int(6), dado_soma(7), carry_out);



end Behavioral;
```

# Somador de 8 bits

Uso de componentes e do comando FOR - GENERATE

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity somador8bits_vfor is
    Port ( dadoa : in  STD_LOGIC_VECTOR (7 downto 0);
           dadob : in  STD_LOGIC_VECTOR (7 downto 0);
           carry_in : in  STD_LOGIC;
           dado_soma : out  STD_LOGIC_VECTOR (7 downto 0);
           carry_out : out  STD_LOGIC);
end somador8bits_vfor;

architecture Behavioral of somador8bits_vfor is

component somador1bit is
    Port ( a : in  STD_LOGIC;
           b : in  STD_LOGIC;
           cin : in  STD_LOGIC;
           sum : out  STD_LOGIC;
           cout : out  STD_LOGIC);
end component;

signal carry_int : std_logic_vector(7 downto 0);
```

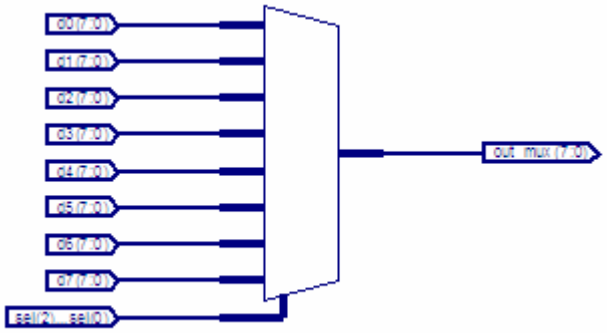
```
begin

bit0: somador1bit
port map(dadoa(0), dadob(0), carry_in, dado_soma(0), carry_int(0));

l1: for i in 1 to 7 generate
bitx: somador1bit
port map(dadoa(i), dadob(i), carry_int(i-1), dado_soma(i), carry_int(i));
end generate;

carry_out <= carry_int(7);

end Behavioral;
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity comparador_4bits is
    Port ( dadox : in  STD_LOGIC_VECTOR (3 downto 0);
           dadoy : in  STD_LOGIC_VECTOR (3 downto 0);
           comp : out  STD_LOGIC_VECTOR (1 downto 0));
end comparador_4bits;

architecture Behavioral of comparador_4bits is

begin

process(dadox, dadoy)
begin
  if (dadox>dadoy) then
     comp <= "10";
  elsif (dadox=dadoy) and dadox="1111" then
     comp <= "11";
  elsif (dadox=dadoy) and dadox="0000" then
     comp <= "00";
  else
     comp <= "01";
end if;
end process;

end Behavioral;
```

# Multiplexador

USA PROCESS e comandos sequenciais como CASE

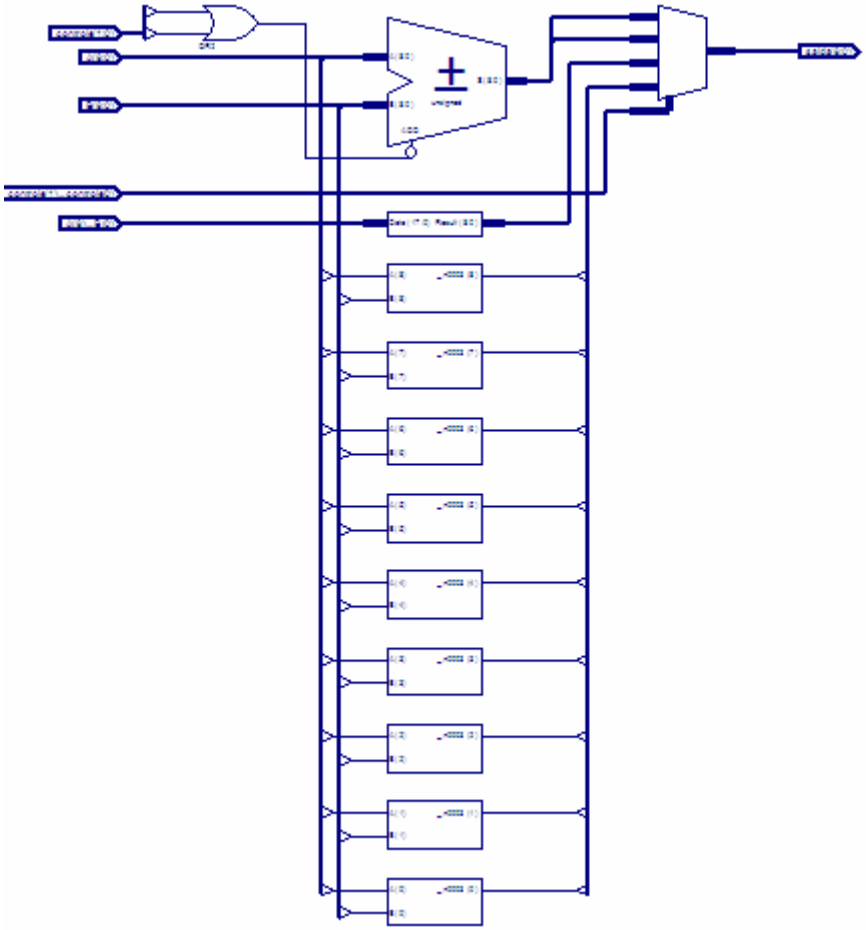
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity mux is
    Port ( d0 : in  STD_LOGIC_VECTOR (7 downto 0);
           d1 : in  STD_LOGIC_VECTOR (7 downto 0);
           d2 : in  STD_LOGIC_VECTOR (7 downto 0);
           d3 : in  STD_LOGIC_VECTOR (7 downto 0);
           d4 : in  STD_LOGIC_VECTOR (7 downto 0);
           d5 : in  STD_LOGIC_VECTOR (7 downto 0);
           d6 : in  STD_LOGIC_VECTOR (7 downto 0);
           d7 : in  STD_LOGIC_VECTOR (7 downto 0);
           sel : in  STD_LOGIC_VECTOR (2 downto 0);
           out_mux : out  STD_LOGIC_VECTOR (7 downto 0));
end mux;
```




```
architecture Behavioral of mux is

begin

process(d0, d1, d2, d3, d4, d5, d6, d7, sel)
begin
case sel is
WHEN "000" => out_mux <=d0;
WHEN "001" => out_mux <=d1;
WHEN "010" => out_mux <=d2;
WHEN "011" => out_mux <=d3;
WHEN "100" => out_mux <=d4;
WHEN "101" => out_mux <=d5;
WHEN "110" => out_mux <=d6;
WHEN others => out_mux <=d7;
end case;
end process;

end Behavioral;
```

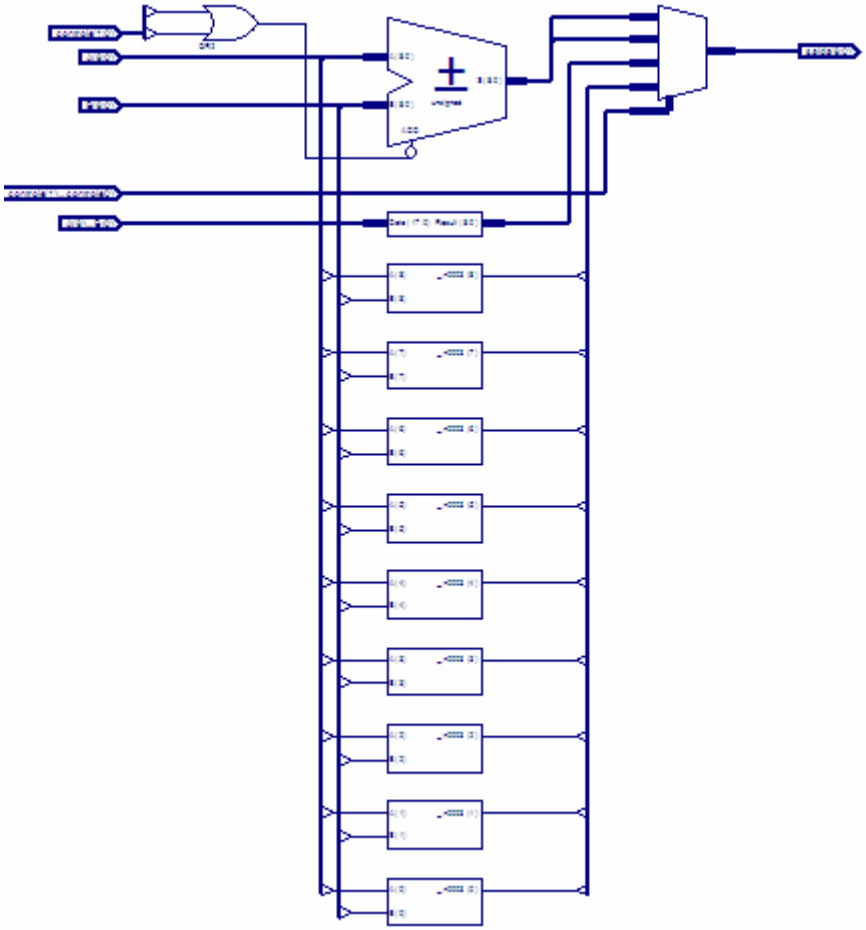
# ULA de 8 bits c/ 4 operações

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity ULA_8bits is
    Port ( A : in  STD_LOGIC_VECTOR (8 downto 0);
           B : in  STD_LOGIC_VECTOR (8 downto 0);
           controle : in  STD_LOGIC_VECTOR(1 downto 0);
           saida : out  STD_LOGIC_VECTOR (8 downto 0));
end ULA_8bits;

architecture Behavioral of ULA_8bits is

begin

process(A, B, controle)
begin
 CASE controle IS
 WHEN "00" => saida <= A + B;
 WHEN "01" => saida <= A - B;
 WHEN "10" => saida <= A xor B;
 WHEN others => saida <= A and (not B);
 END CASE;
end process;

end Behavioral;
```



# ULA de 8 bits c/ 4 operações

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity ULA_8bits_sp is
    Port ( A : in  STD_LOGIC_VECTOR (8 downto 0);
           B : in  STD_LOGIC_VECTOR (8 downto 0);
           controle : in  STD_LOGIC_VECTOR(1 downto 0);
           saida : out  STD_LOGIC_VECTOR (8 downto 0));
end ULA_8bits_sp;

architecture Behavioral of ULA_8bits_sp is

begin

saida <= A+B when controle="00" else
             A-B when controle="01" else
             A xor B when controle="10" else
             A and (not B);


end Behavioral;
```



# Implementação de circuitos sequenciais

Sensíveis ao relógio (clk)

# Clocked Process:
# Detecção da borda do Clock

*if*
**_clock_signal_ name'EVENT and _clock_signal_ name='1'**
**_clock_signal_ name='1' and _clock_signal_ name'EVENT**
**not _clock_signal_ name'STABLE and _clock_signal_ name='1'**
**_clock_signal_ name='1' and not _clock_signal_ name'STABLE**
**RISING_EDGE ( _clock_signal_ name)**

IEEE 1076.6 is not fully supported by all tools

# Clocked Process: Clock Edge Detection

*wait until*

- ***clock_signal_*** **name'EVENT and*clock_signal_* name='1'**
- ***clock_signal_*** **name='1' and c*lock_signal_* name'EVENT**
- **not *clock_signal_* name'STABLE and c*lock_signal_* name='1'**
- ***clock_signal_*** **name='1' and not *clock_signal_* name'STABLE**
- **RISING_EDGE ( *clock_signal_* name)**
- ***clock_signal_*** **name='1'**

IEEE 1076.6 is not fully supported by all tools

# Detection of a Rising Edge by Use of Functions

- Defined in std_logic_1164 package

```
process
begin
   wait until  RISING_EDGE (CLK);
   Q <= D;
end process;
```

```
function RISING_EDGE (signal CLK : std_ulogic)
            return boolean is
begin
    if (CLK`event and CLK = `1` and CLK`last_value = `0`) then
        return true;
    else
        return false;
    end if;
end RISING_EDGE;
```

# Clocked Process: Rules

Elementos de memoria são inferidos, ou seja, gerados sempre que os sinais receberem assinalamentos dentro de uma process controlado por relógio (clock)

```
process
begin
   wait until CLK'event and CLK='1';
   if RESET = '1' then
      -- synchronous register reset
   else
      -- combinatorics
   end if;
end process;
```

**Wait-form:**
**no sensitivity list**
**Synchronous reset**

```
process(CLK, RST)
begin
   if (RST = `1`) then
      -- asynchronous register reset
   elsif (CLK`event and CLK=`1`) then
      -- combinatorics
   end if;
end process;
```

**If-form:**
**only clock and asynchronous signals (reset) in sensitivity list**
**Synchronous and asynchronous reset**

# Registrador: Asynchronous Set/Reset

```
library IEEE;
use IEEE.std_logic_1164.all;

entity ASYNC_FF is
port (
CLK, SET, RST : in std_logic;
D   : in   std_logic_vector(7 downto 0);
Q   : out std_logic_vector(7 downto 0));
end ASYNC_FF;

architecture RTL of ASYNC_FF is
begin
    process  (CLK, RST, SET)
    begin
        if (RST = `1`) then
          Q <= `0`;
       elsif SET ='1' then
          Q <= '1';
        elsif (CLK`event and CLK = `1`) then
          Q <= D;
        end if;
    end process;
end RTL;
```

UFRGS

# Registrador: Synchronous Set/Reset

```
library IEEE;
use IEEE.std_logic_1164.all;

entity ASYNC_FF is
port (
CLK, SET, RST : in std_logic;
D   : in   std_logic_vector(7 downto 0);
Q   : out std_logic_vector(7 downto 0));
end ASYNC_FF;

architecture RTL of ASYNC_FF is
begin
   process  (CLK)
   begin
      if (CLK'event and CLK = '1') then
         if (RST = '1') then
            Q <= `0`;
         elsif SET ='1' then
            Q <= '1';
         else
            Q <= D;
         end if;
      end if;
   end process;
end RTL;
```

# Registrador: Synchronous Set/Reset

```
library IEEE;
use IEEE.std_logic_1164.all;

entity ASYNC_FF is
port (D, CLK, SET, RST : in   std_logic;
      Q                : out std_logic);
end ASYNC_FF;

architecture RTL of ASYNC_FF is
begin
   process  (CLK)
   begin
      if (CLK`event and CLK = `1`) then
         if (RST = `1`) then
            Q <= `0`;
          elsif SET ='1' then
             Q <= '1';
          else
              Q <= D;
         end if;
      end if;
   end process;
end RTL;
```

UFRGS

# Registrador Contador

```
library IEEE;
use IEEE.std_logic_1164.all;

entity COUNTER is
port (CLK:  in std_logic;
        Q    :  out std_logic_vector(3 downto 0);
end COUNTER;

architecture RTL of COUNTER is
signal  COUNT  :  std_logic_vector(3 downto 0);
begin
   process (CLK)
   begin
      if CLK'event and CLK = '1' then
          if (COUNT >= "1001") then
             COUNT <= "0000";
          else
             COUNT <= COUNT +1;
          end if;
      end if;
   end process;

   Q <=  COUNT ;
```

**COUNT: 4 flip flops**
**Q: not used in clocked process pois é um pino de saida e NÃO pode ser lido no process.**