

Curso de Modelagem de Circuitos Digitais em VHDL - Aula 2

Alceu Bernardes Castanheira de Farias¹

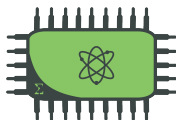
¹Semana Universitária

Universidade de Brasília, campus Gama - FGA

Códigos e slides do curso disponíveis aqui:

<https://github.com/alceu-castanheira/Curso-VHDL-Eletronjun-2020>

24 de setembro de 2019



eletronjun
Engenharia Eletrônica Júnior

Sumário

1 Circuitos aritméticos

2 Processos

3 Circuitos sequenciais

4 Contadores

5 Exercício Final

6 Dicas e Atalhos

1 Circuitos aritméticos

2 Processos

3 Circuitos sequenciais

4 Contadores

5 Exercício Final

6 Dicas e Atalhos

Circuitos aritméticos

- Circuitos aritméticos são responsáveis por realizar operações aritméticas em sistemas digitais;

Circuitos aritméticos

- Circuitos aritméticos são responsáveis por realizar operações aritméticas em sistemas digitais;
- São circuitos muito utilizados;

Circuitos aritméticos

- Circuitos aritméticos são responsáveis por realizar operações aritméticas em sistemas digitais;
- São circuitos muito utilizados;
- Alguns sistemas possuem Unidades Lógico-Aritméticas (ULAs) responsáveis por implementar as operações aritméticas necessárias para o funcionamento correto do sistema;

Circuitos aritméticos

- Circuitos aritméticos são responsáveis por realizar operações aritméticas em sistemas digitais;
- São circuitos muito utilizados;
- Alguns sistemas possuem Unidades Lógico-Aritméticas (ULAs) responsáveis por implementar as operações aritméticas necessárias para o funcionamento correto do sistema;
- Em VHDL, circuitos aritméticos podem ser implementados de maneiras simples por meio de bibliotecas apropriadas.

Circuitos aritméticos em VHDL: somadores

- Utilizando IEEE_NUMERIC_STD:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity somador is
    Port ( a : in STD_LOGIC_VECTOR (3 downto 0);
          b : in STD_LOGIC_VECTOR (3 downto 0);
          output : out STD_LOGIC_VECTOR (3 downto 0));
end somador;

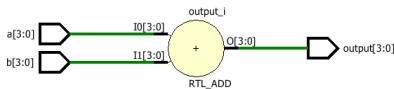
architecture Behavioral of somador is

begin

    output <= std_logic_vector(unsigned(a) + unsigned(b));

end Behavioral;
```

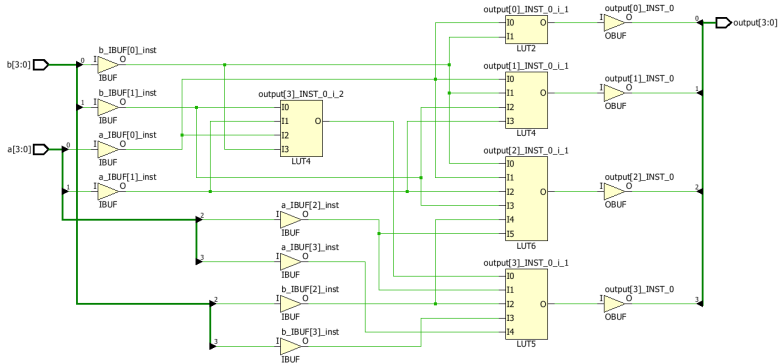

Circuitos aritméticos em VHDL: somadores

**Utilization - Post-Synthesis**

Resource	Estimation	Available	Utilization %
LUT	4	20800	0.02
IO	12	106	11.32

Graph **Table**

Circuitos aritméticos em VHDL: somadores



Circuitos aritméticos em VHDL: subtratores

- Utilizando IEEE_ NUMERIC_ STD:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

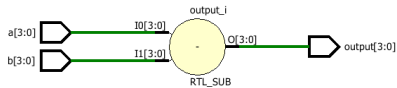
entity somador is
    Port ( a : in STD_LOGIC_VECTOR (3 downto 0);
          b : in STD_LOGIC_VECTOR (3 downto 0);
          output : out STD_LOGIC_VECTOR (3 downto 0));
end somador;

architecture Behavioral of somador is
begin

    output <= std_logic_vector(unsigned(a) - unsigned(b));

end Behavioral;
```

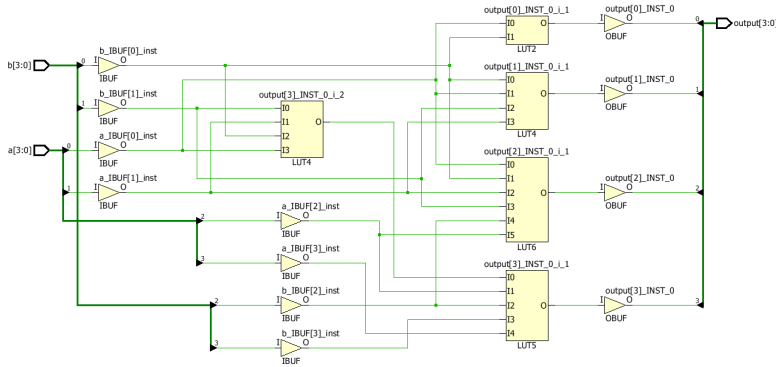
Circuitos aritméticos em VHDL: subtratores

**Utilization - Post-Synthesis**

Resource	Estimation	Available	Utilization %
LUT	4	20800	0.02
IO	12	106	11.32

Graph **Table**

Circuitos aritméticos em VHDL: subtratores



Sumário

1 Circuitos aritméticos

2 Processos

3 Circuitos sequenciais

4 Contadores

5 Exercício Final

6 Dicas e Atalhos

Processos

```
optional_label: process (optional sensitivity list)
    declarations
begin
    sequential statements
end process optional_label;
```

- Implementam instruções sequenciais;

Processos

```
optional_label: process (optional sensitivity list)
    declarations
begin
    sequential statements
end process optional_label;
```

- Implementam instruções sequenciais;
- Declarados dentro da arquitetura de um sistema;

Processos

```
optional_label: process (optional sensitivity list)
    declarations
begin
    sequential statements
end process optional_label;
```

- Implementam instruções sequenciais;
- Declarados dentro da arquitetura de um sistema;
- Possuem uma lista de sensibilidade: sempre que um dado na lista de sensibilidade muda de valor, o processo é executado novamente.

Processos

```
process (ALARM_TIME, CURRENT_TIME)
begin
  if (ALARM_TIME = CURRENT_TIME) then
    SOUND_ALARM <= '1';
  else
    SOUND_ALARM <= '0';
  end if;
end process;
```

- **MUITO IMPORTANTE:**

- 1 Processos ocorrem em paralelo em relação a outros processos.

Processos

```
process (ALARM_TIME, CURRENT_TIME)
begin
    if (ALARM_TIME = CURRENT_TIME) then
        SOUND_ALARM <= '1';
    else
        SOUND_ALARM <= '0';
    end if;
end process;
```

- **MUITO IMPORTANTE:**

- 1 Processos ocorrem em paralelo em relação a outros processos.
- 2 Processos são atualizados somente ao final de todas as instruções, **com exceção de variáveis**.

Processos

```
process (ALARM_TIME, CURRENT_TIME)
begin
    if (ALARM_TIME = CURRENT_TIME) then
        SOUND_ALARM <= '1';
    else
        SOUND_ALARM <= '0';
    end if;
end process;
```

- **MUITO IMPORTANTE:**

- 1 Processos ocorrem em paralelo em relação a outros processos.
- 2 Processos são atualizados somente ao final de todas as instruções, **com exceção de variáveis**.
- 3 Os processos ocorrem em paralelo, mas as instruções do processo ocorrem sequencialmente.

Processos



Mandamentos de processos

- 1 Não escreverás nos mesmos sinais, variáveis e saídas em processos diferentes.
- 2 Não esquecerás da lista de sensibilidade.
- 3 Não esquecerás do *begin*.
- 4 Não nomearás dois processos com o mesmo nome.



Estrutura condicional - *when/case*

```
case expression is
  when choice =>
    sequential statements
  when choice =>
    sequential statements
end case;
```

- Dado valor de uma entrada ou sinal, realiza a instrução correspondente ao valor lido;
- Declarados dentro de um processo;
- Todas as opções devem ser declaradas, a não ser quando utilizada a opção *when others*.
- As opções não podem conflitar entre si.

Estrutura condicional - *when/case*

```
case SEL is
  when "01" =>    Z <= A;
  when "10" =>    Z <= B;
  when others =>  Z <= 'X';
end case;
```


Estrutura condicional - *if*

```
if condition_1 then
    sequential statements
elsif condition2 then
    sequential statements
else
    sequential statements
end if;
```

- Dentre um determinado números de condições, executa a que for verdadeira;
- Declarados dentro de um processo;
- O uso de *elsif* é opcional.
- Se existirem condições que não forem especificadas, um *latch* pode ser criado e problemas de temporização do circuito podem acontecer.

Estrutura condicional - *if* : uso desejado

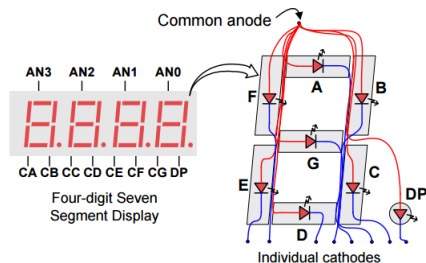
```
if (X = 5) and (Y = 9) then
    Z <= A;
elsif (X >= 5) then
    Z <= B;
else
    Z < C;
end if;
```

Estrutura condicional - *if* : uso indesejado

```
process (EN, D)
begin
    if (EN = '1') then Q <= D;
    end if;
end process;
```

Exercício 2 - Processos e estruturas condicionais

- Projetar em VHDL um decodificador para *display* de 7 segmentos;
- O sistema recebe uma entrada pelas chaves e exibe o valor correspondente em um *display*;
- Cada segmento do display corresponde é ativo em nível lógico baixo;
- Cada display é acionado por um ânodo ativo em nível lógico baixo.



Exercício 2 - Processos e estruturas condicionais

HEX	dp	A	B	C	D	E	F	G
0	1	0	0	0	0	0	0	1
1	1	1	0	0	1	1	1	1
2	1	0	0	1	0	0	1	0
3	1	0	0	0	0	1	1	0
4	1	1	0	0	1	1	0	0
5	1	0	1	0	0	1	0	0
6	1	0	1	0	0	0	0	0
7	1	0	0	0	1	1	1	1
8	1	0	0	0	0	0	0	0
9	1	0	0	0	0	1	0	0
A	1	0	0	0	1	0	0	0
B	1	1	1	0	0	0	0	0

Exercício 2 - Processos e estruturas condicionais

HEX	dp	A	B	C	D	E	F	G
C	1	0	1	1	0	0	0	1
D	1	1	0	0	0	0	1	0
E	1	0	1	1	0	0	0	0
F	1	1	1	1	0	0	0	0

Exercício 2 - Processos e estruturas condicionais

- E para testar no laboratório remoto?

Exercício 2 - Processos e estruturas condicionais

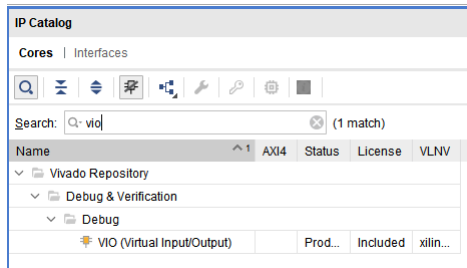
- E para testar no laboratório remoto?
- Precisamos combinar o nosso circuito com um IP da Xilinx: o VIO IP Core

Exercício 2 - Processos e estruturas condicionais

- E para testar no laboratório remoto?
- Precisamos combinar o nosso circuito com um IP da Xilinx: o VIO IP Core
- O VIO irá gerar entradas controladas virtualmente por nós e enviá-las para o FPGA para que possamos testar o circuito no FPGA.

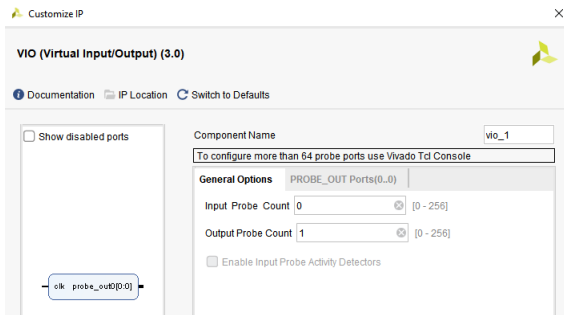
Exercício 2 - Processos e estruturas condicionais

- O VIO pode ser gerado em IP Catalog -> VIO.



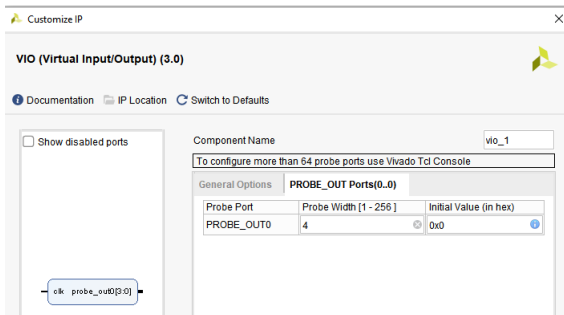
Exercício 2 - Processos e estruturas condicionais

- Temos que definir o número de entradas e saídas do VIO: não teremos entradas e o número de saídas do VIO será igual ao número de entradas do circuito a ser testado no laboratório remoto.



Exercício 2 - Processos e estruturas condicionais

- Temos que definir o número de entradas e saídas do VIO: não teremos entradas e o número de saídas do VIO será igual ao número de entradas do circuito a ser testado no laboratório remoto.



Modularização de um sistema

- Uma estratégia muito útil quando se trabalha com sistemas grandes e complexos é dividir o mesmo em módulos menores, que, quando combinados, resultam no sistema desejado.

Modularização de um sistema

- Uma estratégia muito útil quando se trabalha com sistemas grandes e complexos é dividir o mesmo em módulos menores, que, quando combinados, resultam no sistema desejado.
- Cada módulo pode ser desenvolvido e testado isoladamente, para posteriormente serem integrados.

Modularização de um sistema

- Uma estratégia muito útil quando se trabalha com sistemas grandes e complexos é dividir o mesmo em módulos menores, que, quando combinados, resultam no sistema desejado.
- Cada módulo pode ser desenvolvido e testado isoladamente, para posteriormente serem integrados.
- Metodologia *bottom-up*.

Instanciação de componentes

- Em VHDL, é possível utilizar essa metodologia para implementar circuitos maiores.

Instanciação de componentes

- Em VHDL, é possível utilizar essa metodologia para implementar circuitos maiores.
- Os módulos VHDL podem ser desenvolvidos individualmente e unidos em um arquivo, denominado *top module*.

Instanciação de componentes

- Em VHDL, é possível utilizar essa metodologia para implementar circuitos maiores.
- Os módulos VHDL podem ser desenvolvidos individualmente e unidos em um arquivo, denominado *top module*.
- Nível hierárquico mais alto.

Instanciação de componentes

- Em VHDL, é possível utilizar essa metodologia para implementar circuitos maiores.
- Os módulos VHDL podem ser desenvolvidos individualmente e unidos em um arquivo, denominado *top module*.
- Nível hierárquico mais alto.
- Conecta entradas e saídas dos componentes internos entre si, além de conectar as entradas e saídas do sistema final às entradas e saídas adequadas dos módulos que o compõem.

Instanciação de componentes

- Composto por três partes:

top.vhd (Top-level file)

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

ENTITY top IS
    PORT(w_in, x_in, y_in :IN std_logic;
         clock             :IN std_logic;
         z_out             :OUT std_logic);
END top;

ARCHITECTURE a OF top IS

    COMPONENT logic
        PORT(a,b,c      :IN std_logic;
             x          :OUT std_logic);
    END COMPONENT;

    SIGNAL w_reg, x_reg, y_reg, z_reg  :std_logic;

    BEGIN
        low_logic      : logic PORT MAP (a => w_reg, b => x_reg, c => y_reg, x => z_reg);
        process(w_in, x_in, y_in, clock)
        begin
            w_reg <= w_in;
            x_reg <= x_in;
            y_reg <= y_in;
            z_reg <= z_out;
        end process;
    end a;
```



Instanciação de componentes

● Instanciação de componentes

top.vhd (Top-level file)

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

ENTITY top IS
    PORT(w_in, x_in, y_in :IN std_logic;
         clock             :IN std_logic;
         z_out             :OUT std_logic);
END top;

ARCHITECTURE a OF top IS

    COMPONENT logic
        PORT(a,b,c      :IN std_logic;
             x          :OUT std_logic);
    END COMPONENT;

    SIGNAL w_reg, x_reg, y_reg, z_reg  :std_logic;

BEGIN
    low_logic      : logic PORT MAP (a => w_reg, b => x_reg, c => y_reg, x => z_reg);

    process(w_in, x_in, y_in, clock)
    begin
        if clock'event and clock = '1' then
            w_reg <= w_in;
            x_reg <= x_in;
            y_reg <= y_in;
            z_reg <= w_reg and x_reg and y_reg;
        end if;
    end process;
```



Instanciação de componentes

● Sinais de conexão

top.vhd (Top-level file)

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

ENTITY top IS
    PORT(w_in, x_in, y_in :IN std_logic;
         clock             :IN std_logic;
         z_out             :OUT std_logic);
END top;

ARCHITECTURE a OF top IS

    COMPONENT logic
        PORT(a,b,c      :IN std_logic;
             x          :OUT std_logic);
    END COMPONENT;

    SIGNAL w_reg, x_reg, y_reg, z_reg :std_logic;

    BEGIN
        low_logic      : logic PORT MAP (a => w_reg, b => x_reg, c => y_reg, x => z_reg);

        PROCESS(clock)
```



Instanciação de componentes

- Mapeamento de entradas e saídas do componente

top.vhd (Top-level file)

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

ENTITY top IS
    PORT(w_in, x_in, y_in :IN std_logic;
          clock            :IN std_logic;
          z_out            :OUT std_logic);
END top;

ARCHITECTURE a OF top IS

    COMPONENT logic
        PORT(a,b,c      :IN std_logic;
              x          :OUT std_logic);
    END COMPONENT;

    SIGNAL w_reg, x_reg, y_reg, z_reg    :std_logic;

    BEGIN
        low_logic      : logic PORT MAP (a => w_reg, b => x_reg, c => y_reg, x => z_reg);
```



Exercício 2 - Processos e estruturas condicionais

- Criamos um arquivo que conecta nosso circuito ao VIO (*top module*);
- OBS: O VIO precisa de um sinal de clk, que dita a frequência de funcionamento do mesmo.

```
entity top_module_display_7seg is
  Port (
    -- Entrada
    --
    -- Entrada de clock de 1-bit.
    -- Descrição: Entrada de clock que implementa a frequência de operação
    -- do VIO Core. O circuito 'display_7seg' é puramente combinacional,
    -- não necessita do clock, mas o VIO IP core sim.
    --
    clk : in STD_LOGIC;

    -- Saídas
    --
    -- Saída an de 4-bits.
    -- Descrição: Cada bit dessa saída corresponde ao anodo de um display
    -- da Base 3; o bit menos significativo controla o anodo do display
    -- mais à direita no kit e o bit mais significativo controla o anodo
    -- mais à esquerda do kit. Os anodos habilitam seu respectivo display
    -- em '0', e desabilitam o mesmo em '1'.
    --
    an : out STD_LOGIC_VECTOR (3 downto 0);

    -- Pino de saída referente aos 7 segmentos dos displays da Base 3: 7-bits
    -- Descrição: Cada bit dessa saída corresponde a um dos 7 segmentos dos
    -- displays da Base 3 na ordem "gfedcba". Os segmentos são ligados em
    -- '0' e desligados em '1'.
    --
    seg : out STD_LOGIC_VECTOR (6 downto 0);
  end top_module_display_7seg;
```


Exercício 2 - Processos e estruturas condicionais

- Criamos um arquivo que conecta nosso circuito ao VIO (*top module*);
- OBS: O VIO precisa de um sinal de clk, que dita a frequência de funcionamento do mesmo.

```
architecture Behavioral of top_module_display_7seg is

    -- Instanciação do componente display_7seg
    --
    component display_7seg is
        Port(
            data_in : in STD_LOGIC_VECTOR(3 DOWNTO 0);
            an : out STD_LOGIC_VECTOR(3 DOWNTO 0);
            seg : out STD_LOGIC_VECTOR(6 DOWNTO 0));
    end component;

    -- Instanciação do componente vio_0
    component vio_0 is
        Port(
            clk : in STD_LOGIC;
            probe_out0 : out STD_LOGIC_VECTOR(3 DOWNTO 0));
    end component;

    -- Sinal de conexão entre a saída 'probe_out0' do VIO core e
    -- a entrada 'data_in' do módulo display_7seg.
    --
    signal s_data_in : std_logic_vector(3 downto 0) := (others => '0');
```

Exercício 2 - Processos e estruturas condicionais

- Criamos um arquivo que conecta nosso circuito ao VIO (*top module*);
- OBS: O VIO precisa de um sinal de clk, que dita a frequência de funcionamento do mesmo.

```
-- Conexão dos pinos do módulo display_7seg a seus respectivos sinais, entradas e saídas
--
MODULO_DISPLAY_7SEG: display_7seg port map
(
    data_in => s_data_in,
    an => an,
    seg => seg
);

-- Conexão dos pinos do módulo vio_0 a seus respectivos sinais, entradas e saídas.
--
VIO_CORE: vio_0 port map
(
    clk => clk,
    probe_out0 => s_data_in
);

end Behavioral;
```

Exercício 2 - Processos e estruturas condicionais

- Por fim, precisamos do arquivo de *constraints*, que realiza o mapeamento das entradas e saídas do circuito para os recursos disponíveis no kit da Basys 3.
- Há um arquivo .xdc da Basys3 disponível na internet que facilita esse processo: descomentamos os recursos que serão utilizados e colocamos o nome das nossas entradas e saídas.

```
## This file is a general .xdc for the Basys3 rev B board
## To use it in a project:
## - uncomment the lines corresponding to used pins
## - rename the used ports (in each line, after get_ports) according to the top level signal names in the project

## Clock signal
set_property PACKAGE_PIN W5 [get_ports clk]
set_property IOSTANDARD LVCMOS33 [get_ports clk]
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports clk]
```

Exercício 2 - Processos e estruturas condicionais

- Por fim, precisamos do arquivo de *constraints*, que realiza o mapeamento das entradas e saídas do circuito para os recursos disponíveis no kit da Basys 3.
- Há um arquivo .xdc da Basys3 disponível na internet que facilita esse processo: descomentamos os recursos que serão utilizados e colocamos o nome das nossas entradas e saídas.

```
#7 segment display
set_property PACKAGE_PIN W7 [get_ports {seg[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg[0]}]
set_property PACKAGE_PIN W6 [get_ports {seg[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg[1]}]
set_property PACKAGE_PIN U8 [get_ports {seg[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg[2]}]
set_property PACKAGE_PIN V8 [get_ports {seg[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg[3]}]
set_property PACKAGE_PIN U5 [get_ports {seg[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg[4]}]
set_property PACKAGE_PIN V5 [get_ports {seg[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg[5]}]
set_property PACKAGE_PIN U7 [get_ports {seg[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg[6]}]
```



Exercício 2 - Processos e estruturas condicionais

- Por fim, precisamos do arquivo de *constraints*, que realiza o mapeamento das entradas e saídas do circuito para os recursos disponíveis no kit da Basys 3.
- Há um arquivo .xdc da Basys3 disponível na internet que facilita esse processo: descomentamos os recursos que serão utilizados e colocamos o nome das nossas entradas e saídas.

```
set_property PACKAGE_PIN U2 [get_ports {an[0]]}
set_property IOSTANDARD LVCMOS33 [get_ports {an[0]]}
set_property PACKAGE_PIN U4 [get_ports {an[1]]}
set_property IOSTANDARD LVCMOS33 [get_ports {an[1]]}
set_property PACKAGE_PIN V4 [get_ports {an[2]]}
set_property IOSTANDARD LVCMOS33 [get_ports {an[2]]}
set_property PACKAGE_PIN W4 [get_ports {an[3]]}
set_property IOSTANDARD LVCMOS33 [get_ports {an[3]]}
```

Sumário

1 Circuitos aritméticos

2 Processos

3 Circuitos sequenciais

4 Contadores

5 Exercício Final

6 Dicas e Atalhos

Circuitos sequenciais

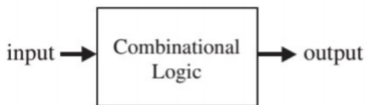
Circuitos sequenciais

Definição

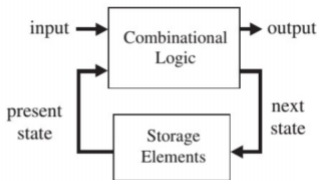
Circuitos sequenciais são circuitos digitais cuja(s) saída(s) depende(m) não somente das entradas atuais do circuito, mas também da sequência passada de entradas.

Circuitos sequenciais

- Esse tipo de circuito, diferentemente dos circuitos combinacionais, necessita armazenar informações referentes aos estados do sistema em alguma espécie de memória.



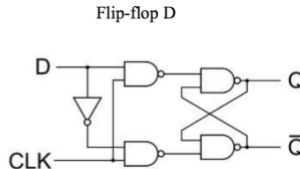
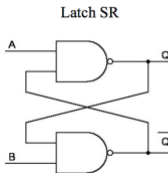
Lógica combinacional



Lógica sequencial

Circuitos sequenciais

- Os elementos de memória mais básicos em eletrônica digital são representados por dois tipos de componentes:
 - » *Latches*.
 - » *Flip-flops*.



Latches vs. Flip-flops

- *Latches* atualizam a saída quando a combinação das entradas *S* e *R* muda (*latch SR*) ou quando a entrada *Enable* é ativada (*latch D*).

Latches vs. Flip-flops

- *Latches* atualizam a saída quando a combinação das entradas *S* e *R* muda (*latch SR*) ou quando a entrada *Enable* é ativada (*latch D*).
- *Latches* são componentes assíncronos, não possuem um sinal que sincroniza a sua operação.

Latches vs. Flip-flops

- *Latches* atualizam a saída quando a combinação das entradas *S* e *R* muda (*latch SR*) ou quando a entrada *Enable* é ativada (*latch D*).
- *Latches* são componentes assíncronos, não possuem um sinal que sincroniza a sua operação.
- *Flip-flops* atualizam as saídas na borda de subida (*rising edge*) ou descida (*falling edge*) do sinal de *clock*.

Latches vs. Flip-flops

- *Latches* atualizam a saída quando a combinação das entradas *S* e *R* muda (*latch SR*) ou quando a entrada *Enable* é ativada (*latch D*).
- *Latches* são componentes assíncronos, não possuem um sinal que sincroniza a sua operação.
- *Flip-flops* atualizam as saídas na borda de subida (*rising edge*) ou descida (*falling edge*) do sinal de *clock*.
- *Flip-flops* são componentes síncronos.

Latches vs. Flip-flops

- Tanto *latches* quanto *flip-flops* são inferidos quando nem todas as condições de uma estrutura condicional são explícitas;

Latches vs. Flip-flops

- Tanto *latches* quanto *flip-flops* são inferidos quando nem todas as condições de uma estrutura condicional são explícitas;
- Se isso ocorre na borda de subida ou descida de um **sinal de clock**, um *flip-flop* é inferido;

Latches vs. Flip-flops

- Tanto *latches* quanto *flip-flops* são inferidos quando nem todas as condições de uma estrutura condicional são explícitas;
- Se isso ocorre na borda de subida ou descida de um *signal de clock*, um *flip-flop* é inferido;
- Quando deseja-se armazenar mais de um *bit* por meio de *flip-flops*, formam-se os registradores (conjuntos de *flip-flops*);

Latches vs. Flip-flops

- Tanto *latches* quanto *flip-flops* são inferidos quando nem todas as condições de uma estrutura condicional são explícitas;
- Se isso ocorre na borda de subida ou descida de um *signal de clock*, um *flip-flop* é inferido;
- Quando deseja-se armazenar mais de um *bit* por meio de *flip-flops*, formam-se os registradores (conjuntos de *flip-flops*);
- *Caso contrário*, infere-se um *latch*;

Latches vs. Flip-flops

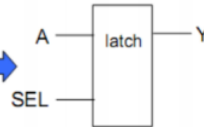

- Tanto *latches* quanto *flip-flops* são inferidos quando nem todas as condições de uma estrutura condicional são explícitas;
- Se isso ocorre na borda de subida ou descida de um **sinal de clock**, um *flip-flop* é inferido;
- Quando deseja-se armazenar mais de um *bit* por meio de *flip-flops*, formam-se os registradores (conjuntos de *flip-flops*);
- **Caso contrário**, infere-se um *latch*;
- *Latches* devem ser evitados pelo projetista, a não ser que o mesmo tenha domínio total da lógica do circuito;

Latches vs. Flip-flops

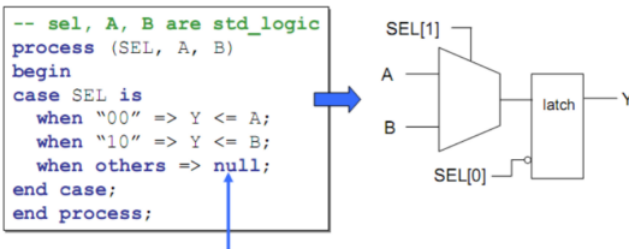
- Tanto *latches* quanto *flip-flops* são inferidos quando nem todas as condições de uma estrutura condicional são explícitas;
- Se isso ocorre na borda de subida ou descida de um *signal de clock*, um *flip-flop* é inferido;
- Quando deseja-se armazenar mais de um *bit* por meio de *flip-flops*, formam-se os registradores (conjuntos de *flip-flops*);
- *Caso contrário*, infere-se um *latch*;
- *Latches* devem ser evitados pelo projetista, a não ser que o mesmo tenha domínio total da lógica do circuito;
 - » Podem ser introduzidos problemas de temporização no sistema.

Latches vs. Flip-flops

```
process (SEL, A)
begin
  if (SEL = '1') then Y <= A;
  end if ;
end process;
```



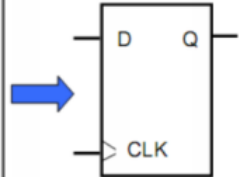
Latches vs. Flip-flops



Para evitar latches, defina a saída para as outras condições, por exemplo,
`when others => Y <= '0' ;`

Latches vs. Flip-flops

```
architecture BEHAVE of DF is
begin
  INFER: process (CLK) begin
    if (CLK'event and CLK = '1') then
      Q <= D;
    end if ;
  end process INFER;
end BEHAVE;
```



Sumário

1 Circuitos aritméticos

2 Processos

3 Circuitos sequenciais

4 Contadores

5 Exercício Final

6 Dicas e Atalhos

Contadores

- Circuitos que realizam modificam a saída de acordo com alguma sequência numérica.

Contadores

- Circuitos que realizam modificam a saída de acordo com alguma sequência numérica.
- A ordem da sequência não importa.

Contadores

- Circuitos que realizam modificam a saída de acordo com alguma sequência numérica.
- A ordem da sequência não importa.
- O número de estados do contador define o seu módulo.

Contadores

- Circuitos que realizam modificam a saída de acordo com alguma sequência numérica.
- A ordem da sequência não importa.
- O número de estados do contador define o seu módulo.
- Um contador pode ser visto como uma máquina de estados.

Contadores

- Circuitos fundamentais em aplicações que envolvem temporização de eventos;

Contadores

- Circuitos fundamentais em aplicações que envolvem temporização de eventos;
- Existem diversos tipos de contadores:

Contadores

- Circuitos fundamentais em aplicações que envolvem temporização de eventos;
- Existem diversos tipos de contadores:
 - » *Free-Running*: contagem crescente seguindo ordem numérica;

Contadores

- Circuitos fundamentais em aplicações que envolvem temporização de eventos;
- Existem diversos tipos de contadores:
 - » *Free-Running*: contagem crescente seguindo ordem numérica;
 - » *Up-down*: contagem crescente ou decrescente seguindo ordem numérica;

Contadores

- Circuitos fundamentais em aplicações que envolvem temporização de eventos;
- Existem diversos tipos de contadores:
 - » *Free-Running*: contagem crescente seguindo ordem numérica;
 - » *Up-down*: contagem crescente ou decrescente seguindo ordem numérica;
 - » Contador Johnson (anel);

Contadores

- Circuitos fundamentais em aplicações que envolvem temporização de eventos;
- Existem diversos tipos de contadores:
 - » *Free-Running*: contagem crescente seguindo ordem numérica;
 - » *Up-down*: contagem crescente ou decrescente seguindo ordem numérica;
 - » Contador Johnson (anel);
 - » Sequência especial.

Contadores

- Circuitos fundamentais em aplicações que envolvem temporização de eventos;
- Existem diversos tipos de contadores:
 - » *Free-Running*: contagem crescente seguindo ordem numérica;
 - » *Up-down*: contagem crescente ou decrescente seguindo ordem numérica;
 - » Contador Johnson (anel);
 - » Sequência especial.
- Um contador é geralmente associado a seu módulo: quantidade de valores diferentes assumidos durante a sequência de contagem.

Contadores

- Circuitos fundamentais em aplicações que envolvem temporização de eventos;
- Existem diversos tipos de contadores:
 - » *Free-Running*: contagem crescente seguindo ordem numérica;
 - » *Up-down*: contagem crescente ou decrescente seguindo ordem numérica;
 - » Contador Johnson (anel);
 - » Sequência especial.
- Um contador é geralmente associado a seu módulo: quantidade de valores diferentes assumidos durante a sequência de contagem.

Aplicações de contadores

- Algumas das principais aplicações de contadores incluem:
 - » Divisores de frequência;
 - » Circuitos de multiplexação;
 - » Temporizador.

Sumário

1 Circuitos aritméticos

2 Processos

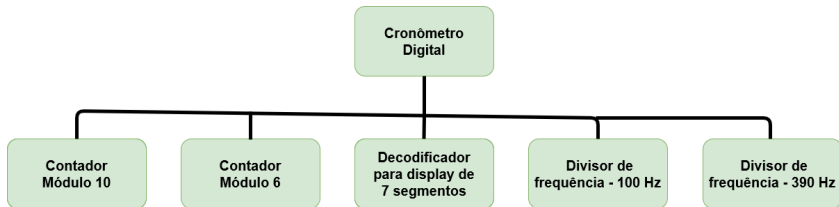
3 Circuitos sequenciais

4 Contadores

5 Exercício Final

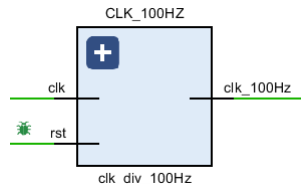
6 Dicas e Atalhos

Exercício Final: Cronômetro Digital



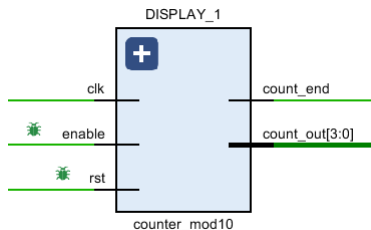
Exercício Final: Cronômetro Digital

- Divisão de frequência 100Hz.
- **Descrição:** Com base no sinal de *clk* original da Basys 3 (100 MHz) gera um sinal de *clk* de 100 Hz para contagem dos centésimos de segundo.



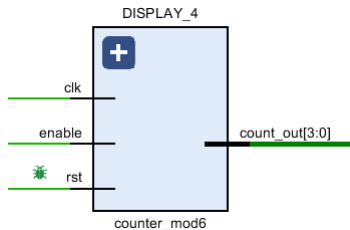
Exercício Final: Cronômetro Digital

- Contador de módulo 10.
- **Descrição:** Realiza a contagem de 0 a 9. É utilizado como displays dos centésimos e décimos de segundo, além da unidade dos segundos.



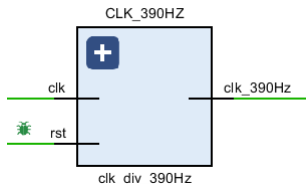
Exercício Final: Cronômetro Digital

- Contador de módulo 6.
- **Descrição:** Realiza a contagem de 0 a 5. É utilizado como display da dezena dos segundos.



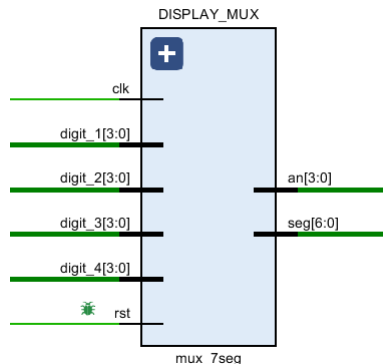
Exercício Final: Cronômetro Digital

- Divisão de frequência 390Hz.
- **Descrição:** Com base no sinal de *clk* original da Basys 3 (100 MHz) gera um sinal de *clk* de 390 Hz para multiplexação dos displays de 7 segmentos da Basys 3.



Exercício Final: Cronômetro Digital

- Controle dos displays.
- **Descrição:** Indica qual número deve ser exibido em cada display e realiza a multiplexação de displays.



45 / 50

Sumário

1 Circuitos aritméticos

2 Processos

3 Circuitos sequenciais

4 Contadores

5 Exercício Final

6 Dicas e Atalhos

Dicas e Atalhos no Vivado

- Você pode desassociar janelas do Vivado e maximizá-las em uma janela separada, com a opção **Float** no canto superior direito;
- Para voltar a janela destacada ao layout original, use a opção **Dock** no canto superior direito;
- Acessar esquemático RTL : **RTL Analysis -> Open Elaborated Design -> Schematic**;
- Acessar esquemático pós-síntese : **Synthesis -> Open Synthesized -> Schematic**;
- Acessar relatório de utilização de recursos de cada módulo pós-implementação: **Implementation -> Open Implemented Design -> Report Utilization**.

Dicas e Atalhos no Vivado

- Realizar simulação comportamental : **Simulation -> Run Simulation**
- Reiniciar a simulação e rodar por um período específico : **Restart, Run For;**
- Se você perceber que há um erro no seu circuito ao realizar a simulação, você pode modificar o código sem fechar a aba de simulação e usar a opção **Relaunch Simulation** para simular o código atualizado.

Dicas e Atalhos no Vivado

- Em caso de erro em qualquer etapa (simulação, síntese, implementação, geração de *bitstream*), as mensagens de erro aparecem no console do Vivado, na opção **Messages**.
- Além disso, o *pop up* de erro vai dizer onde foi gerado um arquivo *log*, que contem todas as mensagens geradas, incluindo as de erro;
- Se o erro for na síntese ou na implementação, o arquivo *log* estará na pasta do seu projeto, no subdiretório **nome__do__projeto.runs**, em **synth** ou **impl**;
- Se o erro for na simulação, o *log* estará no subdiretório **nome__do__projeto.sim -> sim -> behav -> xsim**, em **elabore.log** ou **simulate.log**.

Dicas e Atalhos no Vivado

- Atalhos úteis para codificação em VHDL no Vivado:
 - » **Ctrl + D**: Copia a linha atual logo abaixo da mesma;
 - » **Ctrl + Barra**: Comenta ou descomenta um trecho selecionado de código (pode ser feito também com o botão Toggle Line Comment na barra de ferramentas);
 - » **Ctrl + A**: Adicionar um novo arquivo ao projeto.

Fim da 2^a aula.

Muito obrigado pela atenção de todos.

Códigos e apresentação disponíveis aqui: <https://github.com/alceu-castanheira/Curso-VHDL-Eletronjun-2020>

