Relatório

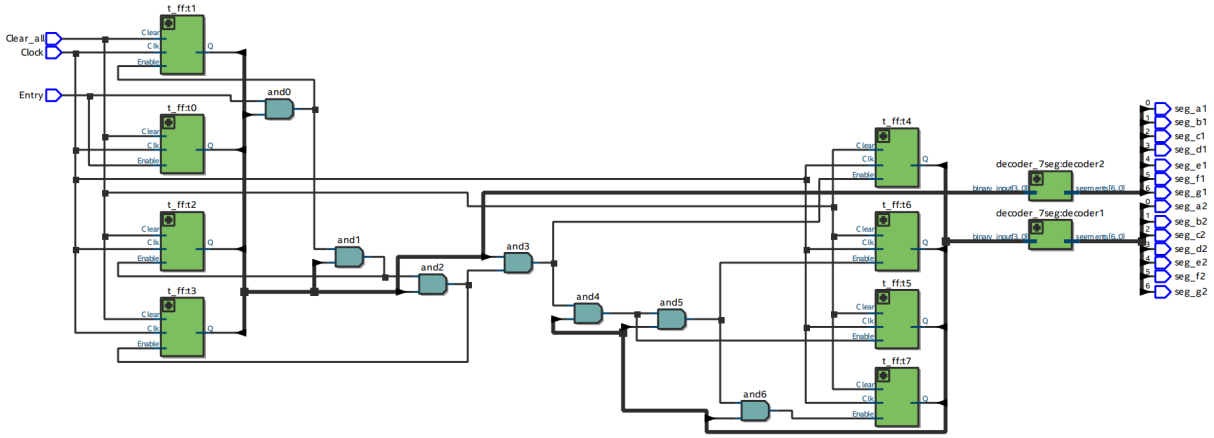
Tópico: Aulas 2 & 3 - Atividade com Latch e FF
Grupo: Adhemar Molon Neto 14687681
Alessandro Rodrigues Pereira da Silva 15436838
Link do Github: <>

Parte 1

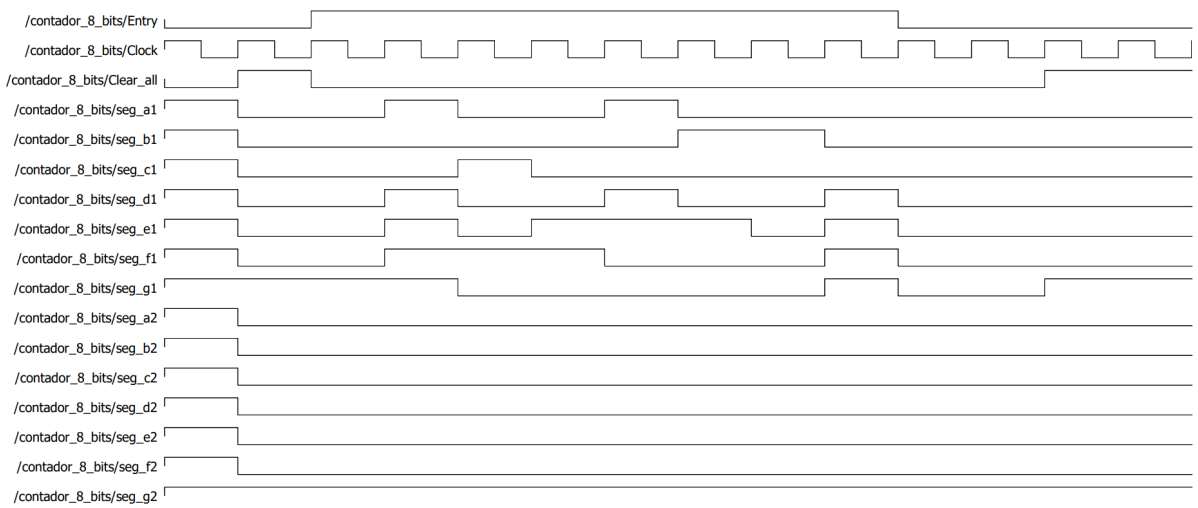
Descrição

Nessa parte, a ideia é criar um contador de 8 bits usando flip-flops do tipo T. A cada vez que o clock sobe (borda positiva), o contador incrementa, desde que o sinal de entrada esteja ativado. Além disso, se o sinal de reset for acionado, o contador volta para zero. O código VHDL basicamente monta esse contador instanciando oito flip-flops T, um para cada bit. Depois, o valor final do contador é dividido em dois blocos de 4 bits (nibbles), que são convertidos em sinais para serem mostrados em dois displays de 7 segmentos, facilitando a visualização do valor atual.

Circuito examinado usando *RTL Viewer*:



Simulação gerada pelo *ModelSim*:



Código VHDL:

```
entity contador_8_bits is
    port (
        Entry : in std_logic;
        Clock : in std_logic;
        Clear_all : in std_logic;
        -- Outputs for the 7-segment displays
        seg_a1 : out std_logic;
        seg_b1 : out std_logic;
        seg_c1 : out std_logic;
        seg_d1 : out std_logic;
        seg_e1 : out std_logic;
        seg_f1 : out std_logic;
        seg_g1 : out std_logic;
        seg_a2 : out std_logic;
        seg_b2 : out std_logic;
        seg_c2 : out std_logic;
        seg_d2 : out std_logic;
        seg_e2 : out std_logic;
        seg_f2 : out std_logic;
        seg_g2 : out std_logic
    );
end entity contador_8_bits;
```

Aqui estamos declarando a entidade do contador de 8 bits. Ela define três sinais de entrada:

- **Entry**: sinal de habilitação, que determina se o contador deve contar ou não.

- **Clock**: o sinal de clock que controla a temporização do contador.
- **Clear_all**: usado para resetar o contador.

Também temos as saídas que serão conectadas aos displays de 7 segmentos, mapeadas como seg_a1 até seg_g2.

```
t0 : t_ff port map(
    Enable => Entry,
    Clk => Clock,
    Clear => Clear_all,
    Q => out0
);

and0 <= out0 and Entry;

t1 : t_ff port map(
    Enable => and0,
    Clk => Clock,
    Clear => Clear_all,
    Q => out1
);

and1 <= out1 and and0;
```

O contador é construído instanciando oito flip-flops T, um para cada bit do contador. Cada flip-flop T só "vira" se o anterior também estiver ativado, o que é garantido pelo uso dos sinais andX. Isso simula o comportamento de um contador síncrono em cascata.

No exemplo acima:

- **t0**: é o primeiro flip-flop, controlado diretamente pelo sinal de habilitação.
- **and0**: garante que o próximo flip-flop (t1) só seja ativado quando t0 estiver ativado e a entrada for alto.
-

```
high_nibble <= out7 & out6 & out5 & out4;  -- High nibble
low_nibble  <= out3 & out2 & out1 & out0;  -- Low nibble
```

O contador de 8 bits é dividido em dois grupos de 4 bits (chamados nibbles). O nibble mais significativo (high_nibble) vai

dos bits 7 a 4, e o nibble menos significativo (low_nibble) vai dos bits 3 a 0. Cada um desses grupos será exibido em um display de 7 segmentos.

```
decoder1 : decoder_7seg port map(
    binary_input => high_nibble,
    segments     => segs1
);

decoder2 : decoder_7seg port map(
    binary_input => low_nibble,
    segments     => segs2
);
```

Para exibir os números binários nos displays de 7 segmentos, o VHDL usa dois decodificadores. Cada decodificador converte um dos nibbles (de 4 bits) em sinais que acendem os segmentos corretos nos displays de 7 segmentos.

- **decoder1**: Decodifica o nibble mais significativo.
- **decoder2**: Decodifica o nibble menos significativo.

```
-- Map the segments of the first display

seg_a1 <= segs2(0);

seg_b1 <= segs2(1);

seg_c1 <= segs2(2);

seg_d1 <= segs2(3);

seg_e1 <= segs2(4);

seg_f1 <= segs2(5);

seg_g1 <= segs2(6);


-- Map the segments of the second display

seg_a2 <= segs1(0);

seg_b2 <= segs1(1);

seg_c2 <= segs1(2);

seg_d2 <= segs1(3);
```

```
seg_e2 <= segs1(4);

seg_f2 <= segs1(5);

seg_g2 <= segs1(6);
```

Aqui ocorre o mapeamento dos sinais do decodificador para os segmentos dos displays de 7 segmentos. Cada segmento (de 'a' até 'g') corresponde a uma parte do display e é ativado conforme a saída do decodificador.

Parte 2

Descrição:

Nessa parte, o objetivo é implementar um contador de 16 bits que divide seu valor em quatro nibbles de 4 bits cada, para exibição em quatro displays de 7 segmentos. O contador incrementa a cada pulso de clock na borda ascendente, desde que o sinal de habilitação esteja ativo. Quando o sinal de reset é acionado, o contador é zerado. O código VHDL realiza a contagem usando uma arquitetura comportamental e instanciando um componente de decodificação para os displays. Após a contagem, o valor é dividido em quatro nibbles, que são enviados para os respectivos displays, permitindo uma visualização clara do valor do contador em formato decimal.

Código VHDL:

```
entity part2 is
    Port (
        clk    : in  STD_LOGIC;
        enable : in  STD_LOGIC;
        clear  : in  STD_LOGIC;
        seg1   : out STD_LOGIC_VECTOR(6 downto 0); -- Display 1
        seg2   : out STD_LOGIC_VECTOR(6 downto 0); -- Display 2
        seg3   : out STD_LOGIC_VECTOR(6 downto 0); -- Display 3
        seg4   : out STD_LOGIC_VECTOR(6 downto 0)  -- Display 4
    );
end part2;
```

A entidade part2 define a interface do módulo, ou seja, as entradas e saídas:

- **clk**: O sinal de clock que controla o ritmo do contador.
- **enable**: Entrada que habilita ou desabilita a contagem.
- **clear**: Sinal de reset que, quando ativado, zera o contador.
- **seg1, seg2, seg3, seg4**: Saídas para os displays de 7 segmentos, controlados pelo contador de 16 bits.

```
component display
    Port (
        bin   : in  STD_LOGIC_VECTOR(3 downto 0);
        seg   : out STD_LOGIC_VECTOR(6 downto 0)
    );
end component;


signal Q       : STD_LOGIC_VECTOR(15 downto 0) := (others => '0');
signal nibble1 : STD_LOGIC_VECTOR(3 downto 0);
signal nibble2 : STD_LOGIC_VECTOR(3 downto 0);
signal nibble3 : STD_LOGIC_VECTOR(3 downto 0);
signal nibble4 : STD_LOGIC_VECTOR(3 downto 0);
```

O componente **display** define um decodificador de 7 segmentos que converte um valor binário de 4 bits em sinais adequados para um display de 7 segmentos.

O sinal **Q** é o registrador de 16 bits que armazena o valor atual do contador.

Os sinais **nibble1, nibble2, nibble3, nibble4** armazenam cada parte do contador de 4 bits (nibbles), que são enviados aos decodificadores.

```
process(clk, clear)
begin
    if clear = '1' then
        Q <= (others => '0'); -- Reseta o contador
    elsif rising_edge(clk) then
        if enable = '1' then
            Q <= std_logic_vector(unsigned(Q) + 1); -- Incrementa
o contador
```

```
        end if;
    end if;
end process;
```

Este processo é responsável por controlar o contador:

- **Reset**: Se o sinal `clear` for ativado (`'1'`), o contador `Q` é zerado.
- **Clock**: A cada borda de subida do clock (`rising_edge(clk)`), o contador é incrementado se o sinal `enable` estiver ativado (`'1'`).
- A operação de incremento usa a conversão para `unsigned` para adicionar 1 ao valor binário de `Q`.

```
nibble1 <= Q(3 downto 0);
nibble2 <= Q(7 downto 4);
nibble3 <= Q(11 downto 8);
nibble4 <= Q(15 downto 12);
```

Aqui, o valor de 16 bits do contador `Q` é dividido em quatro partes de 4 bits:

- **nibble1**: Os 4 bits menos significativos de `Q` (bits 0 a 3).
- **nibble2**: Bits 4 a 7 de `Q`.
- **nibble3**: Bits 8 a 11 de `Q`.
- **nibble4**: Os 4 bits mais significativos de `Q` (bits 12 a 15).

Esses nibbles serão usados para controlar cada um dos quatro displays de 7 segmentos.

```
U1: display port map (bin => nibble1, seg => seg1);
U2: display port map (bin => nibble2, seg => seg2);
U3: display port map (bin => nibble3, seg => seg3);
U4: display port map (bin => nibble4, seg => seg4);
```

Cada nibble é enviado para uma instância do componente **display**, que converte o valor binário de 4 bits em sinais que controlam um display de 7 segmentos. As saídas **seg1** a **seg4** correspondem a cada display de 7 segmentos.

Parte 3

Descrição

Código VHDL

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

use IEEE.NUMERIC_STD.ALL;

ENTITY part3comvar is
	port(
		Enable, Clk, Clear	: in std_logic;
		hex0                            : out std_logic_vector(6
downto 0)
	);
END part3comvar;
ARCHITECTURE main of part3comvar is
COMPONENT hex_display IS
	port(
		num4  : in std_logic_vector(3 downto 0);
		hex   : out std_logic_vector(6 downto 0)
	);
END COMPONENT;
signal Qvec : std_logic_vector(25 downto 0);
signal cont : std_logic_vector(3 downto 0);
signal num  : std_logic_vector(3 downto 0);
begin
	process (Clk) is
	begin
	if(rising_edge(Clk)) then
		if(Clear = '0') then
			Qvec <= "00000000000000000000000000";
			cont <= "0000";
		elsif (Enable = '1') then
			if(unsigned(Qvec) < 50000000) then
				Qvec <= std_logic_vector(unsigned(Qvec) + 1);
			else
				Qvec <= "00000000000000000000000000";
				if(unsigned(cont) < 9) then
					cont <= std_logic_vector(unsigned(cont)
+ 1);
```

```
                    else
                         cont <= "0000";
                    end if;
               end if;
          end if;
     end if;
     end process;
     num <= cont;
     hd0 : entity work.hex_display
          port map(
               num4 => num,
               hex => hex0
          );


end main;
```

Parte 4

Descrição

O objetivo da Parte 4 do exercício é a confecção e implementação de um circuito que mostra o modelo da sua placa, em 4 display's de 7 segmentos, permutando para a esquerda, ciclicamente a cada 1 segundo. A placa utilizada foi a DE0-CV, então a palavra impressa foi "dE0", em 4 displays de 7 segmentos que alternam entre as letras.

Código VHDL

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

use IEEE.NUMERIC_STD.ALL;

ENTITY part4 is
     port(
          Enable, Clk, Clear    : in std_logic;
          hex0,hex1,hex2,hex3   : out std_logic_vector(6 downto 0)
     );
END part4;
ARCHITECTURE main of part4 is
signal Tempo : std_logic_vector(25 downto 0);
signal cont : std_logic_vector(1 downto 0);

type ARRAY_PALAVRA is array (0 to 3) of std_logic_vector(6 downto
0);
constant word : ARRAY_PALAVRA := (0 => "1111111", 1 => "0100001",
2 => "0000110", 3 => "1000000");

begin
     process (Clk) is
```

```
	begin
	if(rising_edge(Clk)) then
		if(Clear = '0') then
			Tempo <= "00000000000000000000000000";
			cont <= "00";
		elsif (Enable = '1') then
			if(unsigned(Tempo) < 50000000) then
				Tempo <= std_logic_vector(unsigned(Tempo) +
1);
			else
				Tempo <= "00000000000000000000000000";
				if(unsigned(cont) < 3) then
					cont <= std_logic_vector(unsigned(cont)
+ 1);
				else
					cont <= "00";
				end if;
			end if;
		end if;
	end if;
	end process;
	hex3 <= word(to_integer(unsigned(cont)));
	hex2 <= word(to_integer(unsigned(cont) + 1) mod 4);
	hex1 <= word(to_integer(unsigned(cont) + 2) mod 4);
	hex0 <= word(to_integer(unsigned(cont) + 3) mod 4);


end main;
```

Parte 5

Descrição

O objetivo da Parte 5, assim como da 4, é a confecção e implementação de um circuito que mostra o modelo da sua placa, porém agora em 6 display's de 7 segmentos. A placa utilizada também foi a DE0-CV, então a palavra impressa é "dE0".

Código VHDL

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

use IEEE.NUMERIC_STD.ALL;

ENTITY part5 is
	port(
		Enable, Clk, Clear		: in std_logic;
		hex0,hex1,hex2,hex3,hex4,hex5	: out std_logic_vector(6
downto 0)
	);
```

```
END part5;
ARCHITECTURE main of part5 is
signal Tempo : std_logic_vector(25 downto 0);
signal cont : std_logic_vector(2 downto 0);

type ARRAY_PALAVRA is array (0 to 5) of std_logic_vector(6 downto
0);
constant word : ARRAY_PALAVRA := (0 => "1111111", 1 => "1111111",
2 => "1111111", 3 => "0100001", 4 => "0000110", 5 => "1000000");

begin
     process (Clk) is
     begin
     if(rising_edge(Clk)) then
          if(Clear = '0') then
               Tempo <= "00000000000000000000000000";
               cont <= "000";
          elsif (Enable = '1') then
               if(unsigned(Tempo) < 50000000) then
                    Tempo <= std_logic_vector(unsigned(Tempo) +
1);
               else
                    Tempo <= "00000000000000000000000000";
                    if(unsigned(cont) < 5) then
                         cont <= std_logic_vector(unsigned(cont)
+ 1);
                    else
                         cont <= "000";
                    end if;
               end if;
          end if;
     end if;
     end process;
     hex5 <= word(to_integer(unsigned(cont)));
     hex4 <= word((to_integer(unsigned(cont)) + 1) mod 6);
     hex3 <= word((to_integer(unsigned(cont)) + 2) mod 6);
     hex2 <= word((to_integer(unsigned(cont)) + 3) mod 6);
     hex1 <= word((to_integer(unsigned(cont)) + 4) mod 6);
     hex0 <= word((to_integer(unsigned(cont)) + 5) mod 6);


end main;
```