Timer de Segundos com Mostrador SSD

Eletrônica Digital II

Bernardo Souza Muniz, Ygor Martins e Germano Coelho.

03 de Julho de 2025

Engenharia de Telecomunicações - IFSC-SJ

Sumário

1. Introdução

O objetivo deste documento é apresentar a implementação de um timer de segundos utilizando a linguagem ***VHDL (VHSIC Hardware Description Language)*** que realiza a contagem de 00 até DU (Dezena e unidade) atraveś de dois displays de sete segmentos. Todo o projeto será estruturado no modelo hierárquico, onde terá uma classe Top Level que faz a instanciação dos demais componentes presentes no programa. Além do desenvolvimento do código, são apresentados dados de verificação de desempenho, como a frequência máxima (Fmax), número de pinos utilizados e quantidade de elementos lógicos consumidos. A plataforma de desenvolvimento utilizada foi o Quartus Prime e a simulação foi realizada com o ModelSim.

2. Contador BCD

O contador BCD tem como objetivo realizar a contagem do tempo, esta contagem por sua vez é determinada através de dois parâmetros genéricos, D (Dezena máxima) e U (Unidade máxima).

Além disso, o a descrição de hardware vem consigo uma configuração com relação a forma que o contador se comporta. Onde este através de um parâmetro genérico pode realizar o overflow da contagem, ou, parar a contagem no valor máximo definido préviamente.

Vejamos agora separadamente, tanto sua entidade quanto sua arquitetura:

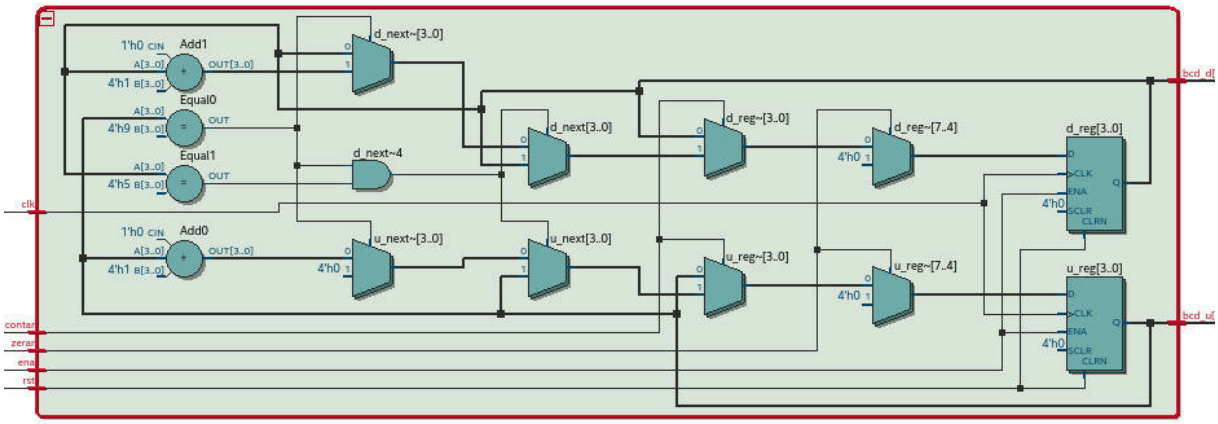
```
entity contaBCD is
   generic (D: natural :=5; U : natural := 9; tipo : natural := 1); --0
=> overflow, 1 => parada
  port (
    clk, rst : in std_logic;
    ena   : in std_logic;
    contar  : in std_logic;
    zerar   : in std_logic;
    bcd_d   : out std_logic_vector(3 downto 0);
    bcd_u   : out std_logic_vector(3 downto 0)
  );
end entity;
```

```
architecture rtl of bcd2ssd is
    signal seg, seg_final : std_logic_vector(6 downto 0);
   signal bcd_aux : std_logic_vector(3 downto 0);
begin
    process(bcd_in, oculta_zero, bcd_aux, seg)
    begin

   bcd_aux <= bcd_in;

        case bcd_aux is
            when "0000" => seg <= "0000001"; -- Padrão mais comum
(segmento 'a' é o LSB)
            when "0001" => seg <= "1001111";
            when "0010" => seg <= "0010010";
```

```
            when "0011" => seg <= "0000110";
            when "0100" => seg <= "1001100";
            when "0101" => seg <= "0100100";
            when "0110" => seg <= "0100000";
            when "0111" => seg <= "0001111";
            when "1000" => seg <= "0000000";
            when "1001" => seg <= "0000100";
            when others => seg <= "1111111";
        end case;

        if oculta_zero = '1' and bcd_aux = "0000" then
            seg_final <= (others => '1');
        else
            seg_final <= seg;
        end if;
    end process;

    anode_common: if tipo = 0 generate
        ssd_out <= seg_final;
    end generate;

    cathode_common: if tipo = 1 generate
        ssd_out <= not seg_final;
    end generate;
end architecture;
```

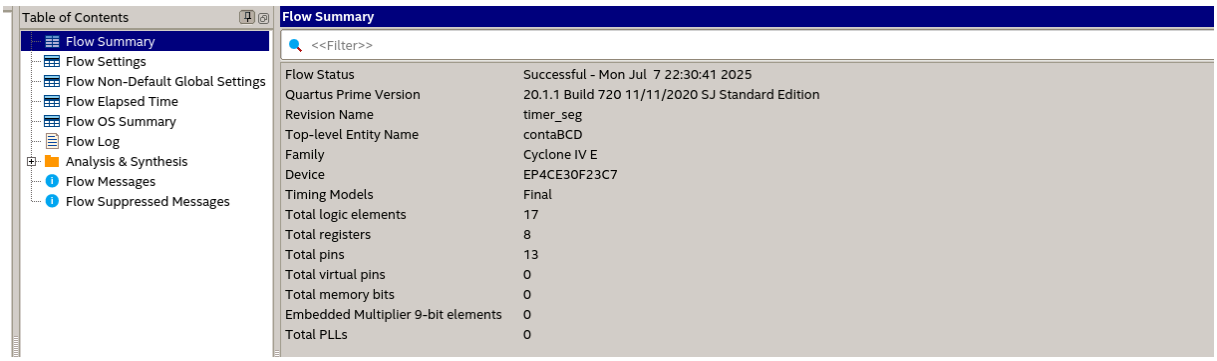
Como resultado desta descrição o seguinte RTL foi obtido:

Figura 1: Elaborada pelo Autor



RTL Viewer do contador BCD

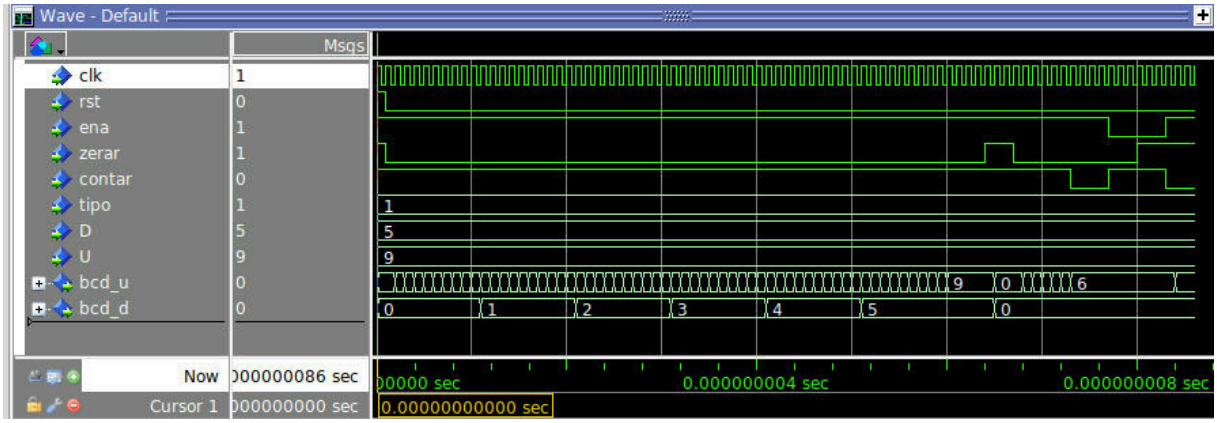
Figura 2: Elaborada pelo Autor



Sumário do contador BCD

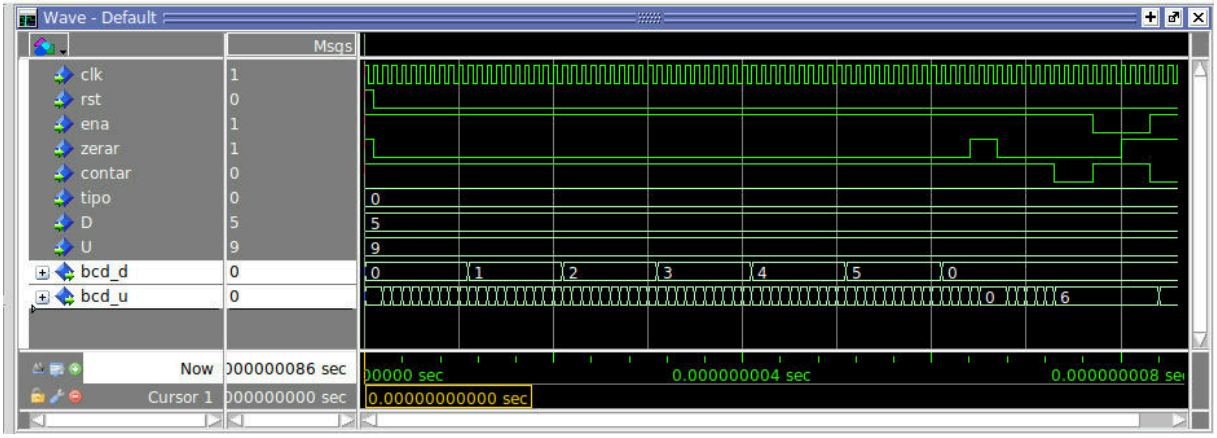
E com este circuito, através de um arquivo .do, é possível obter duas simulações, uma com critério de parada, e a outra com overflow, vejamos ambas conforme a ordem dita acima:

Figura 3: Elaborada pelo Autor



Simulação com critério de parada

Figura 4: Elaborada pelo Autor



Simulação com overflow

3. Divisor de clock

O objetivo principal do circuito divisor de clock dentro do projeto do timer de segundos, é gerar um pulso de habilitação, descrito como `ena_out` na entidade, para o contador a cada 1 segundo a partir do clock disponível no kit de implementação. O pulso de habilitação dura 20 ns.

```
entity div_clk is
    generic (Nbits : natural := 6; fclk : integer := 50); #Apenas para simulacao
    port(
      clk: in std_logic;
      reset :in std_logic;
      ena_out : out std_logic;
      clk_out: out std_logic
    );
end entity;
```

Abaixo é possível ver a descrição da arquitetura do projeto separada em dois segmentos: combinacional e sequencial.

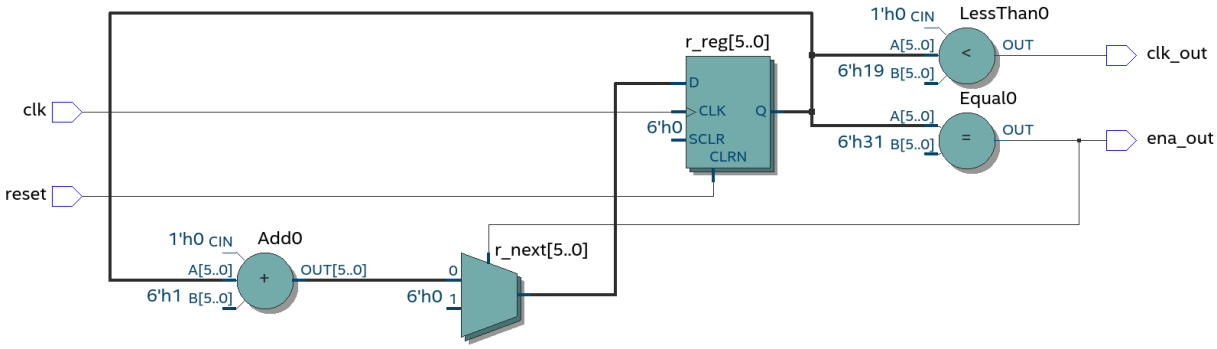
```
architecture clk_division of div_clk is
  signal r_reg, r_next : unsigned (Nbits-1 downto 0);
begin
    process(clk_in,reset)
    begin
        if (reset = '1') then
            r_reg <= (others => '0');
        elsif rising_edge(clk_in) then
            r_reg <= r_next;
        end if;
    end process;
    r_next <= (others => '0') when r_reg = (fclk-1) else r_reg + 1;
    clk_out <= '1' when r_reg < fclk/2 else '0';
    ena_out <= '1' when r_reg=fclk-1 else '0';
end architecture;
```

Na lógica sequencial, temos que o reset é assíncrono, por mais que esteja dentro de um processo que dependa da entrada `clk_in`, sua função é fora da borda de subida do clock. Quando `reset` é ativado, o contador é zerado, se não, registra a próxima contagem feita no segmento combinacional.

Na lógica combinacional, `r_next` conta de de 0 até `fclk - 1`, quando essa entrada é zerada, temos que o ciclo dividico de `clk_out` chegou ao fim, e no próximo ciclo ele começa zerado. A saída `clk_out` atua como um clock dividido, gerando um sinal de clock mais lento com duty cycle de 50%. Por fim, a saída `ena_out` recebe 1 quando um ciclo do clock divido terminou.

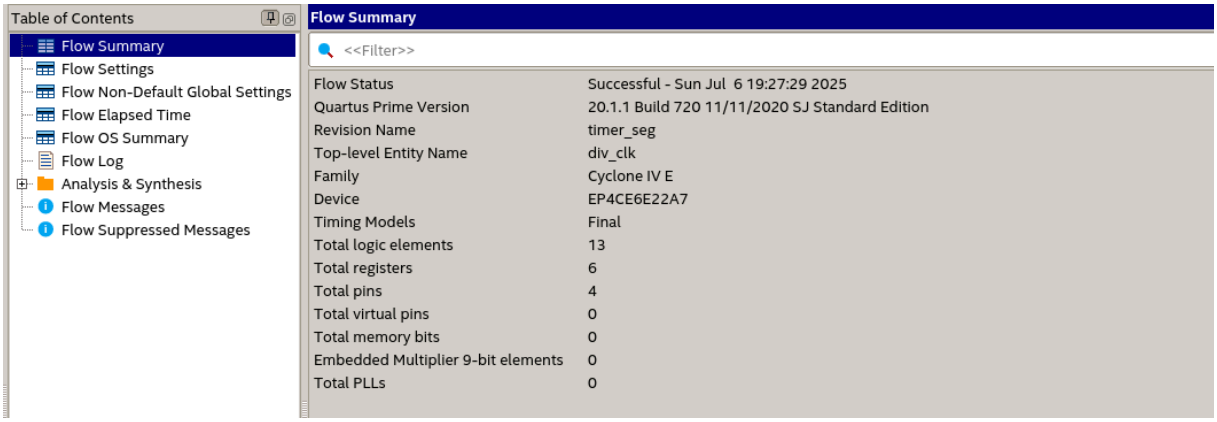
Após fazer a análise de síntese do projeto, foram obtidos os seguintes resultado:

Figura 5: Elaborada pelo Autor



RTL Viewer do divisor de clock

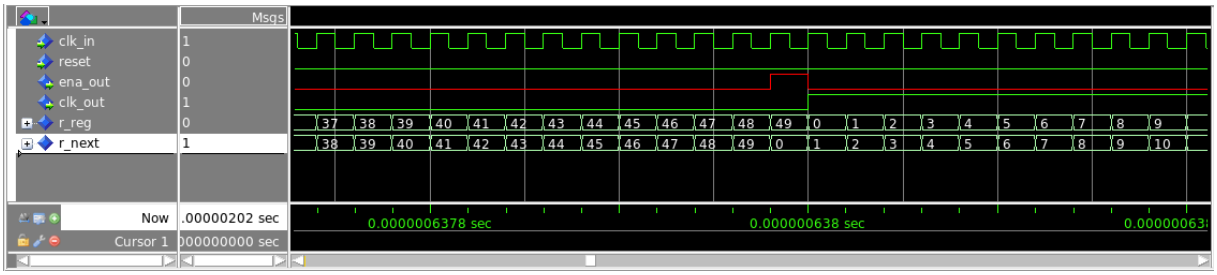
Figura 6: Elaborada pelo Autor



Sumário do divisor de clock

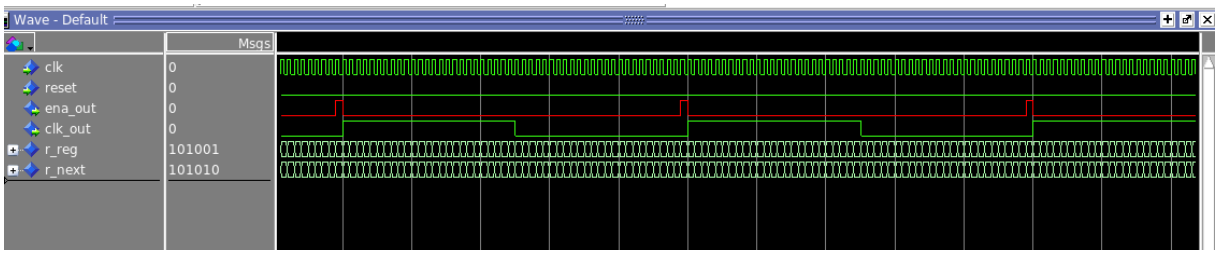
Ao realizar a simulação, foi utilizado um clock de 20 ns e simulado com frequência de 50Hz, neste caso, o parâmetro genérico `fclk` presente na entidade foi definido com valor de 50.

Figura 7: Elaborada pelo Autor



Sumário do divisor de clock

Figura 8: Elaborada pelo Autor



Simulação do divisor de clock

Nota-se que o sinal `r_reg` conta até `fclk - 1` e quando esse evento acontece, o sinal de enable é ativado e `r_reg` zera novamente, demonstrado o comportamento esperado do circuito.

4. Conversor de BCD para SSD

O conversor BCD para Display de Sete Segmentos (SSD) é um componente essencial no projeto do timer, responsável por traduzir os valores numéricos em binário (BCD - Binary Coded Decimal) para os padrões de acionamento dos displays de 7 segmentos.

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity bcd2ssd is
5      #tipo 0 => display anodo comum
6      #tipo 1 => display catodo comum
7      generic (tipo : natural := 1);
8      port (
9          oculta_zero : in std_logic;  -- Nome mais descritivo
10         bcd_in      : in std_logic_vector(3 downto 0);
11         ssd_out     : out std_logic_vector(6 downto 0)
12     );
13 end entity;
```

Como pode ser visto acima, se encontra a entidade do projeto do projeto, onde possui um generic na qual define a configuração do display - valor 0 para ânodo comum (segmentos acionados em nível baixo) ou 1 para cátodo comum (segmentos acionados em nível alto), sendo este último o padrão.

Uma porta oculta_zero que é um sinal de controle que, quando ativado (‘1’), desliga todos os segmentos quando a entrada BCD for zero, útil para eliminar zeros não significativos em visualizações numéricas.

Duas portas de dados: bcd_in e bcd_out.

bdc_in (4 bits): Recebe valores decimais codificados em binário (0-9)

bcd_out (7 bits): Controla individualmente cada segmento (a-g) do display, onde cada bit corresponde a um segmento específico.

```
  architecture rtl of bcd2ssd is
    signal seg, seg_final : std_logic_vector(6 downto 0);
begin
    process(bcd_in, oculta_zero)
    begin
        case bcd_in is
            when "0000" => seg <= "0000001";
            when "0001" => seg <= "1001111";
            when "0010" => seg <= "0010010";
            when "0011" => seg <= "0000110";
            when "0100" => seg <= "1001100";
            when "0101" => seg <= "0100100";
            when "0110" => seg <= "0100000";
            when "0111" => seg <= "0001111";
            when "1000" => seg <= "0000000";
            when "1001" => seg <= "0000100";
            when others => seg <= "1111111";
        end case;

        if oculta_zero = '1' and bcd_in = "0000" then
            seg_final <= (others => '1');
        else
            seg_final <= seg;
        end if;
    end process;

    -- Seleção do tipo de display usando IF GENERATE
    anode_common: if tipo = 0 generate
        ssd_out <= seg_final;
    end generate;

    cathode_common: if tipo = 1 generate
        ssd_out <= not seg_final;
    end generate;
end architecture;
```

A arquitetura RTL implementa a lógica de conversão do código BCD para o padrão de 7 segmentos, com três estágios principais de processamento.

- **Conversão BCD-Segmentos:**

Um bloco combinacional case mapeia cada valor BCD (0-9) para seu padrão correspondente de segmentos. Padrões definidos para display de ânodo comum (ativo em baixo). Exemplo: “0000” (0) ⟶ “0000001” (apenas segmento g desligado). Valores inválidos (≥10) desligam todos segmentos (“1111111”).

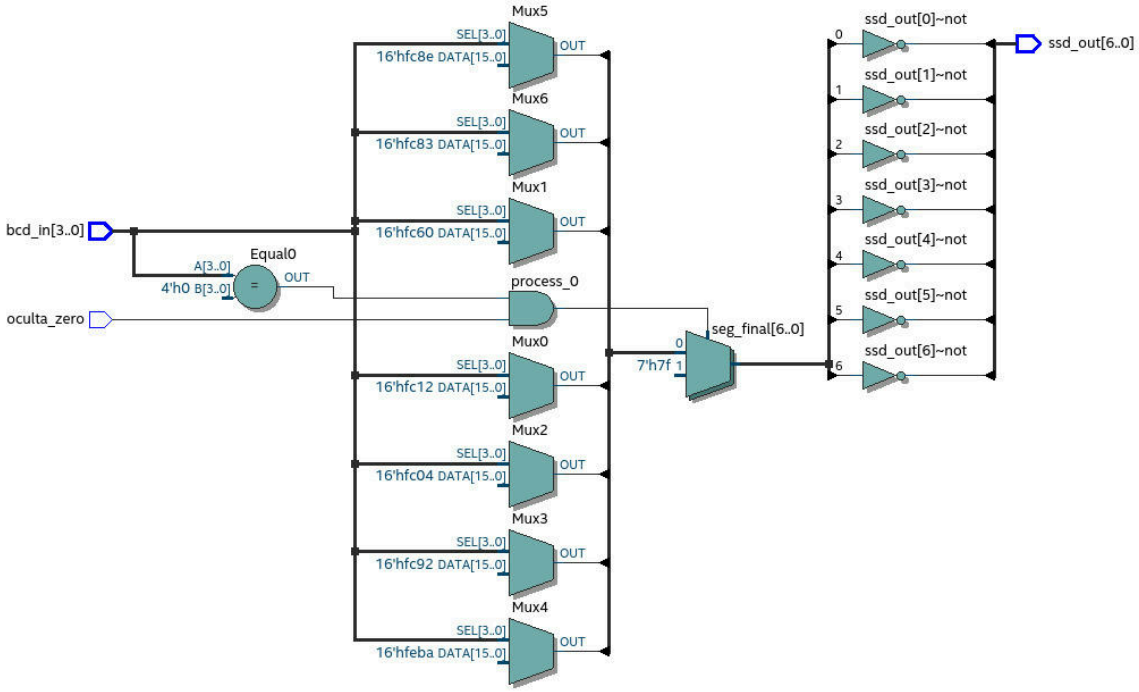
- **Controle de Ocultação de Zero:**

Lógica condicional verifica oculta_zero=‘1’ e valor BCD zero. Quando ativo, força todos segmentos para ‘1’ (desligado). Mantém a conversão normal em outros casos

- **Adaptação para Tipo de Display:**

Uso de if generate para selecionar a polaridade. Ânodo comum (tipo=0): saída direta (seg_final). Cátodo comum (tipo=1): saída invertida (not seg_final). Permite reutilização do mesmo código para ambos tipos de display

Figura 9: Elaborada pelo Autor



RTL do bcd2ssd

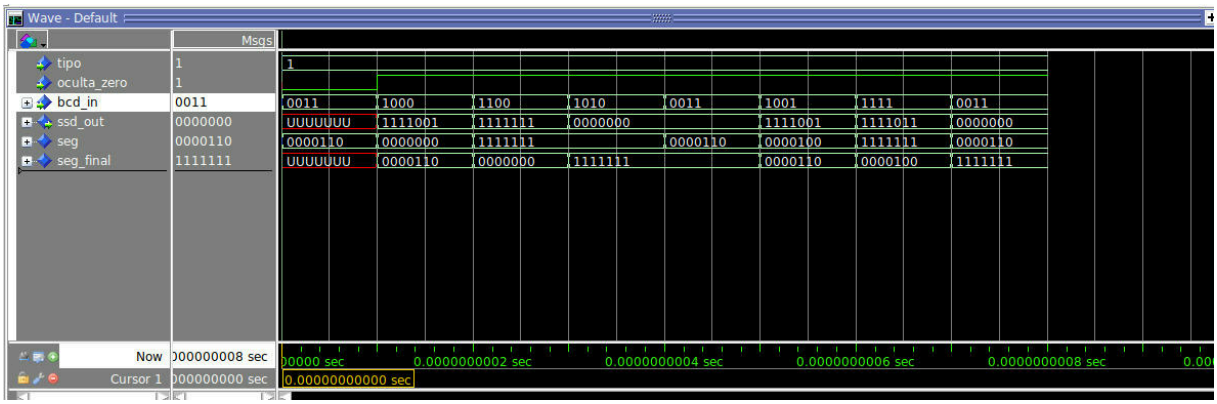
Acima, podemos ver o RTL do bcd2ssd.

Figura 10: Elaborada pelo Autor

Sumário do bcd2ssd

Na figura acima, podemos ver o sumário do circuito bcd2ssd, onde encontra-se 15 elementos lógicos, 0 registradores e 12 pinos.

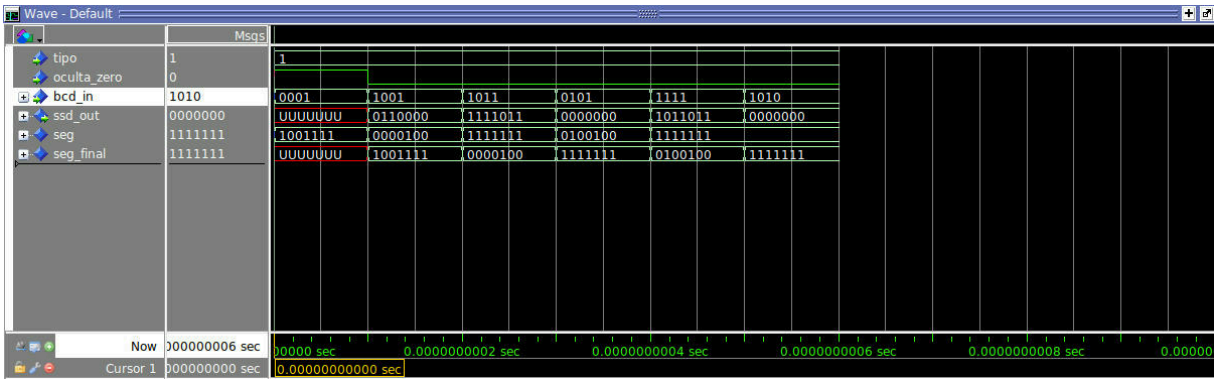
Figura 11: Elaborada pelo Autor



Simulação com ocultando o zero

Na figura acima, podemos observar a simulação do bcd2ssd com o oculta zero em 1.

Figura 12: Elaborada pelo Autor



Simulação sem ocultando o zero

Na figura acima, podemos observar a simulação do bcd2ssd com o oculta zero em 0.

5. Entidade Top Level

Para fazer a junção de todos os circuitos presentes no projeto, foi criada uma entidade Top--Level chamada **`timer_seg.vhd`**. Nela foram instanciados todos os componentes para que o timer de segundos funcione corretamente. No código abaixo é possível verificar a entidade do arquivo:

```
1  entity timer_seg is
2     generic (DEZENA : natural :=5; UNIDADE : natural := 9; FCLOCK :
   natural := 50);
3    port (
4      CLOCK50MHz    : in std_logic;
5      RESET_PB    : in std_logic;
6      CONTAR_SW     : in std_logic;
7      ZERAR_PB      : in std_logic;
8      LED_1SEC      : out std_logic;
9      SSD_DEZENA    : out std_logic_vector(6 downto 0);
10     SSD_UNIDADE   : out std_logic_vector(6 downto 0)
11   );
```

```
12 end entity;
```

Nos parâmetros genéricos foram declarados variáveis que podem ser mudadas conforme a necessidade do contador. Inicialmente foi definido um contador de 00 até 59, onde a dezena e a unidade são atribuídas pelas variáveis naturais **DEZENA** e **UNIDADE**. Além disso, para as simulações foi utilizado um clock de 50Hz, onde a frequência pode ser controlada por **FCLOCK**.

Para fazer a instância de todos os componentes presentes no circuito, foi utilizado uma arquitetura com três sinais internos e utilizado o comando **component**.

```
1  architecture ifsc_de2_115 of timer_seg is
2    -
3    -
4    #DECLARAÇÃO DOS COMPONENTES UTILIZADOS
5    -
6    -
7    signal ENABLE_1SEC : std_logic;
8    signal BCD_UNIDADE : std_logic_vector(3 downto 0); --out
9    signal BCD_DEZENA  : std_logic_vector(3 downto 0); --out
10 begin
```

Os sinais **BCD_UNIDADE** e **BCD_DEZENA** são utilizados na instância do componente **bcd2ssd**, sendo uma instância para dezena e outra para unidade. Além disso , o sinal de enable é conectado com a saída **ena_out** do divisor de clock para que seja recebido o pulso de habilitação.

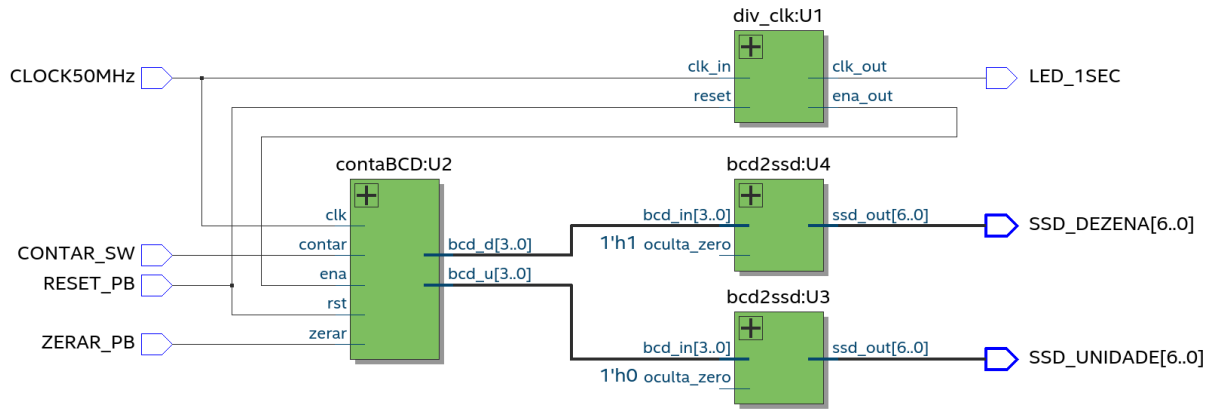
```
1    architecture ifsc_de2_115 of timer_seg is
2     -
3    -
4    #DECLARAÇÃO DOS COMPONENTES E SINAIS...
5    -
6    -
7    begin
8    U1: div_clk generic map (fclk => FCLOCK) --Divisor de clock
9      port map (clk_in => CLOCK50MHz,
10             reset => RESET_PB,
11             clk_out => LED_1SEC,
12             ena_out => ENABLE_1SEC);
13
14   U2 : contaBCD generic map (D => DEZENA, U=> UNIDADE) --Contador BCD
15     port map (clk => CLOCK50MHz,
16           contar => CONTAR_SW,
17           zerar => ZERAR_PB,
18           rst => RESET_PB,
19           ena => ENABLE_1SEC,
20           bcd_u => BCD_UNIDADE,
21           bcd_d => BCD_DEZENA);
22
23      #convertendo unidade de BCD p/ SSD
24   U3 : bcd2ssd
25       port map (bcd_in => BCD_UNIDADE,
26             ssd_out => SSD_UNIDADE,
27             oculta_zero => '0'
```

```
28              );
29          #convertendo dezena de BCD p/ SSD
30      U4 : bcd2ssd
31          port map (bcd_in => BCD_DEZENA,
32                ssd_out => SSD_DEZENA,
33                oculta_zero => '1'
34                );
35  end architecture;
```

O componente **contaBCD** foi projetado para ter overflow quando chegar no falor final controlado pelas variáveis genéricas **UNIDADE** e **BCD_DEZENA**.

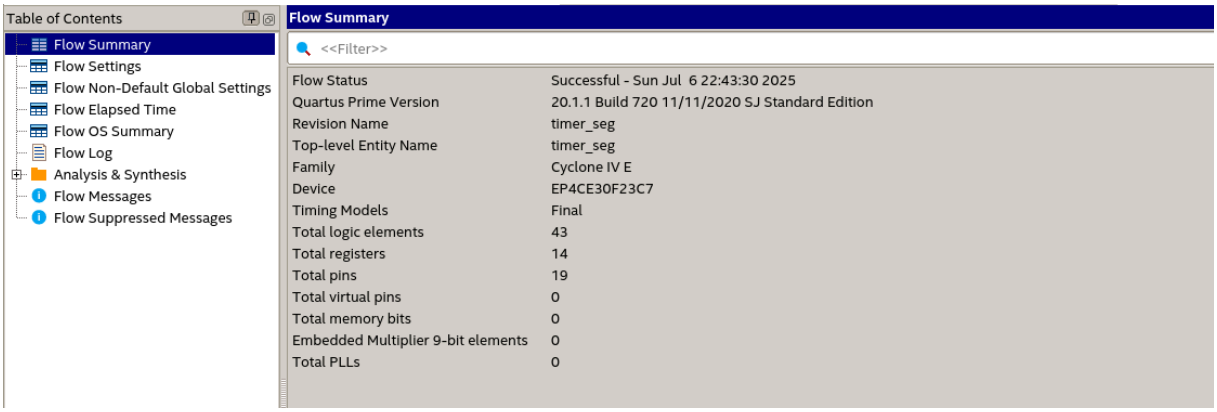
Após fazer a análise de síntese do projeto, foram obtidos os seguintes resultado:

Figura 13: Elaborada pelo Autor



RTL Viewer do timer_seg

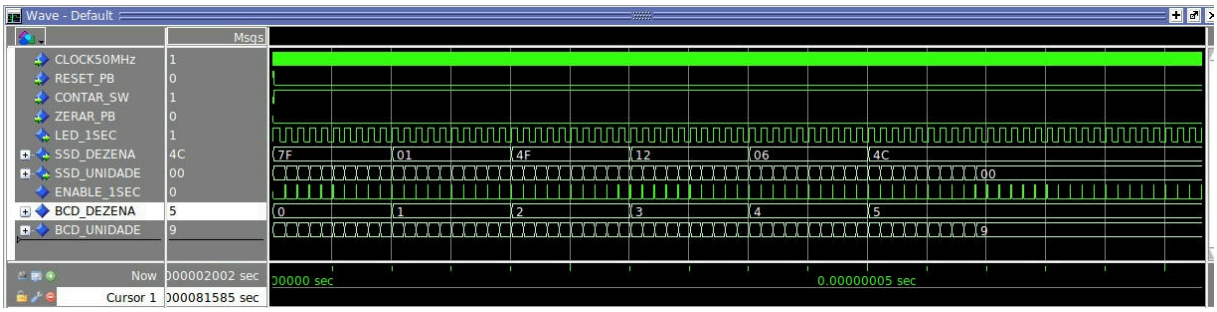
Figura 14: Elaborada pelo Autor



Sumário do timer_seg

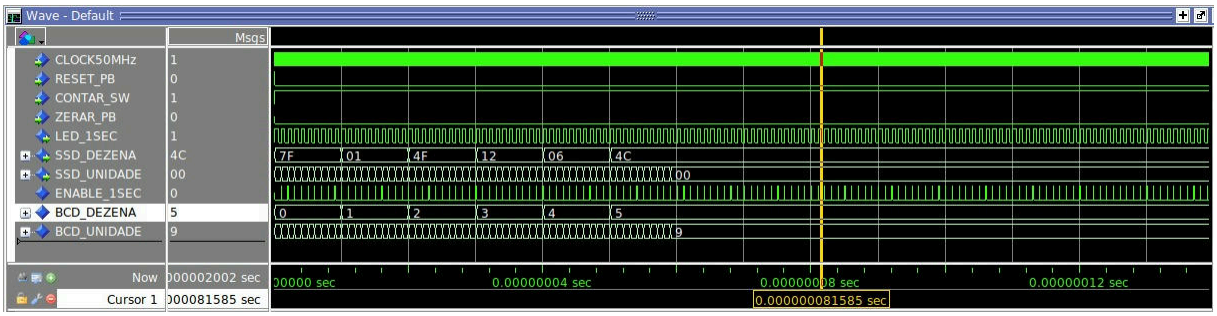
As figuras abaixo mostram o funcionamento da arquitetura do **`timer_seg.vhd`**:

Figura 15: Elaborada pelo Autor



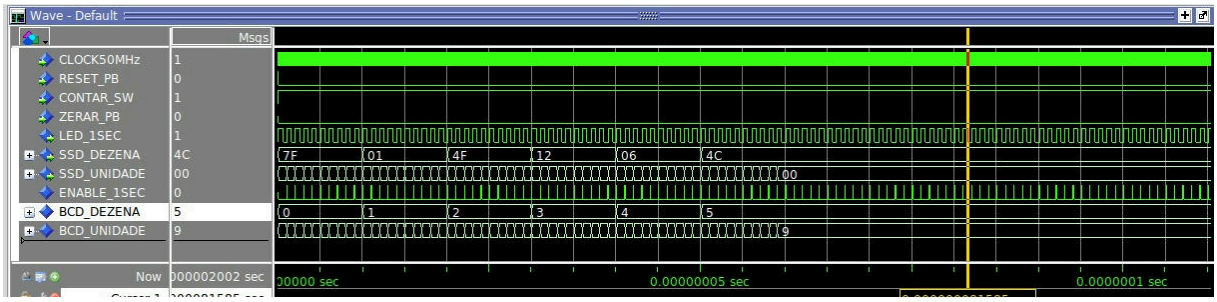
Simulação do timer_seg

Figura 16: Elaborada pelo Autor



Simulação do timer_seg

Figura 17: Elaborada pelo Autor



Simulação do timer_seg

6. Configuração para Placa FPGA

O projeto foi configurado de forma que seja capaz simulá-lo em um kit FPGA. Tal configuração pode ser feita fácilmente através do Quartus Prime. A placa escolhida foi uma EP4CE30F23C7 da família Cyclone IV E.

A seguir, uma imagem das configurações feitas na área do Pin Planner do Quartus:

Figura 18: Elaborada pelo Autor

Node Name	Direction	Location	I/O Bank	VREF Group	I/O Standard	Reserved	Current Strengtl	Slew Rate
CLOCK50MHz	Input	PIN_T1	2	B2_N0	2.5 V (default)		8mA (default)	
CONTAR_SW	Input	PIN_Y22	5	B5_N3	2.5 V (default)		8mA (default)	
LED_1SEC	Output	PIN_D6	8	B8_N3	2.5 V (default)		8mA (default)	2 (default)
RESET_PB	Input	PIN_V22	5	B5_N2	2.5 V (default)		8mA (default)	
SSD_DEZENA[6]	Output	PIN_W2	2	B2_N2	2.5 V (default)		8mA (default)	2 (default)
SSD_DEZENA[5]	Output	PIN_Y1	2	B2_N2	2.5 V (default)		8mA (default)	2 (default)
SSD_DEZENA[4]	Output	PIN_Y2	2	B2_N2	2.5 V (default)		8mA (default)	2 (default)
SSD_DEZENA[3]	Output	PIN_U1	2	B2_N1	2.5 V (default)		8mA (default)	2 (default)
SSD_DEZENA[2]	Output	PIN_U2	2	B2_N1	2.5 V (default)		8mA (default)	2 (default)
SSD_DEZENA[1]	Output	PIN_V1	2	B2_N1	2.5 V (default)		8mA (default)	2 (default)
SSD_DEZENA[0]	Output	PIN_V2	2	B2_N1	2.5 V (default)		8mA (default)	2 (default)

Confirguração dos pinos parte 1

Figura 19: Elaborada pelo Autor

Node Name	Direction	Location	I/O Bank	VREF Group	I/O Standard	Reserved	Current Strengtl	Slew Rate
SSD_UNIDADE[6]	Output	PIN_N6	2	B2_N1	2.5 V (default)		8mA (default)	2 (default)
SSD_UNIDADE[5]	Output	PIN_N7	2	B2_N2	2.5 V (default)		8mA (default)	2 (default)
SSD_UNIDADE[4]	Output	PIN_M6	2	B2_N0	2.5 V (default)		8mA (default)	2 (default)
SSD_UNIDADE[3]	Output	PIN_T4	2	B2_N3	2.5 V (default)		8mA (default)	2 (default)
SSD_UNIDADE[2]	Output	PIN_T3	2	B2_N2	2.5 V (default)		8mA (default)	2 (default)
SSD_UNIDADE[1]	Output	PIN_T5	2	B2_N3	2.5 V (default)		8mA (default)	2 (default)
SSD_UNIDADE[0]	Output	PIN_R5	2	B2_N3	2.5 V (default)		8mA (default)	2 (default)
ZERAR_PB	Input	PIN_U22	5	B5_N2	2.5 V (default)		8mA (default)	

Configuração dos pinos parte 2

7. Tabela de resultados

A fim de fazer um comparativo de cada circuito, foi montado uma tabela que faz a compação dos principais parâmetros de cada circuito.

Tabela 1: Elaborada pelo Autor

Parâmetros	Top Level	Divisor de clock	Conversor BCD p/ SSD	Contador BCD
Elementos lógicos	43	13	15	17
Registers	14	6	0	8
Pinos	19	4	12	13

Tabela de resultados de compilação para os 5 contadores

8. Conclusão

Conclui-se que os resultados esperados para o circuito do Timer de segundos foram satisfatoriamente alcançados por meio da utilização de um projeto hierárquico, estruturado em dois segmentos distintos durante o desenvolvimento do código.