INSTITUTO FEDERAL
Santa Catarina
Câmpus São José

Controlador semafórico para duas vias

Eletrônica Digital II

Bernardo Souza Muniz, Ygor Martins e Germano Coelho.

09 de Julho de 2025

Engenharia de Telecomunicações - IFSC-SJ

Sumário

1. Introdução

O objetivo deste documento é apresentar a implementação de um controlador de semáforos de duas vias utilizando a linguagem ***VHDL (VHSIC Hardware Description Language)***. 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.

Para a implementação da FSM (Máquina de Estados Finitos) foi criado um diagrama de estados que define as condições necessárias para as transições entre estados. A condição para a transição de um estado para o outro pode variar conforme a cor exibida pelo semáforo, refletindo diferentes condicionais de mudança a partir do estado atual. A figura abaixo exibe o diagrama do controlador semafórico.

Figura 1: Elaborado pelo Autor

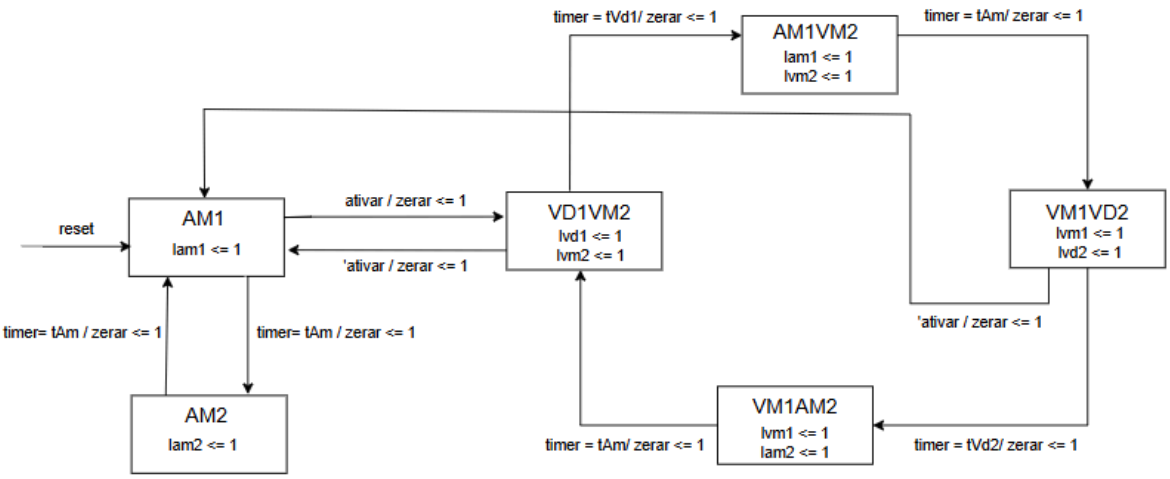


Diagrama de estados do controlador semafórico

Cada estado representado no diagrama foi codificado como um estado distinto na FSM, sendo ativado de acordo com as condições de tempo (sinal timer) e entrada externa (ativar). O sistema foi modelado para garantir a alternância entre os sinais AM1/VM2 e AM2/VM1, permitindo o fluxo completo entre os sinais do semáforo.

2. FSM Semáforo

O FSM_semaforo.vhd tem como objetivo realizar a descrição da máquina de estado que corresponde ao comportamento do semáforo de 2 vias. Além dos estados que ditam quais leds devem estar acesos, e que poderão ser vistos através do diagrama que será apresentado em breve, a máquina contém saídas mealy de zeramento do timer_sec.vhd (componente que será explorado de maneira mais profunda no próximo tópico).

Assim, vamos observar como foi feita a descrição de hardware deste componentes. Vamos iniciar com a entidade:

```
entity FSM_semaforo is
  generic
  (
    temp_amarelo : natural := 5; #tempo em que o semaforo fica amarelo
    temp_verde1  : natural := 60; #tempo que o semaforo 1 fica verde
    temp_verde2  : natural := 30; #tempo em que o semaforo 2 fica verde
    nbits_timer  : natural := 6
  );
  port
  (
    clk, rst : in std_logic;
    ativar : in std_logic; --entrada para mudanca de estados
    timer_std : in std_logic_vector(nbits_timer-1 downto 0); #entrada para mudanca de estados
    lam1, lam2, lvd1, lvd2, lvm1, lvm2 : out std_logic; #saidas indicando qual cor deve ser exibida no semaforo de acordo com o estado atual
    zerar_cnt : out std_logic
  );
end entity;
```

Como observado na figura 1, a mudança de alguns estados para outros são condicionados por um **`timer`**. Nesse sentido, foram criadas variáveis genéricas que definem o tempo de mudança. A comparação é feita utilizando a entrada **`timer_in`**. Além disso foi utilizado uma entrada ativar que atua como outra condicional de mudança de estado.

Os leds de cada semáforo são ascendidos ou apagados quando tem valor ‘1’ ou ‘0’, respectivamente. Nesse sentido, a nomenclatura do led de cada semáforo é descrita de acordo com a cor, sendo: led verde (**`lvd`**), led amarelo (**`lam`**) e led vermelho (**`lvm`**).O parâmetro ‘1’ e ‘2’ do lado de cada variável serve para identificar a qual semáforo pertence.

A máquina de estados foi desenvolvida em dois segmentos, sendo um sequencial e outro combinacional. A parte sequencial trata do estado de início do controlador semafórico e a parte combinacional trata de toda a lógica de mudança de estados.

Abaixo é possível ver a descrição da arquitetura do projeto.

```
architecture semaforos of fsm_semaforo is
  type state is (AM1,AM2,VD1VM2,AM1VM2,VM1VD2,VM1AM2,erro); #estados
  signal pr_state, next_state : state; #variaveis de registro de estado

  #Parte sequencial
  process(clk, rst)
  begin
    if rst = '1' then
      pr_state <= AM1;
    elsif rising_edge(clk) then
      pr_state <= next_state;
    end if;
  end process;

  #Parte combinacional
  process(pr_state, ativar, timer_std)
  begin

    #valores default
    lam1 <= '0';
    lam2 <= '0';
    lvd1 <= '0';
    lvd2 <= '0';
    lvm1 <= '0';
    lvm2 <= '0';
    zerar_cnt <= '0';
    next_state <= pr_state;

    case pr_state is
      when AM1 =>
        lam1 <= '1';
        if timer_std = std_logic_vector(to_unsigned(temp_amarelo,
timer_std'length)) then
          zerar_cnt <= '1';
          next_state <= AM2;
        elsif ativar = '1' then
          zerar_cnt <= '1';
          next_state <= VD1VM2;
        end if;
      when AM2 =>
        lam2 <= '1';
        if timer_std = std_logic_vector(to_unsigned(temp_amarelo,
timer_std'length)) then
          zerar_cnt <= '1';
          next_state <= AM1;
        end if;
      when VD1VM2 =>
        lvd1 <= '1';
        lvm2 <= '1';
        if timer_std = std_logic_vector(to_unsigned(temp_verde1,
timer_std'length)) then
          zerar_cnt <= '1';
          next_state <= AM1VM2;
        elsif ativar = '0' then
          zerar_cnt <= '1';
```

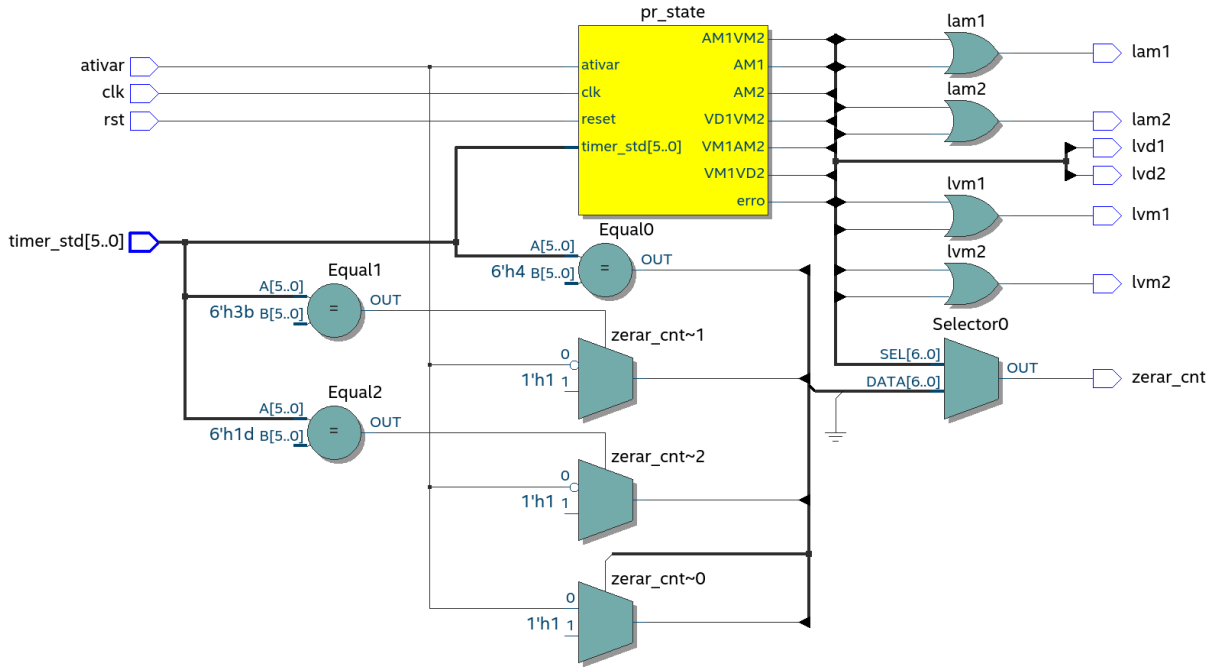
```
53            next_state <= AM1;
54          end if;
55        when AM1VM2 =>
56          lam1 <= '1';
57          lvm2 <= '1';
58          if timer_std = std_logic_vector(to_unsigned(temp_amarelo,
    timer_std'length)) then
59            zerar_cnt <= '1';
60            next_state <= VM1VD2;
61          end if;
62        when VM1VD2 =>
63          lvm1 <= '1';
64          lvd2 <= '1';
65          if timer_std = std_logic_vector(to_unsigned(temp_verde2,
    timer_std'length)) then
66            zerar_cnt <= '1';
67            next_state <= VM1AM2;
68          elsif ativar = '0' then
69            zerar_cnt <= '1';
70            next_state <= AM1;
71          end if;
72        when VM1AM2 =>
73          lvm1 <= '1';
74          lam2 <= '1';
75          if timer_std = std_logic_vector(to_unsigned(temp_amarelo,
    timer_std'length)) then
76            zerar_cnt <= '1';
77            next_state <= VD1VM2;
78          end if;
79        when others =>
80          next_state <= erro;
81        end case;
82    end process;
83  end architecture;
```

Uma vez verificada a arquitetura da FSM, observa-se que a saída **`zerar_cnt`** é implementada como uma saída do tipo Mealy, ou seja, depende tanto do próximo estado quanto da entrada **`timer_std`**. A saída é essencial, pois garante que, ao ocorrer uma transição de estado que exige um novo intervalo de tempo, o contador de segundos seja reiniciado no momento da mudança. Essa lógica é fundamental para assegurar que a troca de cores dos semáforos ocorra de forma sincronizada, evitando comportamentos incorretos no sistema.

Nota-se que foram atribuídos valores ***default*** para todos os leds de ambos os semáforos, bem como para o sinal de zerar e de próximo estado. A atribuição em questão evita a ocorrência de possíveis ***latchs*** no controlador semafórico e de erros no diagrama de estados representado pelo Quartus (ver figura 3).

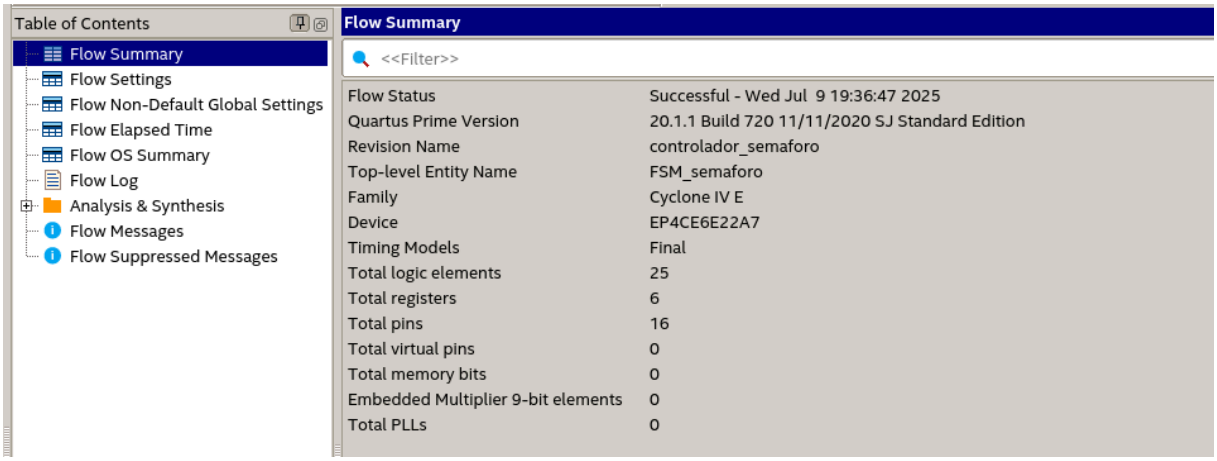
Ao realizar a análise de síntese da FSM, foram obtidos os seguintes resultados:

Figura 2: Elaborada pelo Autor



RTL Viwer da máquina de estado

Figura 3: Elaborada pelo Autor



Sumário da máquina de estado

Vale também observarmos o diagrama representando os estados da máquina, assim:

Figura 4: Elaborada pelo Autor

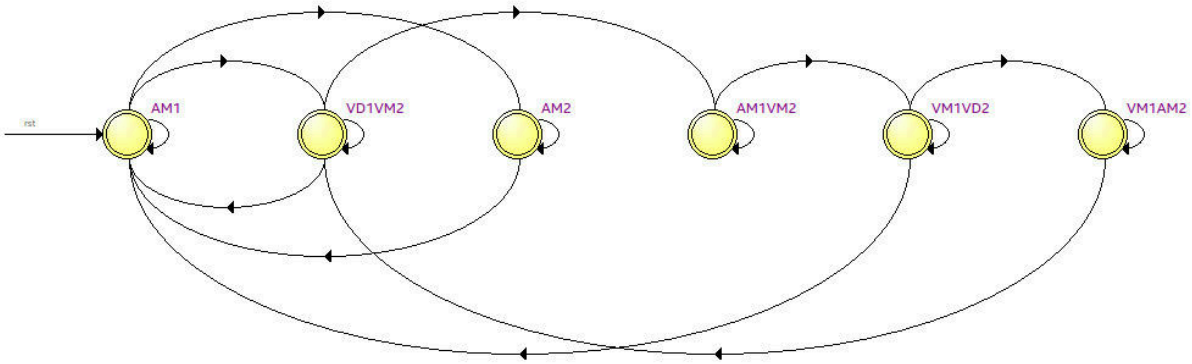
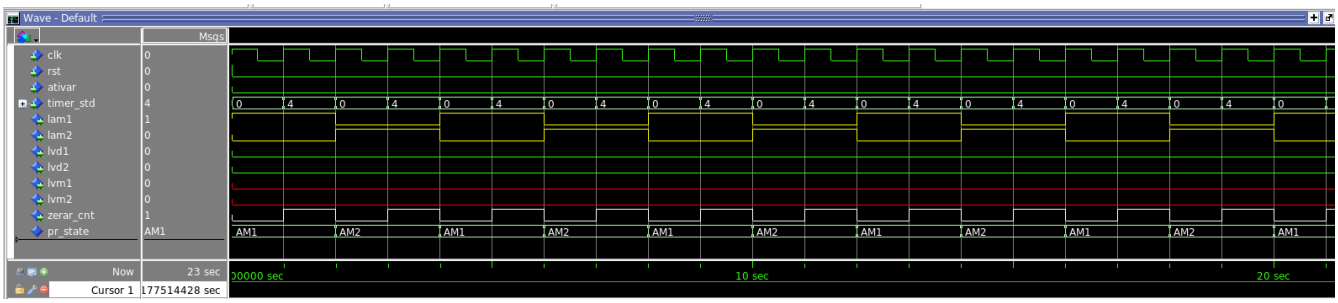


Diagrama de estados da máquina de estado

Também no Quartus, através da simulação do Modelysm, foi obtido o seguinte resultado:

Na figura 5 foi feito a simulação do ciclo amarelo piscante da FSM. Nota-se que de acordo com o diagrama de estados (ver figura 1) a alternância entre os estados AM1 e AM2 ocorre quando o timer de segundos tem valor igual a 5. É possível verificar também quanto a saída zerar, onde a mesma fica inativa durante a permanência nos estados (contador avança) e só é ativada na troca de estados (zera o contador).

Figura 5: Elaborada pelo Autor

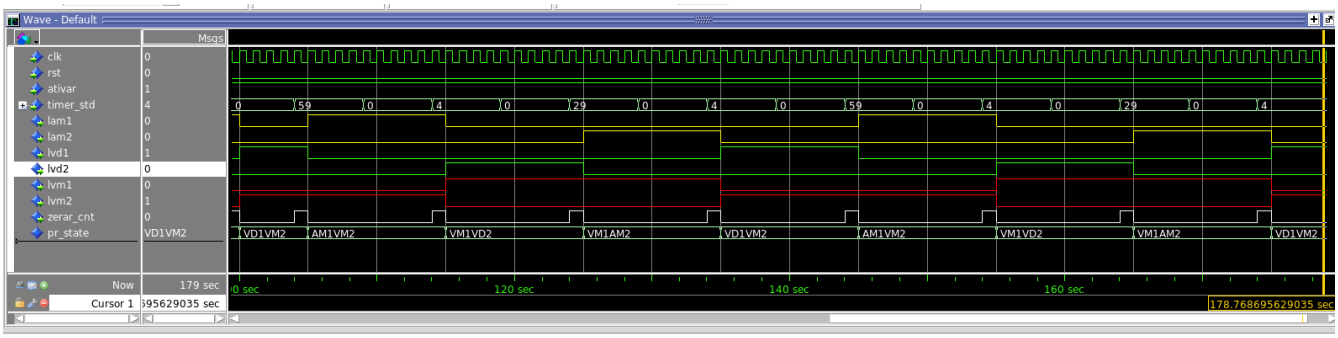


Simulação do Controlador semafórico para duas vias (amarelo piscante)

$$\text{RST} \rightarrow \text{AM1} \rightarrow \text{AM2} \rightarrow \text{AM1} \rightarrow \text{AM2} \rightarrow \text{AM1} \rightarrow \text{AM2} \rightarrow ... \rightarrow ... \rightarrow \quad (1)$$

Na figura 6 é possível verificar a simulação do ciclo verde por duas vezes da máquina de estados. Nota-se que o comportamento da saída zerar é o mesmo que foi identificado no ciclo amarelo piscante.

Figura 6: Elaborada pelo Autor

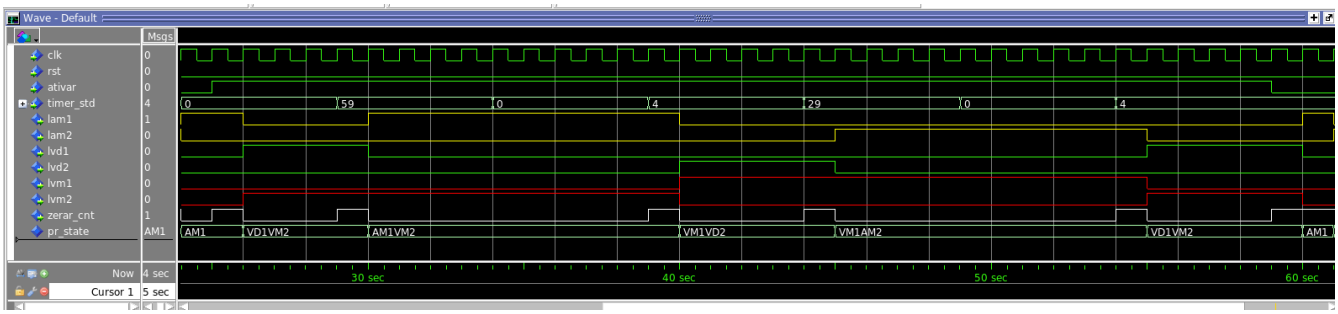


Simulação do Controlador semafórico para duas vias (ciclo verde)

$$\text{VD1VM2} \rightarrow \text{AM1VM2} \rightarrow \text{VM1VD2} \rightarrow \text{VM1AM2} \rightarrow \text{VD1VM2} \rightarrow ... \rightarrow ... \rightarrow \quad (2)$$

Além das simulações passando pelos dois ciclos, foram feitos mais duas simulações que passam por todos os estados do controlador semafórico.

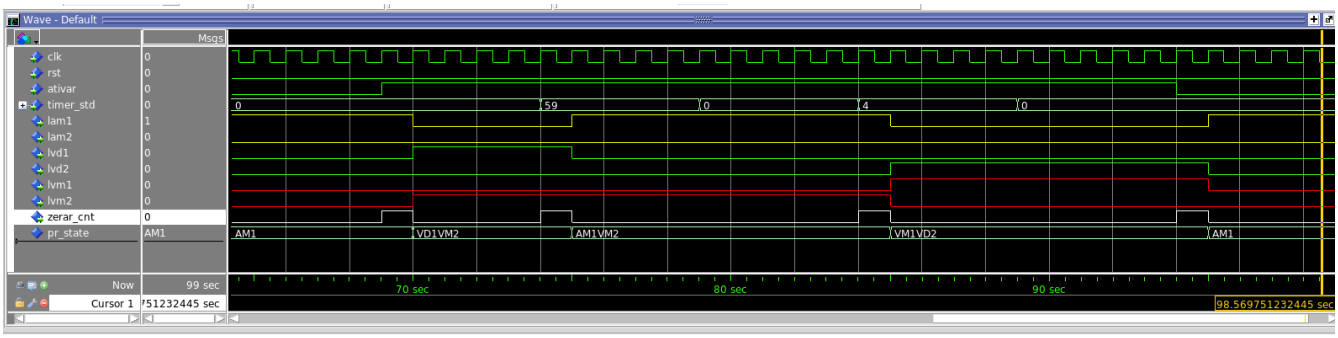
Figura 7: Elaborada pelo Autor



Simulação do Controlador semafórico para duas vias

$$\text{AM1} \rightarrow \text{VD1VM2} \rightarrow \text{AM1VM2} \rightarrow \text{VM1VD2} \rightarrow \text{VM1AM2} \rightarrow \text{VD1VM2} \rightarrow \text{AM1} \quad (3)$$

Figura 8: Elaborada pelo Autor



Simulação do Controlador semafórico para duas vias

$$\text{AM1} \rightarrow \text{VD1VM2} \rightarrow \text{AM1VM2} \rightarrow \text{VM1VD2} \rightarrow \text{AM1} \quad (4)$$

3. Contador de Segundos

O arquivo **timer_sec.vhd** tem como objetivo realizar a contagem de segundos com o fim de ditar as possíveis mudanças de estado do **FSM_semaforo.vhd**, alternando as cores de semáforo quando chega em determinado tempo de contagem.

Abaixo se encontra a descrição da entidade do contador:

```
entity timer_sec is
  generic(MAX : natural := 60; NBITS : natural := 6);
  port
  (
      clk : in std_logic;
      ena : in std_logic;
      contar : in std_logic;
      rst : in std_logic;
      zerar : in std_logic;
      timer : out std_logic_vector(NBITS-1 downto 0)
  );
end entity;
```

O contador por sua vez conta até o maior tempo em segundos que um estado do semáforo verde dura (60 segundos). Além disso,foi implementado critérios de parada na contagem: quando conta até o valor máximo ou quando a entrada de **contar** fica inativa. O contador também volta para zero quando o sinal **zerar** é enviado através de uma saída *Mealy* vinda da troca de estados da FSM.

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

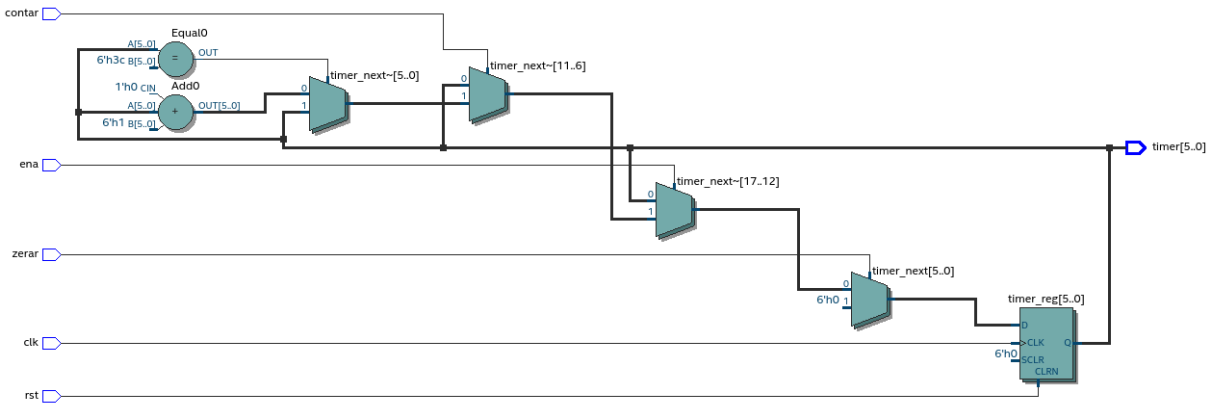
```
architecture ifsc of timer_sec is
   signal timer_reg, timer_next : unsigned(NBITS-1 downto 0);
begin
  #register
  process(clk, rst)
  begin
    if rst = '1' then
      timer_reg <= (others => '0');
    elsif rising_edge(clk) then
      timer_reg <= timer_next;
    end if;
  end process;

  process(timer_reg, ena, zerar, contar)
  begin
    timer_next <= timer_reg;
    if zerar = '1' then
        timer_next <= (others => '0');
    elsif ena = '1' then
        if contar = '1' then
          if timer_reg < to_unsigned(MAX, timer_reg'length) then
            timer_next <= timer_reg + 1;
          end if;
```

```
24          end if;
25        end if;
26      end process;
27
28
29      #output
30      timer <= std_logic_vector(timer_reg);
31  end architecture;
```

Nota-se que a entrada **ena** tem como função controlar o funcionamento da contagem. Quando desativada, o contador mantém seu valor atual, impedindo qualquer avanço. A contagem só é retomada quando ena é receba um novo pulso de habilitação, desde que a entrada **contar** também esteja habilitada. Dessa forma, o incremento ocorre apenas sob a condição conjunta dessas duas entradas.

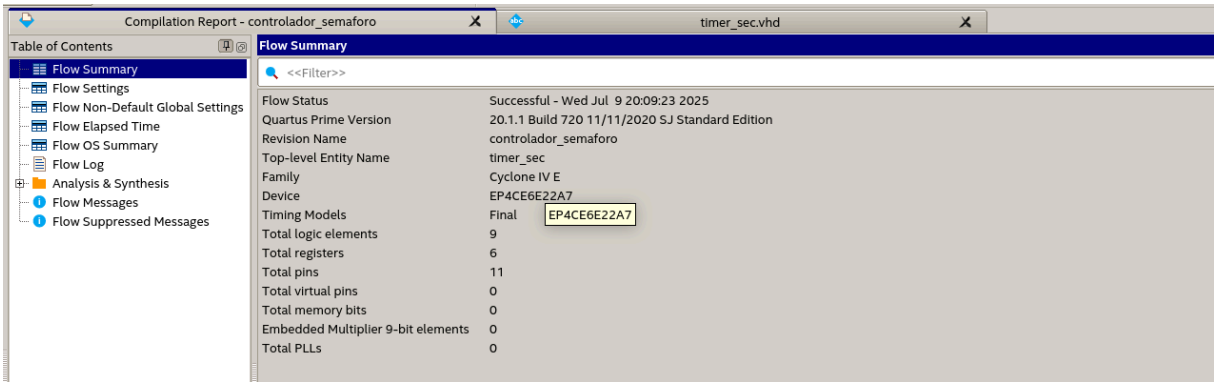
Após fazer a análise de síntese do projeto, foram obtidos os seguintes resultado:

Figura 9: Elaborada pelo Autor



RTL do contador de segundos

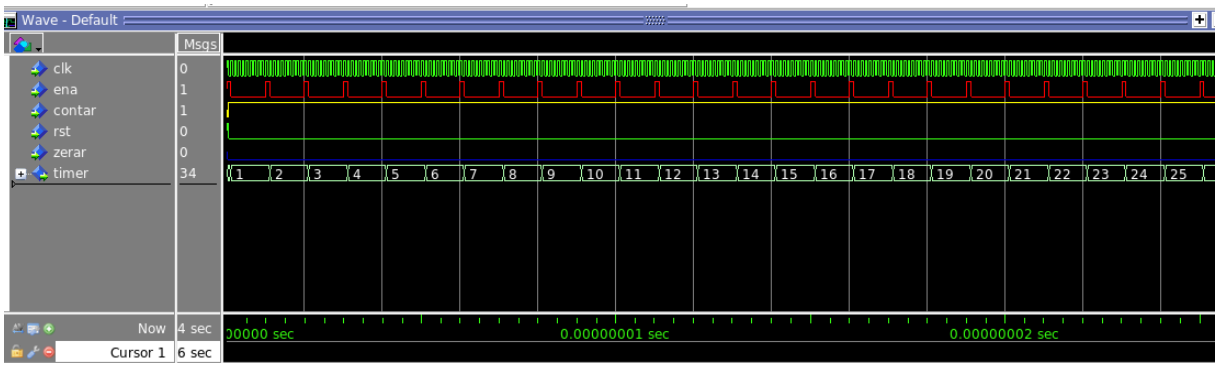
Figura 10: Elaborada pelo Autor



Sumário do contador de segundos

Para fazer a simulação, foi utilizado um clock de 10Hz e um sinal de enable de 1Hz com duty cycle de 10%:

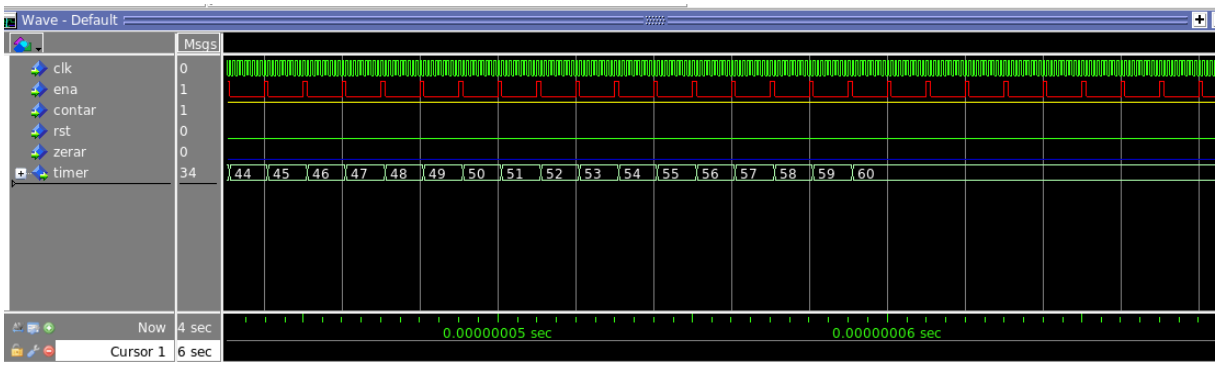
Figura 11: Elaborada pelo Autor



Simulação do contador de segundos

Na figura 8 é possível verificar que quando o contador chega no valor máximo de 60, ele para a contagem.

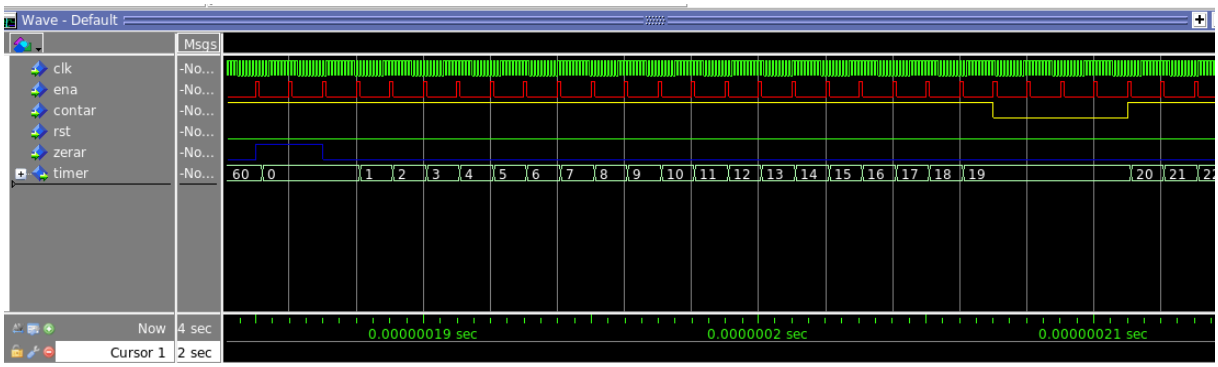
Figura 12: Elaborada pelo Autor



Simulação do contador de segundos com parada de contagem

Por fim, foram testadas as entradas zerar e contar. Nota-se que na simulação o contador havia atingido o valor máximo de 60 segundos, após isso, foi ativado a entrada zerar (onda em azul), e o contador voltou ao estado inicial. Foi retomada a contagem ativando a entrada de contagem e quando o contador estava em 19 a entrada **`contar`** foi desativada (onda em amarelo) e o contador manteve seu valor até que a entrada fosse novamente ativada.

Figura 13: Elaborada pelo Autor



Simulação do contador de segundos com teste das entradas contar e zerar

4. Divisor de Clock

O objetivo principal do circuito divisor de clock dentro do projeto do controlador semafórico, é gerar um pulso de habilitação, descrito como `ena_out` na entidade, para a habilitação de contagem no timer de segundos.

```
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;
```

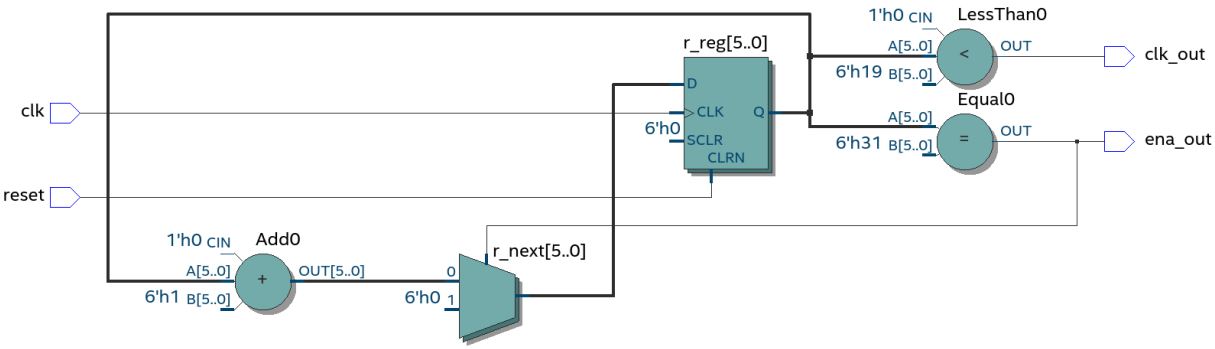
Na lógica sequencial, temos que o reset é assíncrono, por mais que esteja dentro de um processo que dependa da entrada `clk_in`, sua função é fora da borda de subida do clock.

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 dividido 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.

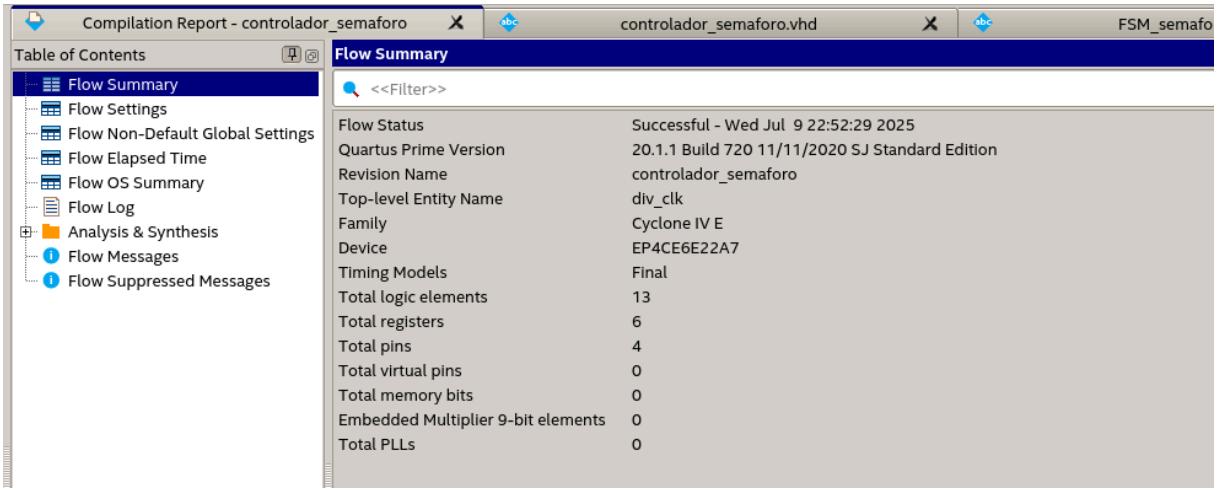
Após fazer a análise de síntese do projeto, foram obtidos os seguintes resultado:

Figura 14: Elaborada pelo Autor



RTL Viewer do divisor de clock

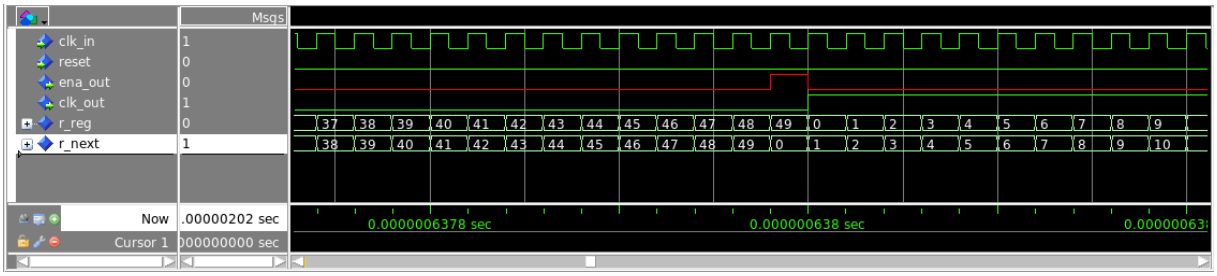
Figura 15: Elaborada pelo Autor



Sumário do divisor de clock

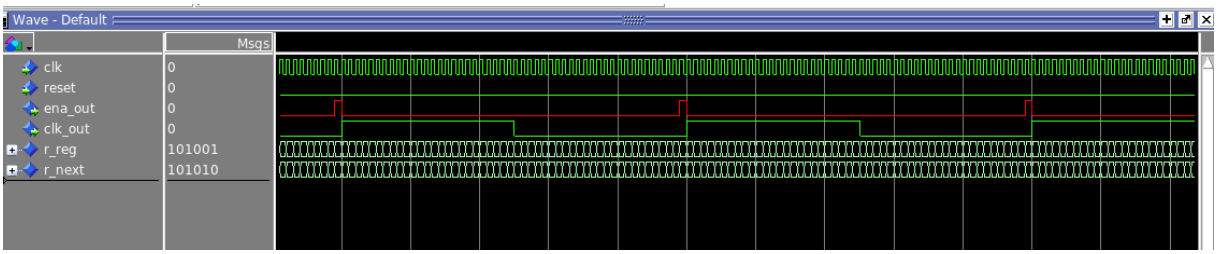
Ao realizar a simulação, foi utilizado um clock de 20 ns e simulado com frequência de 50Hz, neste caso, o parâmetro genérico `fclk` presente na entidade foi definido com valor de 50.

Figura 16: Elaborada pelo Autor



Simulação do divisor de clock

Figura 17: 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.

5. Controlador Semáforo (Top-Level)

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

```
entity controlador_semaforo IS
generic (
  tAMG : natural := 5; tVD1G : natural := 60; tVD2G : natural := 30;
  Nbits_timer :natural := 6;
  FCLK : natural := 50; Nbits_clk : natural := 6);
port (
  CLK50MHz : in std_logic;
  RESET_PB : in std_logic;
  ATIVAR_SW : in std_logic;
  CONTAR_SW : in std_logic;
  LED_1sec  : out std_logic;
  vm1_LED, vd1_LED, am1_LED, vm2_LED, vd2_LED, am2_LED : out std_logic
  );
end entity;
```

Nos parâmetros genéricos foram declarados os tempos de mudança de cada semáforo, podendo ser alterado conforme a necessidade.

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

```
architecture ifsc of controlador_semaforo is

  -
  #DECLARAÇÃO DOS COMPONENTES UTILIZADOS
  -
    #sinais internos: timer_std, zerar_cnt, ena_1sec, etc.
  signal ena_1sec, ZERAR_PB : std_logic;
  signal timer_std : std_logic_vector(Nbits_timer-1 downto 0);

begin
```

Por fim, vamos analisar a arquitetura e como estes componentes estão sendo conectados:

```
architecture ifsc of controlador_semaforo is
  -
  -
  #DECLARAÇÃO DOS COMPONENTES UTILIZADOS
  -
  -
begin
    U1 : div_clk generic map(fclock => FCLK, Nbits => Nbits_timer)
      port map(clk_in => CLK50MHz,
            reset => RESET_PB,
            ena_out => ena_1sec,
            clk_out => LED_1sec
            );
   #Instância do divisor de clock
   U2 : timer_sec generic map(MAX => tVD1G, NBITS => Nbits_timer)
      port map(clk => CLK50MHz,
            contar => CONTAR_SW,
            ena => ena_1sec,
            rst => RESET_PB,
            zerar => ZERAR_PB,
            timer => timer_std
            );

    #Instância do timer binario de 1 sec
    U3 : FSM_semaforo generic map (temp_amarelo => tAMG, temp_verde1 =>
tVD1G, temp_verde2 => tVD2G, nbits_timer => Nbits_timer) --Contador BCD
      port map (clk => CLK50MHz,
            zerar_cnt => ZERAR_PB,
            ativar => ATIVAR_SW,
            rst => RESET_PB,
            timer_std => timer_std,
            lam1 => am1_LED,
            lam2 => am2_LED,
            lvd1 => vd1_LED,
            lvd2 => vd2_LED,
            lvm1 => vm1_LED,
            lvm2 => vm2_LED);
   #Instância da FSM do semaforo
end architecture;
```

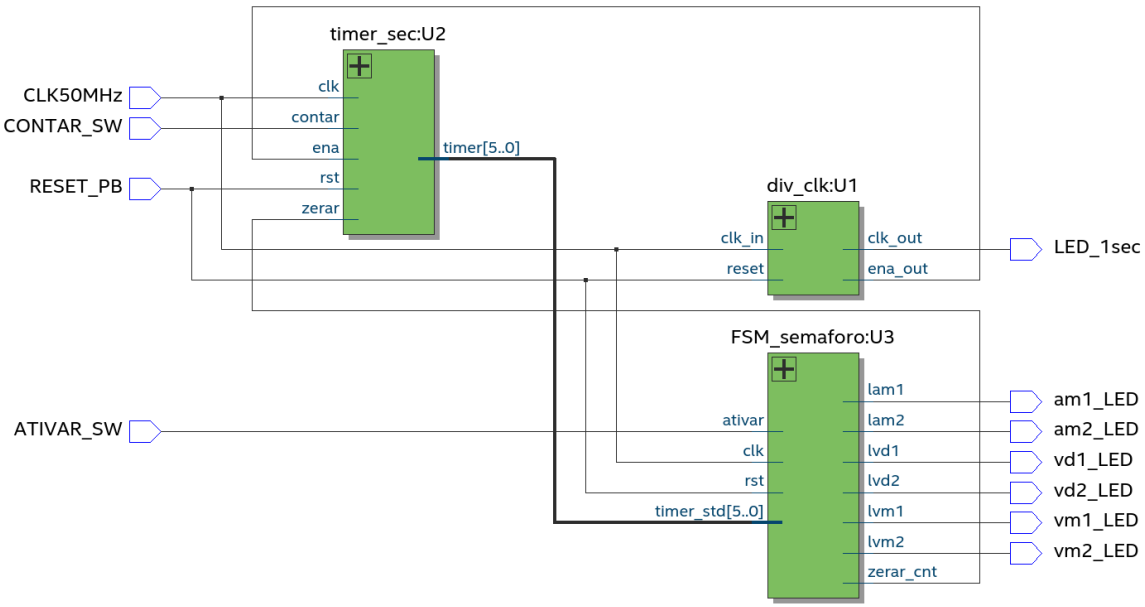
Com a arquitetura declarada e os componentes devidamente instanciados, temos agora todos os blocos fundamentais do circuito conectados de forma estruturada. O divisor de clock

(**div_clk**) é responsável por gerar um pulso com período de 1 segundo, o qual serve de referência temporal para o restante do sistema. Esse pulso habilita o **timer_sec**, que conta os segundos conforme configurado, e envia esse valor para a FSM (**FSM_semaforo**), responsável por coordenar a lógica de transição dos sinais do semáforo.

Cada componente cumpre um papel específico dentro do controlador, e os sinais internos, como **ena_1sec**, **ZERAR_PB** e **timer_std**, fazem a interligação entre eles, garantindo o fluxo correto de transição de estados do controlador semafórico.

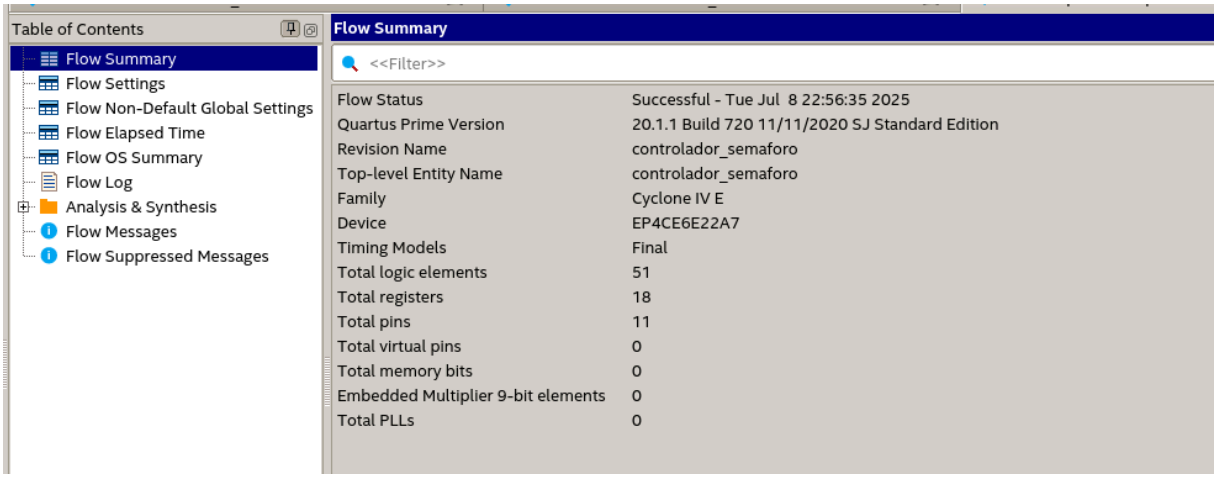
Após fazer a análise de síntese do projeto, foram obtidos os seguintes resultado:

Figura 18: Elaborada pelo Autor



RTL Viewer do Top Level

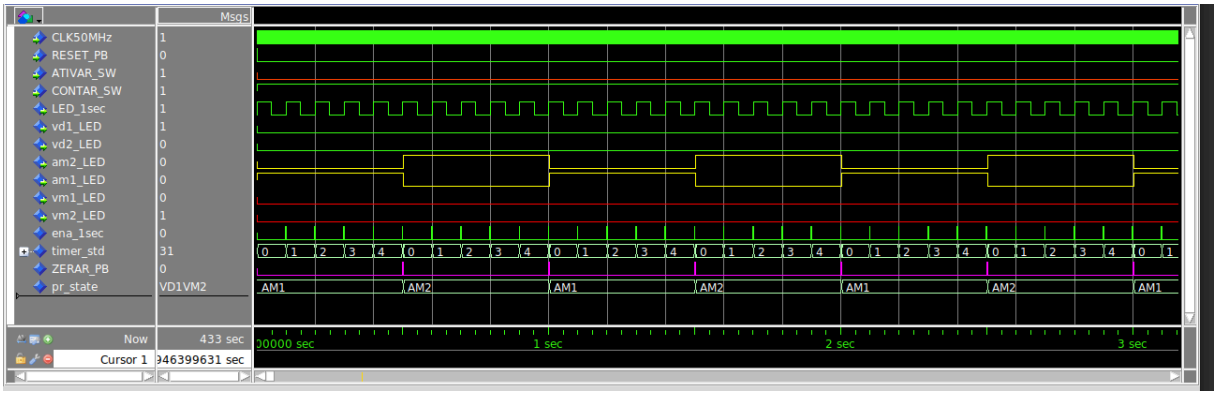
Figura 19: Elaborada pelo Autor



Sumário do Top Level

As figuras abaixo mostram o funcionamento da arquitetura do **`controlador_semaforo.vhd`** e como o projeto funciona por completo:

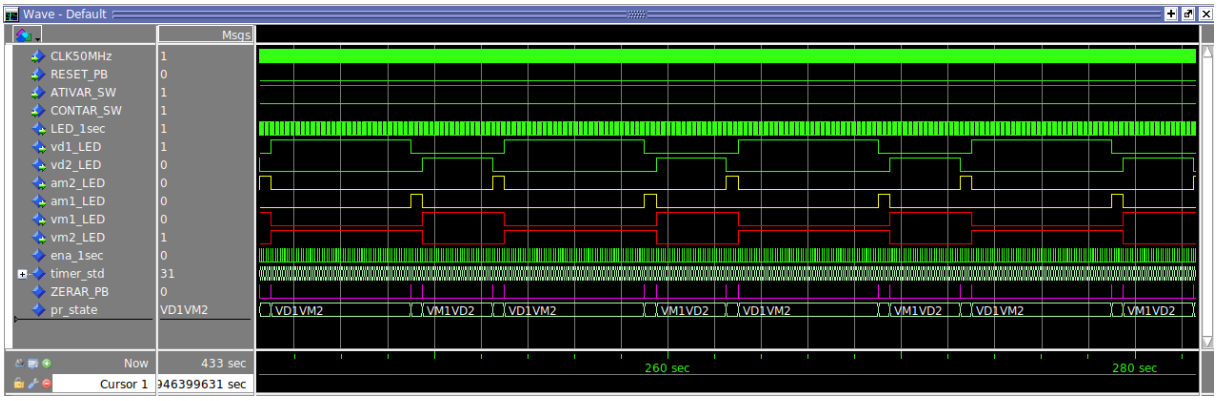
Figura 20: Elaborada pelo Autor



Simulação do Top Level para o ciclo amarelo piscante

Nota-se que a simulação do ciclo amarelo piscante corresponde ao que se esperava do sinal de zerar. O quando o estado atual é AM1 e o contador está em 4 (contou 5 no total pois começou em zero) o sinal de zerar é ativado no mesmo instate de transição para o estado AM2, que por sua vez inicia novamente a contagem em 0.

Figura 21: Elaborada pelo Autor



Simulação do Top Level para o ciclo verde

6. Tabela de resultados

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

Tabela 1: Elaborada pelo Autor

Parâmetros	Top Level	Divisor de clock	Timer	FSM
Elementos lógicos	51	13	9	25
Registers	18	6	6	6
Pinos	11	4	11	16

Tabela de resultados de compilação

7. Conclusão

Desta maneira, podemos observar que ao longo do desenvolvimento deste semáforo de 2 vias, diversos conceitos e aplicações da linguagem de descrição de hardware VHDL em nosso mundo e cotidiano.

O destaque desse projeto podemos afirmar com certeza que se trata da máquina de estado do semáforo (FSM_semaforo.vhd), isto pois através desta fomos capazes de observar a importância e o poder de se utilizar máquinas de estado em projetos VHDL.

Portanto, podemos concluir que ao fim, o objetivo que era descrever o funcionamento de um semáforo de 2 vias utilizando conceitos como projetos hierárquicos e máquinas de estado foi concluída com êxito.