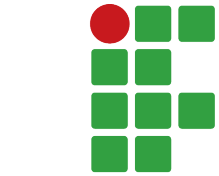

INSTITUTO
FEDERAL
Santa Catarina

Câmpus
São José

Relogio HHMMSS em VHDL

Dispositivos Lógicos Progamáveis I

Arthur Cadore Matuella Barcella

26 de Novembro de 2023

Sumário

1 Desenvolvimento

1.1 Objetivos

- Passo 1: Projete um Relogio-HHMMSS utilizando linguagem VHDL.
- Passo 2: Simule os componentes e o relógio completo no ModelSim
- Passo 3: Implemente o relógio no kit DE2-115 e teste-o.

1.2 Passo 1: Projete um Relogio-HHMMSS utilizando linguagem VHDL

Para iniciar o desenvolvimento do relógio, criei os códigos auxiliares para operar como componentes do relógio, os quais são os seguintes:

- div-clk: Responsavel pelo processo de divisão do clock do chipset para a frequência correta.
- bcd2ssd: Responsavel por fazer a conversão de código BCD para SSD (para impressão nos displays)
- counter Responsavel pela contagem do tempo.

O primeiro código, respectivo ao div-clk está apresentado abaixo:

```
1  library ieee;
2    use ieee.numeric_std.all;
3    use ieee.std_logic_1164.all;
4
5  entity div_clk is
6    generic (
7      div : natural := 50
8    );
9    port (
10     -- in
11     clk_in : in    std_logic;
12     rst    : in    std_logic;
13     -- out
14     clk_out : out   std_logic
15   );
16 end entity div_clk;
17
18 architecture v1 of div_clk is
19
20 begin
21
22   l1 : process (clk_in, rst) is
23
24     variable count : integer range 0 to div - 1;
25
26   begin
27
28     if (rst = '1') then
29       count   := 0;
30       clk_out <= '1';
31     elsif rising_edge(clk_in) then
32       if (count = div - 1) then
33         count   := 0;
34         clk_out <= '1';
35       else
36         count   := count + 1;
37         clk_out <= '0';
```

```
      end if;
    end if;

  end process l1;

end architecture v1;

configuration conf_div_clk of div_clk is
    for v1 end for;
-- for v2 end for;

end conf_div_clk;
```

O segundo código, respectivo a tradução de BCD para SSD, está apresentado abaixo:

```
library ieee;
  use ieee.std_logic_1164.all;
  use ieee.numeric_std.all;

entity bcd2ssd is
  port (
    bin     : in    std_logic_vector(3 downto 0);
    ssd_out : out   std_logic_vector(6 downto 0);
    ac_ccn  : in    std_logic
  );
end entity bcd2ssd;

architecture rtl1 of bcd2ssd is

  signal ssd     : std_logic_vector(6 downto 0);
  signal bin_int : integer range 0 to 9;

begin

  ssd_out <= ssd when ac_ccn = '1' else
             not ssd;
  bin_int <= to_integer(unsigned(bin));

  with bin_int select ssd <=
    "0111111" when 0,
    "0000110" when 1,
    "1011011" when 2,
    "1001111" when 3,
    "1100110" when 4,
    "1101101" when 5,
    "1111101" when 6,
    "0000111" when 7,
    "1111111" when 8,
    "1101111" when 9,
    "1000000" when others;

end architecture rtl1;
```

O terceiro código, respectivo a contagem de tempo, está apresentado abaixo:

```
library ieee;
  use ieee.numeric_std.all;
  use ieee.std_logic_1164.all;

entity counter is
  generic (
    max_decimal : integer := 2;
    max_unit    : integer := 3;
```

```
    n           : integer := 9
  );
  port (
    -- in
    clk : in    std_logic;
    rst : in    std_logic;
    -- out
    bcd_unit    : out   std_logic_vector(3 downto 0);
    bcd_decimal : out   std_logic_vector(3 downto 0);
    clk_out     : out   std_logic
  );
end entity counter;

architecture v1 of counter is

begin

  prc : process (clk, rst) is

    variable count_unit, count_decimal : integer range 0 to 9;

  begin

    if (rst = '1') then
      count_unit    := 0;
      count_decimal := 0;
      clk_out       <= '1';
    elsif rising_edge(clk) then
      clk_out <= '0';

      if (count_unit = max_unit and count_decimal = max_decimal) then
        count_decimal := 0;
        count_unit    := 0;
        clk_out       <= '1';
      elsif (count_unit = n) then
        count_unit := 0;
        if (count_decimal = max_decimal) then
          count_decimal := 0;
        else
          count_decimal := count_decimal + 1;
        end if;
      else
        count_unit := count_unit + 1;
      end if;
    end if;

    bcd_unit    <= std_logic_vector(to_unsigned(count_unit, 4));
    bcd_decimal <= std_logic_vector(to_unsigned(count_decimal, 4));

  end process prc;

end architecture v1;
```

Uma vez com os três códigos já estruturados e testados, dei inicio ao desenvolvimento do relógio em si, através de componentes, quando finalizado o código ficou conforme abaixo:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

  entity clock is
port(
  -- Input ports
```

```
  clk    : in    std_logic;
  enable : in    std_logic;
  reset  : in    std_logic;
  ac_ccn : in    std_logic;

  -- Output SSD ports, to connect on 7segment display.
  ssd_seconds_unit    : out std_logic_vector(6 downto 0);
  ssd_seconds_decimal : out std_logic_vector(6 downto 0);
  ssd_minutes_unit    : out std_logic_vector(6 downto 0);
  ssd_minutes_decimal : out std_logic_vector(6 downto 0);
  ssd_hours_unit      : out std_logic_vector(6 downto 0);
  ssd_hours_decimal   : out std_logic_vector(6 downto 0)
);
end entity;

architecture ifsc of clock is

  component div_clk is
    generic (
      div : natural := 50
    );
    port (
      clk_in  : in    std_logic;
      rst     : in    std_logic;
      clk_out : out   std_logic
    );
  end component div_clk;

  component bcd2ssd is
    port (
      bin     : in    std_logic_vector(3 downto 0);
      ssd_out : out   std_logic_vector(6 downto 0);
      ac_ccn  : in    std_logic
    );
  end component bcd2ssd;

  component counter is
    generic (
      max_decimal : integer := 2;
      max_unit    : integer := 3;
      n           : integer := 9
    );
    port (
      clk         : in    std_logic;
      rst         : in    std_logic;
      bcd_unit    : out   std_logic_vector(3 downto 0);
      bcd_decimal : out   std_logic_vector(3 downto 0);
      clk_out     : out   std_logic
    );
  end component counter;

  signal clk_seconds : std_logic;
  signal clk_minutes : std_logic;
  signal clk_hours   : std_logic;
  signal clk_days    : std_logic;

  signal bcd_unit_seconds    : std_Logic_vector (3 downto 0);
  signal bcd_decimal_seconds : std_Logic_vector (3 downto 0);

  signal bcd_unit_minutes    : std_Logic_vector (3 downto 0);
  signal bcd_decimal_minutes : std_Logic_vector (3 downto 0);

  signal bcd_unit_hours      : std_Logic_vector (3 downto 0);
```

```
71   signal bcd_decimal_hours   : std_Logic_vector (3 downto 0);
72
73 begin
74
75   div_clk_1 : component div_clk
76    generic map(
77    div => 50
78    )
79     port map (
80       clk_in  => clk,
81       clk_out => clk_seconds,
82       rst     => reset
83     );
84
85 -- Counter Section:
86
87     bcd_counter_seconds : component counter
88     generic map (
89       max_decimal => 5,
90       max_unit    => 9,
91       n           => 9
92     )
93     port map (
94       clk         => clk_seconds,
95       rst         => reset,
96       bcd_unit    => bcd_unit_seconds,
97       bcd_decimal => bcd_decimal_seconds,
98       clk_out     => clk_minutes
99     );
100
101    bcd_counter_minutes : component counter
102    generic map (
103      max_decimal => 5,
104      max_unit    => 9,
105      n           => 9
106    )
107    port map (
108      clk         => clk_minutes,
109      rst         => reset,
110      bcd_unit    => bcd_unit_minutes,
111      bcd_decimal => bcd_decimal_minutes,
112      clk_out     => clk_hours
113    );
114
115    bcd_counter_hours : component counter
116    generic map (
117      max_decimal => 2,
118      max_unit    => 3,
119      n           => 9
120    )
121    port map (
122      clk         => clk_hours,
123      rst         => reset,
124      bcd_unit    => bcd_unit_hours,
125      bcd_decimal => bcd_decimal_hours,
126      clk_out     => clk_days
127    );
128
129 -- SSD section:
130
131 bcd2ssd_seconds_unit : component bcd2ssd
132 port map (
133   bin     => bcd_unit_seconds,
```

```
134   ssd_out => ssd_seconds_unit,
135   ac_ccn  => ac_ccn
136 );
137
138 bcd2ssd_seconds_decimal : component bcd2ssd
139 port map (
140   bin     => bcd_decimal_seconds,
141   ssd_out => ssd_seconds_decimal,
142   ac_ccn  => ac_ccn
143 );
144
145
146 bcd2ssd_minutes_unit : component bcd2ssd
147 port map (
148   bin     => bcd_unit_minutes,
149   ssd_out => ssd_minutes_unit,
150   ac_ccn  => ac_ccn
151 );
152
153 bcd2ssd_minutes_decimal : component bcd2ssd
154 port map (
155   bin     => bcd_decimal_minutes,
156   ssd_out => ssd_minutes_decimal,
157   ac_ccn  => ac_ccn
158 );
159
160
161 bcd2ssd_hours_unit : component bcd2ssd
162 port map (
163   bin     => bcd_unit_hours,
164   ssd_out => ssd_hours_unit,
165   ac_ccn  => ac_ccn
166 );
167
168 bcd2ssd_hours_decimal : component bcd2ssd
169 port map (
170   bin     => bcd_decimal_hours,
171   ssd_out => ssd_hours_decimal,
172   ac_ccn  => ac_ccn
173 );
174
175 end architecture;
```

O codigo apresenta a seguinte estrutura:

As entradas abaixo são respectivas ao clock, reset e modo de operação dos leds (a entrada enable não foi implementada):

- clk : in std-logic;
- enable : in std-logic;
- reset : in std-logic;
- ac-ccn : in std-logic;

As saídas abaixo são conectadas a cada display, cada saída é um vetor de 7bits, e cada bit do vetor alimenta um LED diferente do relógio, que faz com que os LEDs acendam corretamente.

- ssd-seconds-unit : out std-logic-vector(6 downto 0);
- ssd-seconds-decimal : out std-logic-vector(6 downto 0);

- ssd-minutes-unit : out std-logic-vector(6 downto 0);
- ssd-minutes-decimal : out std-logic-vector(6 downto 0);
- ssd-hours-unit : out std-logic-vector(6 downto 0);
- ssd-hours-decimal : out std-logic-vector(6 downto 0)

A conexão que existe entre os componentes é realizada através dos sinais abaixo. Os sinais de clocks foram utilizados para conectar a saída de um contador na entrada de outro, assim, quando um contador estoura sua contagem (ou seja, chega no 59s por exemplo), ele dispara um sinal de clock (assincrono) para que o proximo contador faça uma contagem.

O processo se repete para minutos, horas e dias (sendo que o clock de dias não é utilizado):

```
1
2   signal clk_seconds : std_logic;
3   signal clk_minutes : std_logic;
4   signal clk_hours   : std_logic;
5   signal clk_days    : std_logic;
6
7   signal bcd_unit_seconds    : std_Logic_vector (3 downto 0);
8   signal bcd_decimal_seconds : std_Logic_vector (3 downto 0);
9
10  signal bcd_unit_minutes    : std_Logic_vector (3 downto 0);
11  signal bcd_decimal_minutes : std_Logic_vector (3 downto 0);
12
13  signal bcd_unit_hours      : std_Logic_vector (3 downto 0);
14  signal bcd_decimal_hours   : std_Logic_vector (3 downto 0);
```

Os sinais BCD são utilizados para levar os vetores de quatro bits em BCD para os conversores de BCD para SSD. Assim o valor que está sendo contado e modificado a cada segundo é impresso após ser convertido pelo componente de SSD.

Em seguida, temos a sessão de componentes, onde o primeiro componente descrito é o div-clk, responsavel por gerar e dividir o clock, o valor de "div"neste caso 50, é o valor pela qual o clock será dividido. Neste exemplo o tempo só seria contado corretamente caso o chip tivesse um clock de 50Hz que fosse inserido no clk-in.

```
1  div_clk_1 : component div_clk
2    generic map(
3    div => 50
4    )
5     port map (
6       clk_in  => clk,
7       clk_out => clk_seconds,
8       rst     => reset
9     );
```

Como é possivel verificar acima, a saída deste elemento é levada para o clk-seconds, que irá contar as unidades de segundo.

O componente mencionado (primeiro contador) é exibido abaixo, note que sua entrada vem do div-clk, e este contador é responsável por contador os segundos até o limite de 59.

Cada vez que chega neste valor, o contador estoura e inicia sua contagem, neste momento ele envia um pulso de clock para o clk-out. A cada instante, o valor contador é enviado para os componentes de BCD para impressão:

```
1     bcd_counter_seconds : component counter
2     generic map (
3       max_decimal => 5,
4       max_unit    => 9,
```

```
    n           => 9
  )
  port map (
    clk         => clk_seconds,
    rst         => reset,
    bcd_unit    => bcd_unit_seconds,
    bcd_decimal => bcd_decimal_seconds,
    clk_out     => clk_minutes
  );
```

Os componentes de BCD para SSD estão descritos abaixo, note que suas entradas (BCD) vem do componente explicado anteriormente. Uma vez que recebeu os dados para imprimir, basta que este componente saiba qual a forma de alimentação dos leds (catodo comum ou anodo comum), isto é configuravel através da porta ac-ccn.

O componente então converte os dados de BCD para SSD que são levados para as saídas expostas anteriormente.

```
bcd2ssd_seconds_unit : component bcd2ssd
port map (
  bin     => bcd_unit_seconds,
  ssd_out => ssd_seconds_unit,
  ac_ccn  => ac_ccn
);

bcd2ssd_seconds_decimal : component bcd2ssd
port map (
  bin     => bcd_decimal_seconds,
  ssd_out => ssd_seconds_decimal,
  ac_ccn  => ac_ccn
);
```

1.3 Passo 2: Simule os componentes e o relógio completo no ModelSim

Uma vez com o código definido, fiz sua compilação para iniciar os testes no ModelSim:

FONTE: Elaborado pelo autor

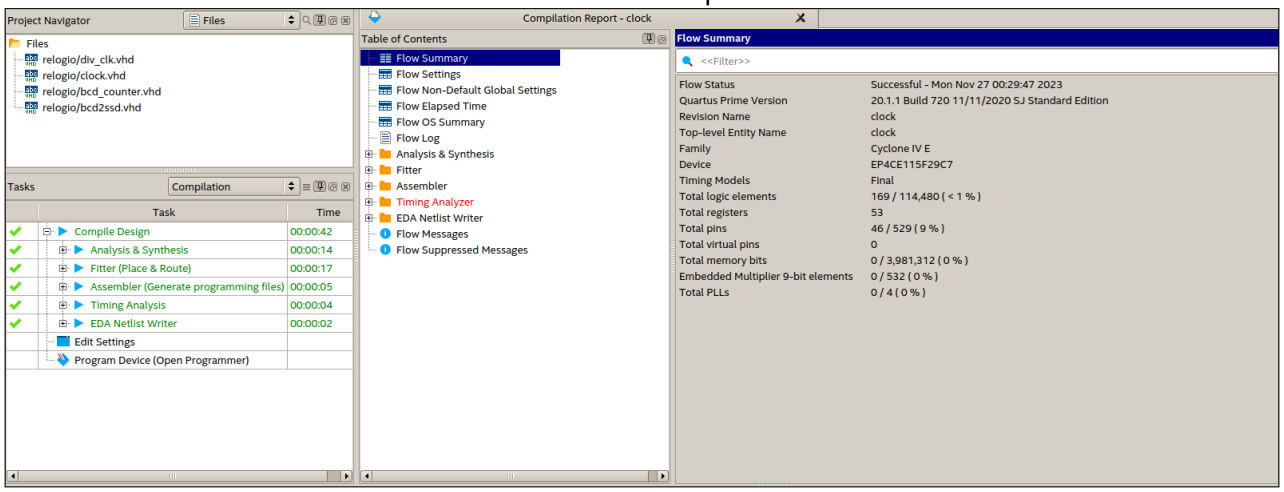

Figura 1: Compilação no Quartus

A compilação resultou no seguinte RTL:

FONTE: Elaborado pelo autor

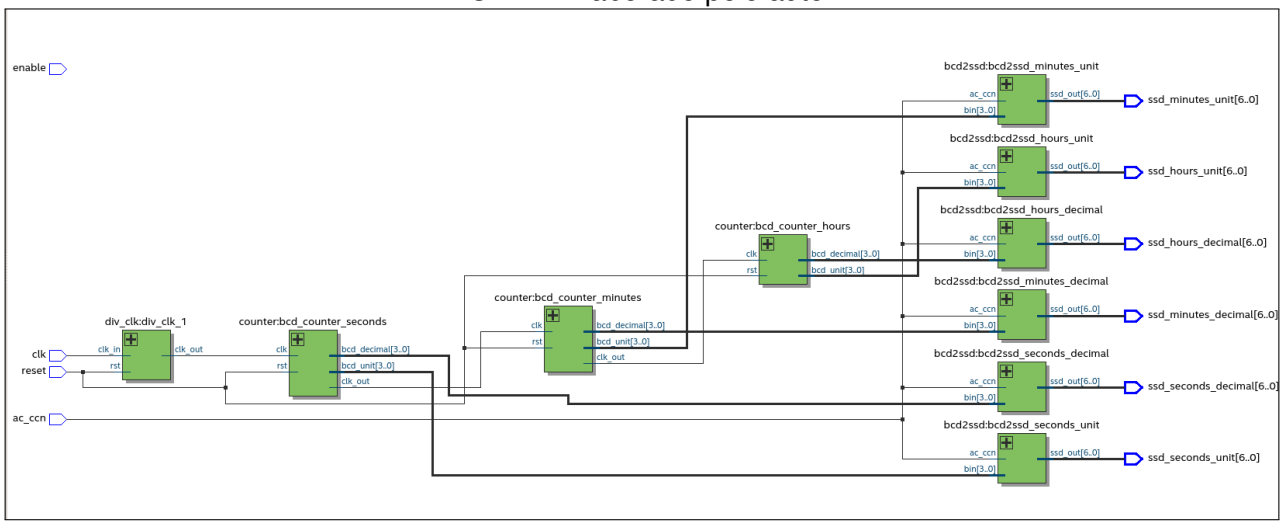

Figura 2: RTL do relógio

Para fazer os testes no modelSim, montei o seguinte wave.do:

```
onerror {resume}
quietly WaveActivateNextPane {} 0
add wave -noupdate -divider CLOCKs
add wave -noupdate /clock/clk_days
add wave -noupdate /clock/clk_hours
add wave -noupdate /clock/clk_minutes
add wave -noupdate /clock/clk_seconds
add wave -noupdate /clock/clk
add wave -noupdate /clock/ac_ccn
add wave -noupdate -divider Inputs
add wave -noupdate /clock/enable
add wave -noupdate /clock/reset
add wave -noupdate -divider {BCD OUTPUTS}
add wave -noupdate /clock/bcd_decimal_hours
add wave -noupdate /clock/bcd_unit_hours
add wave -noupdate /clock/bcd_decimal_minutes
add wave -noupdate /clock/bcd_unit_minutes
add wave -noupdate /clock/bcd_decimal_seconds
add wave -noupdate /clock/bcd_unit_seconds
add wave -noupdate -divider {SSD OUTPUTS}
add wave -noupdate /clock/ssd_hours_decimal
add wave -noupdate /clock/ssd_hours_unit
add wave -noupdate /clock/ssd_minutes_decimal
add wave -noupdate /clock/ssd_minutes_unit
add wave -noupdate /clock/ssd_seconds_decimal
add wave -noupdate /clock/ssd_seconds_unit
TreeUpdate [SetDefaultTree]
WaveRestoreCursors {{Cursor 1} {69240520 ps} 0}
quietly wave cursor active 1
configure wave -namecolwidth 242
configure wave -valuecolwidth 40
configure wave -justifyvalue left
configure wave -signalnamewidth 0
configure wave -snapdistance 10
configure wave -datasetprefix 0
configure wave -rowmargin 4
configure wave -childrowmargin 2
configure wave -gridoffset 0
configure wave -gridperiod 1
configure wave -griddelta 40
```

```
configure wave -timeline 0
configure wave -timelineunits hr
update
WaveRestoreZoom {999999995686 ps} {999999996280 ps}
```

Também montei o tb.do para realizar a configuração automatizada:

```

vlib rtl_work
vmap work rtl_work

vcom -93 -work work {../../clock.vhd}

vcom -93 -work work {../../bcd2ssd.vhd}

vcom -93 -work work {../../counter.vhd}

vcom -93 -work work {../../div_clk.vhd}

vsim work.clock(ifsc)

do wave.do

force -freeze sim:/clock/ac_ccn 1 0

force -freeze sim:/clock/reset 1 0, 0 10

force -freeze sim:/clock/clk 1 0, 0 {100ms } -r 200ms

run 1min
```

Os códigos apresentados acima geram o seguinte RTL:

FONTE: Elaborado pelo autor

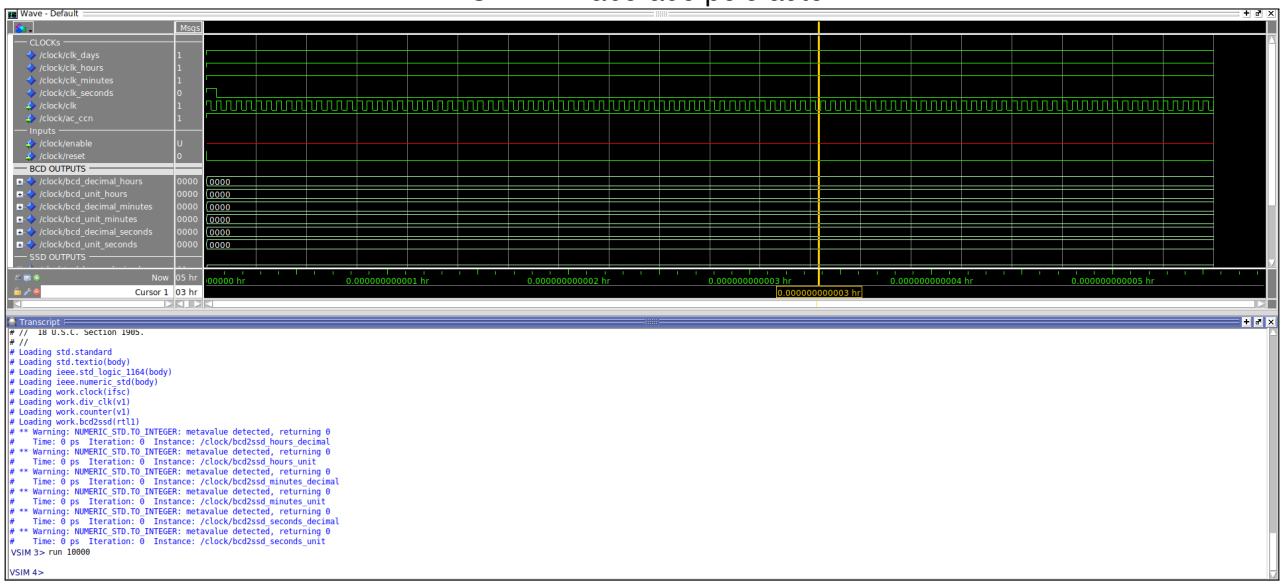

Figura 3: RTL simulado no modelsim

OBS: Irei conversar com o professor na proxima aula sobre o RTL apresentado, pois não apresenta o mesmo formato de onda obtido em sala.

1.4 Passo 3: Implemente o relógio no kit DE2-115:

Após testar o circuito (em sala) e validar seu funcionamento, configurei os pinos do chip para testa-lo utilizando um kit de desenvolvimento.

Para isso, a seguinte tabela de pinagem foi montada no pin-planner seguindo as especificações da wiki.

FONTE: Elaborado pelo autor

Node Name	Direction	Location	I/O Bank	VREF Group	I/O Standard	Reserved	Current Strengtl	Slew Rate	Differential Pair	trict Preservatic
ac_ccn	Input	PIN_AA22	5	B5_N2	2.5 V (default)		8mA (default)			
clk	Input	PIN_Y2	2	B2_N0	2.5 V (default)		8mA (default)			
enable	Input	PIN_Y23	5	B5_N2	2.5 V (default)		8mA (default)			
reset	Input	PIN_Y24	5	B5_N2	2.5 V (default)		8mA (default)			
ssd_hour...cimal[6]	Output	PIN_AA14	3	B3_N0	2.5 V (default)		8mA (default)	2 (default)		
ssd_hour...cimal[5]	Output	PIN_AG18	4	B4_N2	2.5 V (default)		8mA (default)	2 (default)		
ssd_hour...cimal[4]	Output	PIN_AF17	4	B4_N2	2.5 V (default)		8mA (default)	2 (default)		
ssd_hour...cimal[3]	Output	PIN_AH17	4	B4_N2	2.5 V (default)		8mA (default)	2 (default)		
ssd_hour...cimal[2]	Output	PIN_AG17	4	B4_N2	2.5 V (default)		8mA (default)	2 (default)		
ssd_hour...cimal[1]	Output	PIN_AE17	4	B4_N2	2.5 V (default)		8mA (default)	2 (default)		
ssd_hour...cimal[0]	Output	PIN_AD17	4	B4_N2	2.5 V (default)		8mA (default)	2 (default)		
ssd_hours_unit[6]	Output	PIN_AC17	4	B4_N2	2.5 V (default)		8mA (default)	2 (default)		
ssd_hours_unit[5]	Output	PIN_AA15	4	B4_N2	2.5 V (default)		8mA (default)	2 (default)		
ssd_hours_unit[4]	Output	PIN_AB15	4	B4_N2	2.5 V (default)		8mA (default)	2 (default)		
ssd_hours_unit[3]	Output	PIN_AB17	4	B4_N1	2.5 V (default)		8mA (default)	2 (default)		
ssd_hours_unit[2]	Output	PIN_AA16	4	B4_N2	2.5 V (default)		8mA (default)	2 (default)		
ssd_hours_unit[1]	Output	PIN_AB16	4	B4_N2	2.5 V (default)		8mA (default)	2 (default)		
ssd_hours_unit[0]	Output	PIN_AA17	4	B4_N1	2.5 V (default)		8mA (default)	2 (default)		
ssd_minu...cimal[6]	Output	PIN_AH18	4	B4_N2	2.5 V (default)		8mA (default)	2 (default)		
ssd_minu...cimal[5]	Output	PIN_AF18	4	B4_N1	2.5 V (default)		8mA (default)	2 (default)		
ssd_minu...cimal[4]	Output	PIN_AG19	4	B4_N2	2.5 V (default)		8mA (default)	2 (default)		
ssd_minu...cimal[3]	Output	PIN_AH19	4	B4_N2	2.5 V (default)		8mA (default)	2 (default)		
ssd_minu...cimal[2]	Output	PIN_AB18	4	B4_N0	2.5 V (default)		8mA (default)	2 (default)		
ssd_minu...cimal[1]	Output	PIN_AC18	4	B4_N1	2.5 V (default)		8mA (default)	2 (default)		
ssd_minu...cimal[0]	Output	PIN_AD18	4	B4_N1	2.5 V (default)		8mA (default)	2 (default)		
ssd_minu..._unit[6]	Output	PIN_AE18	4	B4_N2	2.5 V (default)		8mA (default)	2 (default)		
ssd_minu..._unit[5]	Output	PIN_AF19	4	B4_N1	2.5 V (default)		8mA (default)	2 (default)		
ssd_minu..._unit[4]	Output	PIN_AE19	4	B4_N1	2.5 V (default)		8mA (default)	2 (default)		
ssd_minu..._unit[3]	Output	PIN_AH21	4	B4_N2	2.5 V (default)		8mA (default)	2 (default)		
ssd_minu..._unit[2]	Output	PIN_AG21	4	B4_N2	2.5 V (default)		8mA (default)	2 (default)		
ssd_minu..._unit[1]	Output	PIN_AA19	4	B4_N0	2.5 V (default)		8mA (default)	2 (default)		
ssd_minu..._unit[0]	Output	PIN_AB19	4	B4_N0	2.5 V (default)		8mA (default)	2 (default)		
ssd_seco...cimal[6]	Output	PIN_Y19	4	B4_N0	2.5 V (default)		8mA (default)	2 (default)		
ssd_seco...cimal[5]	Output	PIN_AF23	4	B4_N0	2.5 V (default)		8mA (default)	2 (default)		
ssd_seco...cimal[4]	Output	PIN_AD24	4	B4_N0	2.5 V (default)		8mA (default)	2 (default)		

Figura 4: Pin-planner

Uma vez com a tabela preenchida, obtive a seguinte imagem do chip (vista do pin-planner):

FONTE: Elaborado pelo autor

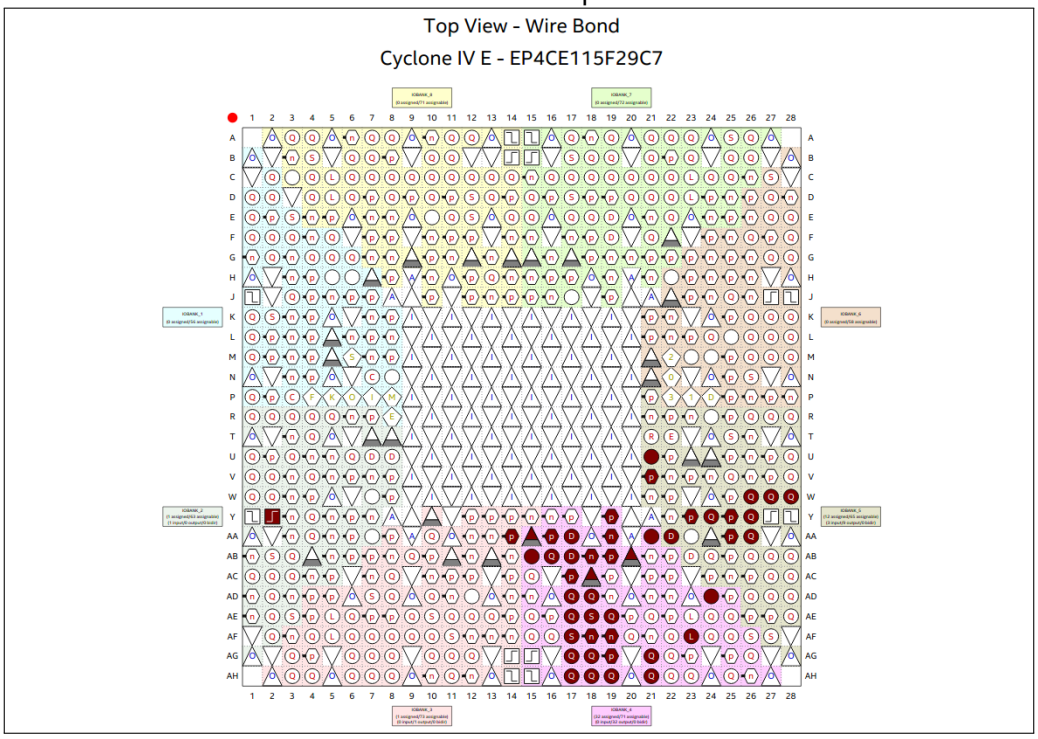

Figura 5: Configuração do CHIP após pinagem

Em seguida, fiz o envio do código para a placa e fiz testes aumentando e diminuindo o valor do clock para validar a troca de sec, min e hora.

FONTE: Elaborado pelo autor

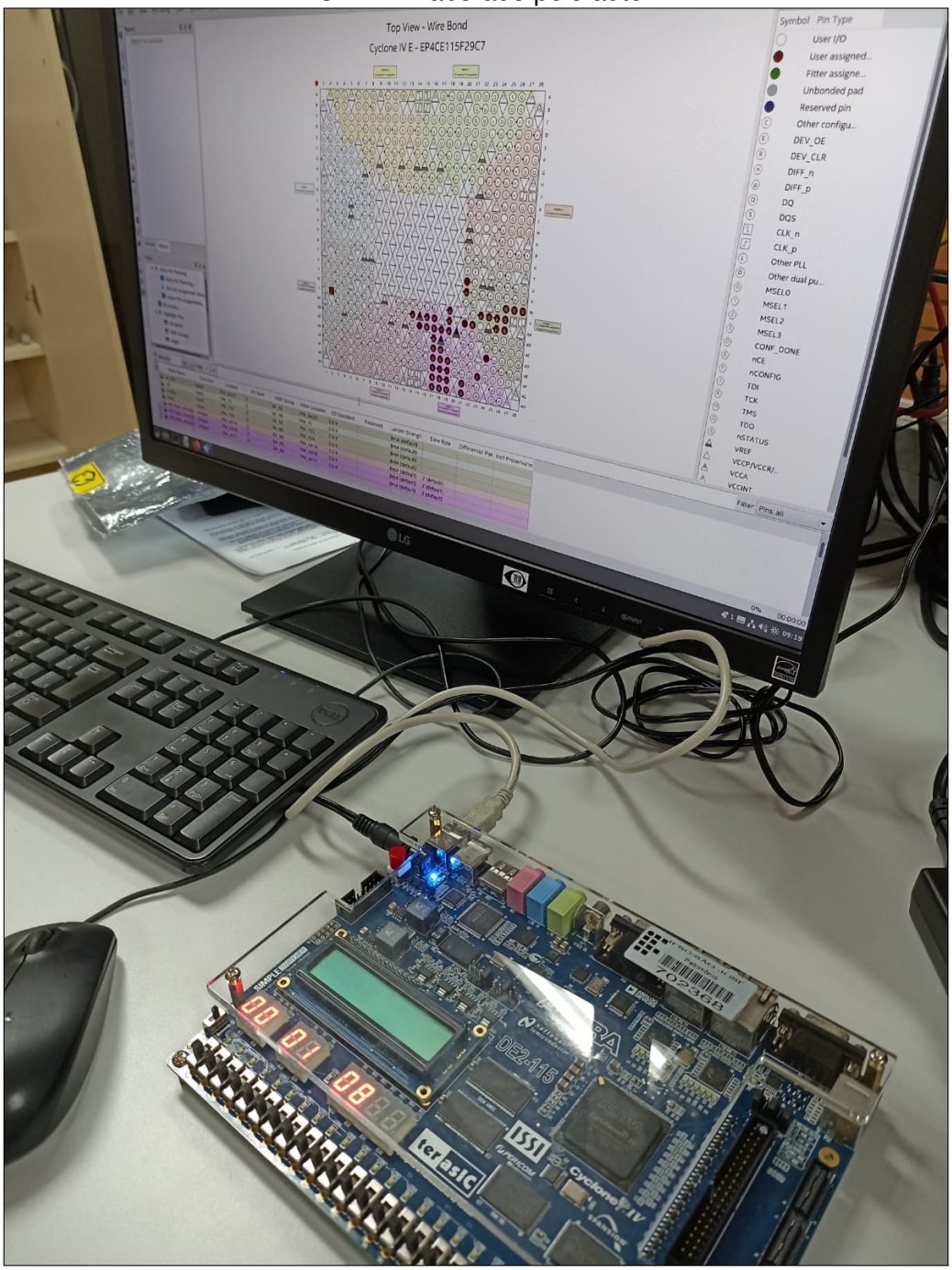

Figura 6: Relógio em operação após envio para a placa

Também alterei a polaridade dos leds para verificar se a inversão ocorre corretamente. Acima há uma imagem mostrando o relógio contando um horário corretamente.

2 Referências bibliográficas

Orientações do laboratório

Simulação Funcional usando o ModelSim