FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Conceção de Sistemas Digitais

1º Semestre 2020 / 2021

Relatório

Modelação e implementação de um controlador de parque de estacionamento

Docentes: Luís Gomes
Anikó Costa

Número de Aluno	Nome	Turno Prático
49341	João Carvalho	P1
52619	João Almeida	P1

1. Introdução

Neste trabalho foi-nos proposto implementar um controlador de um parque de estacionamento.

A modelação e simulação do controlador foi feita em rede de Petri através do programa IOPT-Tools, responsável também por converter o modelo em código VHDL, utilizado posteriormente no programa Xilinx.

Os objetivos neste trabalho eram, para além de, controlar as entrada e saídas de carros no parque através do controlo das cancelas presentes nas mesmas, associar um relógio responsável pela abertura e fecho do parque, também ter em conta passagem de carros entre os andares e ao mesmo tempo obter informação sobre a lotação do parque de estacionamento.

Sendo que as características deste parque de estacionamento ficaram ao nosso critério, optámos por escolher um parque de estacionamento com 4 lugares no total (distribuídos simetricamente por andar), com dois andares, duas entradas (uma em cada piso), 2 rampas no interior (com sentidos únicos e opostos) e duas saídas, também uma em cada piso. Na figura abaixo encontra-se a representação do esquemático da implementação do modelo final:

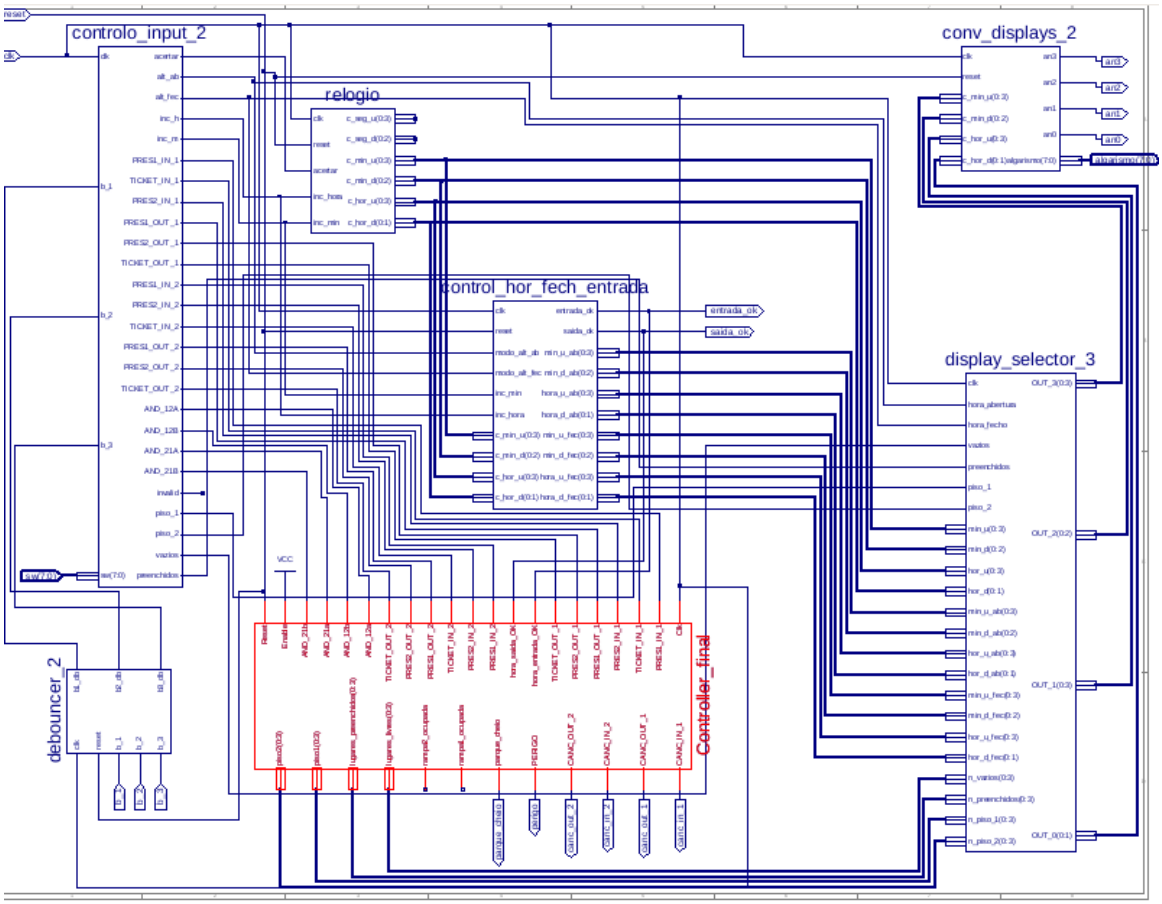

Figura.1- Esquemático do modelo global do controlador do parque.

2. Modelação do problema em Redes de Petri

O controlador foi modelado através da plataforma IOPT-Tools, fornecida pela UNINOVA, e permitiu obter uma rede de Petri que com base nas características do parque de estacionamento em cima referidas. Para tal, concluímos que necessitamos de 18 variáveis de entrada e 11 variáveis de saída:

Variáveis de entrada:

- PRES1_IN_1- sensor que deteta a presença de um carro à entrada do piso 1.
- PRES2_IN_1- sensor que confirma a entrada correta do carro no piso 1.
- TICKET_IN_1- bilhete retirado pelo condutor à entrada do piso 1.
- PRES1_OUT_1- sensor que deteta a presença de um carro à saída do piso 1.
- PRES2_OUT_1- sensor que confirma a saída correta do carro no piso 1.
- TICKET_OUT_1- sensor que deteta a introdução do bilhete de pagamento.

- PRES1_IN_2- sensor que deteta a presença de um carro na entrada do piso 2.
- PRES2_IN_2- sensor que confirma a entrada correta do carro no piso 2.
- TICKET_IN_2-bilhete retirado pelo condutor à entrada do piso 2.
- PRES1_OUT_2- sensor que deteta a presença um carro à saída do piso 2.
- PRES2_OUT_2- sensor que confirma a saída correta do carro no piso 2.
- TICKET_OUT_2- sensor que deteta a introdução do bilhete de pagamento.

- AND_12a- primeiro sensor de passagem na rampa que liga o piso 1 ao piso 2.
- AND_12b- segundo sensor de passagem na rampa que liga o piso 1 ao piso 2.
- AND_21a- primeiro sensor de passagem na rampa que liga o piso 2 ao piso 1.
- AND_21b- segundo sensor de passagem na rampa que liga o piso 2 ao piso 1.

- hora_entrada_OK- caso a hora de entrada seja após a hora de abertura.
- hora_saida_OK- caso a hora de saída seja até uma hora após a hora de fecho.

Variáveis de saída:

- CANC_IN_1- cancela de entrada no piso 1.
- CANC_OUT_1- cancela de saída do piso 1.
- CANC_IN_2- cancela de entrada no piso 2.
- CANC_OUT_2- cancela de saída do piso 2.
- piso1(0:2)- número de lugares ocupados no piso 1, em binário.
- piso2(0:2)- número de lugares ocupados no piso 2, em binário.
- piso1_cheio- piso 1 lotado.
- piso2_cheio- piso 2 lotado.
- PERIGO- carro em marcha-trás/ sentido proibido numa das rampas de comunicação.
- lugares_livres (0:2)- parque com lugares disponíveis, em binário.
- lugares_preenchidos (0:2)- número de lugares de ocupados no total do parque, em binário.

No modelo das redes de petri encontram-se os sub-modelos responsáveis por funções específicas no parque que interligadas garantem o total e correto funcionamento do controlador, os sub-modelos que o dividem são:

Sub-modelo de Entrada:

Cada zona de entrada dispõe de um sensor PRES1_IN instalado no pavimento que deteta a entrada de um novo carro na zona de receção. Para que a cancela levante permitindo o acesso ao parque, é necessário que, para além de existir pelo menos um lugar disponível para estacionar, o condutor retire um bilhete de acesso (que se admite que tenha registado a sua hora de entrada) e provocará a ativação da entrada TICKET_IN. A cancela é levantada através da ativação da variável de saída CANC_IN.

Existe um segundo sensor de pavimento PRES2_IN instalado após a cancela que confirma que a viatura entrou efetivamente no parque. Admite-se que a viatura atua com ambos os sensores quando entra (isto é, que não é possível o carro ficar entre os dois sensores sem atuar pelo menos um deles). A cancela deverá manter-se aberta enquanto qualquer das variáveis de entrada PRES1_IN ou PRES2_IN se mantiverem ativas. Pretende-se garantir a deteção de anomalias em que o condutor faça marcha-atrás em qualquer situação.

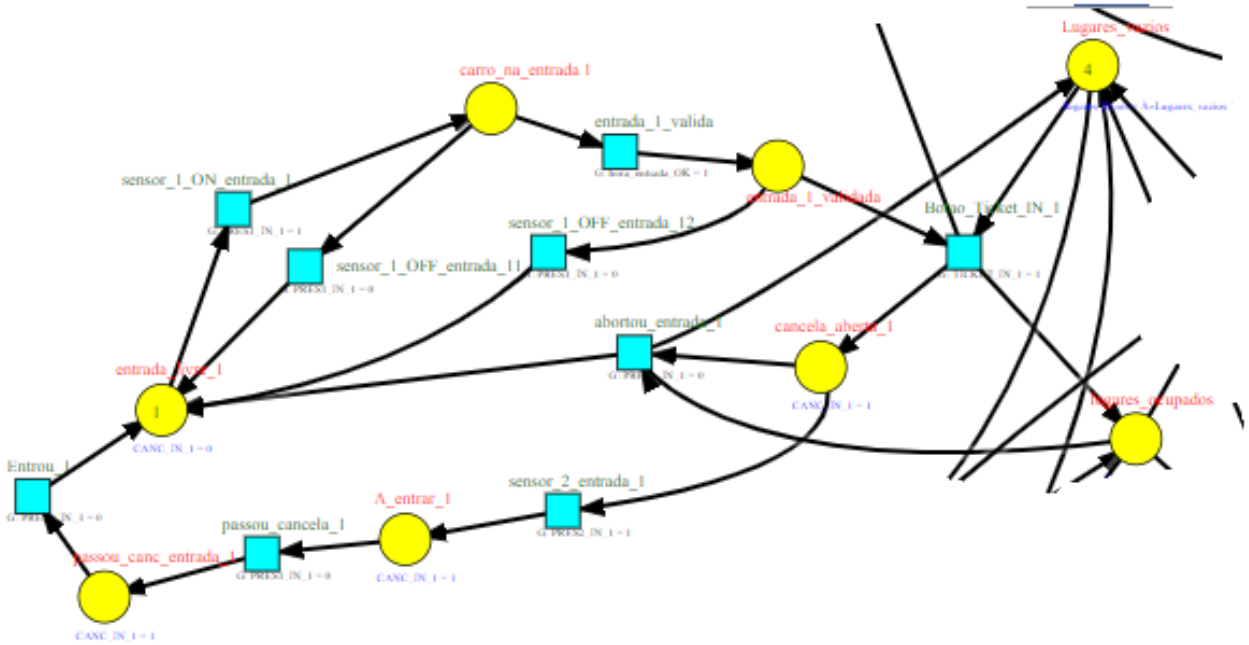

Figura 2- Modelo de RdP para a entrada de carros nos pisos 1.

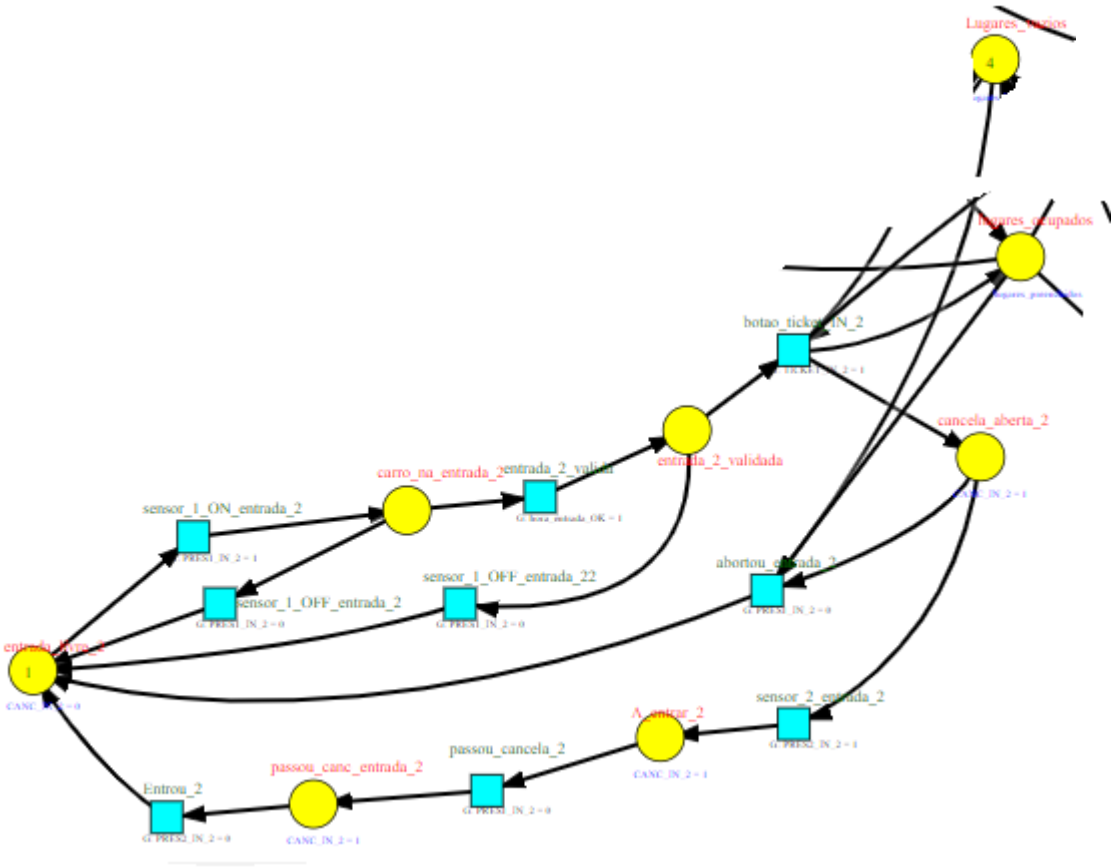

Figura 3- Modelo de RdP para a entrada de carros nos pisos 2.

Sub-modelo de Saída:

Cada zona de saída dispõe de dois sensores de pavimento PRES1_OUT e PRES2_OUT, de um sinal TICKET_OUT confirmando a introdução de um bilhete de pagamento, e de variável de saída CANC_OUT, de forma semelhante a zona de entrada.

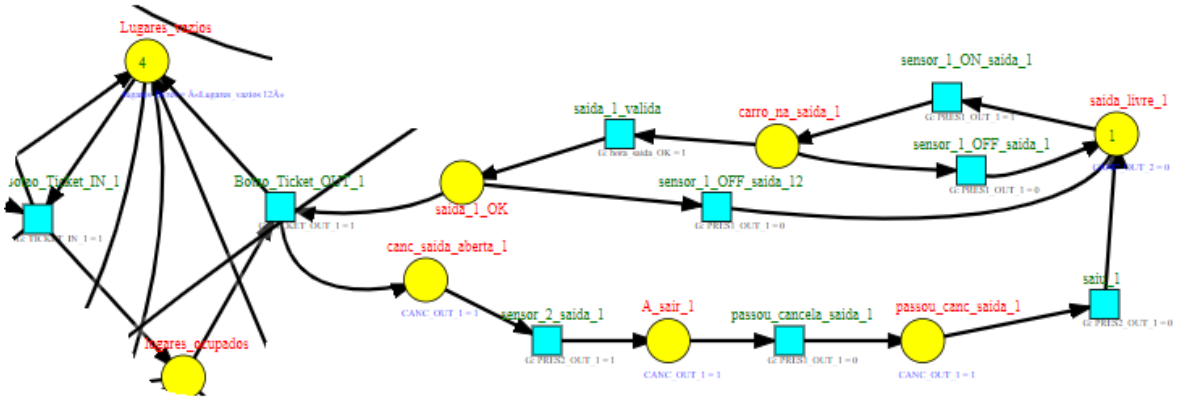

Figura 4- Modelo de RdP para a saída de carros do piso 1.

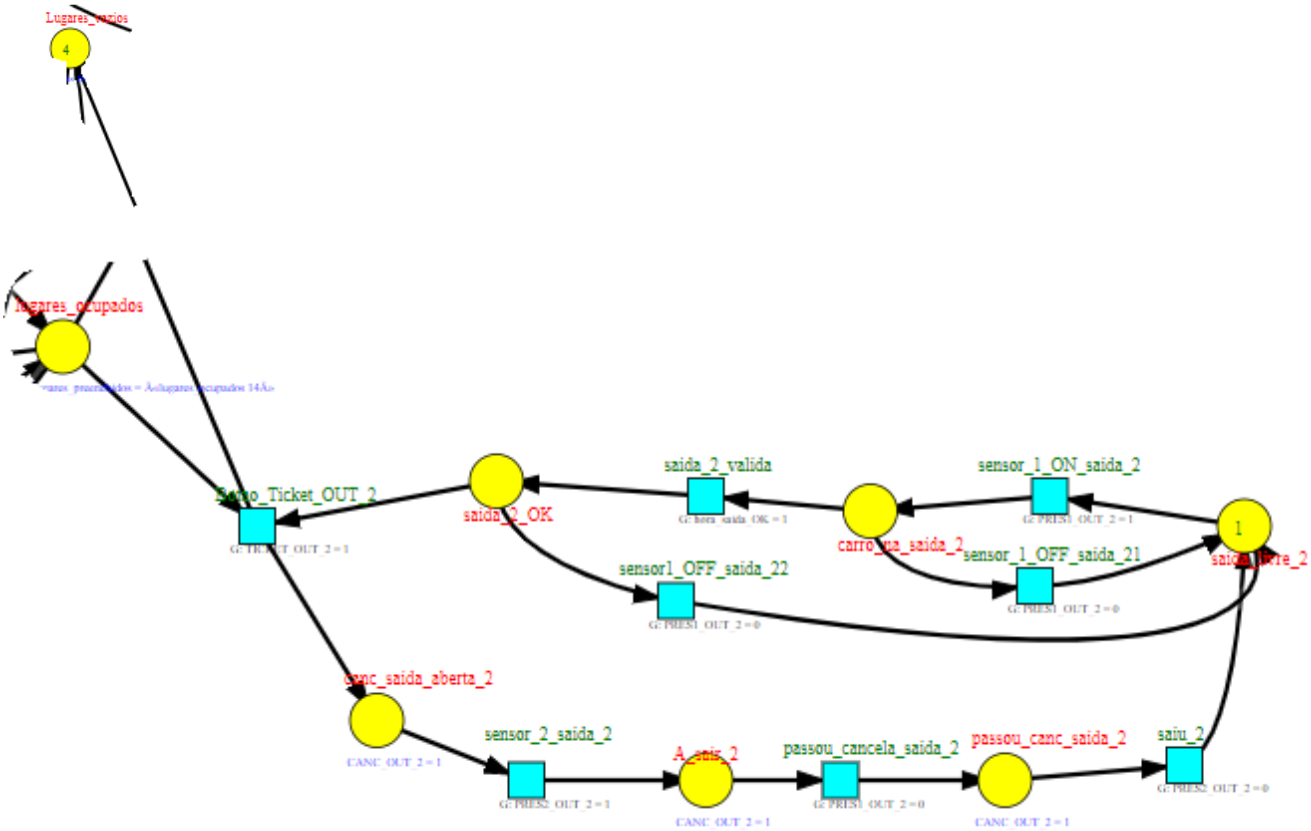

Figura 5- Modelo de RdP para a saída de carros do piso 2.

Sub-modelo da Rampa:

As rampas de comunicação entre andares dispõem de um par de sensores de passagem AND_12a e AND_12b (em que 1 designa o andar de origem e 2 o andar de destino), AND_21a e AND_21b (designa a rampa no sentido contrário) que, quando atuados em sequência garantem que a viatura entrou para o andar respetivo, (a distância entre estes sensores é inferior ao comprimento das viaturas, permitindo detetar situações de "marcha-atrás", bem como situações de "trânsito em sentido proibido")

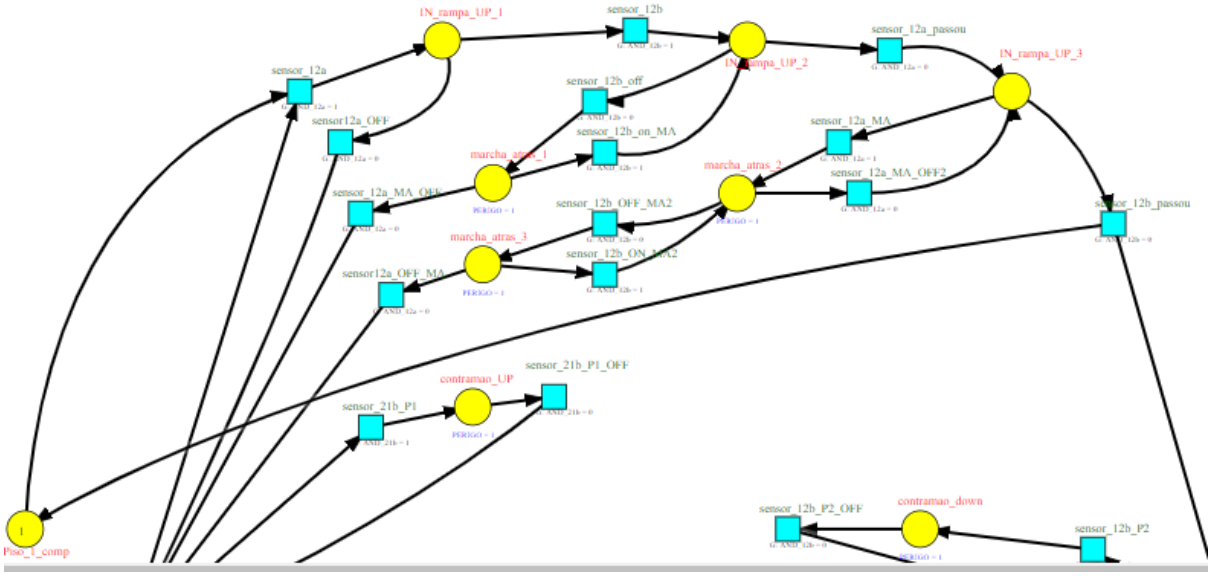

Figura 6- Modelo de RdP da rampa de comunicação entre andar 1 e 2.

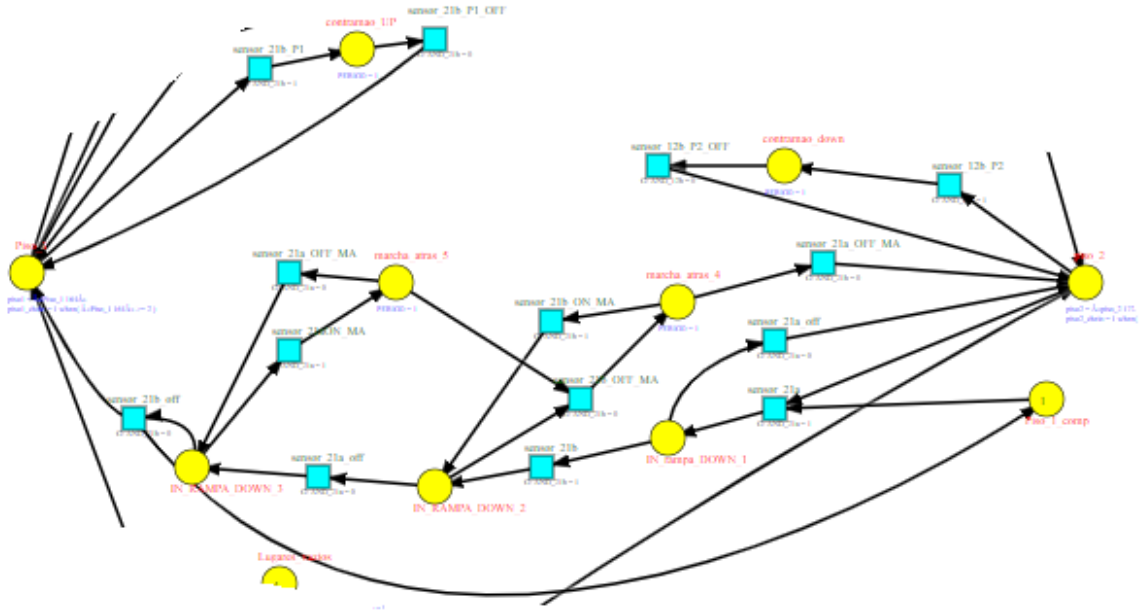

Figura 7- Modelo de RdP da rampa de comunicação entre andar 2 e 1.

3. Implementação em VHDL

Todos os sub-módulos têm interligados o mesmo sinal de clock (clk) e de reset para garantir o sincronismo com a placa e a capacidade de voltar ao estado inicial e limpar todas as variáveis alteradas.

Sub Módulo do Controlador

O código VHDL correspondente ao módulo de controlo do parque implementado em Redes de Petri corresponde ao seguinte módulo na Figura 8 abaixo. De notar que as saídas se encontram à esquerda e as entradas à direita do símbolo.

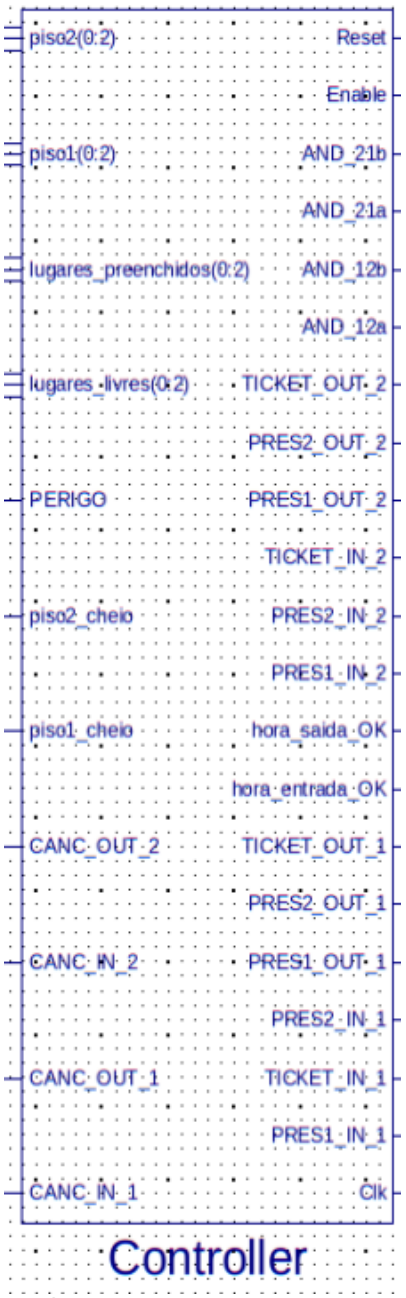

Figura.8- Módulo do controlador do parque de estacionamento.

Sub módulo Input Selector

Como a FPGA não tem sinais de entrada suficientes para a quantidade de variáveis de entrada que o controlador modelado requer, foi necessário criar uma codificação de estados com as 8 entradas existentes na placa, este “manual de utilização” da placa é explicado abaixo juntamente com os processos que o descrevem nas Figuras 10, 11, 12.

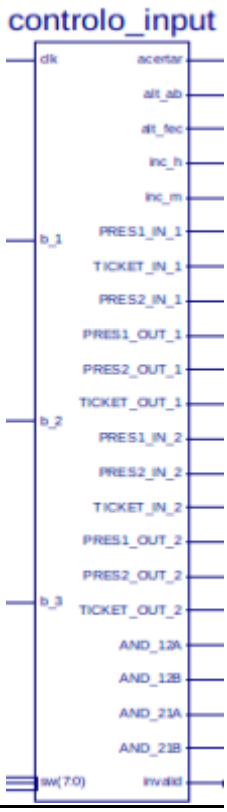

Figura 9- Módulo do controlo de entradas na placa.

O sistema apresenta 4 modos de utilização controláveis pelos SW6, SW5 e SW4 da placa, sendo eles:

- **Modo de controlo do parque**
  1. 000 – Mostrar horas no display
  2. 100 – Mostrar lugares vazios no parque
  3. 101 – Mostrar lugares preenchidos no parque
  4. 110 – Mostrar lugares preenchidos no piso 1
  5. 111 – Mostrar lugares preenchidos no piso 2

  Dentro do modo de controlo do parque, é possível selecionar com os SW2, SW1 e SW0, a parcela do parque que queremos controlar. As diferentes combinações destas entradas, dão significados diferentes aos BTN2, BTN1 e BTN0 da placa, conseguindo assim abranger todos os sensores do sistema.

  1. 000 – Entrada 1
     - BN1 - PRES1_IN_1
     - BN2 -TICKET_IN_1
     - BN3 - PRES1_IN_1
  2. 001 – Entrada 2
     - BN1 - PRES1_OUT_1
     - BN2 – TICKET_OUT_1
     - BN3 - PRES1_OUT_1
  3. 010 – Entrada 2
     - BN1- PRES1_IN_2
     - BN2- TICKET_IN_2

        - BN3 – PRES2_IN_2
    4. 011 – Saída 2
        - BN1 - PRES1_OUT_1
        - BN2 - TICKET_OUT_2
        - BN3 – PRES2_OUT_2
    5. 100 – Rampa 1-2
        - BN1 – AND_12a
        - BN2 – AND_12b
    6. 101 – Rampa 2-1
        - BN1 – AND21a
        - BN2 – AND21b
- **Modo de acerto do relógio**
    - BN1 – incrementar hora
    - BN2 – incrementar minuto

- **Modo acerto hora de entrada**
    - BN1 – incremento hora
    - BN2 – incremento minuto

- **Modo acerto hora de fecho**
    - BN1 – incremento hora
    - BN2 – incremento minuto

Para que estas operações ocorram com sucesso é preciso garantir que os últimos 3 bits (5 a 7) do bus estão a 0.

Abaixo está uma amostra representativa do código para introdução e remoção de um carro no piso 1:

```
case sw(2 downto 0) is
   when "000" =>  if b_1 = '1' then
                     PRES1_IN_1_aux <= '1';
                  else PRES1_IN_1_aux <= '0';
                  end if;
                  if b_2 = '1' then
                     TICKET_IN_1_aux <= '1';
                  else TICKET_IN_1_aux <= '0';
                  end if;
                  if b_3 = '1' then
                     PRES2_IN_1_aux <= '1';
                  else PRES2_IN_1_aux <= '0';
                  end if;
   when "001" => if b_1 = '1' then
                     PRES1_OUT_1_aux <= '1';
                  else PRES1_OUT_1_aux <= '0';
                  end if;
                  if b_2 = '1' then
                     TICKET_OUT_1_aux <= '1';
                  else TICKET_OUT_1_aux <= '0';
                  end if;
                  if b_3 = '1' then
                     PRES2_OUT_1_aux <= '1';
                  else PRES2_OUT_1_aux <= '0';
                  end if;
   when "010" => if b_1 = '1' then
                     PRES1_IN_2_aux <= '1';
                  else PRES1_IN_2_aux <= '0';
                  end if;
```

Figura 10- Amostra de código responsável pelo controlo do parque.

Foram também definidos outros modos de utilização da placa, colocando os primeiros bits a 0, a codificação dos últimos 3 bits rege-se da seguinte maneira:

- 100- Modo de acertar o relógio.

```
---------------- Modo acertar relogio----------------
if sw(5) = '1' and sw(6) = '0' and sw(7) = '0' then
   alt_ab_aux <= '0';
   alt_fec_aux <= '0';
   acertar_aux <= '1';
   if b_1 = '1' then
      inc_h_aux <= '1';
   else inc_h_aux <= '0';
   end if;
   if b_2 = '1' then
      inc_m_aux <= '1';
   else inc_m_aux <= '0';
   end if;
end if;
```

FIgura 11- Amostra de código responsável por colocar em modo de acerto de relógio.

- 010- Modo de alterar hora de abertura do parque de estacionamento.
- 110- Modo de alterar hora de fecho do parque de estacionamento.

```
------------------ Modo alterar hora de abertura----
if sw(5) = '0' and sw(6) = '1' and sw(7) = '0' then
   alt_ab_aux <= '1';
   alt_fec_aux <= '0';
   acertar_aux <= '0';
   if b_1 = '1' then
      inc_h_aux <= '1';
   else inc_h_aux <= '0';
   end if;
   if b_2 = '1' then
      inc_m_aux <= '1';
   else inc_m_aux <= '0';
   end if;
end if;
----------------------- Modo alterar hora de fecho---
if sw(5) = '1' and sw(6) = '1' and sw(7) = '0' then
   alt_fec_aux <= '1';
   acertar_aux <= '0';
   alt_ab_aux <= '0';
   if b_1 = '1' then
      inc_h_aux <= '1';
   else inc_h_aux <= '0';
   end if;
   if b_2 = '1' then
      inc_m_aux <= '1';
   else inc_m_aux <= '0';
   end if;
end if;
```

Figura 12- Amostra de código responsável por colocar em modo de acerto de horas de abertura e fecho

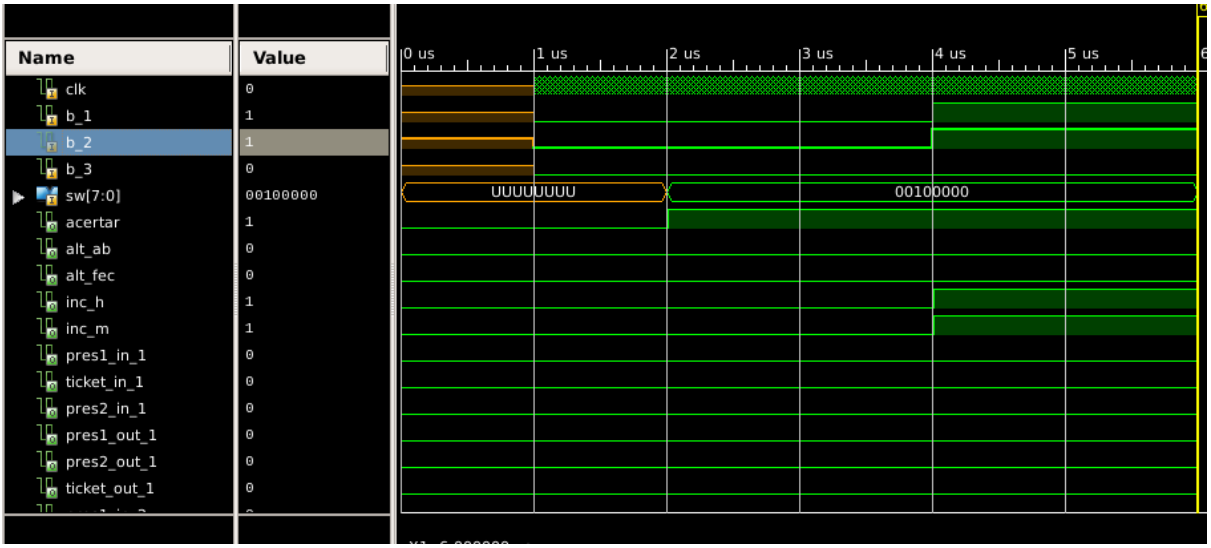

Figura 13 – Simulação do controlador de hora de abertura e fecho

Sub Módulo Relógio

Para a implementação do relógio, utilizámos o código dado no Lab. 2 e acrescentámos as horas e forma de incrementar os minutos e os segundos manualmente. Na figura abaixo encontra-se o símbolo gerado deste módulo.

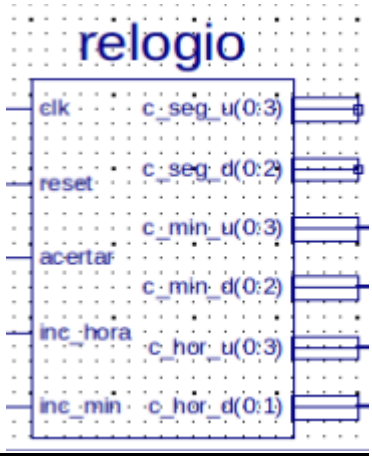

Figura 10- Símbolo do Módulo do relógio

Como se pode verificar, na figura abaixo, utiliza-se para as horas o mesmo método que se utiliza para implementar os segundos e os minutos.

```
process (clk, reset)
begin
   if reset = '1' then
      cont_hor_d <= 0;
   elsif clk'event and clk = '1' then
      if horas_d = '1' then
         if cont_hor_d = 2 then
            cont_hor_d <= 0;
         else
            cont_hor_d <= cont_hor_d + 1;
         end if;
      end if;
   end if;
end process;


c_hor_d <= cont_hor_d;
```

Figura 11- Implementação segundo dígito horas

Os dois processos abaixo representados são responsáveis por inicializar a incrementação manual das horas e dos minutos.

```
process (clk)
begin
   if clk'event and clk = '1' then
      if largou_h = '1' and inc_hora = '1' and acertar = '1' then
         inc_hora_aux <= '1';
         largou_h <= '0';
      end if;
      if largou_h = '0' and inc_hora = '0' then
         largou_h <= '1';
      end if;
   end if;
   if check_h = '1' then
      inc_hora_aux <= '0';
   end if;
end process;

inc_hora_aux2 <= '1' when inc_hora_aux = '1' else '0';
```

```
process (clk)
begin
   if clk'event and clk = '1' then
      if largou_m = '1' and inc_min = '1' and acertar = '1' then
         inc_min_aux <= '1';
         largou_m <= '0';
      end if;
      if largou_m = '0' and inc_min = '0' then
         largou_m <= '1';
      end if;
   end if;
   if check_m = '1' then
      inc_min_aux <= '0';
   end if;
end process;
inc_min_aux2 <= '1' when inc_min_aux = '1' else '0';
```

Figura 12 e 13- Implementação incrementação manual horas e minutos, respetivamente.

Podemos verificar que a variável inc_min_aux2 irá incrementar o algarismo das dezenas dos minutos, fazendo avançar 10 minutos. O processo é semelhante para as horas com a variável inc_hora_aux2 mas para o algarismo das unidades, fazendo avançar uma hora

```
process (clk, reset)
begin
   if reset = '1' then
     cont_min_d <= 0;
   elsif clk'event and clk = '1' then
     if check_m_aux = '1' then
      check_m_aux <= '0';
     end if;
     if min_d = '1' or inc_min_aux2 = '1' then
       if cont_min_d = 5 then
         cont_min_d <= 0;
       else
         cont_min_d <= cont_min_d + 1;
         check_m_aux <= '1';
       end if;
     end if;
   end if;
end process;
check_m <= '1' when check_m_aux = '1' else '0';
horas_u <= '1' when (cont_min_d = 5 and min_d = '1' )  else '0';
c_min_d <= cont_min_d;
```

Figura 14- Incrementação minutos, seja manual ou automática

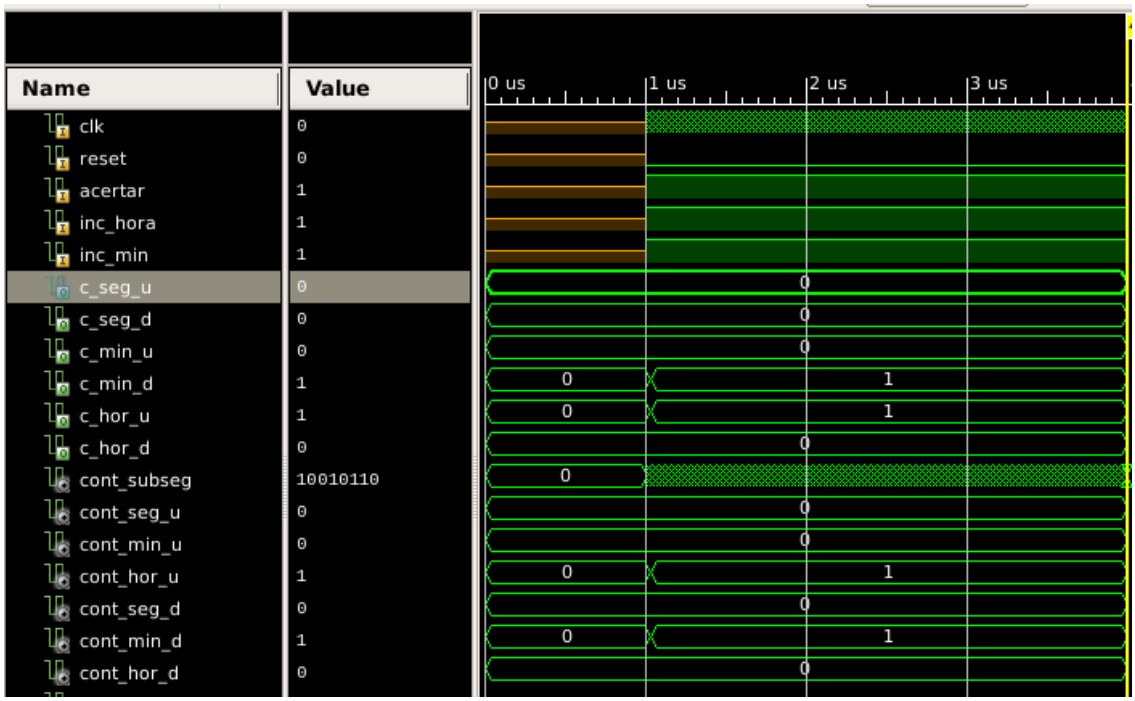

Figura 15- simulação do relógio

Sub Módulo de Display Selector

Este é o módulo responsável pela multiplexagem dos sinais que podem aparecer no display de 7 segmentos e está diretamente relacionado com o módulo “conv_display_2”, sendo que as saídas do módulo display_selector são as entradas do mencionado acima. Na figura abaixo está representado o símbolo gerado do mesmo.

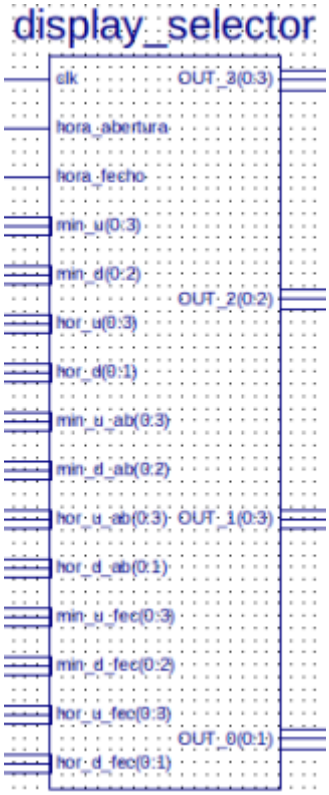

Figura 15 - Símbolo do Módulo responsável pela seleção do output no display de 7 segmentos

No display podem ser apresentados as seguintes informações: hora de abertura do parque, hora de fecho do mesmo, horas e minutos em tempo real.

A escolha destas saídas depende de se entradas hora_abertura ou hora_fecho que são estimuladas pelo módulo de controlo da hora de abertura e de fecho.

```
process(clk)
begin
   if clk'event and clk = '1' then
      if hora_abertura = '1' then
         OUT_3 <= min_u_ab;
         OUT_2 <= min_d_ab;
         OUT_1 <= hor_u_ab;
         OUT_0 <= hor_d_ab;
      elsif hora_fecho = '1' then
         OUT_3 <= min_u_fec;
         OUT_2 <= min_d_fec;
         OUT_1 <= hor_u_fec;
         OUT_0 <= hor_d_fec;
      else
         OUT_3 <= min_u;
         OUT_2 <= min_d;
         OUT_1 <= hor_u;
         OUT_0 <= hor_d;
      end if;
   end if;
end process;
```

Figura 16 - Amostra de código com o processo de multiplexagem das saídas para o display.

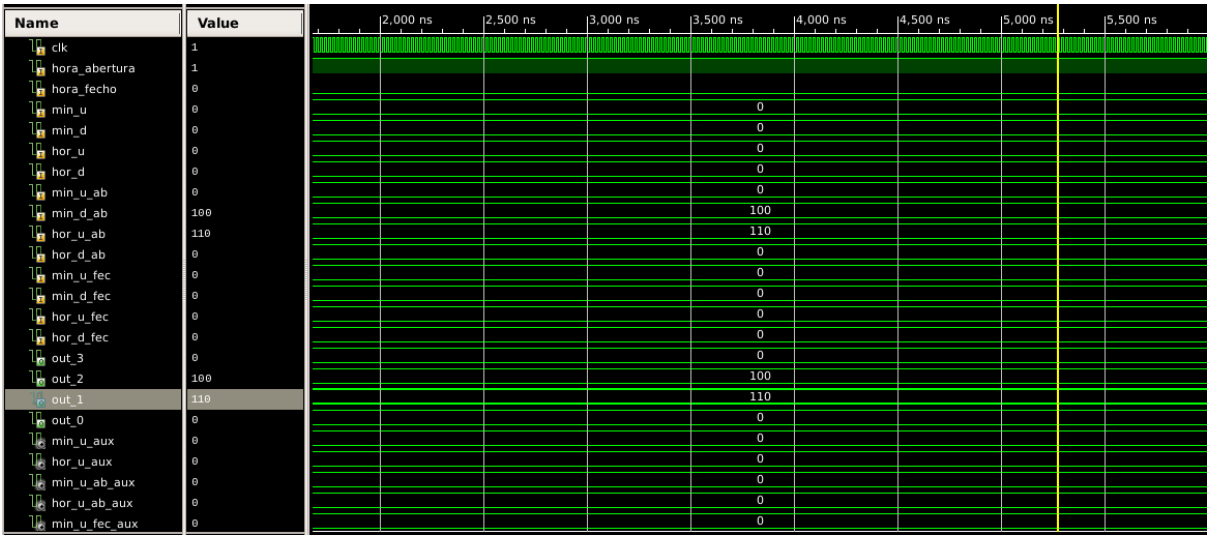

Figura 17 – Simulação do selector de informação a apresentar no display

Sub Módulo de Display

Este módulo foi nos dado no lab.2 quando se deu o começo da implementação do relógio e da primeira implementação num placa FPGA e o seu propósito é converter código para displays e a atualizar os valores nos mesmos.

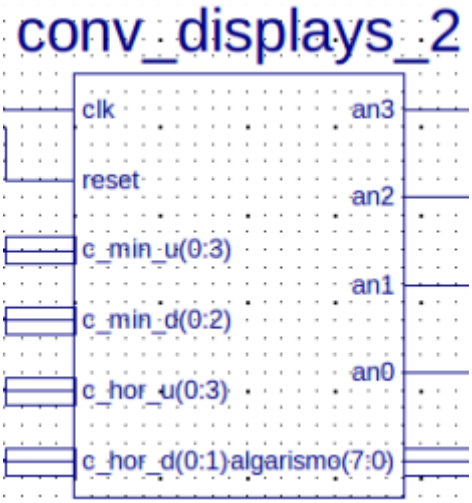

Figura 17 - Símbolo do Módulo responsável pela colocação de

As entradas deste módulo, à excepção do *clk* e do *reset* provém do módulo de multiplexagem de saídas para o display analisado anteriormente, o "*display_selector*". Estas entradas escolhidas previamente correspondem, em binário, aos valores que vão ser convertidos em decimal e apresentados no display.

Em relação às saídas as variáveis an0, an1, an2 e an3 são os dígitos que controlam a atualização no display da FPGA.
O bus algarismo é o código de leds para representar o número no respetivo dígito.

Sub Módulo Controlo de horas de entrada e de fecho

Este módulo é responsável pela comparação entre a hora de entrada/saída de um carro com a hora de abertura/fecho do parque de estacionamento e conforme o resultado dessas comparações vão variando os valores das saídas “entrada_ok” e “saída_ok”.Foi então definido que deve ser negado o acesso ao parque a um carro cuja hora de entrada precede a hora de abertura e deve ser negada a saída a um carro cuja hora de saída exceda em mais de uma hora a hora do fecho.

Este módulo é também responsável pela alteração das horas de abertura e de fecho e esse estímulo vem do módulo “controlo_input”, caso seja escolhido algum destes modos através dos interruptores.

Na figura abaixo encontra-se o símbolo gerado deste módulo.

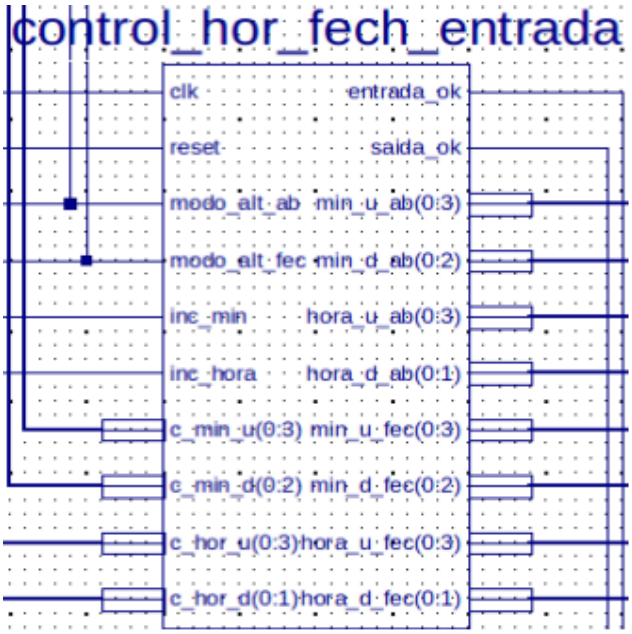

Figura 18 - Símbolo do Módulo responsável pelo o controlo de entradas e saídas de acordo com as horas

As alterações na hora de abertura e de fecho são feitas através dos seguintes processos:

- Alteração das dezenas de minutos na hora de fecho ou de entrada, como se definiu que apenas se incrementa de dez em dez minutos, não houve necessidade de um processo específico para essa componente.

```
process (clk, reset)
   begin
   if reset = '1' then
     min_d_ab_aux <= 0;
   elsif clk'event and clk = '1' then
      if check_ab_m_aux = '1' then
         check_ab_m_aux <= '0';
      end if;
      if modo_alt_ab = '1' then
        if  inc_min_ab_aux2 = '1' then
          if min_d_ab_aux = 5 then
            check_ab_m_aux <= '1';
            min_d_ab_aux <= 0;
          else
            min_d_ab_aux <= min_d_ab_aux + 1;
            check_ab_m_aux <= '1';
          end if;
        end if;
        elsif modo_alt_fec = '1' then
           if  inc_min_ab_aux2 = '1' then
            if min_d_fec_aux = 5 then
              check_ab_m_aux <= '1';
              min_d_fec_aux <= 0;
            else
              min_d_fec_aux <= min_d_fec_aux + 1;
              check_ab_m_aux <= '1';
            end if;
           end if;
       end if;
   end if;
end process;

min_d_ab <= min_d_ab_aux;
min_d_fec <= min_d_fec_aux;
```

Figura 19 - Amostra de código com o processo de alteração dos minutos das horas de funcionamento do parque.

- Alteração das unidades da hora de abertura e fecho:

```
process (clk, reset)
begin
   if reset = '1' then
      hora_u_ab_aux <= 0;
   elsif clk'event and clk = '1' then
      if check_ab_h_aux = '1' then
         check_ab_h_aux <= '0';
      end if;
      if modo_alt_ab = '1' then
         if inc_hora_ab_aux2 = '1' then
            if hora_d_ab_aux = 2 and hora_u_ab_aux = 3 then
               hora_u_ab_aux <= 0;
               check_ab_h_aux <= '1';
            elsif hora_u_ab_aux = 9 then
               hora_u_ab_aux <= 0;
               check_ab_h_aux <= '1';
            else
               hora_u_ab_aux <= hora_u_ab_aux + 1;
               check_ab_h_aux <= '1';
            end if;
         end if;
      elsif modo_alt_fec = '1' then
         if inc_hora_ab_aux2 = '1' then
            if hora_d_fec_aux = 2 and hora_u_fec_aux = 3 then
               hora_u_fec_aux <= 0;
               check_ab_h_aux <= '1';
            elsif hora_u_fec_aux = 9 then
               hora_u_fec_aux <= 0;
               check_ab_h_aux <= '1';
            else
               hora_u_fec_aux <= hora_u_fec_aux + 1;
               check_ab_h_aux <= '1';
            end if;
         end if;
      end if;
   end if;
end process;
h_d_ab_signal <= '1' when (hora_u_ab_aux = 9 and  inc_hora_ab_aux2 = '1') or (hora_u_ab_aux = 3 and hora_d_ab_aux = 2 and  inc_hora_ab_aux2 = '1') else '0';
h_d_fec_signal <= '1' when (hora_u_fec_aux = 9 and  inc_hora_ab_aux2 = '1') or (hora_u_fec_aux = 3 and hora_d_fec_aux = 2 and  inc_hora_ab_aux2 = '1') else '0';
hora_u_ab <= hora_u_ab_aux;
hora_u_fec <= hora_u_fec_aux;
check_ab_m <= '1' when check_ab_m_aux = '1' else '0';
check_ab_h <= '1' when check_ab_h_aux = '1' else '0';
```

Figura 20- Processo responsável pela alteração das unidades da hora de abertura e de fecho.

- Alteração das dezenas da hora de abertura e fecho:

```
process(clk, reset)
begin
   if reset = '1' then
      hora_d_ab_aux <= 0;
   elsif clk'event and clk = '1' then
      if modo_alt_ab = '1' then
         if h_d_ab_signal = '1' then
            if hora_d_ab_aux = 2 then
               hora_d_ab_aux <= 0;
            else
               hora_d_ab_aux <= hora_d_ab_aux + 1;
            end if;
         end if;
      elsif modo_alt_fec = '1' then
         if h_d_fec_signal = '1' then
            if hora_d_fec_aux = 2 then
               hora_d_fec_aux <= 0;
            else
               hora_d_fec_aux <= hora_d_fec_aux + 1;
            end if;
         end if;
      end if;
   end if;
end process;

hora_d_ab <= hora_d_ab_aux;
hora_d_fec <= hora_d_fec_aux;
```

Figura 21 - Processo responsável pela alteração das dezenas da hora de abertura e de fecho.

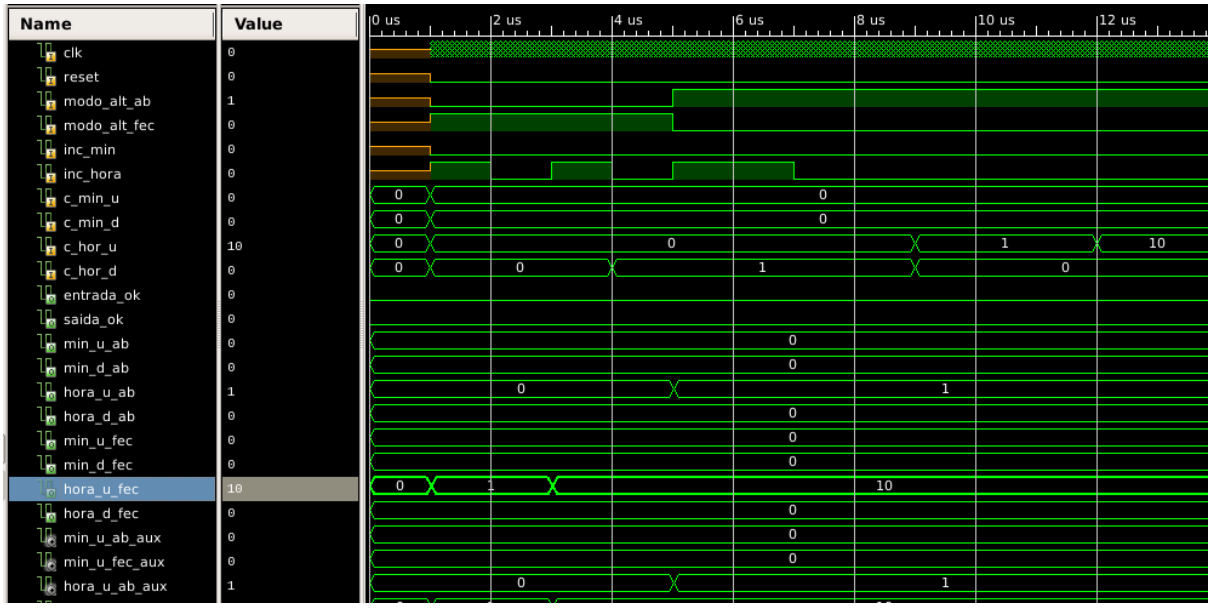

Figura 22- Simulação do módulo para alterar as horas de abertura e fecho

A parte da comparação da hora de entrada/saída com a hora de abertura/ fecho+1 foi feita através de uma conversão em minutos como é descrito na porção de código apresentado na Figura 22.

```
-------------------------------------------------COMPARADOR-------------------------------
total_minutos <= (c_hor_d * 10 * 60) + (c_hor_u * 60) + (c_min_d * 10) + c_min_u;
total_minutos_ab <= (hora_d_ab_aux * 10 * 60) + (hora_u_ab_aux * 60) + (min_d_ab_aux * 10);
total_minutos_fec <= (hora_d_fec_aux * 10 * 60) + (hora_u_fec_aux * 60) + (min_d_fec_aux * 10);

process (clk)
begin
   if clk'event and clk = '1' then
      if total_minutos < total_minutos_ab then
         entrada_aux <= '0';
         saida_aux <= '0';
      elsif total_minutos > total_minutos_ab then
         if total_minutos < total_minutos_fec then
            entrada_aux <= '1';
            saida_aux <= '1';
         elsif (total_minutos > total_minutos_fec) and (total_minutos < total_minutos_fec +60) then
            entrada_aux <= '0';
            saida_aux <= '1';
         else
            entrada_aux <= '0';
            saida_aux <= '0';
         end if;
      end if;
   end if;
end process;

entrada_ok <= entrada_aux;
saida_ok <= saida_aux;
```

Figura 22 - Excerto de código responsável pelos valores dos sinais de saída "entrada_OK" e "saída_OK".

Sub módulo do Debounce

Tal como no primeiro trabalho de avaliação, foi necessário introduzir um módulo capaz de fazer uma leitura limpa dos inputs dados pelos três botões de pressão, visto que alguns geram confusão ao controlador ao enviarem mais valores do que o suposto e errados durante o tempo de pressão do mesmo.

Assim sendo, este módulo fica ao filtrar os sinais de entrada provenientes diretamente dos botões, liga ao módulo "controlo_input" que precisa destes valores corretos para a fazer as alterações necessárias nos vários modos de utilização da placa.

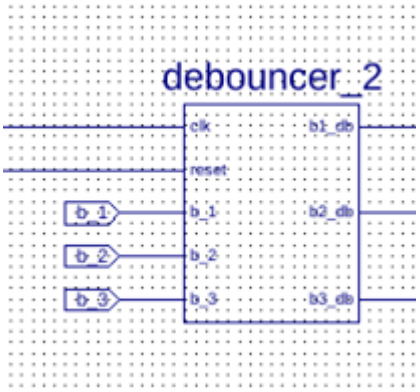

Figura 23 - Símbolo do Módulo responsável pela leitura correta dos botões de pressão.

Segue-se abaixo o processo responsável pela leitura correta do valor botão 1 da placa, seguido de um processo contador auxiliar, que assim que detecta '1', prende o sinal a '1' durante MAXCOUNT.

Aplicou-se o mesmo raciocínio aos processos que se referem aos botões.

```
---------------------------------- Botao b_1------------
process(clk, reset)
begin
      if reset = '1' then
         Stateb1_aux <= 0;
      elsif clk'event and clk = '1' then
         if Stateb1_aux = 0 and b_1 = '1' then
            Stateb1_aux <= 1;
            b1_db_aux <= '1';
         end if;
         if Stateb1_aux = 1 and count_maxb1 = '1' then
            Stateb1_aux <= 2;
            b1_db_aux <= '1';
         end if;
         if Stateb1_aux = 2 and b_1 = '1' then
            Stateb1_aux <= 1;
            b1_db_aux <= '1';
         elsif Stateb1_aux = 2 and b_1 = '0' then
            Stateb1_aux <= 0;
            b1_db_aux <= '0';
         end if;
      end if;
end process;

Stateb1 <= Stateb1_aux;
b1_db <= '1' when b1_db_aux = '1' else '0';
```

Figura 24 - Código do processo de debounce do botão 1.

```
process (clk, reset)
begin
   if reset = '1' then
   elsif clk'event and clk = '1' then
      if Stateb1 = 1 then
         if b1_count = MAXCOUNT - 1 then
            count_maxb1_aux <= '1';
            b1_count <= 0;
         else
            count_maxb1_aux <= '0';
            b1_count <= b1_count + 1;
         end if;
      end if;
   end if;
end process;
 count_maxb1 <= '1' when count_maxb1_aux = '1' else '0';
```

Figura 25 - Código do processo do contador auxiliar para o debounce do botão 1.

4. Atribuição de PINS

Fizemos a atribuição dos pins para a implementação do programa na FPGA de acordo com as figuras abaixo representadas.

```
NET "sw<0>" LOC = "F12";
NET "sw<1>" LOC = "G12";
NET "sw<2>" LOC = "H14";
NET "sw<3>" LOC = "H13";
NET "sw<4>" LOC = "J14";
NET "sw<5>" LOC = "J13";
NET "sw<6>" LOC = "K14";
NET "sw<7>" LOC = "K13";
NET "b_1" LOC = "L13";
NET "b_2" LOC = "M14";
NET "b_3" LOC = "M13";
```

Figura 26 - Atribuição dos PINS.

```
NET "algarismo<0>" LOC = "N16";
NET "algarismo<1>" LOC = "F13";
NET "algarismo<2>" LOC = "R16";
NET "algarismo<3>" LOC = "P15";
NET "algarismo<4>" LOC = "N15";
NET "algarismo<5>" LOC = "G13";
NET "algarismo<6>" LOC = "E14";
NET "algarismo<7>" LOC = "P16";
NET "an0" LOC = "D14";
NET "an1" LOC = "G14";
NET "an2" LOC = "F14";
NET "an3" LOC = "E13";
NET "entrada_ok" LOC = "P11";
NET "saida_ok" LOC = "P12";
NET "perigo" LOC = "N12";
NET "piso_cheio" LOc = "P13";
NET "canc_out_2" LOC = "N14";
NET "canc_in_2" LOC = "L12";
NET "canc_out_1" LOC = "P14";
NET "canc_in_1" LOC = "K12";


NET "clk" LOC= "T9";
NET "reset" LOC = "L14";
```

Conclusão

Neste trabalho acabámos por não utilizar o UART desenvolvido no 1º trabalho de avaliação da cadeira, porém fizemos um relógio operacional capaz de alterar as horas e minutos. Achamos que ao realizar a opção do protocolo UART no primeiro trabalho ficámos em desvantagem em relação a quem fez o relógio pois nesta última modelação e implementação do controlador do parque era opcional a utilização do UART e obrigatória a realização do relógio.

Para modelar o comportamento do parque em si, pusemos em prática os nossos conhecimentos em redes de petri, e VHDL. Uma vez que o projeto necessita de interligar o comportamento do parque em si ao do relógio, foram necessários conhecimentos de modelação adquiridos ao longo do semestre, de forma a que todos os módulos do sistema coexistissem corretamente.

Tal como aconteceu no trabalho anterior, voltámos a ter problemas de debounce com os botões da placa, mas o mesmo foi ultrapassado como foi mencionado no capítulo da Implementação VHDL deste sub-módulo.

Em última análise podemos afirmar que o projeto foi útil pois conseguimos ter uma melhor noção de como juntar todos os módulos e tendo um módulo exterior ao nosso desenvolvimento.

Esta não é a última versão do trabalho, ainda iremos aperfeiçoar o funcionamento da mesma para que fique totalmente completo.