

# Projeto Final de LSD

Universidade de Aveiro

Guilherme Craveiro, Rafael Pinto



Versão 1

# Projeto Final de LSD

Departamento de Eletrónica, Telecomunicações e  
Informática

Universidade de Aveiro

Guilherme Craveiro, Rafael Pinto  
(103574) gjscraveiro@ua.pt, (103379) rafaelpbpinto@ua.pt

17 de junho de 2021

# Índice

<b>1</b>	<b>Introdução</b>	<b>1</b>
<b>2</b>	<b>Arquitetura</b>	<b>2</b>
2.1	Fase 1 . . . . .	2
2.2	Fase 2 . . . . .	3
<b>3</b>	<b>Implementação</b>	<b>4</b>
3.1	Fase 1 . . . . .	4
3.1.1	Máquina de Estados da Fase 1 . . . . .	4
3.1.2	<i>Display</i> . . . . .	7
3.1.3	Divisor da frequência <i>Clock</i> e Temporizador . . . . .	9
3.1.4	<i>Top-level</i> . . . . .	11
3.2	Fase2 . . . . .	13
3.2.1	Máquina de estados da Fase 2 . . . . .	13
3.2.2	<i>DebounceUnit</i> . . . . .	16
3.2.3	<i>Display</i> do tamanho das garrafas . . . . .	17
3.2.4	<i>Top-level</i> . . . . .	18
<b>4</b>	<b>Validação</b>	<b>19</b>
<b>5</b>	<b>Manual do utilizador</b>	<b>20</b>
<b>6</b>	<b>Conclusões</b>	<b>21</b>

# Capítulo 1

## Introdução

VHSIC Hardware Description Language (VHDL) é uma linguagem usada para modelar o comportamento e a estrutura de sistemas digitais em, por exemplo, Field Programmable Gate Array (FPGA). De forma muito resumida, FPGA é uma matriz de blocos lógicos interligados de modo inteligente que podem ser reprogramados para a aplicação desejada.

Este relatório tem como objetivo explicar o funcionamento da máquina automática de oferta de produtos desenvolvida no nosso projeto. Para que a máquina funcionasse foi necessário a implementação de código em VHDL e procedeu-se a vários testes na FPGA e à análise dos mesmos.

A máquina disponibiliza 3 bebidas diferentes das quais podemos selecionar uma através de *switches*. Após a escolha da bebida o utilizador pode ainda entrar no modo "Modo Escolha tamanho das garrafas", onde pode escolher o tamanho da garrafa, 25cl, 33cl, 50cl ou 10dl. Por defeito está configurado para 33cl. A máquina tem ainda um RESET global que coloca a máquina no estado inicial.

# Capítulo 2

## Arquitetura

### 2.1 Fase 1

Para a construção da fase 1 deste projeto, tive-se de implementar um temporizador, um divisor da frequência do *clock*, uma máquina de estados e uma estrutura que permitisse escrever mensagens nos *displays* de 7 segmentos, que estão representados na figura 2.1 pelos nomes *TimerFSM*, *ClkDivider*, *Fase1FSM* e *Display*, respetivamente.

A estrutura *ClkDivider* permite dividir a frequência do *clock* de 50 MHz e passar um *clk* de frequência 10 Hz para a estrutura *Display* que irá permitir colocar a palavra "OLA" a piscar à frequência de 10 Hz.

A estrutura *TimerFSM* foi implementada com o intuito de contar os tempos que a palavra "OLA" aparece a piscar nos *displays* de 7 segmentos e o *led* vermelho fica aceso. Quando esta estrutura fica ativada irá fazer uma contagem decrescente com o valor que lhe é passado pela máquina de estados *Fase1FSM*, quando chega a zero envia um sinal para a máquina de estados que significa que o tempo terminou.

A estrutura *Display* é a que vai permitir escrever nos *displays* de 7 segmentos. É ativada pelos *switches* ou, no caso de quando se pretender escrever "OLA", ativada quando recebe um sinal da máquina de estados *Fase1FSM*.

A máquina de estados *Fase1FSM* é a estrutura que nos permite colocar tudo a funcionar de forma síncrona e ordenada.

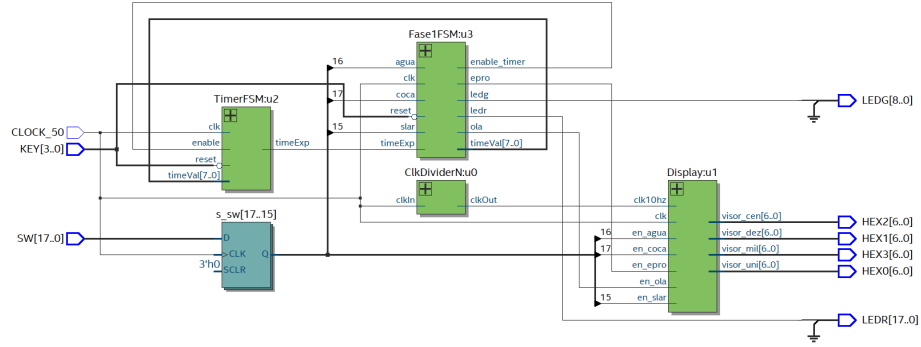


Figura 2.1: Arquitetura da Fase 1

## 2.2 Fase 2

Na implementação da fase 2 acrescentou-se 3 estruturas, que estão representadas na figura 2.2 pelos nomes, *DebounceUnit*, a *Display\_Tam\_Garrafa* e a *Sel\_Tam\_Garrafa*. Esta fase só é ativada quando o *switch* 0 está ativo no momento em que se escolhe a bebida.

A estrutura *DebounceUnit* gera um pulso de relógio que vai servir para escolher o tamanho da garrafa na estrutura *Sel\_Tam\_Garrafa*. Cada vez que é gerado esse sinal a máquina de estados avança para o estado seguinte.

A máquina de estados *Sel\_Tam\_Garrafa* é a estrutura que nos permite escolher o tamanho da garrafa.

A estrutura *Display\_Tam\_Garrafa* permite visualizar nos *displays* de 7 bits qual o tamanho de garrafa que estamos a escolher.

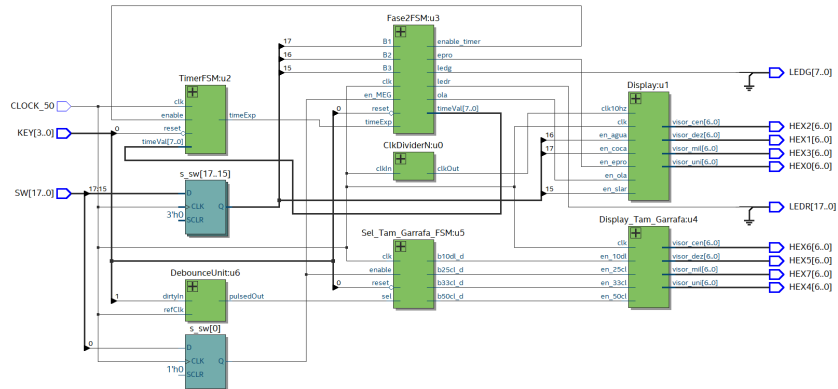


Figura 2.2: Arquitetura da Fase 2

## Capítulo 3

# Implementação

### 3.1 Fase 1

#### 3.1.1 Máquina de Estados da Fase 1

Primeiramente, desenhou-se a máquina de estados que se iria implementar na fase 1 (figura 3.1).

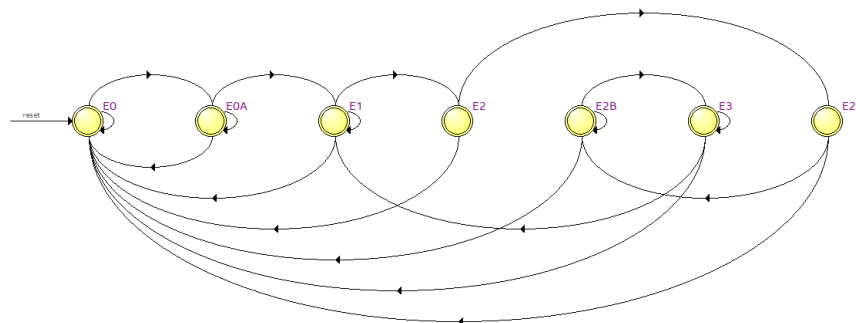


Figura 3.1: Máquina de estados utilizada na Fase 1

A máquina de estados vai permitir que o funcionamento do projeto seja feito de forma síncrona e ordenada. Para a máquina de estados definiu-se um reset que quando ativo põe a máquina de estados no estado inicial e um *clock* que permite que seja criado um processo síncrono. Definiram-se também os *inputs* "coca", "agua" e "slar" que permitem selecionar a bebida pretendida, o *input timeExp* que recebe o sinal do temporizador de que o tempo terminou, os *outputs ola* e *epro* que emitem o sinal ao *Display* para apresentar a mensagem "OLA" e "EPRO", respetivamente, o *output timeVal* que emite o tempo pretendido, o

*output enable\_timer* que emite o sinal para ligar o temporizador e os *outputs ledr* e *ledg* que emitem o sinal para ligar o *led* vermelho e o *led* verde, respectivamente.

Na arquitetura da máquina de estados definiram-se duas constantes com a duração do piscar da palavra "OLA" e do *ledr* aceso, quatro e seis segundos, respectivamente. Optou-se por fazer um só processo síncrono.

No estado E0 os *outputs ola* e *enable\_timer* são ativos e é passado o tempo do *OLA\_TIME* para o *output timeVal* que passará como sinal para o temporizador e a máquina avança para o estado E0A que desativa o *enable\_timer* e quando a contagem do temporizador chega ao fim a máquina avança para o estado E1.

No estado E1 o *output ola* passa a zero e é ativado o *output epro*. Neste estado estamos na fase de escolha de uma bebida, após escolhida a bebida a máquina avança para o estado E2.

No estado E2 estamos na fase de preparação da bebida. Os *outputs ledr* e *enable\_timer* são ativos e é passado o tempo do *LEDR\_TIME* para o *output timeVal* que passará como sinal para o temporizador e a máquina avança para o estado E2A que desativa o *enable\_timer* e avança para o estado E2B que quando a contagem do temporizador chega ao fim avança para o estado E3. Após vários testes da Fase 1 na FPGA concluímos que o estado E2A era necessário, uma vez que este estado atrasa um ciclo de relógio o que é necessário para que o temporizador não tenha problemas na contagem do tempo.

No estado E3 estamos na fase em que a bebida é disponibilizada. O *output ledg* é ativo e se forem desativados todos os *inputs* das bebidas a máquina irá voltar ao estado E1 em que o utilizador poderá escolher outra bebida.

```
entity Fase1FSM is
  port (reset      : in std_logic;
        clk        : in std_logic;
        enable_timer : out std_logic;
        timeVal     : out std_logic_vector(7 downto 0);
        timeExp     : in std_logic;
        ola         : out std_logic;
        epro        : out std_logic;
        coca        : in std_logic;
        agua        : in std_logic;
        slar        : in std_logic;
        ledr        : out std_logic;
        ledg        : out std_logic);
end Fase1FSM;
```

Figura 3.2: Entidade da máquina de estados

```
architecture v1 of Fase1FSM is
  constant OLA_TIME : std_logic_vector(7 downto 0) := x"04"; -- 4 segundos
  constant LEDR_TIME : std_logic_vector(7 downto 0) := x"06"; -- 6 segundos

  type TState is (E0, E0A, E1, E2, E2A, E2B, E3);
  signal s_state : TState;
```

Figura 3.3: Constantes do tempo que o pisca e o led estarão ligados



```

proc_sinc : process(clk)
begin
  if (rising_edge(clk)) then
    if (reset = '1') then
      s_state <= E0;
    else
      case (s_state) is
        when E0 =>
          ola <= '1';
          epro <= '0';
          ledr <= '0';
          ledg <= '0';
          enable_timer <= '1';
          timeVal <= OLA_TIME;
          s_state <= E0A;

        when E0A =>
          enable_timer <= '0';
          if (timeExp = '1') then
            s_state <= E1;
          end if;

        when E1 =>
          ola <= '0';
          epro <= '1';
          ledr <= '0';
          ledg <= '0';
          enable_timer <= '0';
          if ((coca = '1') or (agua = '1') or (slar = '1')) then
            s_state <= E2;
          end if;

        when E2 =>
          ola <= '0';
          epro <= '0';
          ledr <= '1';
          ledg <= '0';
          enable_timer <= '1';
          timeVal <= LEDR_TIME;
          s_state <= E2A;

        when E2A =>
          s_state <= E2B;
          enable_timer <= '0';

        when E2B =>
          if (timeExp = '1') then
            s_state <= E3;
          end if;

        when E3 =>
          ola <= '0';
          epro <= '0';
          ledr <= '0';
          ledg <= '1';
          enable_timer <= '0';
          if ((coca = '0') and (agua = '0') and (slar = '0')) then
            s_state <= E1;
          end if;
        end case;
      end if;
    end if;
  end process;
end process;

```

Figura 3.4: Máquina de estados

### 3.1.2 *Display*

De seguida, implementou-se o código necessário para a visualização do texto pretendido nos *displays* de 7 bits. Na entidade do *Display* (figura 3.5) definiu-se um *clock* de 50MHz e outro de 10Hz, o *clock* de 10Hz é usado para criar o efeito da palavra "OLA" a piscar. Definiu-se também *enables* para cada palavra que ia ser escrita e os visores de 7 bits que iriam ser utilizados.

```
entity Display is
  port (clk          : in std_logic;
        clk10hz      : in std_logic;
        en_ola       : in std_logic;
        en_epro      : in std_logic;
        en_coca      : in std_logic;
        en_agua      : in std_logic;
        en_slar      : in std_logic;
        visor_uni     : out std_logic_vector(6 downto 0);
        visor_dez     : out std_logic_vector(6 downto 0);
        visor_cen     : out std_logic_vector(6 downto 0);
        visor_mil     : out std_logic_vector(6 downto 0));
end Display;
```

Figura 3.5: Entidade do *Display*

Na arquitetura do *Display* começou-se por definir constantes para as letras que irão ser usadas nos *displays* (figura 3.6), para ser de mais fácil compreensão quando usadas no código implementado. Depois, implementou-se um processo síncrono (figura 3.7) em que quando o *enable* da palavra é ativo as letras dessa palavra aparecem colocadas no *display* correspondente. Para criar o efeito de piscar da palavra "OLA" usa-se um *clock* de frequência 10Hz (*clk10Hz*). Cada vez que esse *clock* é ativo a palavra é apresentada nos *displays*, quando este está a zero não aparece nada.

```
architecture v1 of Display is
  -- letras da palavra "OLA"
  constant letraO : std_logic_vector(6 downto 0) := "1000000";
  constant letraL : std_logic_vector(6 downto 0) := "1000111";
  constant letraA : std_logic_vector(6 downto 0) := "0001000";

  -- letras da palavra "EPRO"
  constant letraE : std_logic_vector(6 downto 0) := "0000110";
  constant letraP : std_logic_vector(6 downto 0) := "0001100";
  constant letraR : std_logic_vector(6 downto 0) := "0001000";

  -- letras da palavra "COCA"
  constant letraC : std_logic_vector(6 downto 0) := "1000110";

  -- letras da palavra "AGUA"
  constant letraG : std_logic_vector(6 downto 0) := "0000010";
  constant letraU : std_logic_vector(6 downto 0) := "1000001";

  -- letras da palavra "SLAR"
  constant letraS : std_logic_vector(6 downto 0) := "0010010";
```

Figura 3.6: Constantes para representar as letras nos *displays* de 7 segmentos

```

begin
  process(clk)
  begin
    if(rising_edge(clk)) then
      if((en_ola = '1')) then
        if(clk10hz = '1') then
          visor_uni <= letraA;
          visor_dez <= letraL;
          visor_cen <= letraO;
          visor_mil <= (others => '1');
        else
          visor_uni <= (others => '1');
          visor_dez <= (others => '1');
          visor_cen <= (others => '1');
          visor_mil <= (others => '1');
        end if;
      elsif(en_epro = '1') then
        visor_uni <= letraO;
        visor_dez <= letraR;
        visor_cen <= letraP;
        visor_mil <= letraE;
      elsif(en_coca = '1') then
        visor_uni <= letraA;
        visor_dez <= letraC;
        visor_cen <= letraO;
        visor_mil <= letraC;
      elsif(en_agua = '1') then
        visor_uni <= letraA;
        visor_dez <= letraU;
        visor_cen <= letraG;
        visor_mil <= letraA;
      elsif(en_slar = '1') then
        visor_uni <= letraR;
        visor_dez <= letraA;
        visor_cen <= letraL;
        visor_mil <= letraS;
      else
        visor_uni <= "1111111";
        visor_dez <= "1111111";
        visor_cen <= "1111111";
        visor_mil <= "1111111";
      end if;
    end if;
  end process;
end v1;

```

Figura 3.7: Processo usado para meter as letras nos *displays* corretos

### 3.1.3 Divisor da frequência *Clock* e Temporizador

Para que fosse possível a criação de um *clock* de frequência de 10Hz criou-se um código que divide a frequência do *clock* (figura 3.8). É passado um valor natural à estrutura e esse número vai ser o divisor da frequência do *clock*, neste projeto o valor a ser passado tem de ser cinco milhões.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.NUMERIC_STD.all;

entity ClkDividerN is
    generic(divFactor : natural);
    port(clkIn : in std_logic;
         clkOut : out std_logic);
end ClkDividerN;

architecture RTL of ClkDividerN is
    signal s_divCounter : natural;
begin
    process(clkIn)
    begin
        if (rising_edge(clkIn)) then
            if (s_divCounter = divFactor - 1) then
                clkOut <= '0';
                s_divCounter <= 0;
            else
                if (s_divCounter = (divFactor / 2 - 1)) then
                    clkOut <= '1';
                end if;
                s_divCounter <= s_divCounter + 1;
            end if;
        end if;
    end process;
end RTL;
```

Figura 3.8: Divisor da frequência do *Clock*

Teve-se de implementar, também, um temporizador (figura 3.9) para que seja possível temporizar o tempo do piscar da palavra "OLA" e o tempo que o *led* vermelho deve estar aceso. É passado um valor inteiro à estrutura, esse valor é o tempo em segundos pretendido e é multiplicado pela frequência do *clock* de 50MHz para dar o número de ciclos de relógio que são necessários contar. A cada ciclo de relógio é subtraído um valor ao valor anteriormente calculado, quando esse valor chega a zero é enviado um sinal que significa que o tempo em segundos anteriormente atribuído já passou.

```

library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.NUMERIC_STD.all;

entity TimerFSM is
    port(reset      : in std_logic;
         clk        : in std_logic;
         enable     : in std_logic;
         timeVal    : in std_logic_vector(7 downto 0);
         timeExp    : out std_logic);
end TimerFSM;

architecture Behavioral of TimerFSM is
    signal s_counter : integer := 0;
    signal s_cntZero : std_logic := '0';
begin
    process(clk)
    begin
        if (rising_edge(clk)) then
            if (reset = '1') then
                s_counter <= 1;
                s_cntZero <= '0';
            elsif (enable = '1') then
                s_counter <= to_integer(unsigned(timeVal)) * 50e6;
                s_cntZero <= '0';
            else
                if (s_counter = 0) then
                    s_cntZero <= '1';
                else
                    s_counter <= s_counter - 1;
                    s_cntZero <= '0';
                end if;
            end if;
        end if;
    end process;

    timeExp <= s_cntZero;
end Behavioral;

```

Figura 3.9: Temporizador

### 3.1.4 Top-level

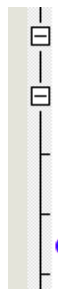
No *Top-level* é onde se interliga as diferentes estruturas que foram implementadas para que tudo se torne funcional e também é onde são definidos os *switches*, as *keys*, os *displays* de 7 segmentos e os *leds* que serão usados na FPGA.

Os *enables* de seleção das bebida *coca*, *agua* e *slar*, ligou-se aos *switches* 17, 16 e 15 da FPGA, respetivamente. Usou-se apenas estes três *switches* na fase 1 do projeto, porém, foi necessário definir todos os *switches* da FPGA, representado na figura 3.10, para a máquina funcionar corretamente.

Os visores *visor\_unidades*, *visor\_dezenas*, *visor\_centenais* e *visor\_milhares* ligou-se ao HEX0, HEX1, HEX2 e HEX3, respetivamente.

O *ledr* ligou-se ao *LEDR0* e o *ledg* ligou-se ao *LEDG7*, de forma a ficarem próximos um do outro.

Para o reset global decidiu-se usar uma *key*, neste caso a *KEY0*.



```
process(CLOCK_50)
begin
  if(rising_edge(CLOCK_50)) then
    S_SW <= SW;
  end if;
end process;

end shell;
```

Figura 3.10: Switches

```

architecture shell of Fase1 is
    signal s_clk10hz : std_logic;
    signal s_enable  : std_logic;
    signal s_timeExp : std_logic;
    signal s_timeVal  : std_logic_vector(7 downto 0);
    signal s_ola      : std_logic;
    signal s_epro     : std_logic;
    signal s_sw       : std_logic_vector(17 downto 0);
begin
    u0: entity work.ClkDividerN(RTL)
        generic map(divFactor => 5000000)
        port map(clkIn      => CLOCK_50,
                 clkOut     => s_clk10hz);
    u1: entity work.Display(v1)
        port map(clk          => CLOCK_50,
                 clk10hz     => s_clk10hz,
                 en_ola      => s_ola,
                 en_epro     => s_epro,
                 en_coca     => s_SW(17),
                 en_agua     => s_SW(16),
                 en_slar     => s_SW(15),
                 visor_uni   => HEX0,
                 visor_dez   => HEX1,
                 visor_cen   => HEX2,
                 visor_mil   => HEX3);
    u2: entity work.TimerFSM(Behavioral)
        port map(reset      => not KEY(0),
                 clk        => CLOCK_50,
                 enable     => s_enable,
                 timeVal    => s_timeVal,
                 timeExp    => s_timeExp);
    u3: entity work.Fase1FSM(v1)
        port map(reset      => not KEY(0),
                 clk        => CLOCK_50,
                 enable_timer => s_enable,
                 timeVal    => s_timeVal,
                 timeExp    => s_timeExp,
                 ola        => s_ola,
                 epro       => s_epro,
                 coca       => s_SW(17),
                 agua       => s_SW(16),
                 slar       => s_SW(15),
                 ledr       => LEDR(0),
                 ledg       => LEDG(7));
end architecture shell of Fase1;

```


Figura 3.11: *Top-level* da Fase 1

## 3.2 Fase2

### 3.2.1 Máquina de estados da Fase 2

Na fase 2 foi necessário a implementação de mais uma máquina de estados para que fosse possível criar o "modo escolha tamanho das garrafas" (figura 3.15). Na máquina de estados principal implementou-se mais um estado que só é ativo quando o *enable* do "modo escolha tamanho das garrafas" é ativo (figura 3.12) e é neste estado que se escolhe o tamanho da bebida.

A máquina só avança para o novo estado implementado, na figura 3.12 representado por *MEG*, quando o *enable en\_MEG* estiver ativo. Essa passagem de estado só acontece depois de escolher a bebida, a contagem do tempo do *ledr* do temporizador para e o *ledr* só se desliga quando a máquina sai do estado *MEG*. Quando o *enable* é desativo a máquina avança para o estado E3.



```
when E2B =>
  if((timeExp = '1') and (en_MEG = '0')) then
    s_state <= E3;
  elsif((timeExp = '1') and (en_MEG = '1')) then
    s_state <= MEG;
  end if;

when E3 =>
  ola <= '0';
  epro <= '0';
  ledr <= '0';
  ledg <= '1';
  enable_timer <= '0';
  timeVal <= (others => '-');
  if((B1 = '0') and (B2 = '0') and (B3 = '0') and (en_MEG = '0')) then
    s_state <= E1;
  end if;

when MEG =>
  ola <= '0';
  epro <= '0';
  ledr <= '1';
  ledg <= '0';
  enable_timer <= '0';
  timeVal <= (others => '-');
  if(en_MEG = '0') then
    s_state <= E3;
  end if;
end case;
```

Figura 3.12: Novo estado implementado



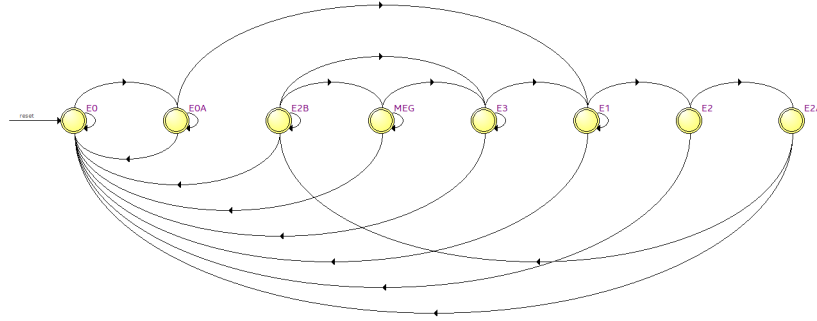


Figura 3.13: Máquina de estados utilizada na Fase 2

Para a máquina de estados implementada para fazer a seleção do tamanho da garrafa (figura 3.14) definiu-se três *inputs*: *sel* que é o sinal enviado pelo *DebounceUnit* implementado posteriormente, *reset* e o *clk* que permite fazer tudo num processo síncrono. E definiu-se quatro *outputs*: *b33cl\_d*, *b25cl\_d*, *b50cl\_d* e *b10dl\_d* que servem para ativar a quantidade de bebida representada nos *displays*.

Para que a máquina avance de estado é necessário receber um sinal do *DebounceUnit* (*sel*). Por defeito a máquina está configurada para estar no estado 33cl.

Cada vez que há um reset a máquina volta ao estado de 33cl.

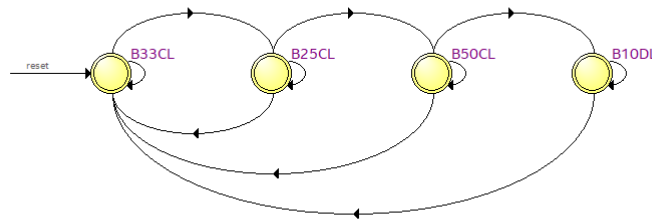


Figura 3.14: Máquina de estados da seleção da garrafa

```

process(clk)
begin
    if(rising_edge(clk)) then
        if(enable = '1') then
            if(reset = '1') then
                s_state <= B33CL;
            else
                case (s_state) is
                    when B33CL =>
                        b33cl_d <= '1';
                        b25cl_d <= '0';
                        b50cl_d <= '0';
                        b10dl_d <= '0';
                        if(sel = '1') then
                            s_state <= B25CL;
                        end if;

                    when B25CL =>
                        b33cl_d <= '0';
                        b25cl_d <= '1';
                        b50cl_d <= '0';
                        b10dl_d <= '0';
                        if(sel = '1') then
                            s_state <= B50CL;
                        end if;

                    when B50CL =>
                        b33cl_d <= '0';
                        b25cl_d <= '0';
                        b50cl_d <= '1';
                        b10dl_d <= '0';
                        if(sel = '1') then
                            s_state <= B10DL;
                        end if;

                    when B10DL =>
                        b33cl_d <= '0';
                        b25cl_d <= '0';
                        b50cl_d <= '0';
                        b10dl_d <= '1';
                        if(sel = '1') then
                            s_state <= B33CL;
                        end if;
                end case;
            end if;
        end if;
    end if;
end process;

```

Figura 3.15: Máquina de estados do modo escolha tamanho das garrafas

### 3.2.2 DebounceUnit

A única função do *DebounceUnit* é gerar um pulso de relógio cada vez que *lhe* é enviado algum *input*.

Quando *lhe* é gerado um *input* é gerado um pulso de relógio que é enviado para a máquina de estados do modo de seleção de bebida e a máquina avança um estado.

```
generic(kHzClkFreq : positive := 50000;
        mSecMinInWidth : positive := 100;
        inPolarity : std_logic := '1';
        outPolarity : std_logic := '1');
port(refClk : in std_logic;
     dirtyIn : in std_logic;
     pulsedOut : out std_logic);
end DebounceUnit;

architecture Behavioral of DebounceUnit is
    constant MIN_IN_WIDTH_CYCLES : positive := mSecMinInWidth * kHzClkFreq;
    subtype TCounter is natural range 0 to MIN_IN_WIDTH_CYCLES;
    signal s_debounceCnt : TCounter := 0;
    signal s_dirtyIn, s_previousIn, s_pulsedOut : std_logic;
begin
    in_sync_proc : process(refClk)
    begin
        if (rising_edge(refClk)) then
            if (inPolarity = '1') then
                s_dirtyIn <= dirtyIn;
            else
                s_dirtyIn <= not dirtyIn;
            end if;
            s_previousIn <= s_dirtyIn;
        end if;
    end process;

    count_proc : process(refClk)
    begin
        if (rising_edge(refClk)) then
            if ((s_dirtyIn = '0') or
                (s_debounceCnt > MIN_IN_WIDTH_CYCLES)) then
                s_debounceCnt <= 0;
                s_pulsedOut <= '0';
            elsif (s_dirtyIn = '1') then
                if (s_previousIn = '0') then
                    s_debounceCnt <= MIN_IN_WIDTH_CYCLES;
                    s_pulsedOut <= '0';
                else
                    if (s_debounceCnt >= 1) then
                        s_debounceCnt <= s_debounceCnt - 1;
                    end if;
                    if (s_debounceCnt = 1) then
                        s_pulsedOut <= '1';
                    else
                        s_pulsedOut <= '0';
                    end if;
                end if;
            end if;
        end if;
    end process;

    pulsedOut <= s_pulsedOut when (outPolarity = '1') else
        not s_pulsedOut;
```

Figura 3.16: DebounceUnit

### 3.2.3 *Display* do tamanho das garrafas

A implementação desta estrutura *Display* é idêntica à estrutura implementada na fase 1. Começou-se também por definir constantes para as letras e números a ser usados de forma a que o código escrito posteriormente seja de mais fácil compreensão. Criaram-se *enables* para cada tamanho de garrafa que quando ativo apresenta nos *displays* o tamanho de garrafa correspondente. Quando nenhum está ativo nos *displays* não aparece nada.

```
architecture v1 of Display_Tam_Garrafa is
-- display "33c1"
constant N3      : std_logic_vector(6 downto 0) := "0110000";
constant LetraC  : std_logic_vector(6 downto 0) := "1000110";
constant LetraL  : std_logic_vector(6 downto 0) := "1000111";

-- display "25c1"
constant N2      : std_logic_vector(6 downto 0) := "0100100";
constant N5      : std_logic_vector(6 downto 0) := "0010010";

-- display "50c1"
constant N0      : std_logic_vector(6 downto 0) := "1000000";

-- display "10d1"
constant N1      : std_logic_vector(6 downto 0) := "1111001";
constant LetraD  : std_logic_vector(6 downto 0) := "0100001";

begin
    process(clk)
    begin
        if(rising_edge(clk)) then
            if(en_33c1 = '1') then
                visor_uni <= LetraL;
                visor_dez <= LetraC;
                visor_cen <= N3;
                visor_mil <= N3;
            elsif(en_25c1 = '1') then
                visor_uni <= LetraL;
                visor_dez <= LetraC;
                visor_cen <= N5;
                visor_mil <= N2;
            elsif(en_50c1 = '1') then
                visor_uni <= LetraL;
                visor_dez <= LetraC;
                visor_cen <= N0;
                visor_mil <= N5;
            elsif(en_10d1 = '1') then
                visor_uni <= LetraL;
                visor_dez <= LetraD;
                visor_cen <= N0;
                visor_mil <= N1;
            else
                visor_uni <= (others => '1');
                visor_dez <= (others => '1');
                visor_cen <= (others => '1');
                visor_mil <= (others => '1');
            end if;
        end if;
    end process;
end v1;
```

Figura 3.17: Display do tamanho das garrafas

### 3.2.4 Top-level

No *Top-level* da fase 2 definiu-se sinais do tamanho da garrafa e do pulso de relógio gerado pelo *DebounceUnit* (figura 3.18). Os sinais do tamanho da garrafa servem para ativar os *enables* que ativam a representação do tamanho da bebida que está ser selecionado nos *displays*.

Para gerar o pulso de relógio que permite mudar de estado na máquina de estados da seleção do tamanho da garrafa definiu-se a *KEY1*. Quando se carrega nessa *key* o *DebounceUnit* envia um sinal para a máquina de estados da seleção do tamanho da garrafa e a máquina avança de estado.

O *enable* definido para o "modo Escolha tamanho das garrafas" foi o *switch SW0*.

```
signal s_33c1      : std_logic;
signal s_50c1      : std_logic;
signal s_25c1      : std_logic;
signal s_10d1      : std_logic;
signal s_sel       : std_logic;
```

Figura 3.18: Sinais definidos

```
u4: entity work.Display_Tam_Garrafa(v1)
    port map(clk      => CLOCK_50,
             en_33c1  => s_33c1,
             en_25c1  => s_25c1,
             en_50c1  => s_50c1,
             en_10d1  => s_10d1,
             visor_uni => HEX4,
             visor_dez => HEX5,
             visor_cen => HEX6,
             visor_mil => HEX7);

u5: entity work.Sel_Tam_Garrafa_FSM(v1)
    port map(reset    => not KEY(0),
             clk      => CLOCK_50,
             enable   => s_SW(0),
             sel      => s_sel,
             b33c1_d  => s_33c1,
             b25c1_d  => s_25c1,
             b50c1_d  => s_50c1,
             b10d1_d  => s_10d1);

u6: entity work.DebounceUnit(Behavioral)
    port map(refClk   => CLOCK_50,
             dirtyIn  => KEY(1),
             pulsedOut => s_sel);
```

Figura 3.19: Top-level Fase 2

## Capítulo 4

# Validação

Neste capítulo deveria estar a simulação dos principais módulos do projeto, porém houve problemas ao tentar simular o projeto pelo que não foi possível através de simulações confirmar os resultados esperados. Todavia, efetuaram-se inúmeros testes na FPGA para confirmar que o que tinha sido implementado estava de acordo com o suposto.

## Capítulo 5

# Manual do utilizador

Após ligar e programar a FPGA pressionar a *key* 0 para inicializar a máquina da maneira adequada. Após inicializada a máquina, aparecerá a mensagem "OLA" a piscar durante quatro segundos.

Passados os quatro segundos, nos *displays*, aparecerá a mensagem "EPRO" que significa "Escolha um Produto". Para escolher o produto desejado terá de ativar um dos seguintes *switches*:

- *Switch* 17: Coca-cola;
- *Switch* 16: Água;
- *Switch* 15: Sumo de Laranja.

Após a escolha liga-se um *led* vermelho durante seis segundos que significa que a bebida está a ser disponibilizada. Terminando os seis segundos o *led* vermelho desliga-se e liga-se um *led* verde que significa que a bebida já está disponibilizada.

Se o utilizador pretender escolher outra bebida, basta desativar o *switch* que selecionou e aparecerá no *display* a mensagem "EPRO" podendo, assim, selecionar outra bebida.

O utilizador pode ainda entrar no modo "Modo Escolha tamanho das garrafas", onde poderá escolher o tamanho da garrafa, tendo como opções 25cl, 33cl, 50cl ou 10dl. Por defeito a máquina está configurada para disponibilizar garrafas de 33cl. Para entrar neste modo basta, ou antes de disponibilizar a bebida ou enquanto a bebida estiver a ser disponibilizada, ativar o *switch* 0. Para escolher de entre os vários tamanhos de garrafa pressionar a *key* 1. Após selecionar o tamanho da garrafa desative o *switch* 0 e a bebida será disponibilizada na garrafa desejada ligando-se o *led* verde quando estiver pronta.

Para que a máquina volte ao estado inicial é só pressionar a *key* 0.

## Capítulo 6

# Conclusões

Em suma, pode-se afirmar que o projeto cumpre com as especificações e requisitos definidos, à exceção das simulações. Obteve-se uma máquina totalmente funcional, em que é possível escolher a bebida pretendida, bem como o tamanho da garrafa desejado.

Este projeto permitiu-nos perceber melhor como funcionam os sistemas digitais, principalmente a linguagem VHDL que foi usada para modelar o comportamento e a estrutura de tais sistemas. É de salientar que as temáticas abordadas neste projeto são bastante relevantes não só para o nosso percurso académico mas também para uma futura vida profissional.

Sistemas digitais estão presentes em todas as máquinas que nos rodeiam, pelo que o ser humano, mesmo sem se aperceber, se tornou dependente de tais sistemas. Dito isto, achamos que é de extrema importância perceber como os sistemas digitais funcionam e este trabalho ajudou-nos nesse sentido.



# Contribuições dos autores

Este trabalho foi realizado por Guilherme Craveiro (GC) e Rafael Pinto (RP), alunos do primeiro ano do Mestrado Integrado em Engenharia de Computadores e Telemática (MIECT). Cada um dos autores participou ativamente e de forma conjunta neste trabalho de aprofundamento, permitindo afirmar que tanto o autor GC como o autor RP contribuíram 50% cada para o aspeto final deste projeto.

# Acrónimos

**MIECT** Mestrado Integrado em Engenharia de Computadores e Telemática

**VHDL** VHSIC Hardware Description Language

**FPGA** Field Programmable Gate Array