UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
DEPARTAMENTO DE ENGENHARIA DE COMPUTAÇÃO E AUTOMAÇÃO
DCA0212.1 – CIRCUITOS DIGITAIS – LABORATÓRIO

PROJETO MÁQUINA DE SORVETE

DISCENTES:
EFRAIN MARCELO PULGAR PANTALEON
FERNANDO LUCAS SOUSA SILVA
MATHEUS GOMES DINIZ ANDRADE
TEÓPHILO VITOR DE CARVALHO CLEMENTE

DOCENTE:
TIAGO BARROS

NATAL/RN
2022

1. Introdução

A máquina produtora de sorvete consiste em uma importante ferramenta utilizada para produzir comercialmente sorvetes de massa, picolés, gelatos e entre outros tipos. Em seu interior, os materiais serão ao mesmo tempo batidos e congelados, o que lhes dará a consistência adequada para serem vendidos. Como nosso projeto final da disciplina de Circuitos Digitais DCA0212.1, iremos apresentar o controlador utilizado no projeto de uma máquina de sorvete expresso, aqui mostraremos a descrição dele em VHDL e sua respectiva simulação.

2. Controlador

Aqui descreveremos as etapas para obter o controlador do projeto e como foi seu desenvolvimento em VHDL.

2.1. Projeto da Máquina de Estados Finitos (FSM)

Na FSM temos a substituição das operações que envolvem dados por sinais de controle do tipo booleano. Assim, após a obtenção da FSM é possível criar a lógica combinacional que de acordo com equações da obtidas na tabela verdade que veremos posteriormente, e com isso o bloco de controle que irá gerir o funcionamento da máquina de sorvete.

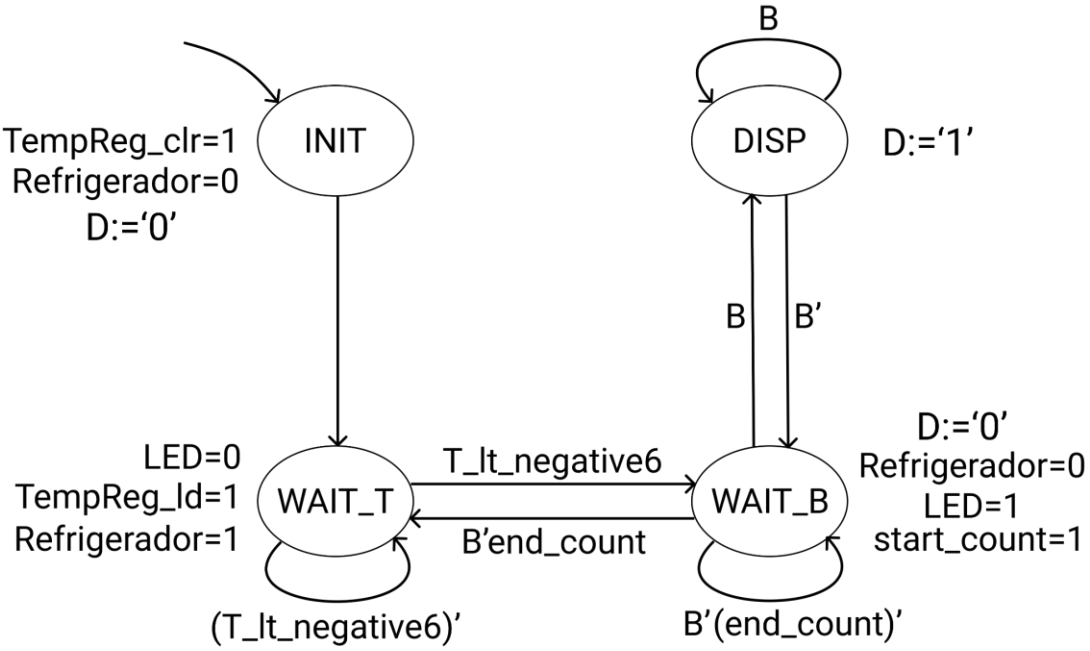


Figura 1 - Máquina de estados (FSM).

2.2. Tabela Verdade

Aqui foi construída uma tabela verdade que descreve o comportamento do sistema, obtivemos uma tabela de 32 (2^5) linhas, porém foi notado os casos que não dependiam de todas e as variáveis, assim foi realizada a simplificação e obtida uma tabela verdade de melhor compreensão. Com isso, analisando os sinais de entrada e saída, percebe-se que para alguns casos os sinais: T_lt_negative6, end_count e B, não fazem diferença para o valor resultante das saídas: n0 e n1, já para outros casos um único deles ou um par é a responsável pela mudança de n0 e n1. Desse modo, a tabela resultante é mostrada na figura 2.

	Inputs					Outputs							
	T_lt_negative6	end_count	B	s0	s1	n0	n1	TempReg_clr	TempReg_ld	start_count	LED	Refrigerador	D
INIT	x	x	x	0	0	0	1	1	0	0	0	0	0
WAIT_T	0	x	x	0	1	0	1	0	1	0	0	1	0
	1	x	x	0	1	1	0	0	1	0	0	1	0
WAIT_B	x	0	0	1	0	1	0	0	0	1	1	0	0
	x	x	1	1	0	1	1	0	0	1	1	0	0
	x	1	0	1	0	0	1	0	0	1	1	0	0
DISP	x	x	0	1	1	1	0	0	0	0	1	0	1
	x	x	1	1	1	1	1	0	0	0	1	0	1

Figura 2 – Tabela verdade dos estados simplificada.

2.3. Lógica Combinacional

Mediante a criação da tabela verdade podemos extrair as equações lógicas de cada estado. Para isso, são utilizados os minitermos que permitem retirar informações da tabela verdade e representar em forma de equações.

n0 = (t_et_n6 · s0' · s1) + (end_count' · B' · s0 · s1') + (B · s0 · s1') + (B' · s0 · s1) + (B · s0 · s1)

n1 = (s0' · s1') + (t_lt_n6' · s0' · s1') + (B · s0 · s1') + (end_count · B' · s0 · s1') + (B · s0 · s1)

TempReg_clr = s0' · s1

TempReg_ld = (t_lt_n6' · s0' · s1) + (t_lt_n6 · s0' · s1)

start_count = (end_count' · B' · s0 · s1') + (B · s0 · s1') + (end_count · B' · s0 · s1')

LED = (end_count' · B' · s0 · s1') + (B · s0 · s1') + (end_count · B' · s0 · s1') + (B' · s0 · s1) + (B · s0 · s1)

Refrigerador = (t_lt_n6' · s0' · s1) + (t_lt_n6 · s0' · s1)

D = s0 · s1

Após isso, foi constatado que as equações eram passíveis de simplificações e assim foi feito e com isso obtemos o seguinte resultado:

n0 = s1 · s0' · t_et_n6 + s0 · (end_count' + B + s1)

n1 = s0' · (s1' + t_lt_n6') + s0 · (B + s1'end_count)

TempReg_clr = s0' · s1

TempReg_ld = s0' · s1'

start_count = s0 · s1'

LED = s0

Refrigerador = s0' · s1

D = s0 · s1

Com isso, projetamos os circuitos lógicos seguindo cada equação obtida, como mostrado nas imagens a seguir:

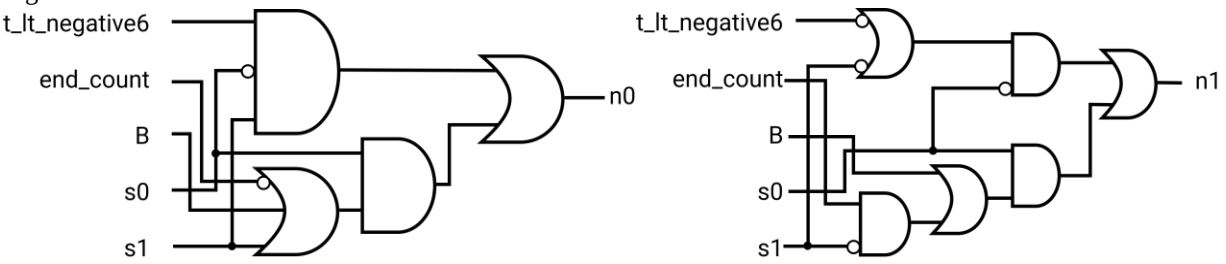


Figura 3 - Circuitos lógicos de n0 e n1.

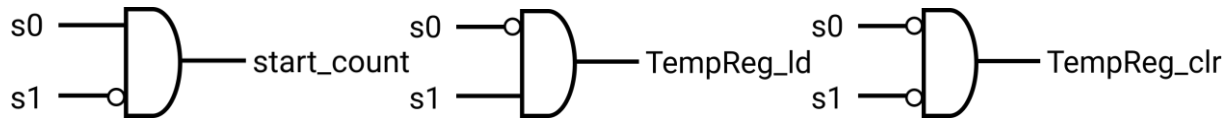


Figura 4 - Circuitos lógicos de start_count, TempReg_Id e TempReg_clr.

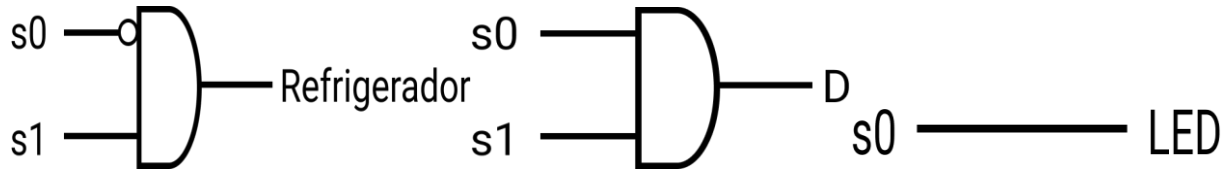


Figura 5 - Circuito lógico refrigerador, D e LED.

2.4. Descrição do controlador em VHDL

Aqui apresentaremos os códigos em VHDL para a construção do controlador, para tal foram criados os devidos componentes e após isso unidos para a formação do controlador, assim apresentaremos os respectivos códigos e o resultado da simulação final.

```
1   entity Refrigerador is
2       port(
3           s0_Refrigerador: in bit;
4           s1_Refrigerador: in bit;
5           Refrigerador: out bit
6       );
7   end;
8
9   architecture behav of Refrigerador is
10  begin
11      Refrigerador <= not(s0_Refrigerador) and s1_Refrigerador;
12  end;
```

Figura 6 - Código VHDL para o refrigerador.

```
1   entity LED is
2       port(
3           s0_LED: in bit;
4           LED: out bit
5       );
6   end;
7
8   architecture behav of LED is
9   begin
10      LED <= s0_LED;
11  end;
```

```
1   entity D is
2       port(
3           s0_D, s1_D: in bit;
4           saida_D: out bit
5       );
6   end;
7
8   architecture behav of D is
9   begin
10      saida_D <= s0_D and s1_D;
11  end;
```

Figura 7 - Código VHDL para o LED e para o D respectivamente.

```vhdl
entity TempReg_clr is
   port(
      s0_TempReg_clr: in bit;
      s1_TempReg_clr: in bit;
      TempReg_clr: out bit
   );
end;

architecture behav of TempReg_clr is
begin
   tempReg_clr <= not(s0_TempReg_clr) and not(s1_TempReg_clr);
end;
```

Figura 8 - Código VHDL para o clear do registrador de temperaturas.

```vhdl
entity TempReg_ld is
   port(
      s0_TempReg_ld: in bit;
      s1_TempReg_ld: in bit;
      TempReg_ld: out bit
   );
end;

architecture behav of TempReg_ld is
begin
   tempReg_ld <= not(s0_TempReg_ld) and s1_TempReg_ld;
end;
```

Figura 9 - Código VHDL para o load do registrador de temperaturas.

```vhdl
entity start_count is
   port(
      s0_start_count: in bit;
      s1_start_count: in bit;
      start_count: out bit
   );
end;

architecture behav of start_count is
begin
   start_count <= s0_start_count and not(s1_start_count);
end;
```

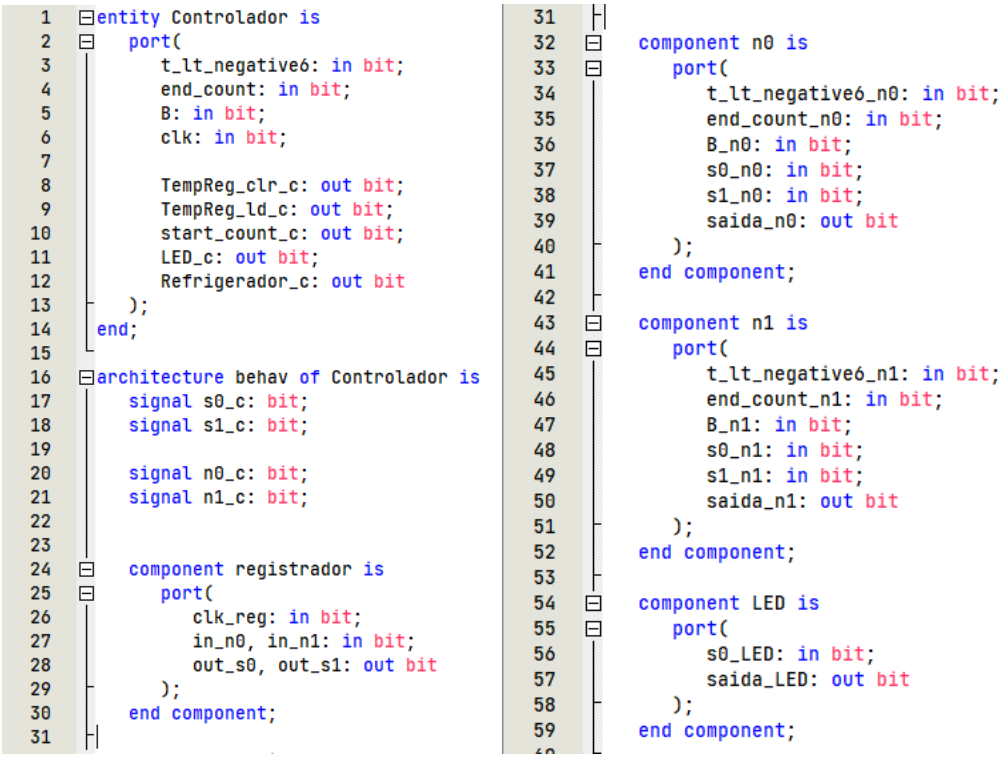
Figura 10 - Código VHDL para a contagem do temporizador.

```vhdl
entity flipflop is
   port(
      d_ff: in bit;
      clk_ff: in bit;
      q_ff: out bit
   );
end;

architecture behav of flipflop is
begin
   process(d_ff, clk_ff)
      begin
         if(clk_ff ' event and clk_ff = '1' and d_ff = '0') then
            q_ff <= '0';
         elsif(clk_ff ' event and clk_ff = '1' and d_ff = '1') then
            q_ff <= '1';
         end if;
   end process;
end;
```

Figura 11 - Código VHDL para a construção dos Flip-flops.

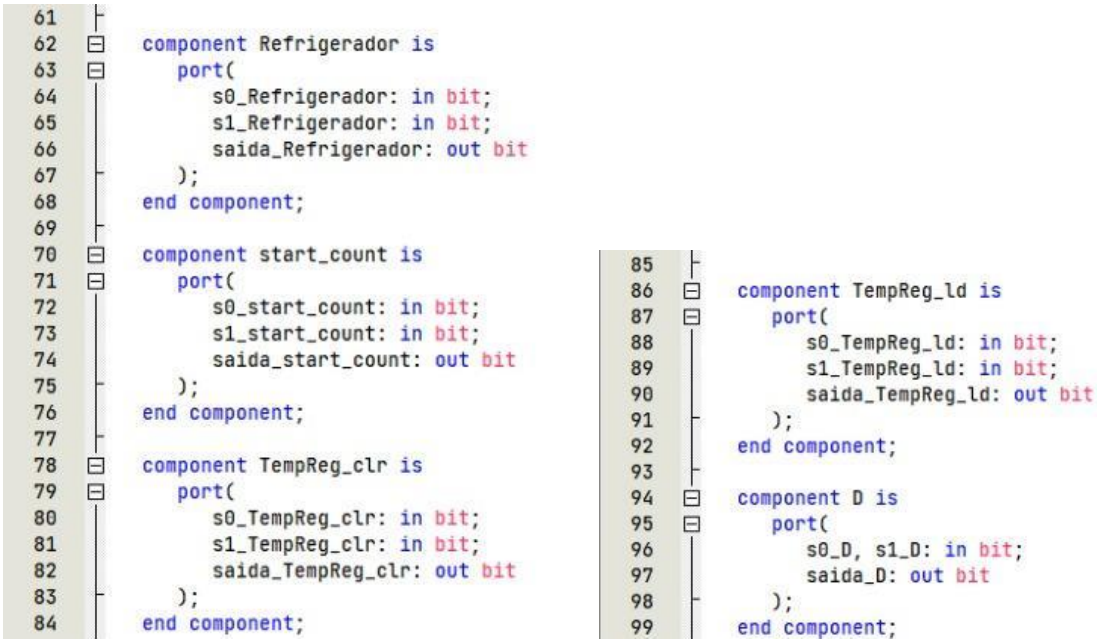
```vhdl
entity registrador is
   port(
      clk_reg: in bit;
      in_n0, in_n1: in bit;
      out_s0, out_s1: out bit
   );
end;

architecture behav of registrador is
   component flipflop is
      port(
         d_ff: in bit;
         clk_ff: in bit;
         q_ff: out bit
      );
   end component;

begin
   u1: flipflop port map(d_ff => in_n0, clk_ff => clk_reg, q_ff => out_s0);
   u2: flipflop port map(d_ff => in_n1, clk_ff => clk_reg, q_ff => out_s1);
end;
```

Figura 12 - Código VHDL para a construção do Registrador de 2 bits a partir dos Flip-flops.



```vhdl
entity Controlador is
    port(
        t_lt_negativeó: in bit;
        end_count: in bit;
        B: in bit;
        clk: in bit;

        TempReg_clr_c: out bit;
        TempReg_ld_c: out bit;
        start_count_c: out bit;
        LED_c: out bit;
        Refrigerador_c: out bit
    );
end;

architecture behav of Controlador is
    signal s0_c: bit;
    signal s1_c: bit;

    signal n0_c: bit;
    signal n1_c: bit;


    component registrador is
        port(
            clk_reg: in bit;
            in_n0, in_n1: in bit;
            out_s0, out_s1: out bit
        );
    end component;

component n0 is
    port(
        t_lt_negativeó_n0: in bit;
        end_count_n0: in bit;
        B_n0: in bit;
        s0_n0: in bit;
        s1_n0: in bit;
        saida_n0: out bit
    );
end component;

component n1 is
    port(
        t_lt_negativeó_n1: in bit;
        end_count_n1: in bit;
        B_n1: in bit;
        s0_n1: in bit;
        s1_n1: in bit;
        saida_n1: out bit
    );
end component;

component LED is
    port(
        s0_LED: in bit;
        saida_LED: out bit
    );
end component;
```

Figura 13 - Junção dos componentes e código final em VHDL.



```vhdl
component Refrigerador is
    port(
        s0_Refrigerador: in bit;
        s1_Refrigerador: in bit;
        saida_Refrigerador: out bit
    );
end component;

component start_count is
    port(
        s0_start_count: in bit;
        s1_start_count: in bit;
        saida_start_count: out bit
    );
end component;

component TempReg_clr is
    port(
        s0_TempReg_clr: in bit;
        s1_TempReg_clr: in bit;
        saida_TempReg_clr: out bit
    );
end component;

component TempReg_ld is
    port(
        s0_TempReg_ld: in bit;
        s1_TempReg_ld: in bit;
        saida_TempReg_ld: out bit
    );
end component;

component D is
    port(
        s0_D, s1_D: in bit;
        saida_D: out bit
    );
end component;
```

Figura 14 - Junção dos componentes e código final em VHDL.

```
93  begin
94
95      u1: n0 port map(
96            t_lt_negativo6_n0 => t_lt_negativo6,
97            end_count_n0 => end_count,
98            B_n0 => B,
99            s0_n0 => s0_c,
100           s1_n0 => s1_c,
101           saida_n0 => n0_c
102       );
103     u2: n1 port map(
104           t_lt_negativo6_n1 => t_lt_negativo6,
105           end_count_n1 => end_count,
106           B_n1 => B,
107           s0_n1 => s0_c,
108           s1_n1 => s1_c,
109           saida_n1 => n1_c
110       );
111
112     u3: registrador port map(
113           clk_reg => clk,
114           in_n0 => n0_c,
115           in_n1 => n1_c,
116           out_s0 => s0_c,
117           out_s1 => s1_c
118       );
```

```
120     u4: LED port map(
121           s0_LED => s0_c,
122           saida_LED => LED_c
123       );
124     u5: Refrigerador port map(
125           s0_Refrigerador => s0_c,
126           s1_Refrigerador => s1_c,
127           saida_Refrigerador => Refrigerador_c
128       );
129     u6: start_count port map(
130           s0_start_count => s0_c,
131           s1_start_count => s1_c,
132           saida_start_count => start_count_c
133       );
134     u7: TempReg_clr port map(
135           s0_TempReg_clr => s0_c,
136           s1_TempReg_clr => s1_c,
137           saida_TempReg_clr => TempReg_clr_c
138       );
139     u8: TempReg_ld port map(
140           s0_TempReg_ld => s0_c,
141           s1_TempReg_ld => s1_c,
142           saida_TempReg_ld => TempReg_ld_c
143       );
144
145 end;
```

Figura 15 - Junção dos componentes e código final em VHDL.

```
152     u9: D port map(
153           s0_D => s0_c,
154           s1_D => s1_c,
155           saida_D => D_c
156       );
157 end;
```

Figura 16 - Junção dos componentes e código final em VHDL.

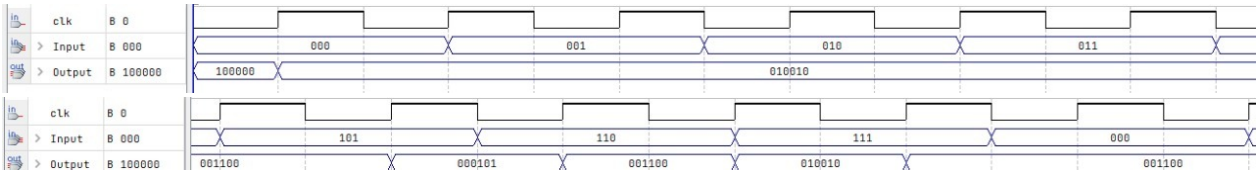


Figura 17 - Resultado da simulação para o controlador em VHDL.

4. Conclusões

A realização do projeto da máquina de sorvete permitiu a aplicação da teoria vista em sala de aula em um modelo real e utilizado no cenário comercial em larga escala, para tal foi de grande importância a compreensão dos elementos que compõem a estrutura da máquina e como seria implementada a FSM no seu contexto. Além disso, também utilizamos da simplificação na tabela verdade e manipulação de equações lógicas através de minitermos para reduzi-las a um tamanho menor. Outrossim, vemos que a implementação do bloco de controle se mostrou satisfatória sendo possível a elaboração do código VHDL e sua simulação, atestando seu funcionamento.

5. Referências

Vahid, Frank. Digital Design with RTL Design, VHDL, and Verilog Solution Manual. 2° Edição.2010.

Quartus II. Software de simulação.