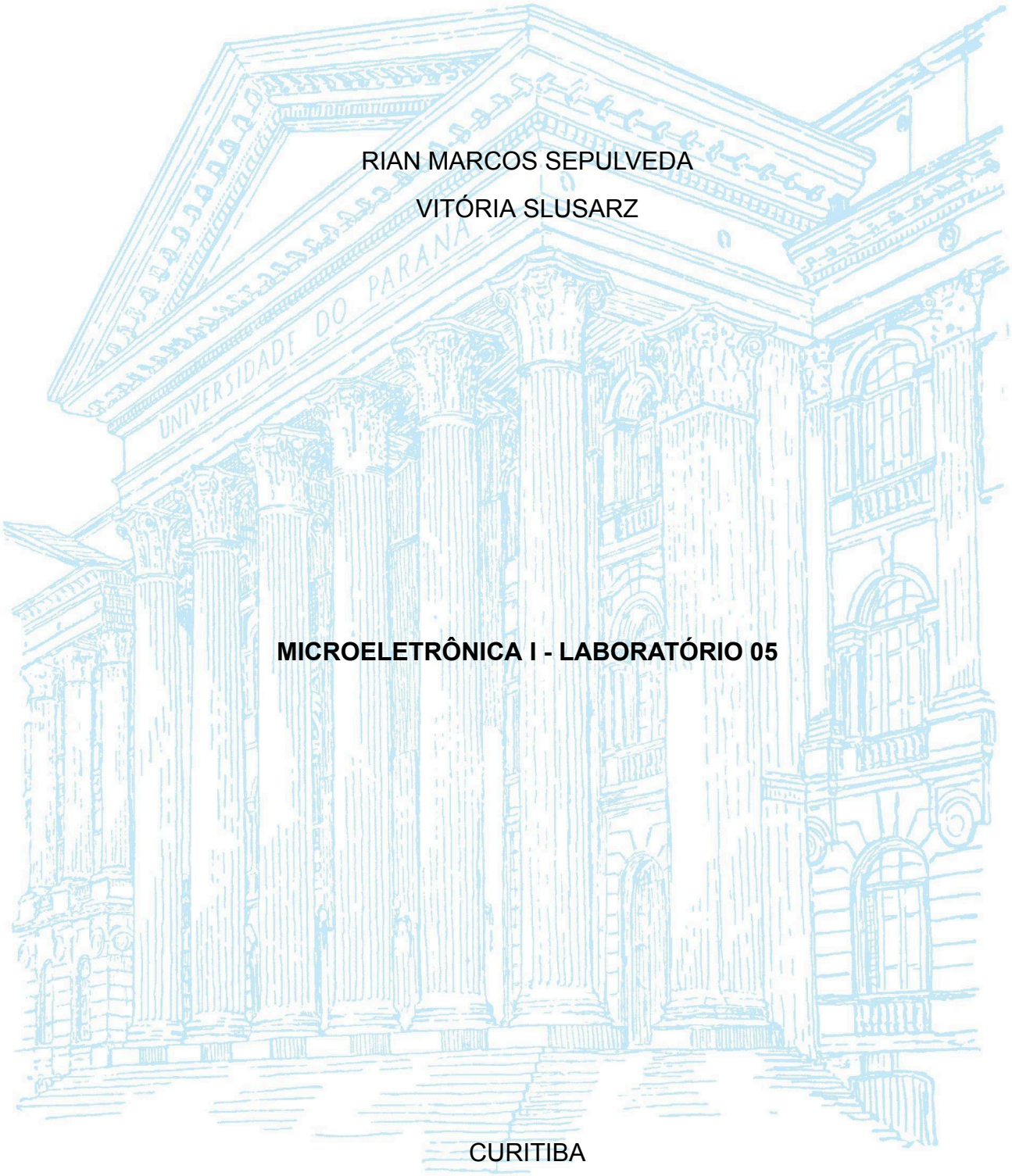
UNIVERSIDADE FEDERAL DO PARANÁ
DEPARTAMENTO DE ENGENHARIA ELÉTRICA
SETOR DE TECNOLOGIA



RIAN MARCOS SEPULVEDA
VITÓRIA SLUSARZ

MICROELETRÔNICA I - LABORATÓRIO 05

CURITIBA
2024

RIAN MARCOS SEPULVEDA
VITÓRIA SLUSARZ

MICROELETRÔNICA I - LABORATÓRIO 05

Trabalho apresentado no curso de Engenharia Elétrica para a disciplina de Microeletrônica I solicitado como requisito de avaliação parcial da disciplina.
Orientador: Prof. Dr. Sibilla Batista da Luz França.

CURITIBA

2024

1 INTRODUÇÃO

Neste relatório, apresentamos os resultados de um laboratório prático dedicado à implementação de sistemas digitais utilizando o FPGA XC3S500E e a linguagem de descrição de hardware VHDL. Inicialmente, abordamos os conceitos teóricos fundamentais, incluindo a natureza dos FPGAs, os princípios da linguagem VHDL e conceitos específicos como o funcionamento do display BCD de 7 segmentos, o peso de Hamming e a lógica de ordenação binária.

Em seguida, detalhamos a metodologia adotada em cada experimento, descrevendo a elaboração dos códigos em VHDL, a simulação dos circuitos e os testes realizados para verificar o funcionamento correto dos dispositivos.

Os resultados obtidos e as conclusões derivadas desses experimentos destacam não apenas a aplicação prática dos conceitos teóricos estudados, mas também a eficácia das ferramentas utilizadas, demonstrando a versatilidade e a importância do FPGA e da VHDL no desenvolvimento de sistemas digitais complexos.

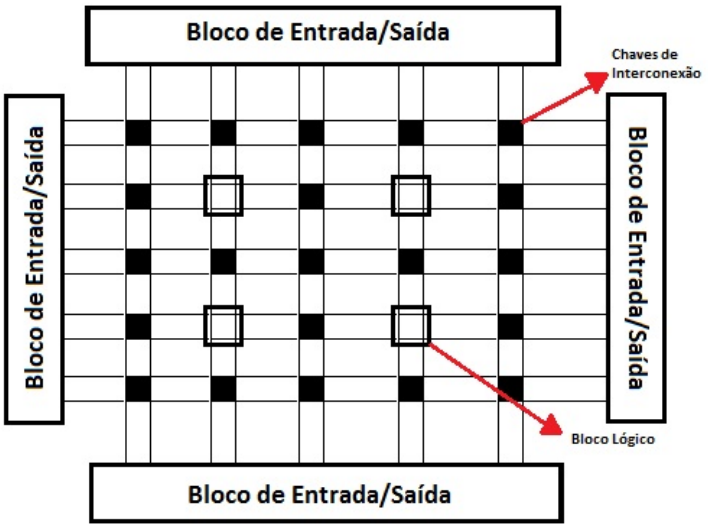
Este relatório tem como objetivo oferecer uma visão ampla das atividades desenvolvidas no laboratório, contribuindo para a compreensão dos conceitos e técnicas empregados na implementação de sistemas digitais.

2 FUNDAMENTAÇÃO TEÓRICA

O presente capítulo visa oferecer uma base conceitual para compreender os princípios relacionados aos temas abordados no laboratório em questão. Dessa forma, serão explorados fundamentos teóricos dos FPGA (Field-Programmable Gate Arrays) e da linguagem de descrição de hardware VHDL (VHSIC Hardware Description Language), bem como também conceitos adicionais utilizados em cada um dos projetos realizados.

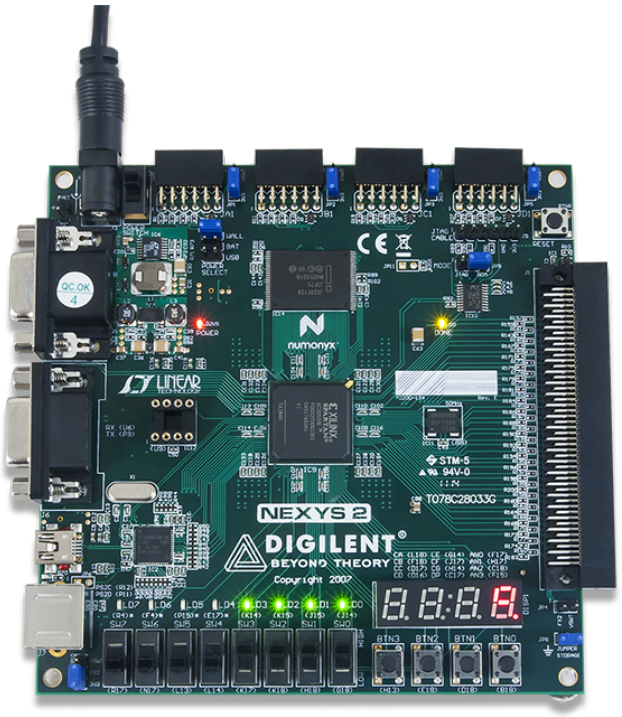
2.1 FPGA - XC3S500E:

O XC3S500E é um FPGA (Field-Programmable Gate Arrays) que pertence à família Spartan-3E da Xilinx e é conhecido por seu custo-benefício. Os arranjos de porta programável em campo (FPGAs) são dispositivos semicondutores que consistem em uma matriz de blocos lógicos configuráveis, blocos de entrada/saída, e interconexões programáveis.



O equipamento utilizado na disciplina conta com quinhentas mil portas lógicas equivalentes (LEs), conferindo a ele uma capacidade considerável para a implementação de circuitos digitais mais complexos. Essa capacidade pode ser traduzida em até 192 blocos lógicos configuráveis (CLBs) e até 4 blocos DSP (Digital Signal Processing), fazendo com que o XC3S500E seja adequado para aplicações

que requerem um processamento intensivo de sinais digitais. Além disso, o XC3S500E possui uma variedade de recursos adicionais que fazem com que ele seja versátil em diferentes contextos de aplicação, como por exemplo sua memória embutida, que proporciona armazenamento local para dados e configurações.



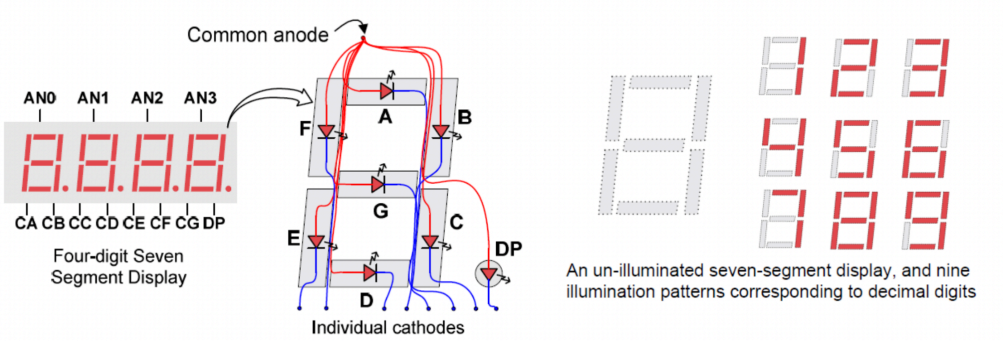
Sendo assim, para que o dispositivo fosse programado e configurado, foi utilizado o software Xilinx ISE (Integrated Software Environment), que permite a descrição do circuito em uma linguagem de descrição de hardware, além da síntese e implementação do design na placa.

2.2 VHDL:

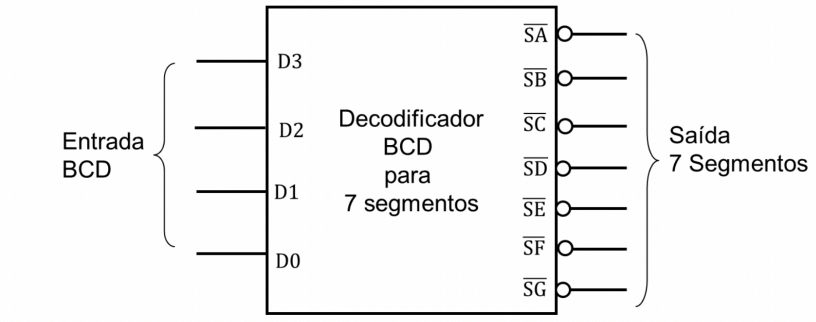
A linguagem VHDL (VHSIC Hardware Description Language) é uma linguagem de descrição de hardware, sendo amplamente utilizada para realizar o design de circuitos integrados e FPGAs. Com ela é possível modelar desde pequenos componentes lógicos até sistemas mais completos, especificando suas funcionalidades, interconexões e comportamentos. Uma das principais vantagens da linguagem é sua capacidade de simulação, o que permite a verificação do funcionamento do sistema antes mesmo de sua implementação na placa

2.3 DISPLAY BCD 7 SEGMENTOS:

O display BCD de 7 segmentos é um dispositivo que exibe números decimais e é muito utilizado para diversas aplicações. Cada dígito é representado por sete segmentos separados que podem ser ligados ou desligados, formando diferentes padrões que representam números de 0 a 9. Cada um desses sete segmentos representa uma parte específica do dígito e recebe uma letra de referência, podendo ser "a", "b", "c", "d", "e", "f" e "g" conforme mostra a imagem abaixo:



Sendo assim, para que o display exiba um valor numérico, o valor decimal precisa ser convertido para sua forma binária codificada em decimal (BCD), onde cada dígito decimal será representado por quatro bits binários. Em seguida, esses valores BCD são aplicados aos segmentos correspondentes do dispositivo, acendendo ou apagando os segmentos conforme necessário para exibir o número desejado.

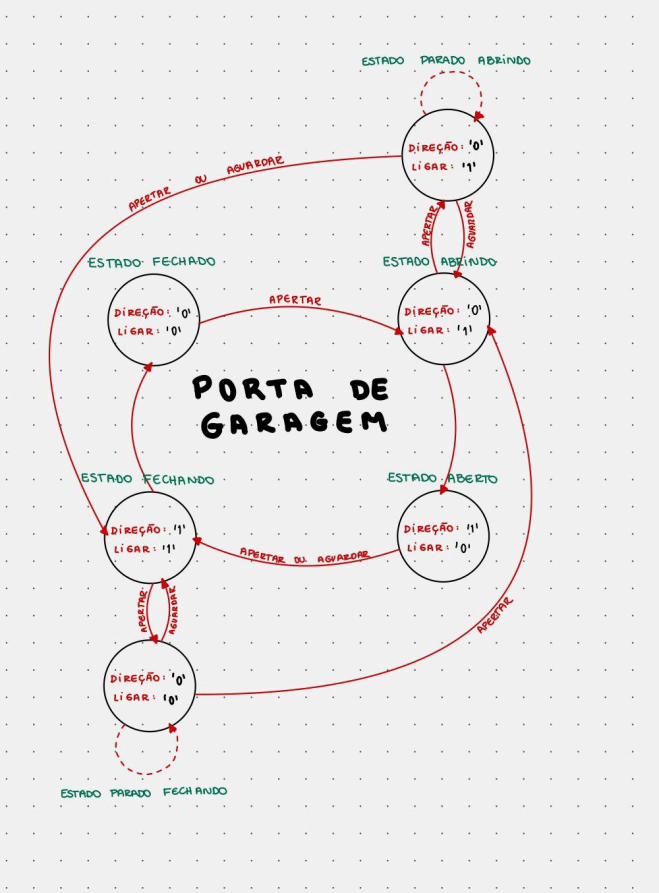


3 METODOLOGIA:

O presente capítulo visa explicar como foram realizados os quatro projetos do segundo laboratório, sendo que em todos eles foram desenvolvidos códigos em VHDL para implementação e teste.

3.1 Controlador de porta de Garagem:

Antes de começar a desenvolver o código VHDL para o circuito, é essencial criar o diagrama de transição de estados. Esse diagrama representa os diferentes estados pelos quais o sistema pode passar durante sua execução. Abaixo segue o diagrama de transição de estados:



3.1.1 Códigos em VHDL:

3.1.1.1 código principal “porta_garagem”:

Para o projeto, o código em VHDL abaixo foi desenvolvido:

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity porta_garagem is
    Port (
        remoto : in  STD_LOGIC;          -- Entrada para o controle remoto
        aberta : in  STD_LOGIC;          -- Sensor indicando se a garagem está aberta
        fechada : in  STD_LOGIC;         -- Sensor indicando se a garagem está fechada
        rst : in  STD_LOGIC;             -- Sinal de reset
        clk : in  STD_LOGIC;             -- Sinal de clock
        ligar : out  STD_LOGIC;          -- Saída para ligar o motor
        direcao : out  STD_LOGIC;        -- Saída para definir a direção do motor
        led: out STD_LOGIC_VECTOR(5 downto 0) -- Saída de debug para monitoramento do estado
    );
end porta_garagem;

architecture Behavioral of porta_garagem is

type state is (st_fechado, st_aberto, abrindo, fechando, parado_abrindo, parado_fechando); -- Definição dos estados
SIGNAL present_state, future_state: state; -- Estado atual e estado futuro
SIGNAL estado_bt: STD_LOGIC; -- Sinal de estado do botão (controle remoto)
SIGNAL pause: STD_LOGIC; -- Sinal de pausa do temporizador
SIGNAL Useg_sig: integer range 0 to 10; -- Sinal para contar segundos
SIGNAL Dseg_sig: integer range 0 to 4; -- Sinal para contar décimos de segundo
SIGNAL flag: std_logic; -- Sinal de flag para alternância de estado
signal flag_timer: std_logic :='0'; -- Sinal para controle do temporizador

-- Componentes para debounce e temporizador
component temporizador is
    PORT(
        rst: in STD_LOGIC;
        clk: in STD_LOGIC;
        pause: in STD_LOGIC;
        Dseg: out integer;
```

```vhdl
        Dseg: out integer;
        Useg: out integer
    );
end component;


component debounce is
    PORT(
        entrada: in std_logic;
        clk: in std_logic;
        saida: out std_logic
    );
end component;



begin

-- Instanciação do componente debounce para tratar o botão remoto
bt: debounce port map(remoto, clk, estado_bt);

-- Instanciação do componente Timer para contar tempo
tempo: temporizador PORT MAP(flag_timer, clk, pause, Dseg_sig, Useg_sig);

-- Processo para atualização do estado presente com base no clock e reset
process(rst, clk)
begin
    if (rst='1') then
        present_state <= st_fechado; -- Reset: Estado inicial é fechado
    elsif rising_edge(clk) then
        present_state <= future_state; -- Atualiza o estado presente com o futuro na borda de subida do clock
    end if;
end process;

```

```
67
68  -- Processo para alternar o sinal de flag baseado no estado do botão
69  -- impede de ficar com o dedo pressionado no botão para sempre e continuar dando sinal
70  process(rst, estado_bt)
71  begin
72      if (rst='1') then
73          flag <= '0'; -- Reset do flag
74      elsif (estado_bt'event and estado_bt='0') then
75          flag <= not(flag); -- Alterna o flag na borda de descida do botão
76      end if;
77  end process;
78
79  -- Processo principal de controle da porta da garagem
80  process (present_state, remoto, aberta, fechada, estado_bt, flag, Dseg_sig)
81  begin
82      case present_state is
83          when st_fechado =>
84              ligar <= '0';
85              direcao <= '0';
86              pause <= '1';
87              led <= "000001";
88              flag_timer <= '1';
89
90              if (estado_bt = '1' and flag = '0' and aberta = '0' and fechada = '1') then
91                  future_state <= abrindo; -- Muda para o estado abrindo
92              else
93                  future_state <= st_fechado; -- Permanece no estado fechado
94              end if;
95
96          when abrindo =>
97              ligar <= '1';
98              direcao <= '0';
99              pause <= '1';
100             led <= "000010";
```

```
100             led <= "000010";
101             flag_timer <= '1';
102
103             if (estado_bt = '1' and aberta = '0' and fechada = '0' and flag = '1') then
104                 future_state <= parado_abrindo; -- Muda para o estado parado abrindo
105             elsif (aberta = '1' and fechada = '0') then
106                 future_state <= st_aberto; -- Muda para o estado aberto
107             else
108                 future_state <= abrindo; -- Permanece no estado abrindo
109             end if;
110
111         when st_aberto =>
112             ligar <= '0';
113             direcao <= '1';
114             pause <= '0';
115             led <= "000100";
116             flag_timer <= '0';
117
118             if (estado_bt = '1' and aberta = '1' and fechada = '0' and flag = '1') then
119                 future_state <= fechando; -- Muda para o estado fechando
120             elsif (Dseg_sig = 3) then --30 segundos
121                 future_state <= fechando; -- Muda para o estado fechando após 3 segundos
122             else
123                 future_state <= st_aberto; -- Permanece no estado aberto
124             end if;
125
126         when fechando =>
127             ligar <= '1';
128             direcao <= '1';
129             pause <= '1';
130             led <= "001000";
131             flag_timer <= '1';
132
133             if (estado_bt = '1' and aberta = '0' and fechada = '0' and flag = '1') then
```

```
133             if (estado_bt = '1' and aberta = '0' and fechada = '0' and flag = '1') then
134                 future_state <= parado_fechando; -- Muda para o estado parado fechando
135             elsif (fechada = '1' and aberta = '0') then
136                 future_state <= st_fechado; -- Muda para o estado fechado
137             else
138                 future_state <= fechando; -- Permanece no estado fechando
139             end if;
140 
141         when parado_abrindo =>
142             ligar <= '0';
143             direcao <= '1';
144             pause <= '0';
145             led <= "010000";
146             flag_timer <= '0';
147 
148             if (estado_bt = '1' and aberta = '0' and fechada = '0' and flag = '1') then
149                 future_state <= fechando; -- Muda para o estado fechando
150             elsif (Dseg_sig = 3) then
151                 future_state <= fechando; -- Muda para o estado fechando após 3 segundos
152             else
153                 future_state <= parado_abrindo; -- Permanece no estado parado abrindo
154             end if;
155 
156         when parado_fechando =>
157             ligar <= '0';
158             direcao <= '0';
159             pause <= '0';
160             led <= "100000";
161             flag_timer <= '0';
162 
163             if (estado_bt = '1' and aberta = '0' and fechada = '0' and flag = '1') then
164                 future_state <= abrindo; -- Muda para o estado abrindo
165             elsif (Dseg_sig = 3) then
166                 future_state <= fechando; -- Muda para o estado fechando após 3 segundos
```

```
166                 future_state <= fechando; -- Muda para o estado fechando após 3 segundos
167             else
168                 future_state <= parado_fechando; -- Permanece no estado parado fechando
169             end if;
170 
171     end case;
172 end process;
173 
174 end Behavioral;
175 
```

O texto descreve a implementação de uma máquina de estados para controlar uma entidade chamada "garagem". Essa entidade possui oito portas: `clk`, `rst`, `remoto`, `aberta`, `fechada`, `ligar`, `direcao` e `led`. Aqui estão as principais informações:

1. Entradas:

- `clk`: Entrada correspondente ao sinal de clock.
- `rst`: Entrada assíncrona que "reseta" o estado da máquina de estados quando atinge nível lógico alto.
- `remoto`: Entrada responsável pelo estado do botão.
- `aberta` e `fechada`: Entradas correspondentes aos sensores de abertura ou fechamento da porta.

2. Saídas:

- `ligar`: Saída que representa o acionamento do motor.

- `direcao`: Saída que indica a direção de abertura/fechamento da porta.
- `led`: Saída codificada em "one hot" dos estados da máquina de estados.

Em seguida, são definidos todos os estados possíveis da máquina de estados, conforme mostrado no diagrama de transição de estados. Além disso, são declarados os sinais atribuídos a outros códigos (`.vhd`) que serão comentados posteriormente quanto às suas funções e funcionalidades.
O sinal `flag` representa a disponibilidade de acionamento do botão. Quando o estado do botão é enviado após o debounce, o código envia apenas um pulso através da `flag`. Também são mencionados os componentes `debounce` e `Timer`, responsáveis por realizar o debounce do botão da entrada `remoto` e a contagem dos segundos.
Por fim, um processo sensível às entradas `rst`, `clk` e `presente_state` é definido. Se a entrada `rst` estiver em nível lógico alto, o estado da máquina de estados é definido com base no sensor (por exemplo, estado "fechado" para `fechada = 1`, estado "aberto" para `aberta = 1`). Caso contrário, verifica-se se o botão está disponível para ser clicado a cada batida do clock, e um bloco de `case` é definido para todos os estados da máquina de estados.

3.1.1.2 código do componente “debounce”:

O componente debounce é usado para eliminar ruídos indesejados (chamados de "bouncing") de sinais digitais, como aqueles provenientes de botões mecânicos. Quando um botão é pressionado, ele pode gerar múltiplas transições antes de se estabilizar, e o circuito debounce garante que apenas uma transição limpa seja passada adiante. O código implementado para tal se encontra a seguir:

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity debounce is
    generic(
        freqclk: integer := 1; -- frequência do clock em kHz
        twindow: integer := 1  -- tempo de intervalo em ms
    );
    Port (
        entrada : in  STD_LOGIC;  -- Entrada do sinal a ser "debounced"
        clk : in  STD_LOGIC;      -- Sinal de clock
        saida : out  STD_LOGIC    -- Saída do sinal "debounced"
    );
end debounce;

architecture Behavioral of debounce is
    signal temp: std_logic := '0';  -- Sinal temporário para armazenar o valor estabilizado
    signal flag: std_logic := '0';  -- Sinal de flag, não utilizado no processo
    constant max: integer := freqclk * twindow;  -- Constante que define o número máximo de contagem
begin

process(clk)
    variable count: integer range 0 to max;  -- Variável para contar ciclos de clock
begin
    if(clk'event and clk = '1') then  -- Detecta a borda de subida do clock
        if(temp /= entrada) then  -- Verifica se a entrada mudou
            count := count + 1;  -- Incrementa o contador
            if(count = max) then  -- Se o contador atingir o valor máximo
                temp <= entrada;  -- Atualiza o sinal temporário com a entrada
                count := 0;  -- Reseta o contador
            end if;
        else
            count := 0;  -- Reseta o contador se a entrada não mudou
        end if;
```

```vhdl
        end if;
    end if;
end process;

saida <= temp;  -- Atribui o sinal temporário à saída

end Behavioral;
```

O componente debounce elimina flutuações indesejadas (bouncing) no sinal de entrada desta forma:

Monitora a entrada e, se detectar uma mudança, começa a contar ciclos de clock.

Se a entrada permanecer estável por um número suficiente de ciclos (definido por max), atualiza a saída com o novo valor.

Se a entrada mudar antes de o contador atingir max, o contador é resetado e começa a contar novamente.

Dessa forma, o componente assegura que apenas transições de sinal estáveis sejam passadas para a saída, eliminando ruídos e garantindo um sinal limpo para outros componentes do sistema

3.1.1.3 código do componente “temporizador”:

O componente temporizador é utilizado para contar o tempo em segundos e dezenas de segundos, controlando uma saída de temporização que pode ser pausada e resetada.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity temporizador is
    PORT(
        rst: in STD_LOGIC;      -- Sinal de reset
        clk: in STD_LOGIC;      -- Sinal de clock
        pause: in STD_LOGIC;    -- Sinal de pausa
        Dseg: out integer;      -- Saída para os dígitos das dezenas de segundos
        Useg: out integer       -- Saída para os dígitos das unidades de segundos
    );
end temporizador;

architecture Behavioral of temporizador is
    CONSTANT Dmax: integer := 4;          -- Valor máximo para os dígitos das dezenas de segundos
    CONSTANT Umax: integer := 10;         -- Valor máximo para os dígitos das unidades de segundos
    CONSTANT Freq: integer := 50000000;   -- Frequência do clock

begin

    process(clk, rst, pause)
        variable count_clk: integer range 0 to Freq := 0;  -- Variável para contar ciclos de clock
        variable Dseg_var: integer range 0 to Dmax := 0;   -- Variável para armazenar os dígitos das dezenas de segundos
        variable Useg_var: integer range 0 to Umax := 0;   -- Variável para armazenar os dígitos das unidades de segundos
    begin
        if(rst = '1') then
            Dseg_var := 0;    -- Reseta os dígitos das dezenas de segundos
            Useg_var := 0;    -- Reseta os dígitos das unidades de segundos
            count_clk := 0;   -- Reseta o contador de ciclos de clock

        elsif(clk'event and clk = '1') then   -- Detecta a borda de subida do clock
            if(pause = '0') then   -- Verifica se o sinal de pausa está desativado
                if(count_clk < Freq) then   -- Verifica se o contador de ciclos não atingiu o máximo
                    count_clk := count_clk + 1;   -- Incrementa o contador de ciclos
```

```
                    count_clk := count_clk + 1;   -- Incrementa o contador de ciclos
                else
                    count_clk := 0;   -- Reseta o contador de ciclos
                    Useg_var := Useg_var + 1;   -- Incrementa os dígitos das unidades de segundos

                    if(Useg_var = Umax) then   -- Verifica se os dígitos das unidades de segundos atingiram o máximo
                        Useg_var := 0;   -- Reseta os dígitos das unidades de segundos
                        Dseg_var := Dseg_var + 1;   -- Incrementa os dígitos das dezenas de segundos

                        if(Dseg_var = Dmax) then   -- Verifica se os dígitos das dezenas de segundos atingiram o máximo
                            Dseg_var := 0;   -- Reseta os dígitos das dezenas de segundos
                        end if;
                    end if;
                end if;
            end if;
        else
            count_clk := count_clk;   -- Mantém o valor do contador de ciclos
        end if;

        Dseg <= Dseg_var;   -- Atribui os dígitos das dezenas de segundos à saída
        Useg <= Useg_var;   -- Atribui os dígitos das unidades de segundos à saída
    end process;

end Behavioral;
```

O componente temporizador conta segundos e dezenas de segundos.

Usa um contador de ciclos de clock (count_clk) para dividir a frequência do clock (Freq) em segundos.

Contabiliza unidades de segundos (Useg_var) e dezenas de segundos (Dseg_var).

Pode ser pausado ou resetado através dos sinais pause e rst.

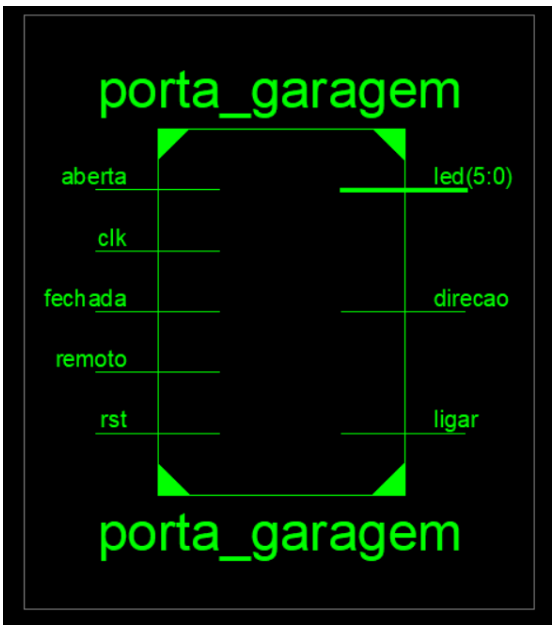
As saídas (Dseg e Useg) são atualizadas com os valores dos contadores internos, representando o tempo decorrido em dezenas e unidades de segundos.

3.1.2 Esquema de pinos e esquemáticos:

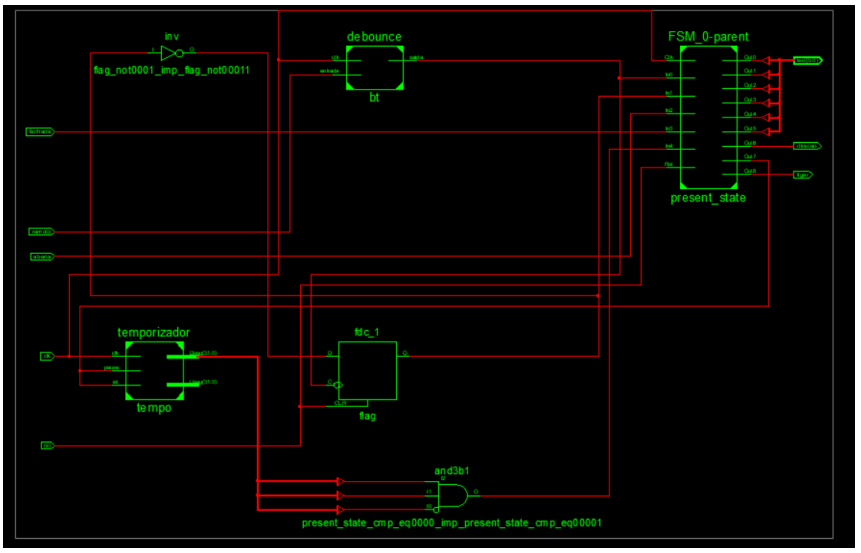
A figura abaixo demonstra o esquema de pinos na placa.

```
1
2  NET "clk" LOC = "B8";
3  NET "remoto" LOC = " B18";
4  NET "aberta" LOC = "G18";
5  NET "fechada" LOC = "H18";
6  NET "rst" LOC = "R17";
7  NET "ligar" LOC = "J15";
8  NET "direcao" LOC = "J14";
9  NET "led(0)" LOC = "K15";
10 NET "led(1)" LOC = "K14";
11 NET "led(2)" LOC = "E17";
12 NET "led(3)" LOC = "P15";
13 NET "led(4)" LOC = "F4";
14 NET "led(5)" LOC = "R4";
15
16
```

O esquemático RTL (Register Transfer Logic) abaixo representa o comportamento do circuito digital a nível de registro:



RTL SIGNALS:



3.1.3 Teste:

Após o desenvolvimento do código e atribuição dos pinos, um *test bench* foi criado a fim de validar o comportamento do hardware descrito em VHDL, para que fosse possível simular seu funcionamento antes mesmo da implementação física na placa.

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.ALL;
3
4  ENTITY garagem_testb IS
5  END garagem_testb;
6
7  ARCHITECTURE behavior OF garagem_testb IS
8
9      -- Component Declaration for the Unit Under Test (UUT)
10
11     COMPONENT garagem
12     PORT(
13          remoto : IN  std_logic;
14          aberta : IN  std_logic;
15          fechada : IN  std_logic;
16          rst : IN  std_logic;
17          clk : IN  std_logic;
18          ligar : OUT  std_logic;
19          direcao : OUT  std_logic;
20          debug : OUT  std_logic_vector(5 downto 0)
21         );
22     END COMPONENT;
23
24
25    --Inputs
26    signal remoto : std_logic := '0';
27    signal aberta : std_logic := '0';
28    signal fechada : std_logic := '0';
29    signal rst : std_logic := '0';
30    signal clk : std_logic := '0';
31
32    --Outputs
33    signal ligar : std_logic;
34    signal direcao : std_logic;
```

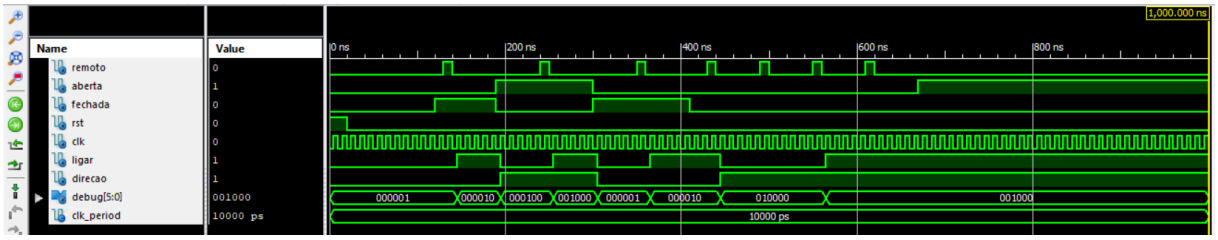
```
34    signal direcao : std_logic;
35    signal debug : std_logic_vector(5 downto 0);
36
37    -- Clock period definitions
38    constant clk_period : time := 10 ns;
39
40  BEGIN
41
42     -- Instantiate the Unit Under Test (UUT)
43     uut: garagem PORT MAP (
44            remoto => remoto,
45            aberta => aberta,
46            fechada => fechada,
47            rst => rst,
48            clk => clk,
49            ligar => ligar,
50            direcao => direcao,
51            debug => debug
52          );
53
54     -- Clock process definitions
55     clk_process :process
56     begin
57        clk <= '0';
58        wait for clk_period/2;
59        clk <= '1';
60        wait for clk_period/2;
61     end process;
62
63
64     -- Stimulus process
65     stim_proc: process
66     begin
67        rst<='1';
```

```
67        rst<='1';
68        -- hold reset state for 100 ns.
69        wait for 20 ns;
70      rst<='0';
71
72        wait for clk_period*10;
73
74       -- Configurar as entradas iniciais
75       remoto <= '0';
76       aberta <= '0';
77       fechada <= '1';
78       rst <= '0';
79
80       wait for 10 ns; -- Fechado
81
82       -- Configurar remoto=1
83       remoto <= '1';
84       wait for 10 ns;
85       remoto <= '0';
86
87       wait for 50 ns; -- Abrindo
88
89       -- Configurar aberta=1
90       aberta <= '1';
91       fechada <= '0';
92       wait for 50 ns; -- Aberto
93
94       -- Configurar remoto=1
95       remoto <= '1';
96       wait for 10 ns;
97       remoto <= '0';
98
99       wait for 50 ns; -- Fechando
100
```

```
100
101      aberta <= '0';
102      fechada <= '1';
103
104      wait for 50 ns; -- Fechado
105
106      -- Configurar remoto=1
107      remoto <= '1';
108      wait for 10 ns;
109      remoto <= '0';
110
111      wait for 50 ns; -- Abrindo
112
113      fechada <= '0';
114      aberta <= '0';
115
116      wait for 20 ns;
117
118      remoto <= '1';
119      wait for 10 ns;
120      remoto <= '0';
121
122      wait for 50 ns; -- Parado abrindo
123
124      remoto <= '1';
125      wait for 10 ns;
126      remoto <= '0';
127
128      wait for 50 ns; -- Fechando
129
130      remoto <= '1';
131      wait for 10 ns;
132      remoto <= '0';
133
```

```
133
134       wait for 50 ns; -- Parado fechando
135
136       remoto <= '1';
137       wait for 10 ns;
138       remoto <= '0';
139
140       wait for 50 ns; -- Abrindo
141
142       aberta <= '1';
143       fechada <= '0';
144
145       wait for 30000 ms;
146
147       -- Aguardar um pouco
148       wait for 100 ns;
149
150        wait;
151     end process;
152
153  END;
```

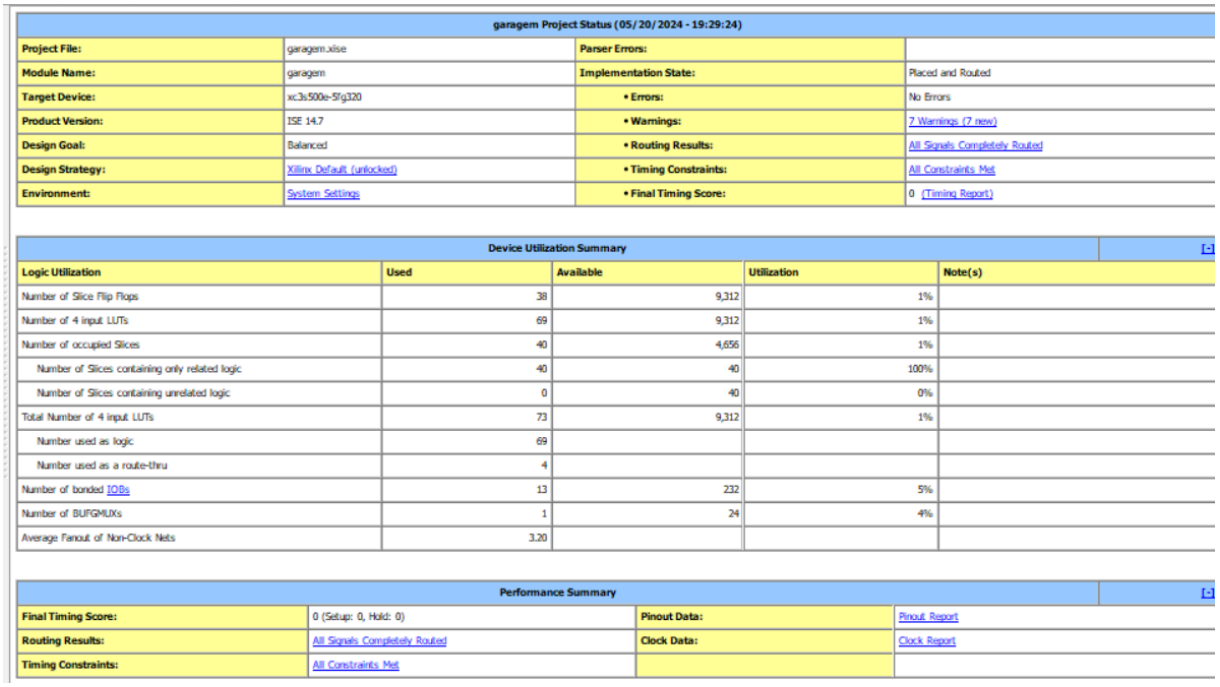
Na figura a seguir, é apresentado o resultado da simulação desenvolvida na seção anterior. Essa simulação ilustra a transição entre os estados e o desempenho do sistema em resposta aos estímulos aplicados. A análise proporciona uma visão abrangente do funcionamento do sistema.



Através do resultado da simulação realizada, é possível notar que a transição de estados ocorre de maneira consistente na borda de subida do sinal de clock, imediatamente após o clique no controle remoto. Além disso, a sequência de estados segue o padrão desejado, evidenciando que o diagrama de estados foi corretamente replicado por meio do código descrito nas seções anteriores.

3.1.4 Utilização do dispositivo e considerações:

Conforme é possível observar na imagem abaixo, apenas 1% de LUTs (estrutura de dados) de 4 inputs foi utilizado.



garagem Project Status (05/20/2024 - 19:29:24)			
Project File:	garagem.xise	**Parser Errors:**	
Module Name:	garagem	**Implementation State:**	Placed and Routed
Target Device:	xc3s500e-5fg320	**• Errors:**	No Errors
Product Version:	ISE 14.7	**• Warnings:**	7 Warnings (7 new)
Design Goal:	Balanced	**• Routing Results:**	All Signals Completely Routed
Design Strategy:	Xilinx Default (unlocked)	**• Timing Constraints:**	All Constraints Met
Environment:	System Settings	**• Final Timing Score:**	0 (Timing Report)

Device Utilization Summary				[-]
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Flip Flops	38	9,312	1%	
Number of 4 input LUTs	69	9,312	1%	
Number of occupied Slices	40	4,656	1%	
Number of Slices containing only related logic	40	40	100%	
Number of Slices containing unrelated logic	0	40	0%	
Total Number of 4 input LUTs	73	9,312	1%	
Number used as logic	69			
Number used as a route-thru	4			
Number of bonded IOBs	13	232	5%	
Number of BUFGMUXs	1	24	4%	
Average Fanout of Non-Clock Nets	3.20			

Performance Summary				[-]
Final Timing Score:	0 (Setup: 0, Hold: 0)	**Pinout Data:**	Pinout Report	
Routing Results:	All Signals Completely Routed	**Clock Data:**	Clock Report	
Timing Constraints:	All Constraints Met			

4 RESULTADOS E CONCLUSÃO:

Em resumo, o projeto e a implementação de circuitos digitais desempenham um papel fundamental na era tecnológica atual, onde dispositivos eletrônicos estão presentes em todos os aspectos de nossa vida cotidiana. O uso de linguagens de descrição de hardware, como VHDL, permite que os projetistas descrevam sistemas digitais de maneira eficaz e eficiente, capturando a lógica e o comportamento dos circuitos em um formato compreensível tanto para humanos quanto para ferramentas de software.

Durante o desenvolvimento deste relatório, exploramos detalhadamente as noções essenciais de VHDL e o processo de projeto de circuitos digitais. Por meio de exemplos práticos, pudemos compreender como descrever componentes, criar módulos, implementar lógica concorrente e, por fim, verificar o comportamento dos sistemas digitais por meio de bancos de testes.

O conhecimento adquirido neste relatório oferece uma base sólida para futuros projetos de circuitos digitais e a compreensão das etapas envolvidas no projeto, simulação, verificação e implementação de sistemas digitais. A aplicação desses princípios não apenas garante a criação de sistemas funcionais, mas também contribui para a otimização do desempenho e a redução de possíveis problemas no design.

Em última análise, o estudo de VHDL e o projeto de circuitos digitais constituem uma jornada empolgante e essencial para qualquer pessoa interessada no campo da eletrônica digital, pois oferecem as ferramentas necessárias para transformar conceitos abstratos em sistemas eletrônicos tangíveis e funcionais que impulsionam nossa sociedade moderna.

5 REFERÊNCIA:

França, S. B. L. (Prof. Dr.). (2024). Aulas de Microeletrônica 1, ministradas pela Prof. Dr. Sibilla Batista da Luz França. UFPR.