Relatório

Tópico: Aula 8 - FSM
Grupo: Maicon Chaves Marques 14593530
Pedro Calciolari Jardim 11233668
João Vitor Pereira Candido 13751131

Parte 1:

Queremos criar uma máquina de estados finitos (FSM) que detecte sequências de quatro símbolos consecutivos iguais na entrada `w`. A saída `z` será 1 sempre que `w` for 1 ou 0 por quatro pulsos de clock consecutivos. Sequências sobrepostas são permitidas, ou seja, se `w` permanecer 1 por cinco pulsos consecutivos, `z` será 1 no quarto e no quinto pulsos. Caso contrário, `z` será 0. Ilustrado na figura abaixo:

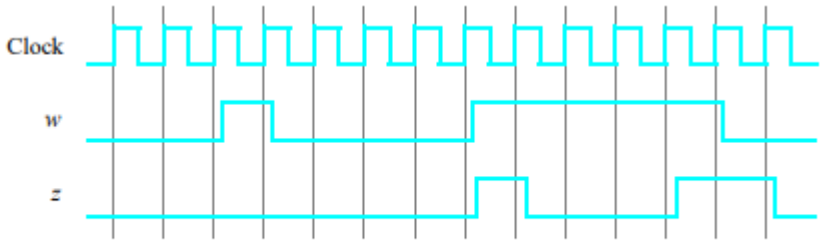
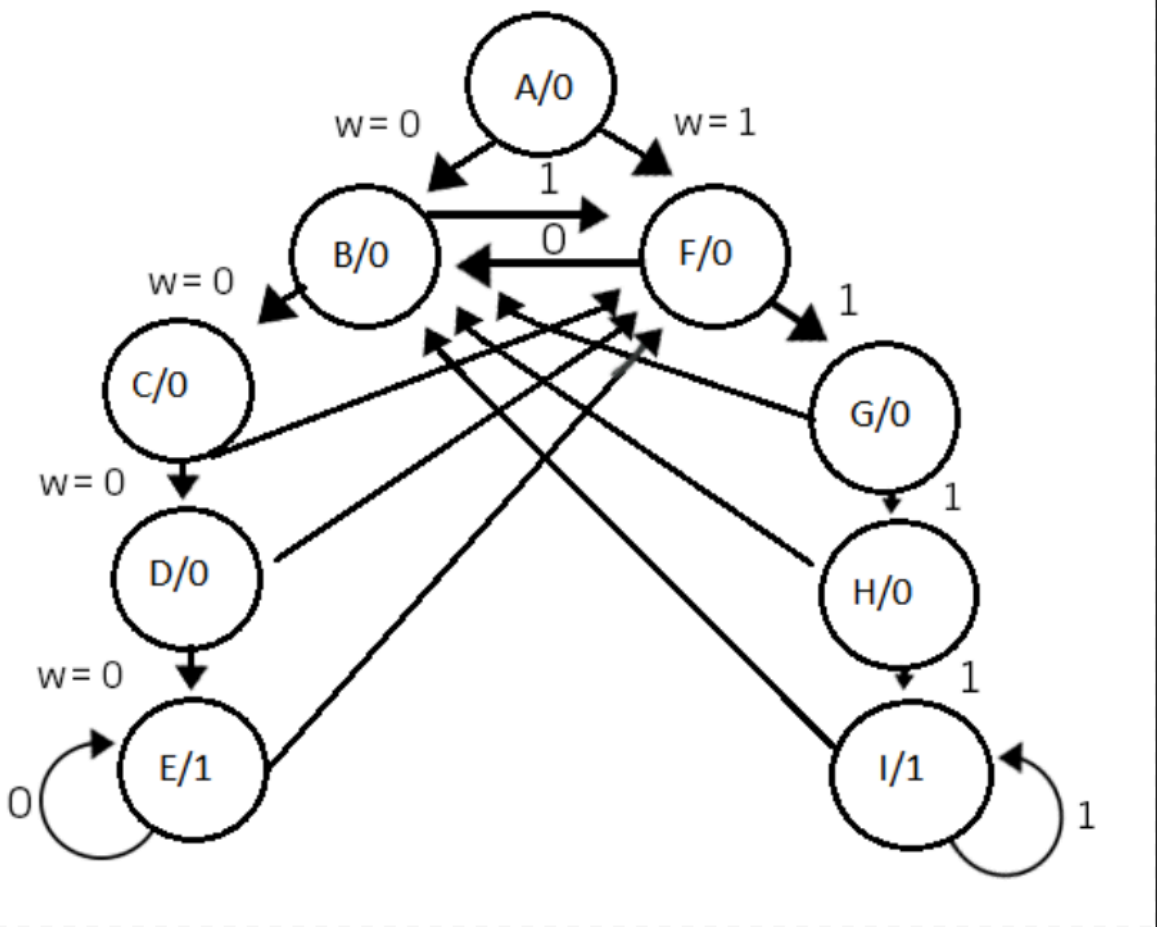


Figure 1: Required timing for the output z.

Esse é o diagrama da máquina:



Uma vez compilado e simulado o circuito, foi feita a configuração da FPGA Cyclone V DE0-CV para testar o circuito de fato.

Código VHDL:

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

entity parte1 is
    Port (
        w : in STD_LOGIC;
        clock : in STD_LOGIC;       -- Sinal de clock
        clear : in STD_LOGIC;       -- Sinal de clear assíncrono
        data_out : out std_logic_vector(8 downto 0);  -- 9 bits de saída de dados
        z : out STD_LOGIC
    );
end parte1;
```

```
architecture Behavioral of parte1 is
   -- Sinais intermediários para conectar os flip-flops
   signal Q_internal : STD_LOGIC_VECTOR (8 downto 0); -- Estados internos dos flip-flops
   signal Not_Q_internal : STD_LOGIC_VECTOR (8 downto 0); -- Inversos dos estados
   signal Q_entrada_internal : STD_LOGIC_VECTOR (8 downto 0); -- Saídas internas dos
flip-flops
   signal Not_Q_entrada_internal : STD_LOGIC_VECTOR (8 downto 0); -- Inversos das
saídas internas
   signal Not_w : STD_LOGIC;
         signal z_in : STD_LOGIC;

   component flipflop is
      port (
         D :  IN  STD_LOGIC;
         CLK :  IN  STD_LOGIC;
         QA :  OUT  STD_LOGIC;
         QB :  OUT  STD_LOGIC
      );
   end component;

begin



process(clock, clear)
   begin
      if clear = '1' then
         -- Se o reset for ativado, zera todos os sinais internos
          Q_entrada_internal(8 downto 1) <= (others => '0');
                                    Q_entrada_internal(0) <= '1';  -- Mantém o último bit como 1
                                    Not_Q_entrada_internal(8 downto 1) <= (others => '1'); --
Inversos dos bits de 0 a 7
                                    Not_Q_entrada_internal(0) <= '0'; -- Inverso do último bit que
é 1
                                    z<= '0';
      elsif falling_edge(clock) then
         -- Atualiza os estados dos flip-flops com base na lógica desejada
         Q_entrada_internal(8) <= (w AND Q_internal(8)) OR (w AND Q_internal(7));
         Q_entrada_internal(7) <= (w AND Q_internal(6));
         Q_entrada_internal(6) <= (w AND Q_internal(5));
         Q_entrada_internal(5) <= (w AND (Q_internal(0) OR Q_internal(1) OR Q_internal(2)
OR Q_internal(3) OR Q_internal(4)));
         Q_entrada_internal(4) <= (Not_w AND Q_internal(4)) OR (Not_w AND
Q_internal(3));
         Q_entrada_internal(3) <= (Not_w AND Q_internal(2));
         Q_entrada_internal(2) <= (Not_w AND Q_internal(1));
         Q_entrada_internal(1) <= (Not_w AND (Q_internal(0) OR Q_internal(5) OR
Q_internal(6) OR Q_internal(7) OR Q_internal(8)));
```

```
        Q_entrada_internal(0) <= '0';

        -- Atualiza os inversos
                                Not_Q_entrada_internal <= NOT Q_entrada_internal;
                        elsif rising_edge(clock) then
                                z <= (Q_entrada_internal(4) OR Q_entrada_internal(8));
    end if;
end process;

                Not_w <= NOT w;
        -- Instanciando os flip-flops
                        dff1: flipflop
                                port map (
                                        D => Q_entrada_internal(0),
                                        CLK => clock,
                                        QA => Q_internal(0),
                                        QB => Not_Q_internal(0)
                                );

                        dff2: flipflop
                                port map (
                                        D => Q_entrada_internal(1),
                                        CLK => clock,
                                        QA => Q_internal(1),
                                        QB => Not_Q_internal(1)
                                );

                        dff3: flipflop
                                port map (
                                        D => Q_entrada_internal(2),
                                        CLK => clock,
                                        QA => Q_internal(2),
                                        QB => Not_Q_internal(2)
                                );

                        dff4: flipflop
                                port map (
                                        D => Q_entrada_internal(3),
                                        CLK => clock,
                                        QA => Q_internal(3),
                                        QB => Not_Q_internal(3)
                                );

                        dff5: flipflop
                                port map (
                                        D => Q_entrada_internal(4),
                                        CLK => clock,
                                        QA => Q_internal(4),
```

```
                QB => Not_Q_internal(4)
        );

    dff6: flipflop
        port map (
                D => Q_entrada_internal(5),
                CLK => clock,
                QA => Q_internal(5),
                QB => Not_Q_internal(5)
        );

    dff7: flipflop
        port map (
                D => Q_entrada_internal(6),
                CLK => clock,
                QA => Q_internal(6),
                QB => Not_Q_internal(6)
        );

    dff8: flipflop
        port map (
                D => Q_entrada_internal(7),
                CLK => clock,
                QA => Q_internal(7),
                QB => Not_Q_internal(7)
        );

    dff9: flipflop
        port map (
                D => Q_entrada_internal(8),
                CLK => clock,
                QA => Q_internal(8),
                QB => Not_Q_internal(8)
        );

     data_out <= Q_internal;

   -- Atribuindo a saída dos flip-flops à saída do contad

end Behavioral;
```

Parte 4:

Nesta parte do exercício, implementamos um codificador de código Morse usando uma FSM por meio de pulsos gerados em um LED (representado por `ledr0`). A1. Definição da Entidade e Arquitetura

A entidade `parte4` define as entradas e saídas do sistema:

- Entradas:
  - `clock` e `clear`: Sinais de controle do sistema.
  - `key0` e `key1`: Botões de reset.
  - `sw`: Conjunto de interruptores para selecionar a letra.
- Saídas:
  - `data_out`: Estados internos dos flip-flops.
  - `ledr0`: Sinal de saída para exibir os pulsos Morse.

A arquitetura `Behavioral` define os sinais intermediários e componentes usados para implementar o sistema.

Código VHDL desta parte:

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;


entity parte4 is

    Port (

        clock : in STD_LOGIC;       -- Sinal de clock

        clear : in STD_LOGIC;       -- Sinal de clear assíncrono

        key0     : IN STD_LOGIC;        -- Botão de reset

                    key1     : IN STD_LOGIC;        -- Botão de reset

                    data_out : out std_logic_vector(8 downto 0);  -- 9 bits de saída de dados

        sw       : IN STD_LOGIC_VECTOR(2 DOWNTO 0); -- Interruptores SW2-0 para a letra

        ledr0 : out STD_LOGIC

    );

end parte4;


architecture Behavioral of parte4 is

    -- Sinais intermediários para conectar os flip-flops
```

```
    signal Q_internal : STD_LOGIC_VECTOR (8 downto 0); -- Estados internos dos flip-flops

    signal Not_Q_internal : STD_LOGIC_VECTOR (8 downto 0); -- Inversos dos estados

    signal Q_entrada_internal : STD_LOGIC_VECTOR (8 downto 0); -- Saídas internas dos
flip-flops

    signal Not_Q_entrada_internal : STD_LOGIC_VECTOR (8 downto 0); -- Inversos das
saídas internas



        SIGNAL morse_seq  : STD_LOGIC_VECTOR(14 DOWNTO 0);  -- Armazena os
pontos e traços

    SIGNAL counter    : INTEGER RANGE 0 TO 50000000;   -- Contador para gerar pulsos
de 0.5 e 1.5s

    SIGNAL pulse_0_5s : STD_LOGIC := '0';             -- Pulso de 0.5 segundos

    SIGNAL current_bit : INTEGER RANGE 0 TO 20 := 0;    -- Índice do bit atual no código
Morse

        SIGNAL next1 : STD_LOGIC;



    -- Definir constantes de pontos e traços para cada letra

        CONSTANT A_morse : STD_LOGIC_VECTOR(14 DOWNTO 0) :=
"000000000011101"; -- A: .-

        CONSTANT B_morse : STD_LOGIC_VECTOR(14 DOWNTO 0) :=
"000000101010111"; -- B: -...

        CONSTANT C_morse : STD_LOGIC_VECTOR(14 DOWNTO 0) :=
"000010111010111"; -- C: -.-.

        CONSTANT D_morse : STD_LOGIC_VECTOR(14 DOWNTO 0) :=
"000000001010111"; -- D: -..

        CONSTANT E_morse : STD_LOGIC_VECTOR(14 DOWNTO 0) :=
"000000000000001"; -- E: .

        CONSTANT F_morse : STD_LOGIC_VECTOR(14 DOWNTO 0) :=
"000000101110101"; -- F: ..-.

        CONSTANT G_morse : STD_LOGIC_VECTOR(14 DOWNTO 0) :=
"000000101110111"; -- G: --.
```

```
        CONSTANT H_morse : STD_LOGIC_VECTOR(14 DOWNTO 0) :=
"000000001010101"; -- H: ....

        CONSTANT I_morse : STD_LOGIC_VECTOR(14 DOWNTO 0) :=
"000000000000000"; -- H: ....



    component flipflop is

        port (

            D :  IN  STD_LOGIC;

            CLK :  IN  STD_LOGIC;

            QA :  OUT  STD_LOGIC;

            QB :  OUT  STD_LOGIC

        );

    end component;


begin




         PROCESS (Q_internal)  -- Adicionado sw à lista de sensibilidade

    BEGIN

        IF falling_edge(clock) AND (next1 = '1') THEN

            CASE Q_internal IS

                                            WHEN "000000001" => morse_seq <= I_morse;  --
Letra I

                WHEN "000000010" => morse_seq <= A_morse;  -- Letra A

                WHEN "000000100" => morse_seq <= B_morse;  -- Letra B

                WHEN "000001000" => morse_seq <= C_morse;  -- Letra C

                WHEN "000010000" => morse_seq <= D_morse;  -- Letra D

                WHEN "000100000" => morse_seq <= E_morse;  -- Letra E
```

```
            WHEN "001000000" => morse_seq <= F_morse;  -- Letra F
            WHEN "010000000" => morse_seq <= G_morse;  -- Letra G
            WHEN "100000000" => morse_seq <= H_morse;  -- Letra H
            WHEN OTHERS => morse_seq <= I_morse; -- Padrão (Letra A)
        END CASE;
                        END IF;
    END PROCESS;

    -- Processo para exibir o código Morse no LEDR0
    PROCESS (clock, key0, key1)
    BEGIN
                IF Key0 = '0' or key1 = '0' then
                    -- Se o reset for ativado, zera todos os sinais internos
                      Q_entrada_internal(8 downto 1) <= (others => '0');
                      Q_entrada_internal(0) <= '1';  -- Mantém o último bit como 1
                      Not_Q_entrada_internal(8 downto 1) <= (others => '1'); -- Inversos
dos bits de 0 a 7
                      Not_Q_entrada_internal(0) <= '0'; -- Inverso do último bit que é 1
                      current_bit <= 0;
        counter <= 0;
        pulse_0_5s <= '0'; -- Adicionado reset do pulso
      ELSIF rising_edge(clock) THEN
        IF current_bit < 15 THEN
          IF morse_seq(current_bit) = '1' THEN
            -- Gera pulso de 0.5 segundos
            IF counter < 25000000 THEN
              pulse_0_5s <= '1';
```

```
                counter <= counter + 1;
            ELSE
                counter <= 0;
                current_bit <= current_bit + 1;
            END IF;
        ELSE
            -- Gera pulso de 0.5 segundos para '0'
            IF counter < 25000000 THEN
                pulse_0_5s <= '0';
                counter <= counter + 1;
            ELSE
                counter <= 0;
                current_bit <= current_bit + 1;
            END IF;
        END IF;
    ELSE
                                    Q_entrada_internal(0) <= '0';
                                    Q_entrada_internal(1) <= ((NOT sw(2)) AND (NOT
sw(1)) AND (NOT sw(0)));
                                    Q_entrada_internal(2) <= ((NOT sw(2)) AND (NOT
sw(1)) AND (sw(0)));
                                    Q_entrada_internal(3) <= ((NOT sw(2)) AND (sw(1))
AND (NOT sw(0)));
                                    Q_entrada_internal(4) <= ((NOT sw(2)) AND (sw(1))
AND (sw(0)));
                                    Q_entrada_internal(5) <= ((sw(2)) AND (NOT sw(1))
AND (NOT sw(0)));
                                    Q_entrada_internal(6) <= ((sw(2)) AND (NOT sw(1))
AND (sw(0)));
```

```
                                        Q_entrada_internal(7) <= ((sw(2)) AND (sw(1)) AND
(NOT sw(0)));
                                        Q_entrada_internal(8) <= ((sw(2)) AND (sw(1)) AND
(sw(0)));

                                        -- Atualiza os inversos
                                        Not_Q_entrada_internal <= NOT Q_entrada_internal;
                                        current_bit <= 0;
                                        counter <= 0;
                                        pulse_0_5s <= '0'; -- Adicionado reset do pulso
                                        next1 <= '1';

        END IF;
      END IF;
   END PROCESS;

        -- Instanciando os flip-flops
                        dff1: flipflop
                                port map (
                                        D => Q_entrada_internal(0),
                                        CLK => clock,
                                        QA => Q_internal(0),
                                        QB => Not_Q_internal(0)
                                );

                        dff2: flipflop
                                port map (
```

```
                D => Q_entrada_internal(1),
                CLK => clock,
                QA => Q_internal(1),
                QB => Not_Q_internal(1)
        );

dff3: flipflop
        port map (
                D => Q_entrada_internal(2),
                CLK => clock,
                QA => Q_internal(2),
                QB => Not_Q_internal(2)
        );

dff4: flipflop
        port map (
                D => Q_entrada_internal(3),
                CLK => clock,
                QA => Q_internal(3),
                QB => Not_Q_internal(3)
        );

dff5: flipflop
        port map (
                D => Q_entrada_internal(4),
                CLK => clock,
```

```
                QA => Q_internal(4),
                QB => Not_Q_internal(4)
        );

dff6: flipflop
        port map (
                D => Q_entrada_internal(5),
                CLK => clock,
                QA => Q_internal(5),
                QB => Not_Q_internal(5)
        );

dff7: flipflop
        port map (
                D => Q_entrada_internal(6),
                CLK => clock,
                QA => Q_internal(6),
                QB => Not_Q_internal(6)
        );

dff8: flipflop
        port map (
                D => Q_entrada_internal(7),
                CLK => clock,
                QA => Q_internal(7),
                QB => Not_Q_internal(7)
```

```
        );

    dff9: flipflop
        port map (
            D => Q_entrada_internal(8),
            CLK => clock,
            QA => Q_internal(8),
            QB => Not_Q_internal(8)
        );

    data_out <= Q_internal;
    ledr0 <= pulse_0_5s;

-- Atribuindo a saída dos flip-flops à saída do contad

end Behavioral;
```