

Máquinas de Estados Finitos

Disciplina:
Laboratório de Sistemas Digitais

Professor:
Jhonattan Córdoba Ramírez

Alunos:
João Pedro Copelli - 2021014414
João Victor Gomes - 2020072690
Marcelle Christine Aquino Silva - 2021014546

15 de outubro de 2022

Sumário

1	Introdução	3
2	Parte Teórica	3
2.1	O que são e para que servem as máquinas de estados finitos? .	3
2.2	Quais as diferenças entre Máquinas de Mealy e Máquinas de Moore?	3
2.3	Como modelar máquinas de estados finitos em VHDL?	4
2.3.1	Usando um único process	4
2.3.2	Usando múltiplos processes	5
2.4	O que é one-hot encoding?	5
2.5	Como modelar os estados de uma Máquina de Estados Finitos em VHDL usando onehot encoding?	6
2.5.1	Usando múltiplos processes	7
3	Parte Prática	8
4	Conclusão	13

1 Introdução

Neste relatório serão discutidos conceitos do que são máquinas de estado, quais as diferenças entre uma máquina de Mealy e uma máquina de Moore, como modelar a FSM com um ou mais process em VHDL e sobre one-hot encoding..

2 Parte Teórica

2.1 O que são e para que servem as máquinas de estados finitos?

Uma máquina de estados finita (FSM – do inglês Finite State Machine) ou autômato finito é um modelo matemático usado para representar programas de computadores ou circuitos lógicos. O conceito é concebido como uma máquina abstrata que deve estar em um de um número finito de estados.

2.2 Quais as diferenças entre Máquinas de Mealy e Máquinas de Moore?

A Máquina de Moore possui uma função que gera uma palavra de saída (que pode ser vazia) para cada estado da máquina. Esta saída só depende do estado atual da máquina. Já a Máquina de Mealy é um Autômato Finito modificado de forma a gerar uma palavra de saída para cada transição entre os estados.

2.3 Como modelar máquinas de estados finitos em VHDL?

2.3.1 Usando um único process

```
PROCESS ( Resetn, Clock )
BEGIN
    IF Resetn...
        y <= A;
    ELSIF (Clock'EVENT AND
            Clock = '1') THEN
        CASE y IS
            WHEN A =>
                IF w = '0'
                    THEN y <= A;
                ELSE y <= B;
                END IF;
            WHEN B =>
                ....
        END CASE;
END PROCESS;
```

MC602 – 2011

Figura 1: FSM em um único process

2.3.2 Usando múltiplos processes

```
PROCESS ( w, y_present )
BEGIN
    CASE y_present IS
        WHEN A =>
            IF w = '0' THEN
                y_next <= A;
            ELSE
                y_next <= B;
            END IF;
        WHEN B => ....
    END CASE;
END PROCESS;

PROCESS (Clock, Resetn)
BEGIN
    IF Resetn ....
        y_present <= A;
    ELSIF (Clock'EVENT AND
            Clock = '1') THEN
        y_present <= y_next;
    END IF;
END PROCESS;
```

Figura 2: FSM em dois processes

2.4 O que é one-hot encoding?

É uma maneira de codificar os estados de algum processo com apenas um bit alto e todos os outros baixos. Por exemplo, o estado 'A' é '001', o 'B' é '010' e o 'C' é '100'.

2.5 Como modelar os estados de uma Máquina de Estados Finitos em VHDL usando onehot encoding?

```
-- library declaration
library IEEE;
use IEEE.std_logic_1164.all;
-- entity
entity my_fsm4_oh is
    port ( X,CLK,RESET : in std_logic;
           Y : out std_logic_vector(3 downto 0);
           Z1,Z2 : out std_logic);
end my_fsm4_oh;
-- architeture
architecture fsm4_oh of my_fsm4_oh is
    type state_type is (ST0,ST1,ST2,ST3);
    attribute ENUM_ENCODING: STRING;
    attribute ENUM_ENCODING of state_type: type is "1000 0100 0010 0001";
    signal PS,NS : state_type;
begin
    sync_proc: process(CLK,NS,RESET)
    begin
        if (RESET = '1') then PS <= ST0;
        elsif (rising_edge(CLK)) then PS <= NS;
        end if;
    end process sync_proc;
```

Figura 3: FSM em um único process

Acima temos a declaração da entidade, da arquitetura com os estados da FSM e de um process que somente fará a troca do estado atual para o próximo estado. É possível perceber o uso do *one-hotencoding* na definição dos códigos dos estados.

2.5.1 Usando múltiplos processes

```

comb_proc: process(PS,X)
begin
    -- Z1: the Moore output; Z2: the Mealy output
    Z1 <= '0'; Z2 <= '0'; -- pre-assign the outputs
    case PS is
        when ST0 =>      -- items regarding state ST0
            Z1 <= '1'; -- Moore output
            if (X = '0') then NS <= ST1; Z2 <= '0';
            else NS <= ST0; Z2 <= '1';
            end if;
        when ST1 =>      -- items regarding state ST1
            Z1 <= '1'; -- Moore output
            if (X = '0') then NS <= ST2; Z2 <= '0';
            else NS <= ST1; Z2 <= '1';
            end if;
        when ST2 =>      -- items regarding state ST2
            Z1 <= '0'; -- Moore output
            if (X = '0') then NS <= ST3; Z2 <= '0';
            else NS <= ST2; Z2 <= '1';
            end if;
        when ST3 =>      -- items regarding state ST3
            Z1 <= '1'; -- Moore output
            if (X = '0') then NS <= ST0; Z2 <= '0';
            else NS <= ST3; Z2 <= '1';
            end if;
        when others => -- the catch all condition
            Z1 <= '1'; Z2 <= '0'; NS <= ST0;
    end case;
end process comb_proc;

-- one-hot encoded approach
with PS select
    Y <= "1000" when ST0,
    "0100" when ST1,
    "0010" when ST2,
    "0001" when ST3,
    "1000" when others;
end fsm4_oh;

```

Figura 4: FSM em dois processes

No process combinacional acima, verificamos qual o estado atual e tomamos as ações com base no estado da chave que realiza a troca de estado. Após o process, temos a atribuição da saída com base no estado atual definido anteriormente.

3 Parte Prática

Criamos um novo projeto no Quartus II e adicionamos o arquivo *pseudo_mux fsm.vhd* ao projeto e modelamos o sistema.

Abaixo podemos ver a FSM e o RTL gerados:

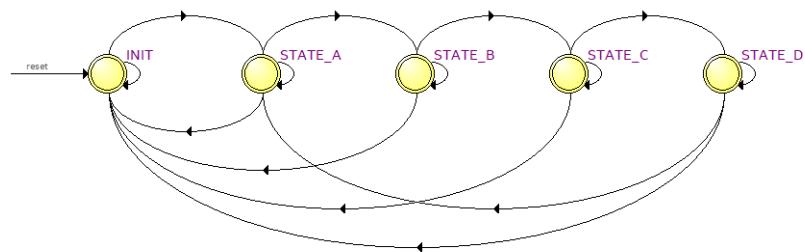


Figura 5: FSM gerada pelo Quartus

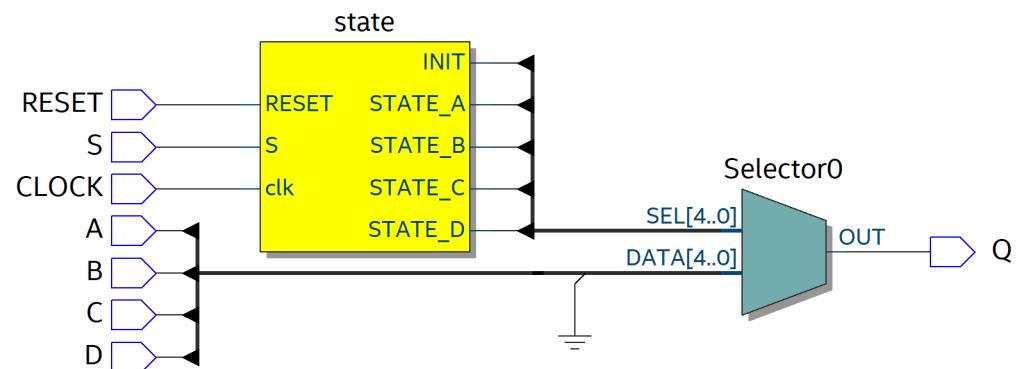


Figura 6: RTL do modelo

Com tudo verificado e o código alterado conforme a descrição acima, fizemos a configuração do *testbench* e executamos a simulação pelo ModelSim. Com a simulação também funcional, podemos fazer o planejamento de quais pinos serão utilizados na placa real. Definimos que os pinos utilizados serão:

- Entradas INPUT:
 - Chave SW0 - PIN_C10
 - Chave SW1 - PIN_C11
 - Chave SW2 - PIN_D12
 - Chave SW3 - PIN_C12
- Entrada S:
 - Chave SW8 - PIN_A14
- Entrada RESET:
 - Chave SW8 - PIN_B14
- Entrada CLOCK:
 - Chave SW9 - PIN_P11
- Saída Q:
 - LED0 - PIN_A8

Para o teste utilizando a placa DE10, construímos um *clock divider* para reduzir o *clock* da DE10 de $50MHz$ para $0,5Hz$. Com isso, realizamos a gravação da placa para os testes.//

Com a placa gravada, realizamos o seguinte teste: Realizamos o acionamento da entrada "S" para realizar a troca da atribuição da saída "Q" e ao final acionamos o RESET para voltarmos ao primeiro estado.

Os resultados obtidos foram esses:



Figura 7: Comportamento Inicial: Acompanhando a entrada 'A'



Figura 8: Acompanhando a entrada 'A'

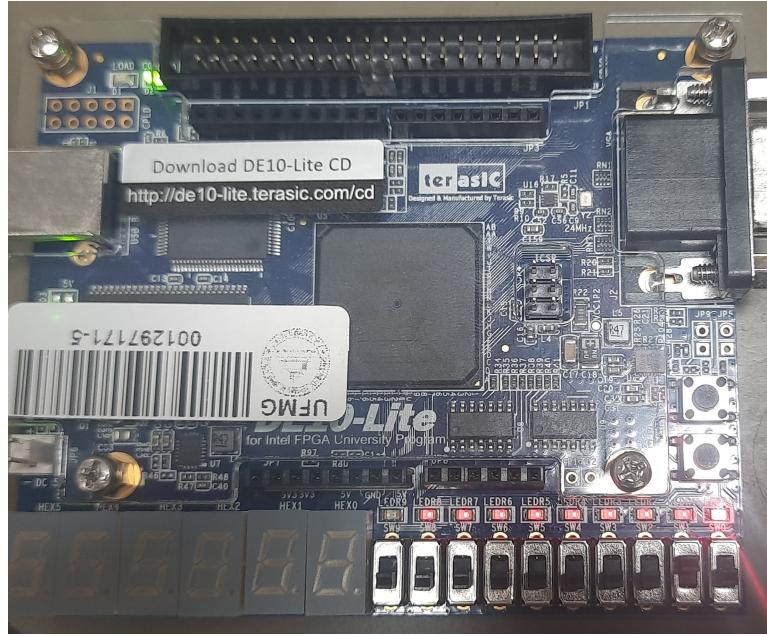


Figura 9: Acionamento de 'S': Acompanhando a entrada 'B'



Figura 10: Acionamento de 'S': Acompanhando a entrada 'C'



Figura 11: Acionamento de 'S': Acompanhando a entrada 'D'



Figura 12: Acionamento do RESET: Volta para 'A'

4 Conclusão

Como podemos observar pelas figuras acimas e corrigindo os erros encontrados, os resultados obtidos foram iguais aos resultados esperados.