Aula 8 – Projeto Lógico Sequencial

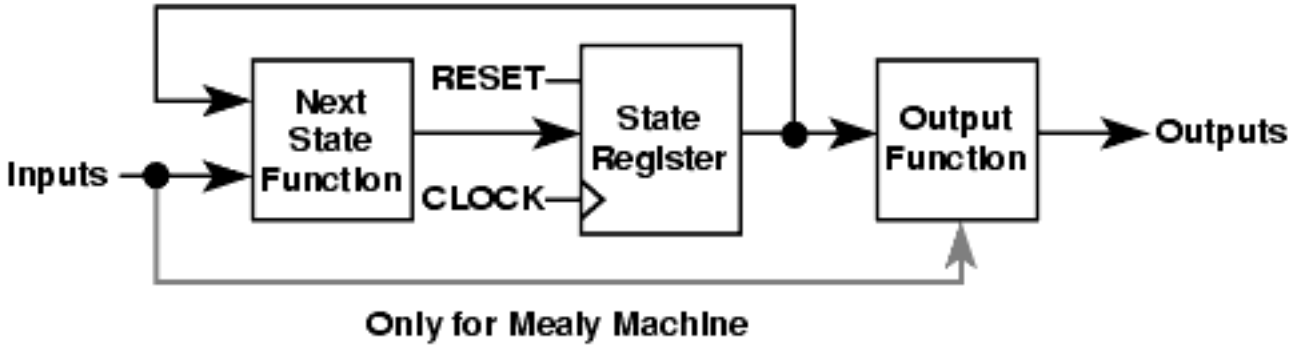
FSMs

FSMs

- **F**inite **S**tate **M**achines: modelam circuitos lógicos sequenciais e são úteis para projetar sistemas cujas tarefas formam uma sequência bem definida (e.g., controladores digitais)
- Possuem **registradores** para armazenar o **estado atual** e uma **combinação lógica** para funções de **próximo estado** e **saída**

FSMs

- **Moore**: saída depende do estado atual
- **Mealy**: saída depende do estado atual e das entradas

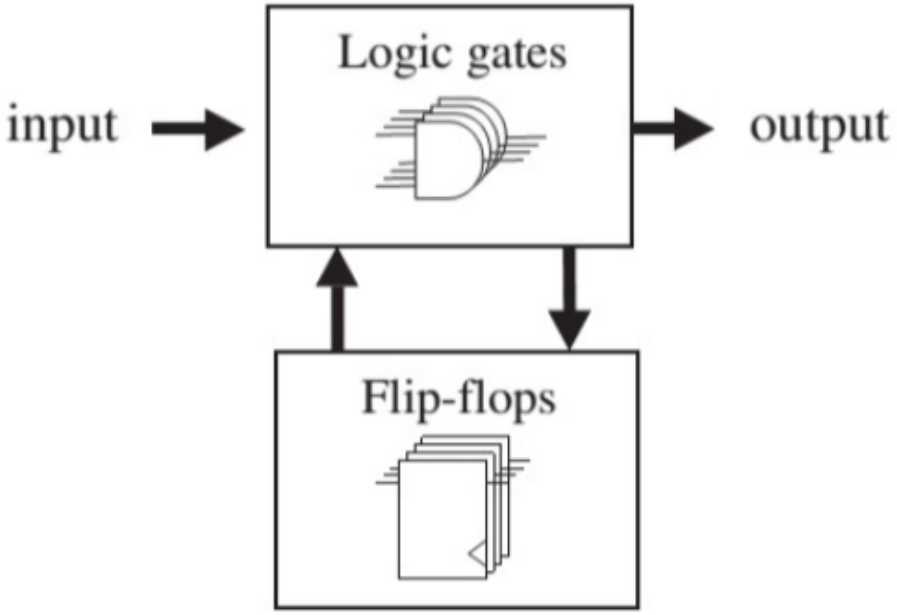


FSMs em VHDL

- FSMs tipicamente incluem:
  - Pelo menos **dois processos**, sendo que um deles controla o clock
  - Sentenças ***if-then-else***
  - Sentenças ***case-when***
  - **Tipos** que o usuário define para armazenar o estado atual e o próximo estado

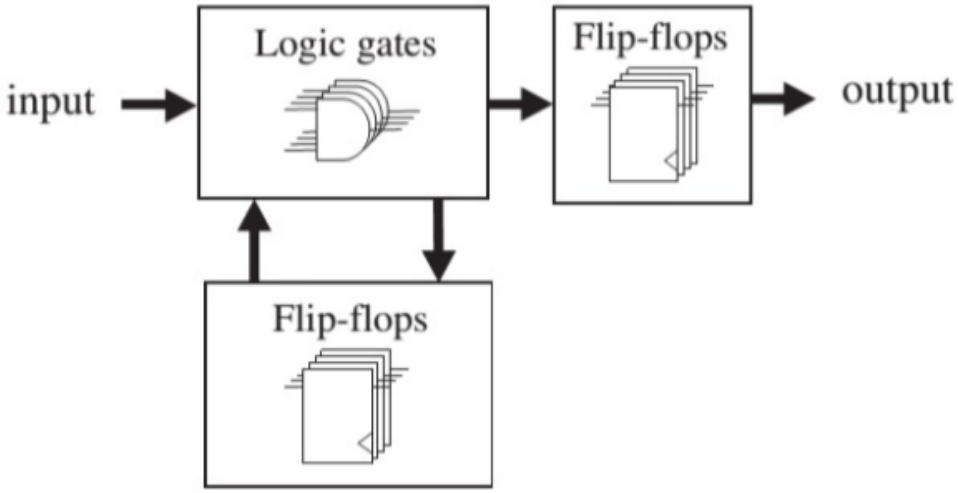
FSMs em VHDL

- **Estilo 1**:
  - Apenas o próximo estado é registrado
  - Saídas são assíncronas
  - Armazenamento do estado separado da lógica de transição e de saída



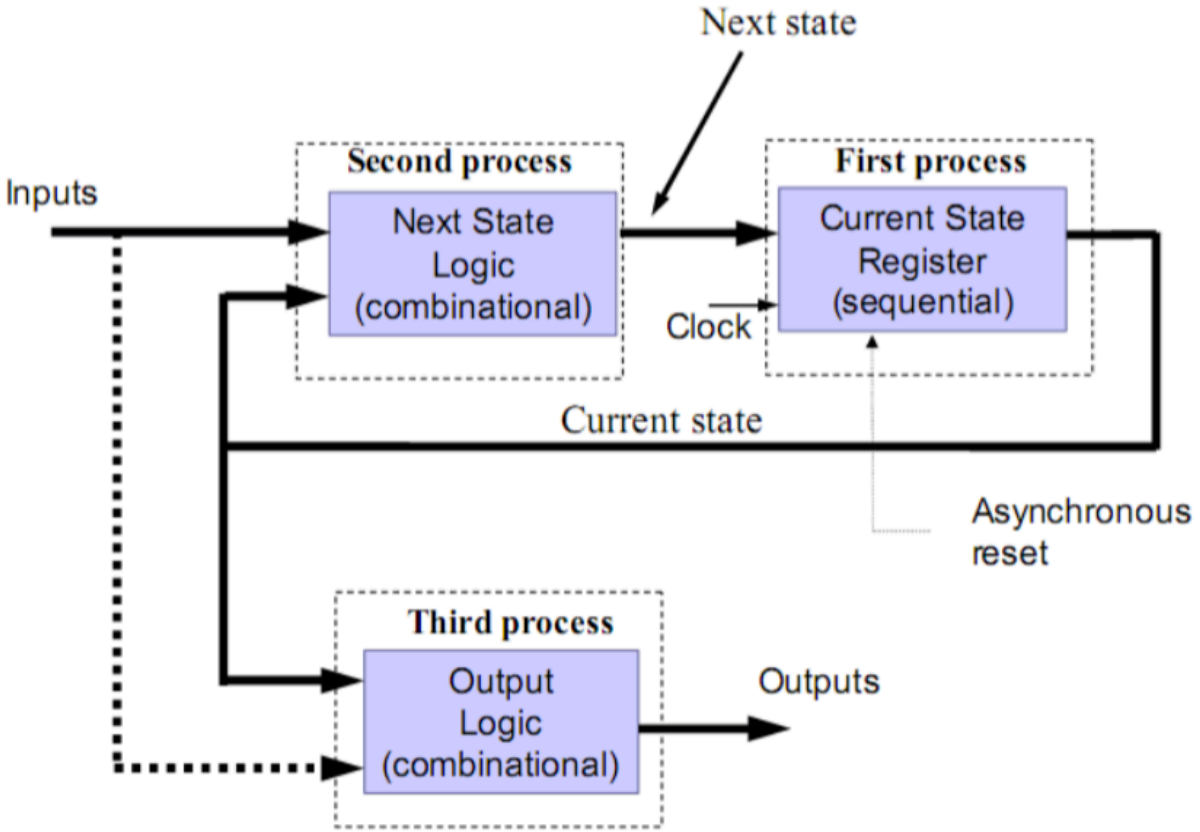
FSMs em VHDL

- **Estilo 2**:
  - As saídas também são registradas
  - Em algumas aplicações, as saídas requerem sincronismo



Estilo 1

- 3 processos



Estilo 1

- **Processo 1:** lógica sequencial do estado atual

```
state_reg: process(clk,reset)
begin
        if rising_edge(clk) then
            if reset='1' then
                current_state <= initial_state;
            else
                current_state <= next_state;
            end if;
        end if
end process state_reg;
```

Estilo 1

- **Processo 2:** lógica combinacional do próximo estado

```
next_state_logic: process(current_state,input1,input2,...)
begin
    case current_state is
        when state1 =>
            if condition1 then
                next_state <= state_value;
            elsif condition2 then
                next_state <= state_value;
            ...
            else
                next_state <= state_value;
            end if;
        when state2=>
            ...
    end case;
end process next_state_logic;
```

Estilo 1

- **Processo 3:** lógica combinacional de saída

```
output_logic: process(current_state,input1,input2,...)
begin
    case current_state is
        when state1 =>
            moore_output1 <= value;
            moore_output2 <= value;
            ...
            if condition1 then
                mealy_output1 <= value;
                mealy_output2 <= value;
                ...
            elsif condition2 then
                mealy_output1 <= value;
                ...
            else
                mealy_output1 <= value;
                ...
            end if;
        when state2=>
            ...
    end case;
end process output_logic;
```

Estilo 1

- Com frequência, os processos 2 e 3 podem ser combinados em um único processo

```
fsm_logic: process(current_state, input1, input2,...)
begin
   case current_state is
      when state1 =>
         if condition1 then
            next_state <= state_value;
            mealy_output1 <= value;
            mealy_output2 <= value;
            ...
         elsif condition2 then
            next_state <= state_value;
            mealy_output1 <= value;
            ...
         ...
         else
            next_state <= state_value;
            mealy_output1 <= value;
            ...
         end if;
         moore_output1 <= value;
         ...
      when state2 =>
         ...
   end case;
end process fsm_logic;
```

Estilo 1

- Usando apenas um processo:

```
architecture Behavioral of MaquinaMoore is
   type os_estados is (e0,e1);
   signal estado: os_estados;
begin

maquina: process(clock, reset)
   begin
      if reset = '1' then
         estado <= e0;
      elsif rising_edge(clock) then
         case estado is
            when e0 =>
               if entrada = '1' then
                  estado <= e1;
               else
                  estado <= e0;
               end if;
            when e1 =>
               if entrada = '1' then
                  estado <= e1;
               else
                  estado <= e0;
               end if;
         end case;
      end if;
   end process maquina;
   saida <= '0' when estado = e0 else '1';
end Behavioral;
```

Definição dos estados

Definição das transições

Definição da saída

Estilo 2

- Saídas registradas

```
ARCHITECTURE <arch_name> OF <ent_name> IS
   TYPE states IS (state0, state1, state2, state3, ...);
   SIGNAL pr_state, nx_state: states;
   SIGNAL temp: <data_type>;
BEGIN

PROCESS (reset, clock)
BEGIN
   IF (reset='1') THEN
      pr_state <= state0;
   ELSIF (clock'EVENT AND clock='1') THEN
      output <= temp;
      pr_state <= nx_state;
   END IF;
END PROCESS;
```

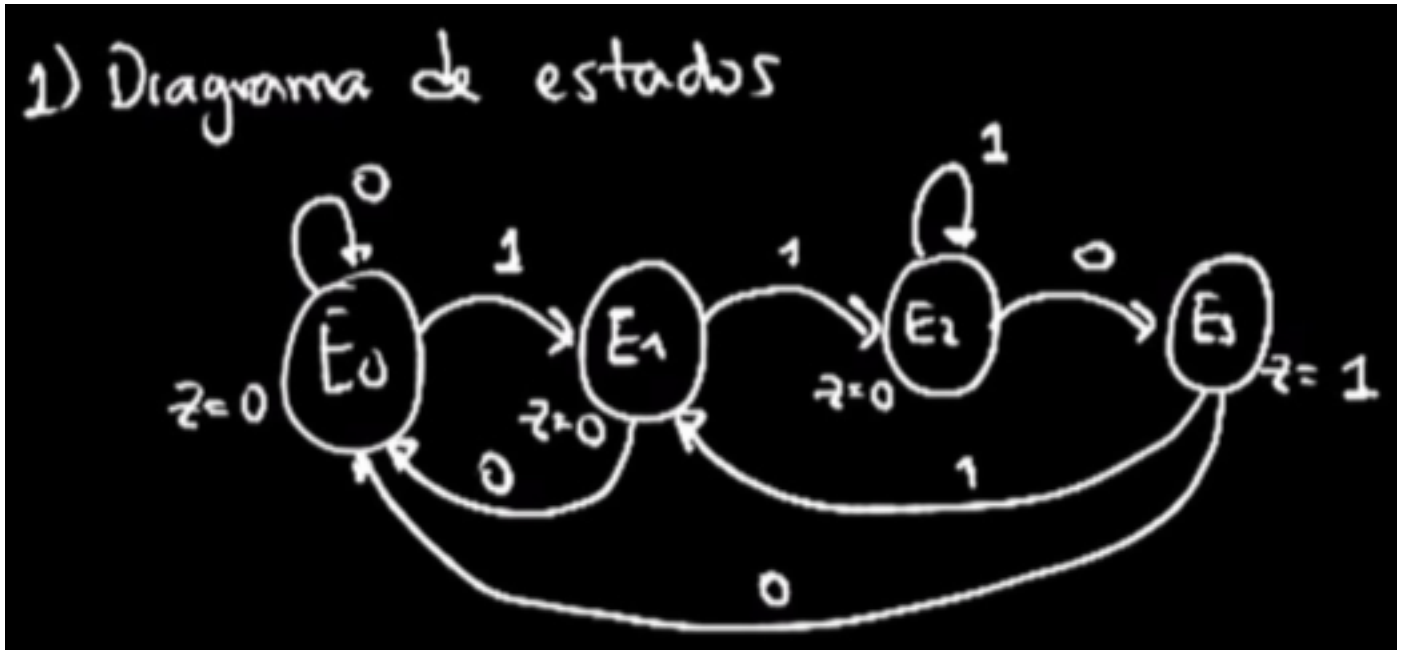
Estilo 2

```
PROCESS (pr_state)
BEGIN
   CASE pr_state IS
      WHEN state0 =>
         temp <= <value>;
         IF (condition) THEN nx_state <= state1;
         ...
         END IF;
      WHEN state1 =>
         temp <= <value>;
         IF (condition) THEN nx_state <= state2;
         ...
         END IF;
      WHEN state2 =>
         temp <= <value>;
         IF (condition) THEN nx_state <= state3;
         ...
         END IF;
      ...
   END CASE;
END PROCESS;
```

Exemplos de FSMs

Detector de sequência

- Implementar uma FSM que detecte (saída Z = 1) a sequência ”**110**” na entrada A.



Detector de sequência

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

entity detector_sequencia_110 is
    Port ( clk : in  STD_LOGIC;
           reset : in  STD_LOGIC;
           A : in  STD_LOGIC;
           saida : out  STD_LOGIC);
end detector_sequencia_110;

architecture Behavioral of detector_sequencia_110 is

type state is (e0,e1,e2,e3);
signal current_state, next_state : state := e0;

begin
```

Detector de sequência

- **Processo 1**: lógica sequencial do estado atual

```
armazena_estado: process(clk,reset)
begin
    if rising_edge(clk) then
        if reset='1' then
            current_state <= e0;
        else
            current_state <= next_state;
        end if;
    end if;
end process;
```

Detector de sequência

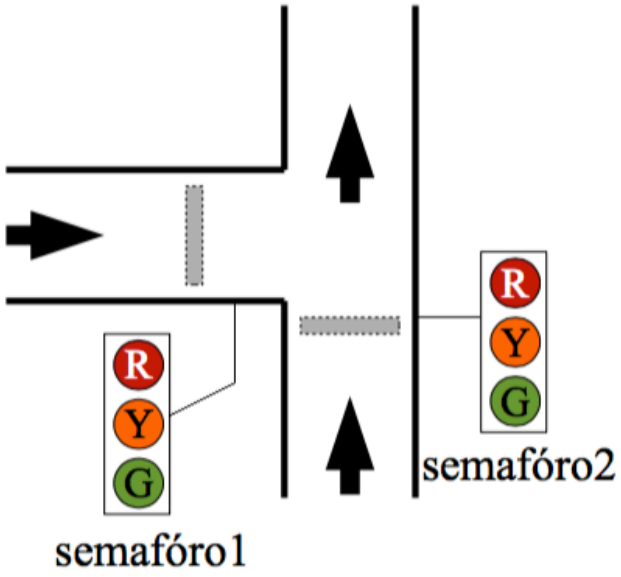
- **Processo 2**: lógica combinacional de próximo estado e saída

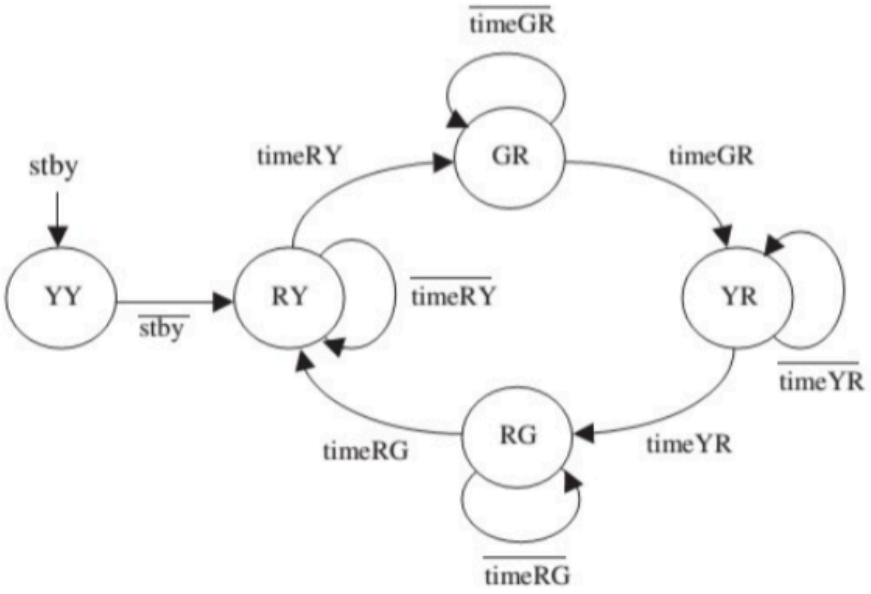
```
transicao_estado: process(current_state,A)
begin
    case current_state is
        when e0 =>
            saida <= '0';
            if A='0' then next_state <= e0;
            else next_state <= e1;
            end if;
        when e1 =>
            saida <= '0';
            if A='0' then next_state <= e0;
            else next_state <= e2;
            end if;
        when e2 =>
            saida <= '0';
            if A='0' then next_state <= e3;
            else next_state <= e2;
            end if;
        when e3 =>
            saida <= '1';
            if A='0' then next_state <= e0;
            else next_state <= e1;
            end if;
    end case;
end process;
```

Controlador de Semáforo

- Especificação:
  - 3 modos de operação:
    - **Regular**: quatro estados, sendo cada um com um timer independente e programável (descrito no circuito como uma constante)
    - **Teste**: permite sobrescrever o valor dos temporizadores de forma a validar o sistema durante a manutenção (1 segundo por estado)
    - **Standby**: neste modo, o sistema ativa as luzes amarelas em ambas as direções enquanto o sinal de standby estiver ativo

Controlador de Semáforo





State	Operation Mode		
	REGULAR	TEST	STANDBY
	Time	Time	Time
RG	timeRG (30s)	timeTEST (1s)	---
RY	timeRY (5s)	timeTEST (1s)	---
GR	timeGR (45s)	timeTEST (1s)	---
YR	timeYR (5s)	timeTEST (1s)	---
YY	---	---	Indefinite

Controlador de Semáforo

```
1  ------------------------------------------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  ------------------------------------------------
5  ENTITY tlc IS
6     PORT ( clk, stby, test: IN STD_LOGIC;
7            r1, r2, y1, y2, g1, g2: OUT STD_LOGIC);
8  END tlc;
9  ------------------------------------------------
10 ARCHITECTURE behavior OF tlc IS
11    CONSTANT timeMAX : INTEGER := 2700;
12    CONSTANT timeRG : INTEGER := 1800;
13    CONSTANT timeRY : INTEGER := 300;
14    CONSTANT timeGR : INTEGER := 2700;
15    CONSTANT timeYR : INTEGER := 300;
16    CONSTANT timeTEST : INTEGER := 60;
17    TYPE state IS (RG, RY, GR, YR, YY);
18    SIGNAL pr_state, nx_state: state;
19    SIGNAL time : INTEGER RANGE 0 TO timeMAX;
20 BEGIN
```

Controlador de Semáforo

```
22    PROCESS (clk, stby)
23       VARIABLE count : INTEGER RANGE 0 TO timeMAX;
24    BEGIN
25       IF (stby='1') THEN
26          pr_state <= YY;
27          count := 0;
28       ELSIF (clk'EVENT AND clk='1') THEN
29          count := count + 1;
30          IF (count = time) THEN
31             pr_state <= nx_state;
32             count := 0;
33          END IF;
34       END IF;
35    END PROCESS;
```

Controlador de Semáforo

```
37    PROCESS (pr_state, test)
38    BEGIN
39       CASE pr_state IS
40          WHEN RG =>
41             r1<='1'; r2<='0'; y1<='0'; y2<='0'; g1<='0'; g2<='1';
42             nx_state <= RY;
43             IF (test='0') THEN time <= timeRG;
44             ELSE time <= timeTEST;
45             END IF;
46          WHEN RY =>
47             r1<='1'; r2<='0'; y1<='0'; y2<='1'; g1<='0'; g2<='0';
48             nx_state <= GR;
49             IF (test='0') THEN time <= timeRY;
50             ELSE time <= timeTEST;
51             END IF;
52          WHEN GR =>
53             r1<='0'; r2<='1'; y1<='0'; y2<='0'; g1<='1'; g2<='0';
54             nx_state <= YR;
55             IF (test='0') THEN time <= timeGR;
56             ELSE time <= timeTEST;
57             END IF;
```

Controlador de Semáforo

```
58          WHEN YR =>
59             r1<='0'; r2<='1'; y1<='1'; y2<='0'; g1<='0'; g2<='0';
60             nx_state <= RG;
61             IF (test='0') THEN time <= timeYR;
62             ELSE time <= timeTEST;
63             END IF;
64          WHEN YY =>
65             r1<='0'; r2<='0'; y1<='1'; y2<='1'; g1<='0'; g2<='0';
66             nx_state <= RY;
67       END CASE;
68    END PROCESS;
69 END behavior;
```