

# Laboratório de Sistemas Digitais

## Aula Teórico-Prática 8

Ano Letivo 2015/16

Modelação, simulação e síntese de  
Máquinas de Estados Finitos

Modelo de Mealy

MEFs comunicantes

Arnaldo Oliveira, Augusto Silva, Iouliia Skliarova, Bernardo Cunha, José L. Azevedo



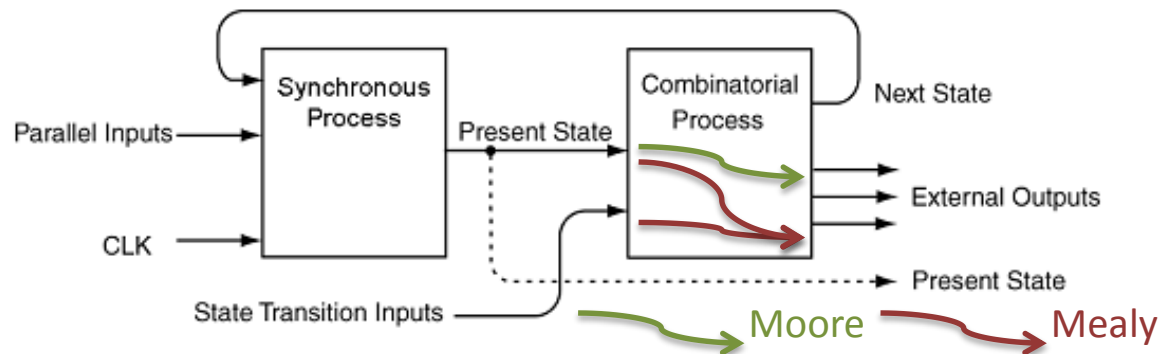
Universidade  
de Aveiro

# Conteúdo

- Abordagens de modelação de Máquinas de Estados Finitos (MEFs) / Finite State Machines (FSMs) baseadas em VHDL
  - Modelo de *Mealy*
- Codificação de estados
  - *One-hot versus* número mínimo de *flip-flops*
  - Observação externa do estado
- Flexibilidade na especificação das transições
  - Utilização de operadores relacionais
- MEFs comunicantes
  - Exemplo

# Máquinas de *Mealy*

- O estilo “2 processos” interdependentes pode facilmente adaptar-se às máquinas de *Mealy*
  - Dependendo da forma como as saídas são atribuídas (no processo combinatório)
    - *Moore* (dependem apenas do estado)
      - Atribuídas diretamente nos **when** de um **case**
    - *Mealy* (dependem do estado e das entradas)
      - Atribuídas em **if...then...else** (dependentes das entradas) dentro dos **when** de um **case**- Cuidado a ter com as saídas
  - Garantir sempre a atribuição (processo combinatório – sem *latches*!!!)
    - Tal como já acontecia no caso de *Moore*!!!



# Exemplo (*Mealy*) – Detetor de Sequência (1101)

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity Seq1101Detector is
    port(reset : in  std_logic;
          clk   : in  std_logic;
          xIn   : in  std_logic;
          zOut  : out std_logic);
end Seq1101Detector;

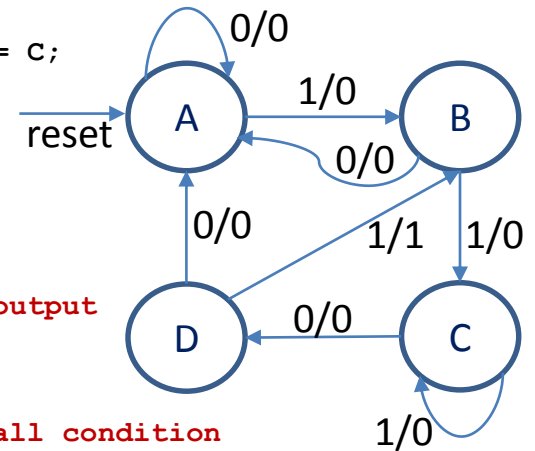
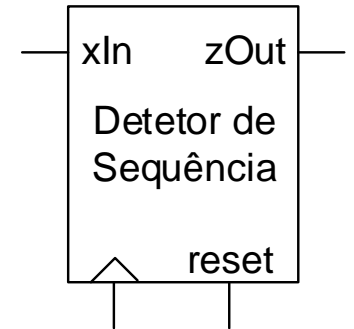
architecture Behav of Seq1101Detector is
    type state is (A, B, C, D);
    signal PS, NS : state;
begin
    sync_proc: process(clk, reset)
    begin
        if (reset = '1') then
            PS <= A;
        elsif (rising_edge(clk)) then
            PS <= NS;
        end if;
    end process;
```

## Exemplo:

```
xIn : 0110100101101101011010111010
zOut: 0000100000001001000010000010
```

```
comb_proc : process(PS, xIn)
begin
    zOut <= '0'; -- Frequent output value, could appear
    -- in all "when" statements, but would require more code

    case PS is
        when A =>
            if (xIn = '1') then NS <= B;
            else NS <= A;
            end if;
        when B =>
            if (xIn = '1') then NS <= C;
            else NS <= A;
            end if;
        when C =>
            if (xIn = '1') then NS <= C;
            else NS <= D;
            end if;
        when D =>
            if (xIn = '1') then
                NS <= B;
                zOut <= '1'; -- Mealy output
            else NS <= A;
            end if;
        when others => -- Catch all condition
            NS <= A;
    end case;
end process;
end Behav;
```

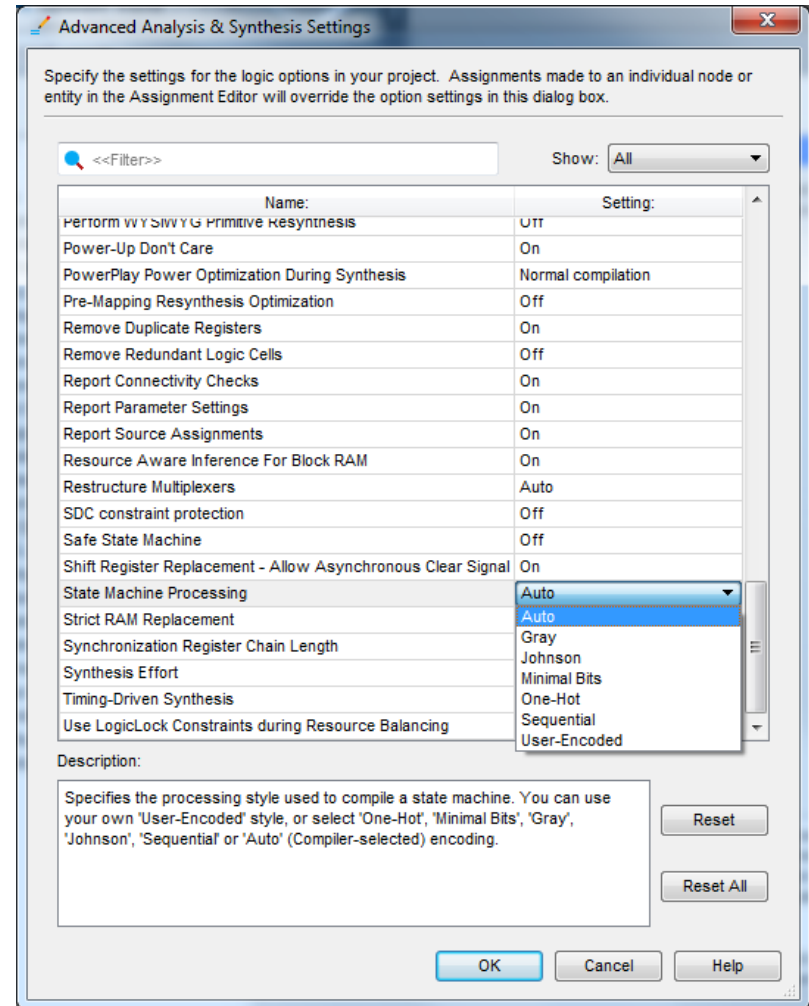


# Codificação dos Estados

- Em VHDL o recurso a um tipo enumerado para codificar simbolicamente os estados
  - Permite uma descrição de alto-nível muito próxima do diagrama de estados
  - Confere ao sintetizador a tarefa da codificação binária dos estados
  - O hardware sintetizado inclui frequente mais *flip-flops* que o nº mínimo (quando a codificação é binária ou *gray*)
    - Codificação frequente – “*One-hot*”
      - Exemplo: 4 estados => 4 flip-flops (000**1**, 00**1**0, 0**1**00, **1**000)
      - Adequada à implementação em FPGA (elevado número de *flip-flops* disponíveis mas LUTs com poucas entradas - tipicamente 4 a 6)
        - » Lógica de estado seguinte e de saída tendencialmente mais simples (com menos níveis e mais rápida)

# Codificação dos Estados

- No “Altera Quartus” é também possível “forçar” a técnica de codificação de estado em “Assignments → Settings → Compiler Settings → Advanced Settings (Synthesis)”
- Seguimento / observação externa dos estados
  - Frequentemente com saídas adicionais



# Output do Estado Interno

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity Seq1101 is
  port(reset : in std_logic;
        clk   : in std_logic;
        xIn   : in std_logic;
        stout : out std_logic_vector(3 downto 0);
        zOut  : out std_logic);
end Seq1101;
```

```
architecture Behav of Seq1101 is
  type state is (A, B, C, D);
  signal PS, NS : state;
begin
  sync_proc: process(clk, reset)
  begin
    if (reset = '1') then
      PS <= A;
    elsif (rising_edge(clk)) then
      PS <= NS;
    end if;
  end process;
```

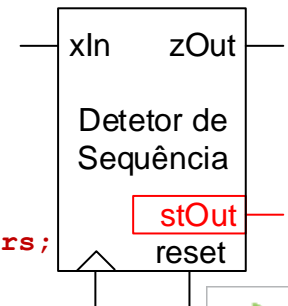
```
  comb_proc : process(PS, xIn)
  begin
    zOut <= '0'; -- Most frequent output value
```

```
  case PS is
    when A =>
      if (xIn = '1') then NS <= B;
      else NS <= A;
      end if;
      . . .
      . . .
    when D =>
      if (xIn = '1') then
        NS <= B;
        zOut <= '1'; -- Mealy output
      else NS <= A;
      end if;
    when others => -- Catch all condition
      NS <= A;
  end case;
end process;
```

```
  with PS select
    stOut <= "0001" when A,
            "0010" when B,
            "0100" when C,
            "1000" when D,
            "0000" when others;
```

```
end Behav;
```

Atribuição concorrente  
com `sync_proc` e  
`comb_proc`



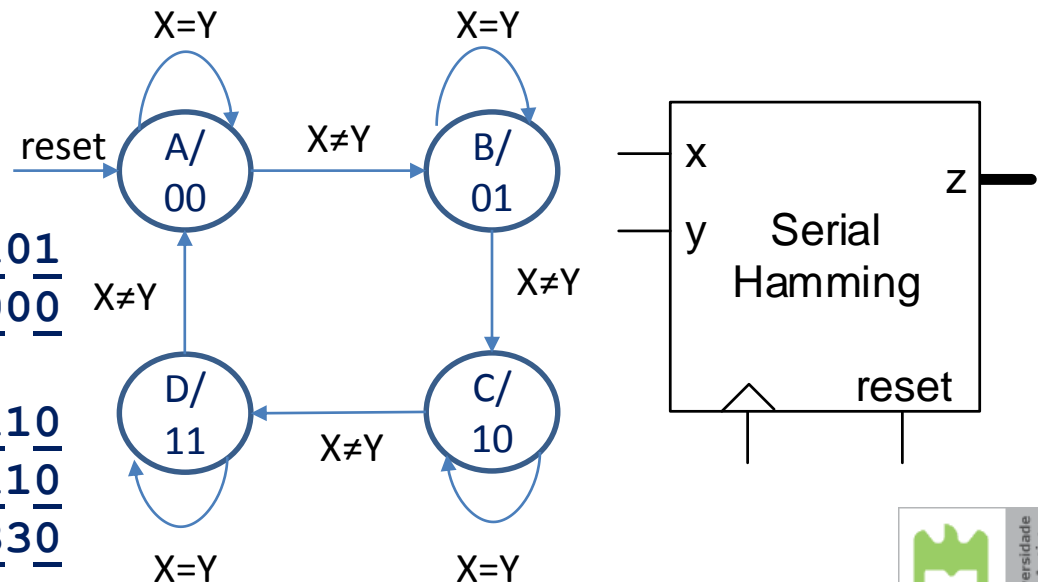
# Especificação Flexível das Transições

- Em VHDL é possível especificar as condições de transição de estado de diversas formas
  - Códigos binários ou valores simbólicos
  - Expressões booleanas ou relacionais
- Exemplo
  - Distância de *Hamming* módulo 4 de duas entradas série

## Exemplo:

**x:**     0001100110110110111101  
**y:**     0011000100110100100000

**z (0):** 00110000111111100010110  
**z (1):** 00001111111111100001110  
**z**     : 0011222233333300012330





# Exemplo – Distância de *Hamming*

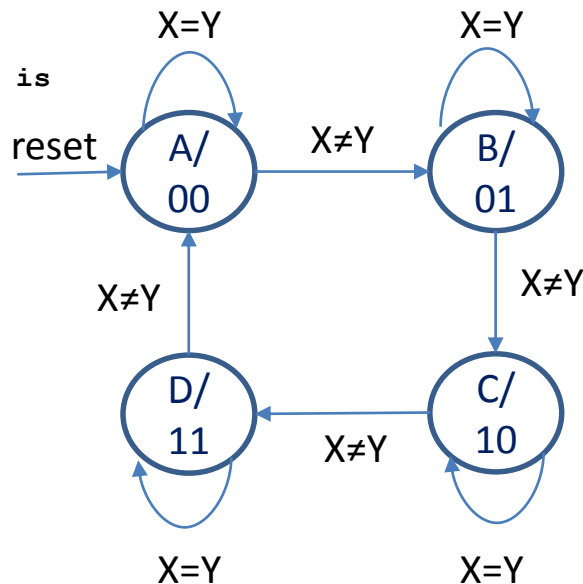
## Módulo 4

```
-- Computes modulo 4 (0...3) Hamming distance
-- between to serial inputs

library IEEE;
use IEEE.std_logic_1164.all;

entity serialHamming is
    port(clk      : in std_logic;
         reset    : in std_logic;
         x,y      : in std_logic;
         z        : out std_logic_vector(1 downto 0));
end serialHamming;
```

```
architecture behav of serialHamming is
    type state is (A, B, C, D);
    signal PS, NS : state;
begin
    sync_proc : process(clk, reset)
    begin
        if (reset = '1') then
            PS <= A;
        elsif (rising_edge(clk)) then
            PS <= NS;
        end if;
    end process;
```



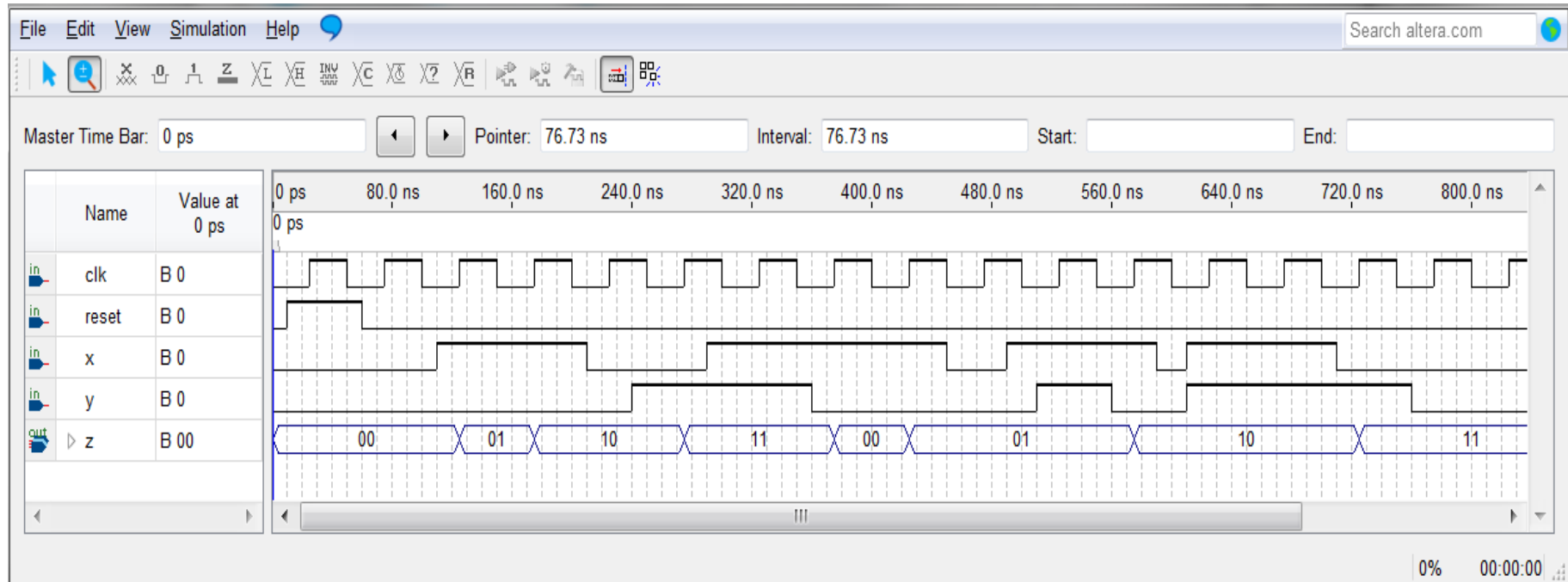
```
comb_proc : process(PS, x, y)
begin
    NS <= PS; -- Otherwise assume
              -- NS equal to PS

    case PS is
        when A =>
            z <= "00";
            if (x /= y) then NS <= B;
            end if;
        when B =>
            z <= "01";
            if (x /= y) then NS <= C;
            end if;
        when C =>
            z <= "10";
            if (x /= y) then NS <= D;
            end if;
        when D =>
            z <= "11";
            if (x /= y) then NS <= A;
            end if;
        when others =>
            z <= "00";
            NS <= A;
        end case;
    end process;
end behav;
```



# Simulação - Distância de *Hamming*

## Módulo 4

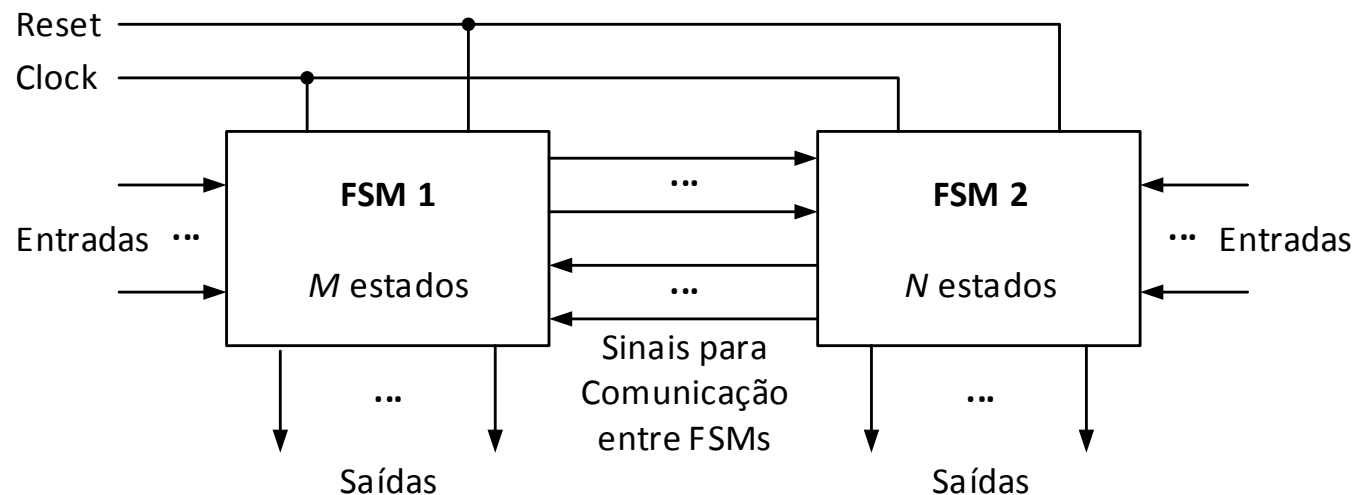


Simulação com *testbench*:

- Gerada a partir do ficheiro VWF
- Gerada diretamente em VHDL de acordo com o *template* para componentes sequenciais

# Máquinas de Estado Finitos Comunicantes

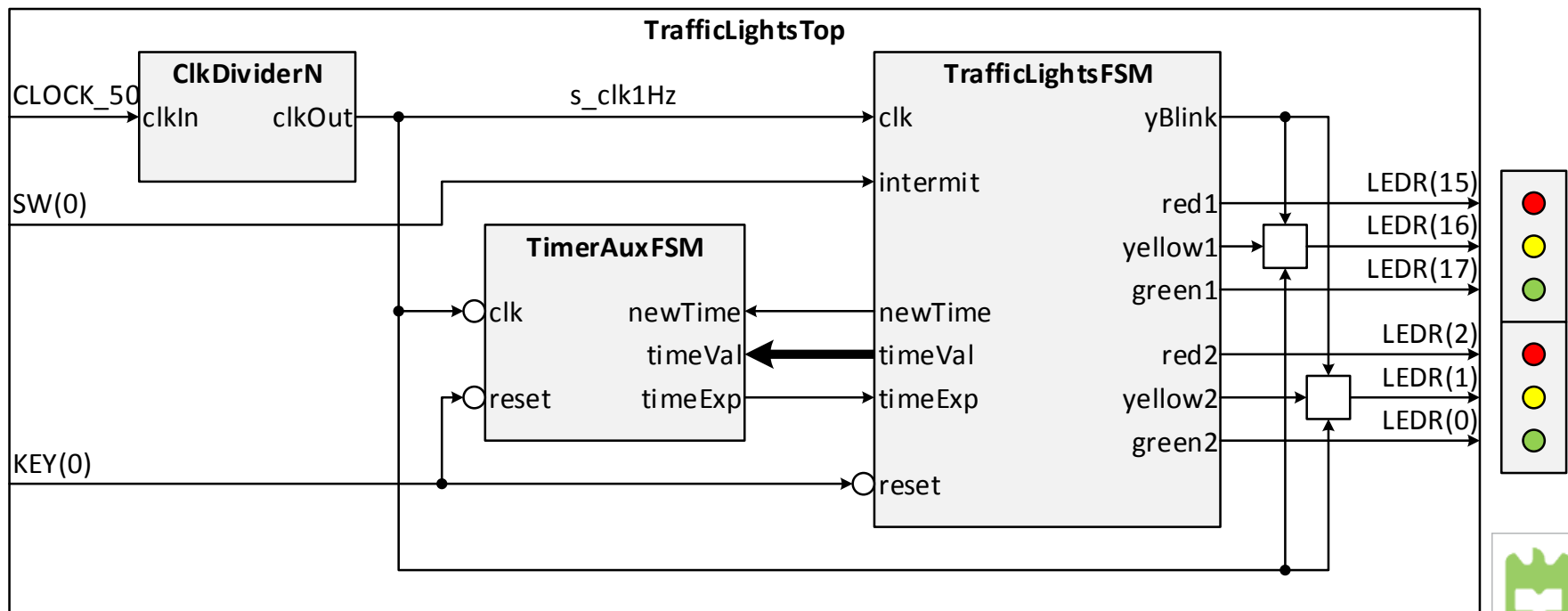
- Partição da funcionalidade do sistema em duas ou mais máquinas paralelas ou sequenciais
  - Decomposição visa facilitar o desenvolvimento e a validação
  - Partilha dos sinais de inicialização e sincronização
    - Frequentemente operam em diferentes flancos do mesmo sinal de *clock*
  - Sinais de entrada e de saída
    - Específicos de cada sub-máquina
    - Partilhados pelas sub-máquinas
    - Comunicação entre sub-máquinas



# Exemplo de MEFs Comunicantes

## Semáforo de Obras

- Duas MEFs comunicantes
  - Sequenciação dos estados dos semáforos - *TrafficLightsFSM*
  - Temporização dos estados dos semáforos - *TimerAuxFSM*
  - Frequência de operação - 1 Hz



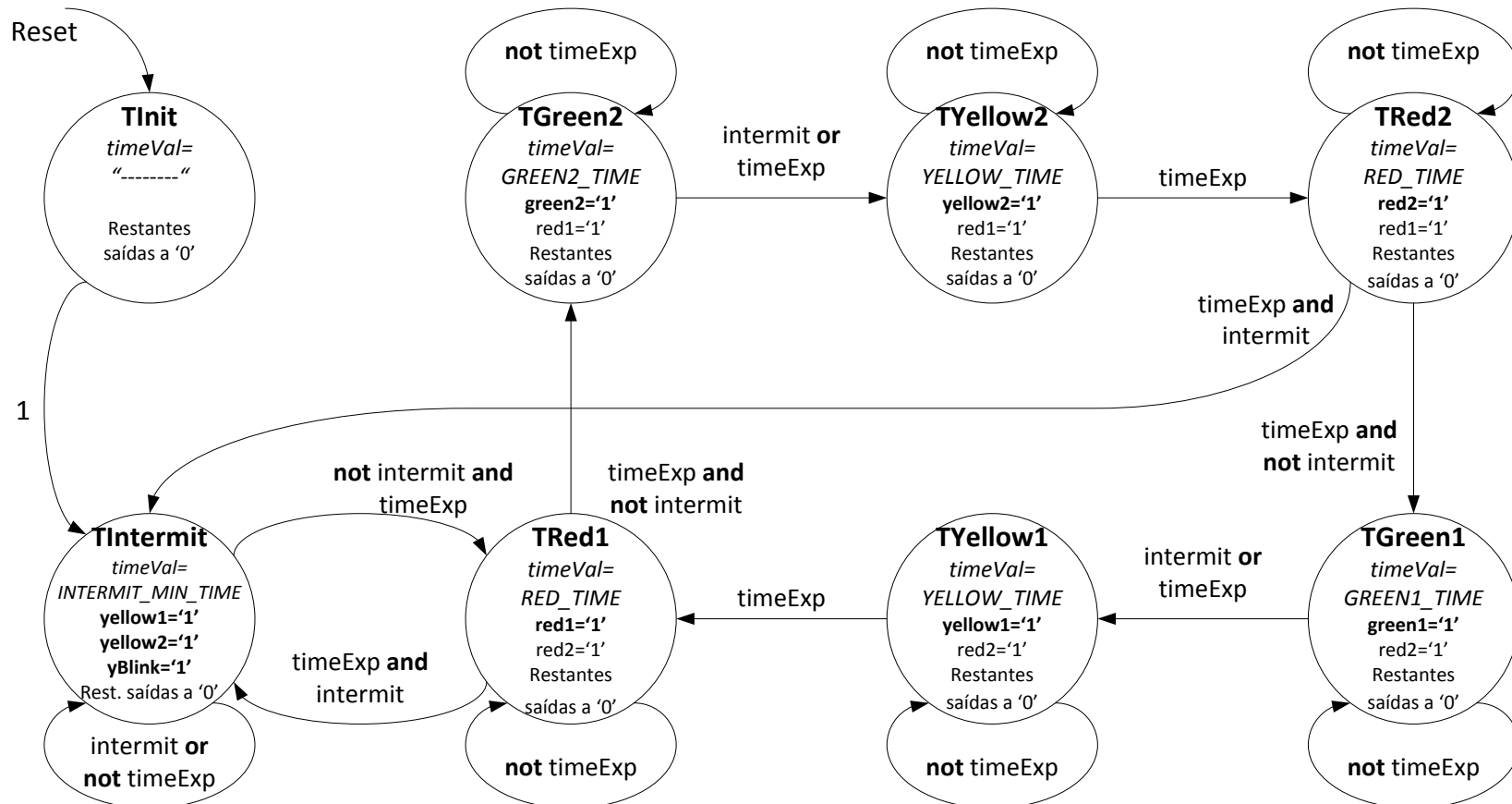
# Controlador do Semáforo

## *TrafficLightsFSM*

### Alguns portos

- **intermit** – entrada de ativação do amarelo intermitente
- **newTime, timeVal** – pedido (à *TimerAuxFSM*) de uma nova temporização
- **timeExp** – temporização terminada (expirada) (notificação da *TrafficLightsFSM*)
- **yBlink** – saída de controlo do amarelo intermitente

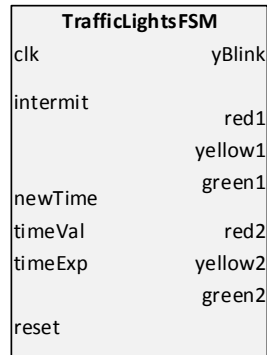
TrafficLightsFSM	
clk	yBlink
intermit	red1
	yellow1
	green1
newTime	red2
timeVal	yellow2
timeExp	green2
reset	



# TrafficLightsFSM – Ent. + Arquitetura (decl. do estado e proc. síncrono)

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity TrafficLightsFSM is
    port(reset      : in  std_logic;
         clk        : in  std_logic;
         intermit   : in  std_logic;
         newTime    : out std_logic;
         timeVal    : out std_logic_vector(7 downto 0);
         timeExp    : in  std_logic;
         yBlink     : out std_logic;
         red1       : out std_logic;
         yellow1    : out std_logic;
         green1     : out std_logic;
         red2       : out std_logic;
         yellow2    : out std_logic;
         green2     : out std_logic);
end TrafficLightsFSM;
```



architecture Behavioral of TrafficLightsFSM is

-- Constantes para definição das temporizações

```
constant RED_TIME      : std_logic_vector(7 downto 0) := "00000100"; -- 4 s
constant YELLOW_TIME   : std_logic_vector(7 downto 0) := "00000011"; -- 3 s
constant GREEN1_TIME   : std_logic_vector(7 downto 0) := "00001110"; -- 14 s
constant GREEN2_TIME   : std_logic_vector(7 downto 0) := "00001100"; -- 10 s
constant INTERMIT_MIN_TIME : std_logic_vector(7 downto 0) := "00000110"; -- 6 s
```

```
type TState is (TInit, TIntermit, TRed1, TYellow1,
               TGreen1, TRed2, TYellow2, TGreen2);
signal s_currentState, s_nextState : Tstate := TInit;
signal s_stateChanged : std_logic := '1';

begin
    sync_proc : process(clk)
    begin
        if (rising_edge(clk)) then
            if (reset = '1') then
                s_currentState <= TInit;
                s_stateChanged <= '1';
            else
                if (s_currentState /= s_nextState) then
                    s_stateChanged <= '1';
                else
                    s_stateChanged <= '0';
                end if;
                s_currentState <= s_nextState;
            end if;
        end if;
    end process;
    newTime <= s_stateChanged;
```

valores de  
inicialização pós  
configuração da  
FPGA

newTime é  
ativado quando  
existe de facto  
uma transição  
de estado



# TrafficLightsFSM – Proc. Combinatório

```
comb_proc : process(s_currentState,
                    intermit, timeExp)
```

```
begin
```

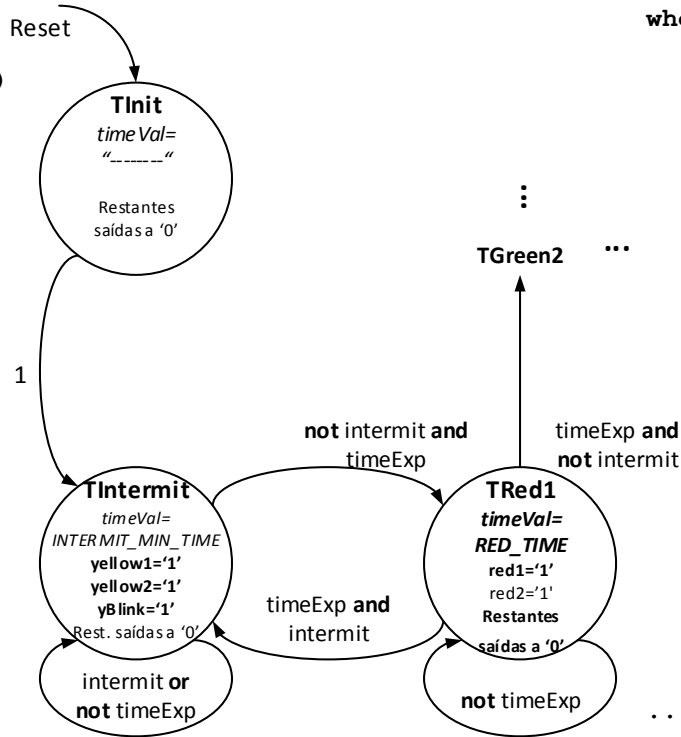
```
  case (s_currentState) is
```

```
    when TInit =>
```

```
      red1      <= '0';
      yellow1   <= '0';
      green1    <= '0';
      red2      <= '0';
      yellow2   <= '0';
      green2    <= '0';
      yBlink    <= '0';
      timeVal   <= (others => '-');
      s_nextState <= TIntermit;
```

```
    when TIntermit =>
```

```
      red1      <= '0';
      yellow1   <= '1';
      green1    <= '0';
      red2      <= '0';
      yellow2   <= '1';
      green2    <= '0';
      yBlink    <= '1';
      timeVal   <= INTERMIT_MIN_TIME; -- Indicação da temporização
      if ((intermit = '0') and (timeExp = '1')) then
        s_nextState <= TRed1;
      else
        s_nextState <= TIntermit;
      end if;
```



```
  when TRed1 =>
```

```
    red1      <= '1';
    yellow1   <= '0';
    green1    <= '0';
    red2      <= '1';
    yellow2   <= '0';
    green2    <= '0';
    yBlink    <= '0';
    timeVal   <= RED_TIME;
    if (timeExp = '1') then
      if (intermit = '1') then
        s_nextState <= TIntermit;
      else
        s_nextState <= TGreen2;
      end if;
    else
      s_nextState <= TRed1;
    end if;
```

```
...
when TYellow1 =>
...
when TGreen1 =>
...
end case;
end process;
end Behavioral;
```

TrafficLightsFSM	
clk	yBlink
intermit	red1
	yellow1
	green1
newTime	red2
timeVal	yellow2
timeExp	green2
reset	



# TimerAuxFSM – Código Completo

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.NUMERIC_STD.all;
```

```
entity TimerAuxFSM is
```

```
    port(reset      : in  std_logic;
          clk       : in  std_logic;
          newTime   : in  std_logic;
          timeVal   : in  std_logic_vector(7 downto 0);
          timeExp   : out std_logic);
```

```
end TimerAuxFSM;
```

```
architecture Behavioral of TimerAuxFSM is
```

```
    signal s_counter : unsigned(7 downto 0) := (others => '1');
    signal s_cntZero  : std_logic := '0';
```

```
begin
```

```
    process(clk)
```

```
    begin
```

```
        if (rising_edge(clk)) then
```

```
            if (reset = '1') then
```

```
                s_counter <= (others => '1');
```

```
                s_cntZero <= '0';
```

```
            elsif (newTime = '1') then
```

```
                s_counter <= unsigned(timeVal) - 1;
```

```
                s_cntZero <= '0';
```

```
            else
```

```
                if (s_counter = "00000000") then
```

```
                    s_cntZero <= '1';
```

```
                else
```

```
                    s_counter <= s_counter - 1;
```

```
                    s_cntZero <= '0';
```

```
                end if;
```

```
            end if;
```

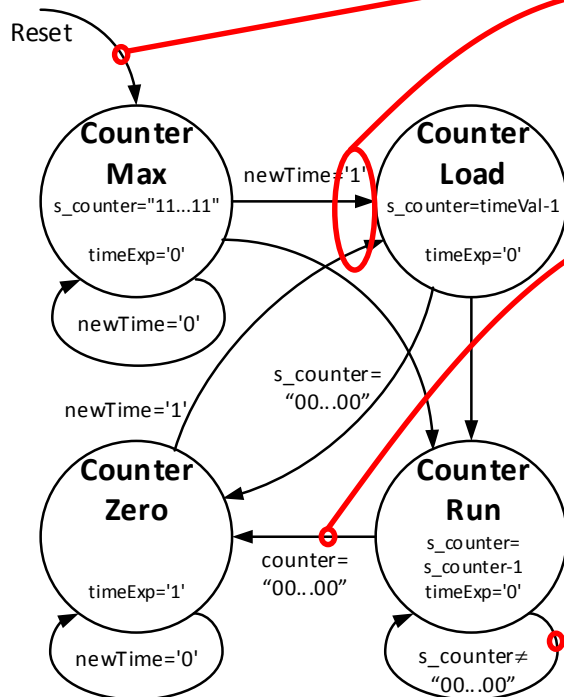
```
        end if;
```

```
    end process;
```

```
    timeExp <= s_cntZero;
```

```
end Behavioral;
```

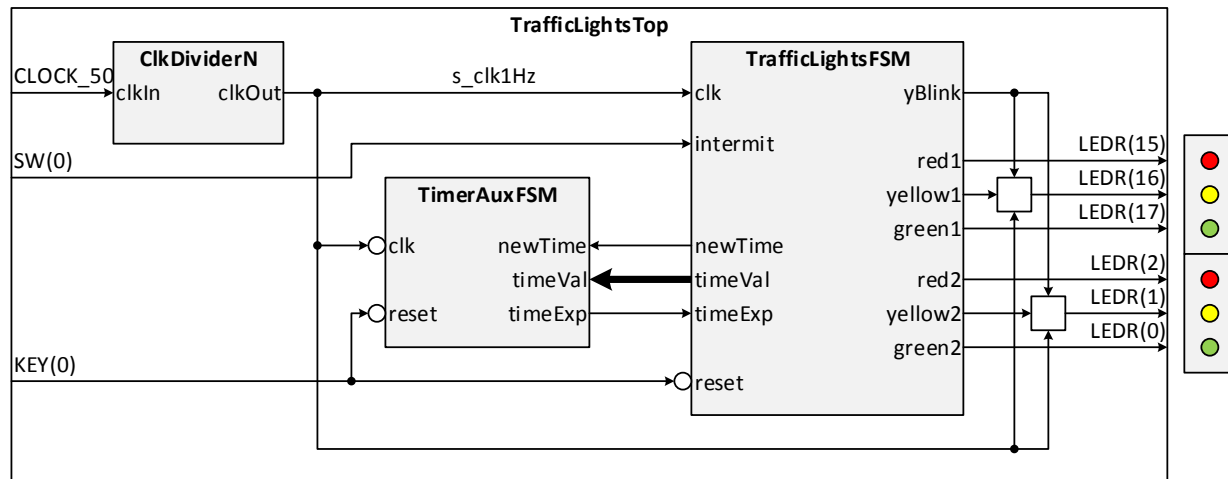
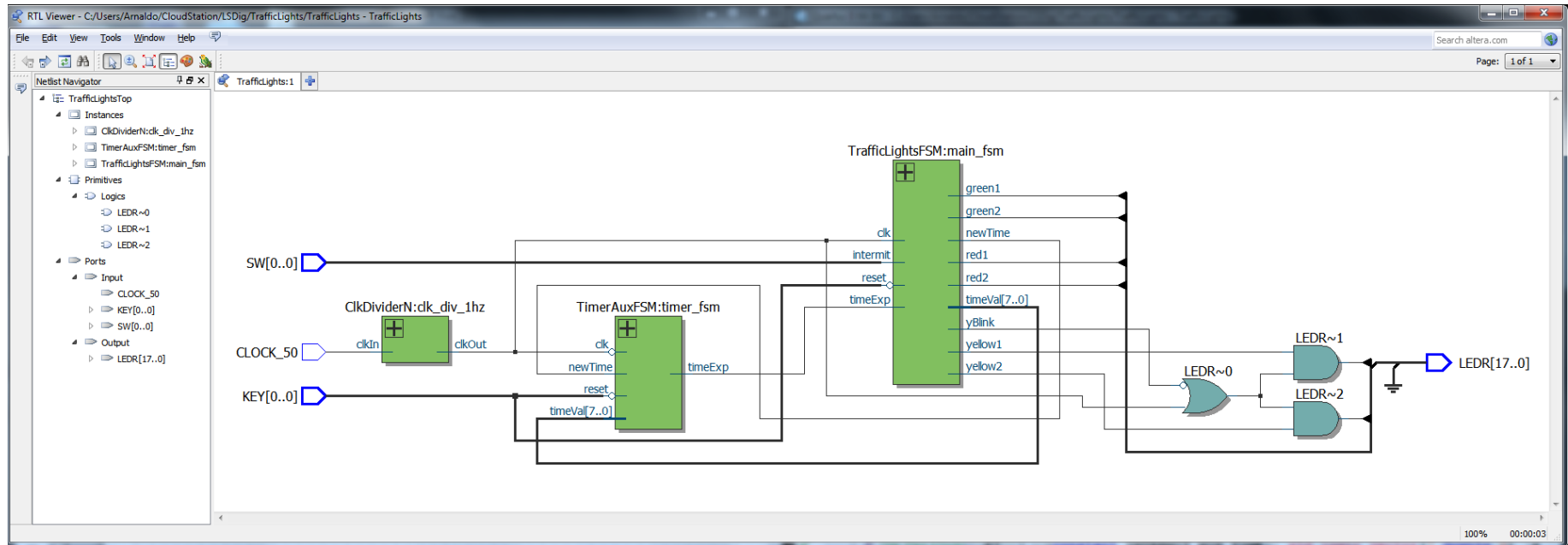
MEF modelada  
com um único  
processo e  
baseada num  
contador



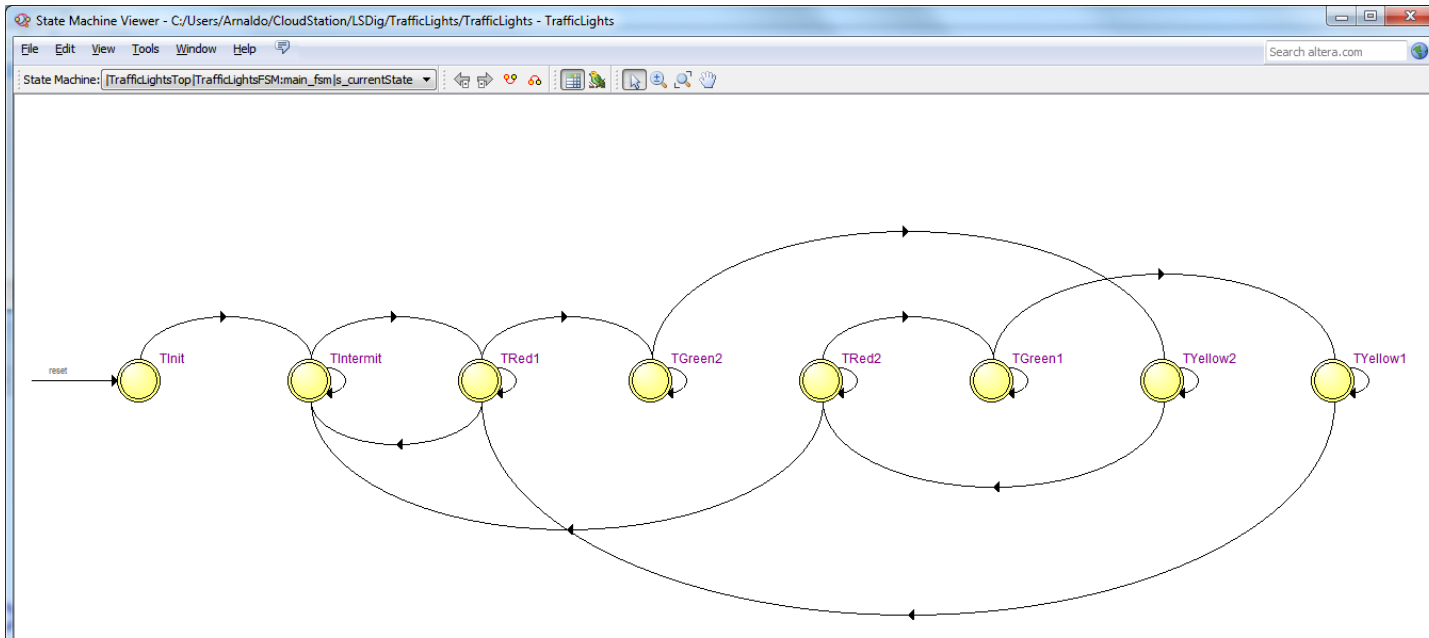
Contador decrescente de  
**timeVal-1**  
a 0 (zero)  
**timeExp** toma o valor  
'1' quando o contador  
atinge o valor zero



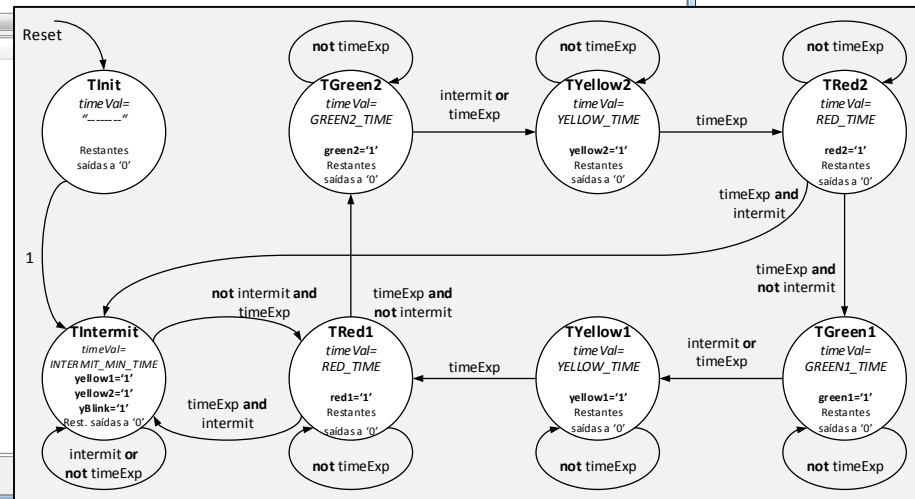
# Quartus Prime RTL Netlist Viewer



# Quartus Prime State Machine Viewer



	Source State	Destination State	Condition
1	TGreen1	TYellow1	(!intermit).(timeExp).(reset) + (!intermit).(reset)
2	TGreen1	TGreen1	(!intermit).(timeExp).(reset)
3	TGreen2	TGreen2	(!intermit).(timeExp).(reset)
4	TGreen2	TYellow2	(!intermit).(timeExp).(reset) + (!intermit).(reset)
5	TInit	TIntermit	(reset)
6	TIntermit	TRed1	(timeExp).(intermit).(reset)
7	TIntermit	TIntermit	(!intermit).(timeExp).(reset) + (!intermit).(reset)
8	TRed1	TRed1	(timeExp).(reset)
9	TRed1	TGreen2	(!intermit).(timeExp).(reset)
10	TRed1	TIntermit	(intermit).(timeExp).(reset)
11	TRed2	TRed2	(timeExp).(reset)
12	TRed2	TIntermit	(intermit).(timeExp).(reset)
13	TRed2	TGreen1	(!intermit).(timeExp).(reset)
14	TYellow1	TYellow1	(timeExp).(reset)
15	TYellow1	TRed1	(intermit).(timeExp).(reset)
16	TYellow2	TRed2	(timeExp).(reset)
17	TYellow2	TYellow2	(timeExp).(reset)



# Comentários Finais

- No final desta aula e do trabalho prático 9 de LSDig, deverá ser capaz de:
  - Conhecer os passos necessários à síntese de máquinas de estados finitos
  - Recorrer a descrições comportamentais VHDL próximas do diagrama de estados saídas
    - Modelo de *Mealy*
    - Modelo de *Moore*
  - Conceber *testbenches* para a simulação funcional das MEF
  - Desenvolver sistemas baseados em máquinas de estados finitos comunicantes
- ... bom trabalho prático 9, disponível no site da UC
  - [elearning.ua.pt](http://elearning.ua.pt)