



**Universidade Federal de Santa Catarina  
Centro Tecnológico – CTC  
Departamento de Engenharia Elétrica**



***<http://gse.ufsc.br>***

# **“Finite-State Machine in VHDL”**

**Prof. Eduardo Augusto Bezerra**

**Eduardo.Bezerra@ufsc.br**

**Florianópolis, August 2020.**

# Learning goals

---

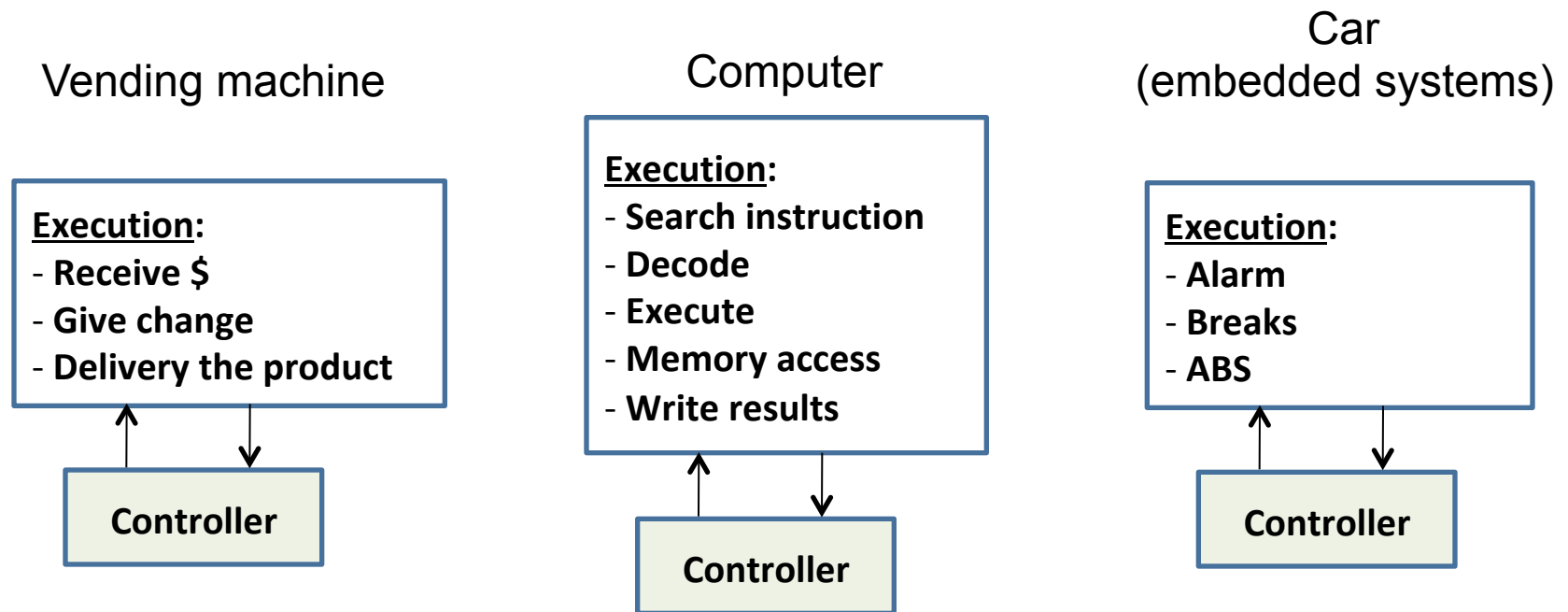
1. To review the Finite-State Machine (FSM) concept.
2. To review the concept of a sequential circuit controlling (driving) a combinational circuit.
3. To understand how to describe an FSM in VHDL.
4. To review the design of counters in VHDL.
5. Case study: design and implementation of a counter in VHDL, using an FSM.

# Overview

# Finite-State Machine (FSM)

---

- Computational systems are usually composed of a control module and a “operations execution” module (data path).



# ***Finite-State Machine (FSM)***

---

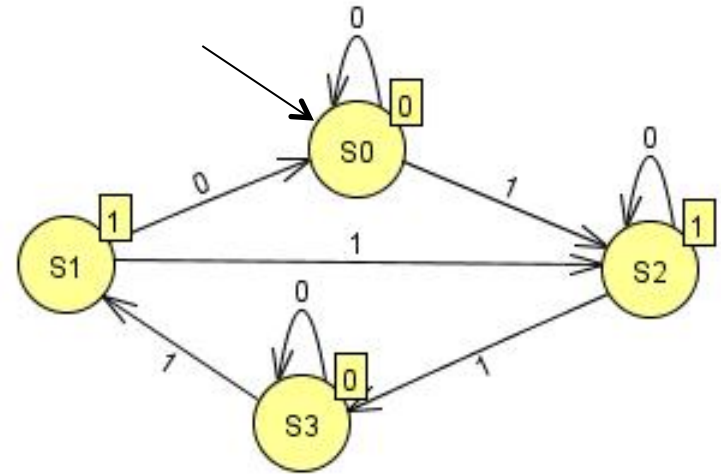
- The “controller” is responsible for coordinating the sequence of activities to be performed in a given process (or system)
- In digital systems, “sequential circuits” are used to generate control signals
- A sequential circuit goes through a series of states and, at each state (at each moment), it can provide a certain output
- Outputs are used to control the execution of activities in a process
- The sequential logic used in the implementation of an FSM has a “finite” number of states.

# Finite-State Machine (FSM)

---

Behavior model composed of:

- States
- Transitions
- Actions



## States

Stores information about the past, taking in account the changes in the inputs from the beginning until the present

## Transition

Indicates a state transition, and it is described by a condition that enables the state modification

## Action

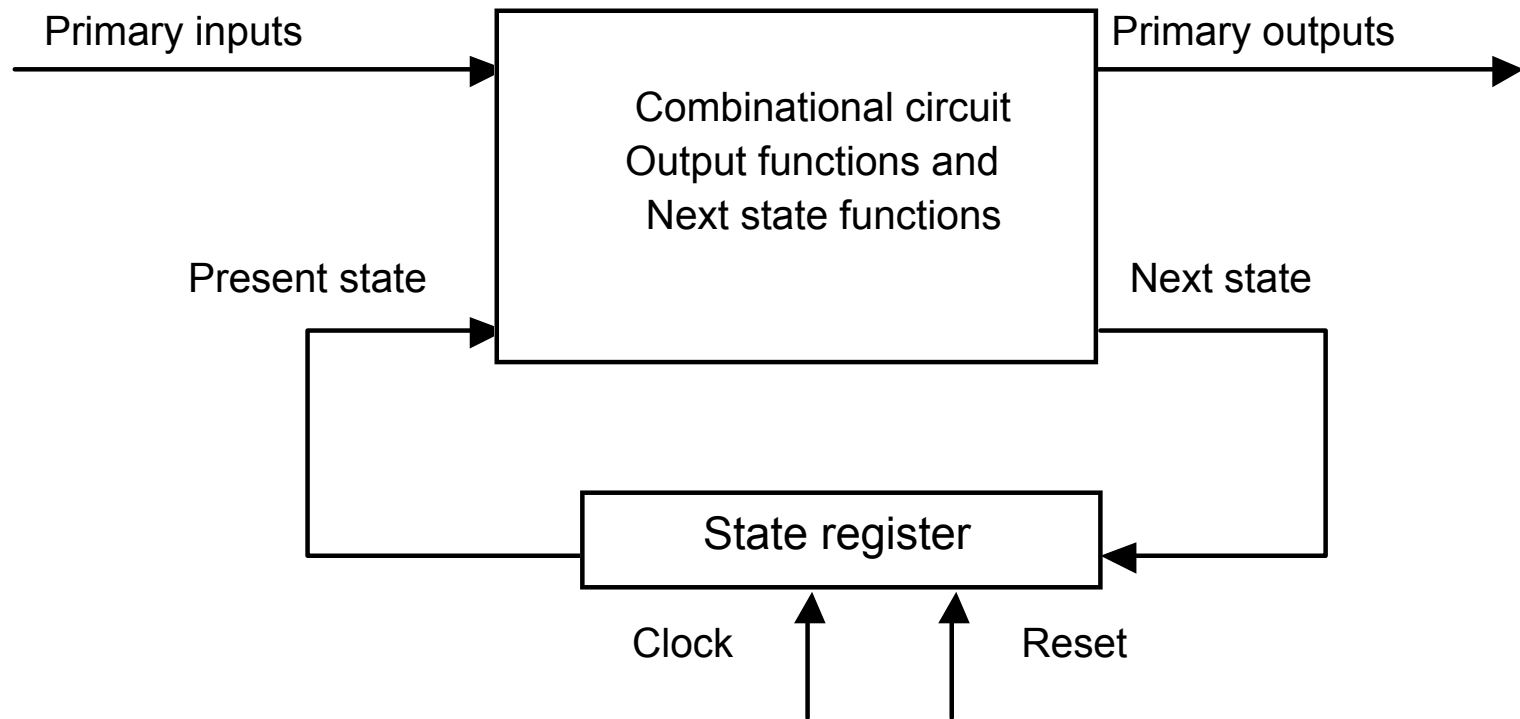
Description of the activity to be performed at a given time

# **FSM structure**

# FSM structure

---

- Two modules:
  - “Present state” storage; and
  - “Outputs” and “Next state” definition





# FSM structure

---

- “Present state” register
  - Can be a flip-flops based register
- “Output” and “Next state” definition
  - Combinacional circuit; or
  - A truth table of “Output” and “Next state” logics stored in a memory (ROM, Flash, RAM, ...)

# FSM synthesis

# Describing an FSM in VHDL

## A **2 processes** FSM description in VHDL

```
entity MOORE is port(X, clock, reset : in std_logic; Z: out std_logic); end;
```

```
architecture A of MOORE is
```

```
  type STATES is (S0, S1, S2, S3);
```

```
  signal CS, NS : STATES;
```

```
begin
```

```
  process (clock, reset)
```

```
  begin
```

```
    if reset= '1' then
```

```
      CS <= S0;
```

```
    elsif clock'event and clock='1' then
```

```
      CS <= NS ;
```

```
    end if;
```

```
  end process;
```

```
  process(CS, X)
```

```
  begin
```

```
    case CS is
```

```
      when S0 =>
```

```
        Z <= '0';
```

```
        if X='0' then NS <=S0; else NS <= S2; end if;
```

```
      when S1 =>
```

```
        Z <= '1';
```

```
        if X='0' then NS <=S0; else NS <= S2; end if;
```

```
      when S2 =>
```

```
        Z <= '1';
```

```
        if X='0' then NS <=S2; else NS <= S3; end if;
```

```
      when S3 =>
```

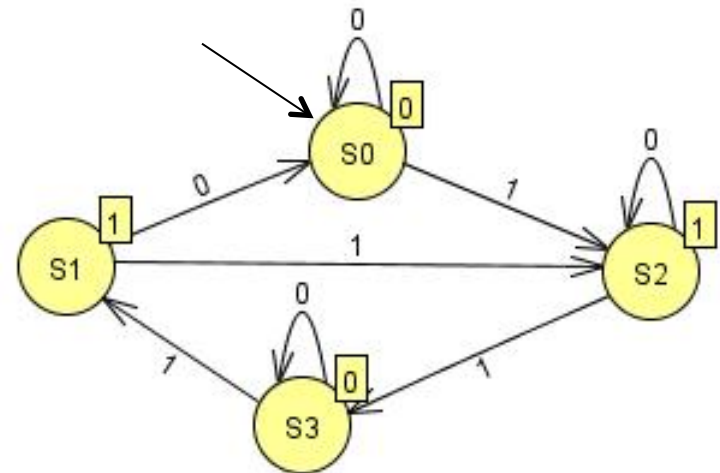
```
        Z <= '0';
```

```
        if X='0' then NS <=S3; else NS <= S1; end if;
```

```
    end case;
```

```
  end process;
```

```
end A;
```



# Describing an FSM in VHDL

## A **2 processes** FSM description in VHDL

```

entity MOORE is  port(X, clock, reset : in std_logic;    Z: out std_logic);  end;

architecture A of MOORE is
  type STATES is (S0, S1, S2, S3);
  signal CS, NS : STATES;
begin
  process (clock, reset)
  begin
    if reset= '1' then
      CS <= S0;
    elsif clock'event and clock='1' then
      CS <= NS ;
    end if;
  end process;

  process(CS, X)
  begin
    case CS is
      when S0 =>    Z <= '0';
                    if X='0' then NS <=S0; else NS <= S2; end if;
      when S1 =>    Z <= '1';
                    if X='0' then NS <=S0; else NS <= S2; end if;
      when S2 =>    Z <= '1';
                    if X='0' then NS <=S2; else NS <= S3; end if;
      when S3 =>    Z <= '0';
                    if X='0' then NS <=S3; else NS <= S1; end if;
    end case;
  end process;
end A;

```

**ENUM TYPE**  
**CS** (current state) and **NS** (next state) signals

# Describing an FSM in VHDL

## A **2 processes** FSM description in VHDL

```
entity MOORE is  port(X, clock, reset : in std_logic;    Z: out std_logic);  end;
```

```
architecture A of MOORE is
```

```
  type STATES is (S0, S1, S2, S3);
```

```
  signal CS, NS : STATES;
```

```
begin
```

```
  process (clock, reset)
```

```
  begin
```

```
    if reset= '1' then
```

```
      CS <= S0;
```

```
    elsif clock'event and clock='1' then
```

```
      CS <= NS ;
```

```
    end if;
```

```
  end process;
```

Register to hold current state (CS),  
which is a function of the next state (NS)

```
  process(CS, X)
```

```
  begin
```

```
    case CS is
```

```
      when S0 =>
```

```
        Z <= '0';
```

```
        if X='0' then NS <=S0; else NS <= S2; end if;
```

```
      when S1 =>
```

```
        Z <= '1';
```

```
        if X='0' then NS <=S0; else NS <= S2; end if;
```

```
      when S2 =>
```

```
        Z <= '1';
```

```
        if X='0' then NS <=S2; else NS <= S3; end if;
```

```
      when S3 =>
```

```
        Z <= '0';
```

```
        if X='0' then NS <=S3; else NS <= S1; end if;
```

```
    end case;
```

```
  end process;
```

```
end A;
```

# Describing an FSM in VHDL

## A **2 processes** FSM description in VHDL

```
entity MOORE is  port(X, clock, reset : in std_logic;  Z: out std_logic);  end;
```

```
architecture A of MOORE is
```

```
  type STATES is (S0, S1, S2, S3);
```

```
  signal CS, NS : STATES;
```

```
begin
```

```
  process (clock, reset)
```

```
  begin
```

```
    if reset= '1' then
```

```
      CS <= S0;
```

```
    elsif clock'event and clock='1' then
```

```
      CS <= NS ;
```

```
    end if;
```

```
  end process;
```

```
  process(CS, X)
```

```
  begin
```

```
    case CS is
```

```
      when S0 =>
```

```
        Z <= '0';
```

```
        if X='0' then NS <=S0; else NS <= S2; end if;
```

```
      when S1 =>
```

```
        Z <= '1';
```

```
        if X='0' then NS <=S0; else NS <= S2; end if;
```

```
      when S2 =>
```

```
        Z <= '1';
```

```
        if X='0' then NS <=S2; else NS <= S3; end if;
```

```
      when S3 =>
```

```
        Z <= '0';
```

```
        if X='0' then NS <=S3; else NS <= S1; end if;
```

```
    end case;
```

```
  end process;
```

```
end A;
```

**NS and Z output generation according  
To CS and X input  
(see the sensitivity list)**

# Describing an FSM in VHDL

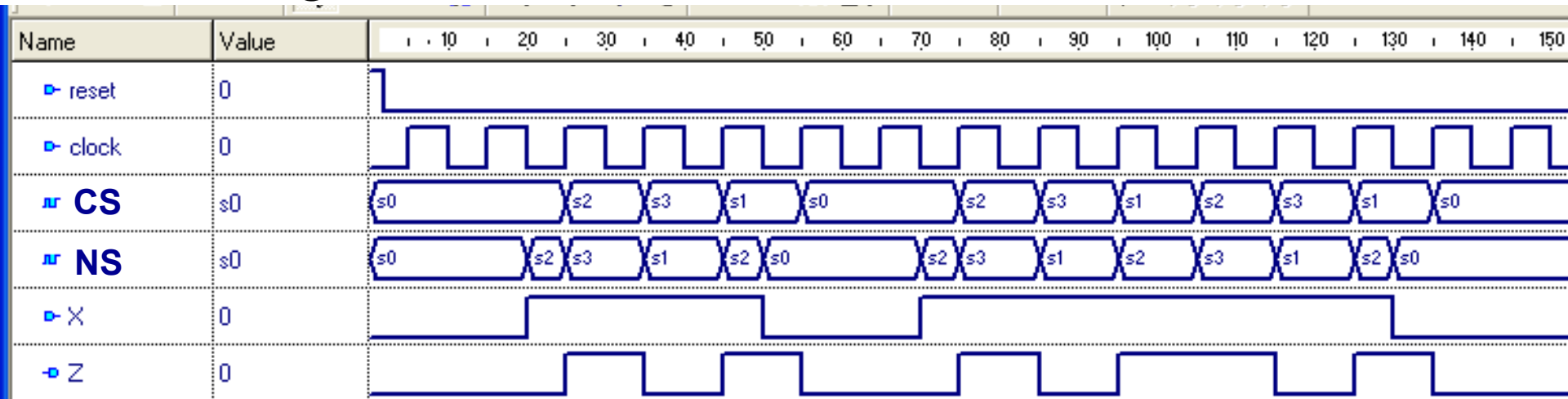
## A **2 processes** FSM description in VHDL

**Draw the FSM according to the transitions in the combinational process:**

```
process(CS, X)  
begin  
  case CS is  
    when S0 =>    Z <= '0';  
                  if X='0' then NS <=S0; else NS <= S2; end if;  
    when S1 =>    Z <= '1';  
                  if X='0' then NS <=S0; else NS <= S2; end if;  
    when S2 =>    Z <= '1';  
                  if X='0' then NS <=S2; else NS <= S3; end if;  
    when S3 =>    Z <= '0';  
                  if X='0' then NS <=S3; else NS <= S1; end if;  
  end case;  
end process;
```

**This is a Moore machine. The output (Z) is determined only by its current state (S0, ...).  
In a Mealy machine, the output values are determined both by its current state and by the values of its inputs.**

# Describing an FSM in VHDL



```

process(CS, X)
begin
    case CS is
        when S0 =>    Z <= '0';
                     if X='0' then NS <=S0; else NS <= S2; end if;
        when S1 =>    Z <= '1';
                     if X='0' then NS <=S0; else NS <= S2; end if;
        when S2 =>    Z <= '1';
                     if X='0' then NS <=S2; else NS <= S3; end if;
        when S3 =>    Z <= '0';
                     if X='0' then NS <=S3; else NS <= S1; end if;
    end case;
end process;

```



# Describing an FSM in VHDL

## A **1 process** FSM description in VHDL

```
entity MOORE is  port(X, clock, reset : in std_logic;    Z: out std_logic);  end;

architecture B of MOORE is
  type STATES is (S0, S1, S2, S3);
  signal CS: STATES;
begin
  process(clock, reset)
  begin
    if reset= '1' then
      CS <= S0;
    elsif clock'event and clock='1' then
      case CS is
        when S0 => Z <= '0';
                     if X='0' then CS <=S0; else CS <= S2; end if;
        when S1 => Z <= '1';
                     if X='0' then CS <=S0; else CS <= S2; end if;
        when S2 =>  Z <= '1';
                     if X='0' then CS <=S2; else CS <= S3; end if;
        when S3 => Z <= '0';
                     if X='0' then CS <=S3; else CS <= S1; end if;
      end case;
    end if;
  end process;
end B;
```

- CS: same behaviour
- The Z output will have a 1 clock cycle delay

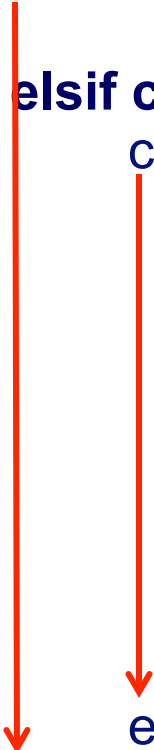
# Describing an FSM in VHDL

## A **1 process** FSM description in VHDL

```

process(clock, reset)
begin
    if reset= '1' then
        CS <= S0;
    elsif clock'event and clock='1' then
        case CS is
            when S0 =>
                Z <= '0';
                if X='0' then
                    CS <=S0;
                else
                    CS <= S2;
                end if;
            when S1 =>
                Z <= '1';
                if X='0' then
                    ...
                end if;
            ...
        end case;
    end if;
end process;

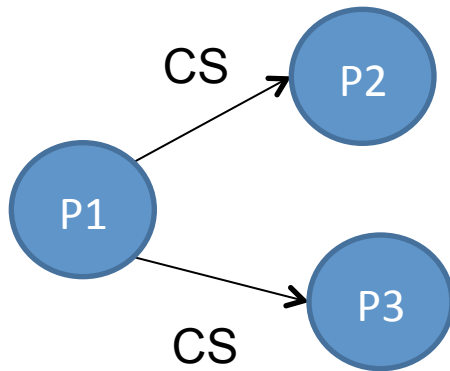
```



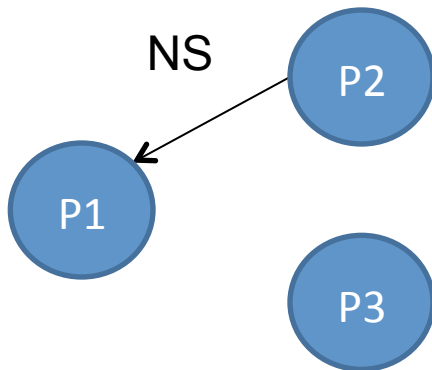
*The NS signal  
(see 2 processes)  
is not used.*

# Describing an FSM in VHDL

## A **3 processes** FSM description in VHDL



- P1 determines the current state (CS), signaling this information to P2 and P3.



- P2 determines the next state, updating the NS signal, with no transition performed (this will be done by P1).
- P3 determines the new signal values (updating the current state).

# Describing an FSM in VHDL

## A **3 processes** FSM description in VHDL

P1 – Process sensitive to clock transitions. It performs the FSM state transitions, making the current state (CS) receive the next state (NS). The transition is sensitive to the clock falling edge.

```
P1: process(clk)
begin
    if clk'event and clk = '0' then
        if rst = '0' then
            CS <= S0;
        else
            CS <= NS;
        end if;
    end if;
end process;
```

# Describing an FSM in VHDL

## A **3 processes** FSM description in VHDL

P2 –Perform the states changes (define the next state). Sensitive to changes in the signals defined in the sensitivity list. It controls the states by defining the flow, that is, it defines what will be the value of the NS signal to be used by the P1 process responsible for performing the state transitions. The construction "case CS is" selects the current state and, according to the FSM signals, a next state is defined in the NS signal.

```
process( CS, X )
begin
    case CS is
        when S0 =>
            NS <= S1;
        when S1 =>
            if X = '1' then
                NS <= S2;
            else
                NS <= S1;
            end if;
        when S2 =>
            NS <= S1;
        when others =>
            end case;
    end process;
```

# Describing an FSM in VHDL

## A **3 processes** FSM description in VHDL

P3 –Performs signal assignments in each state. Signals are changed at the rising edge, and the states at the falling edge. All signals are assigned, including output signals and internal process signals.

```
process (clk)
begin
    if clk'event and clk = '1' then
        case CS is
            when S0 =>
                Z <= '0'
            when S1 =>
                Z <= '0'
            when S2 =>
                Z <= '1';
            when others =>
        end case;
    end if;
end process;
```

## *Using DE2 available signals (pins) for Clock and Reset*

```
library ieee;
use ieee.std_logic_1164.all;
entity FSM is
port (
    LEDR: out std_logic_vector(7 downto 0);
    KEY: in std_logic_vector(3 downto 0);
    CLOCK_50: in std_logic
);
end FSM;
architecture FSM_beh of FSM is
    type states is (S0, S1, S2, S3);
    signal CS, NS: states;
    signal clock: std_logic;
    signal reset: std_logic;
begin
    clock <= CLOCK_50;
    reset <= KEY(3);
```

```
process (clock, reset)
begin
    if reset = '0' then
        CS <= S0;
    elsif clock'event and
        clock = '1' then
        CS <= NS;
    end if;
end process;
```

```
process (CS, KEY(0), KEY(1))
begin
    case CS is
        when S0 => if KEY(0) = '0' then
            NS <= S3; else NS <= S0;
        end if;
        when S1 =>
            LEDR <= "01010101";
            NS <= S0;
        when S2 =>
            case KEY(1) is
                when '0' => LEDR <= "10101010";
                when '1' => LEDR <= "00000000";
                when others => LEDR <= "11111111";
            end case;
            NS <= S1;
        when S3 =>
            NS <= S2;
        end case;
    end process;
end FSM_beh;
```

## Exercise

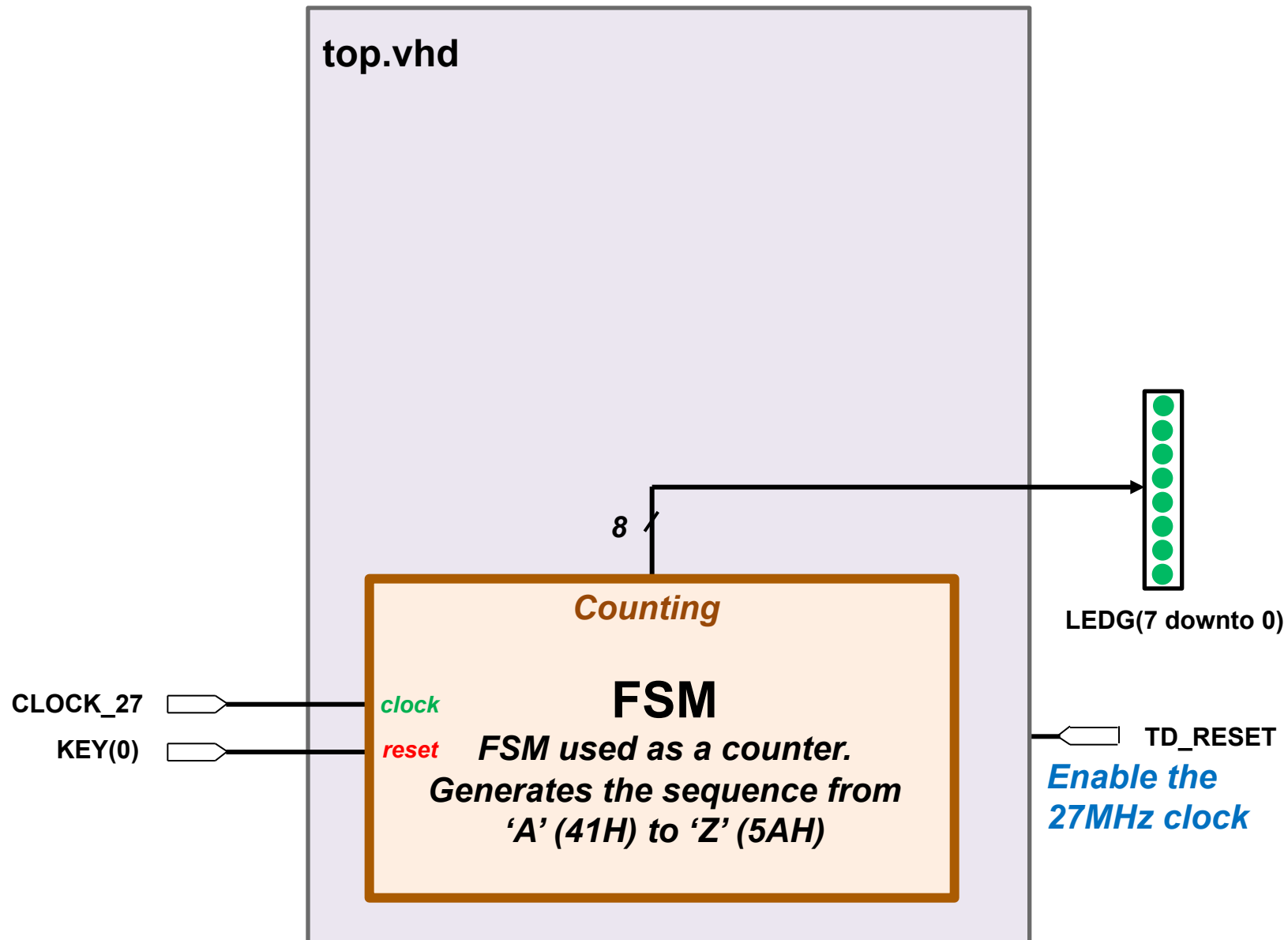


# Tarefa

---

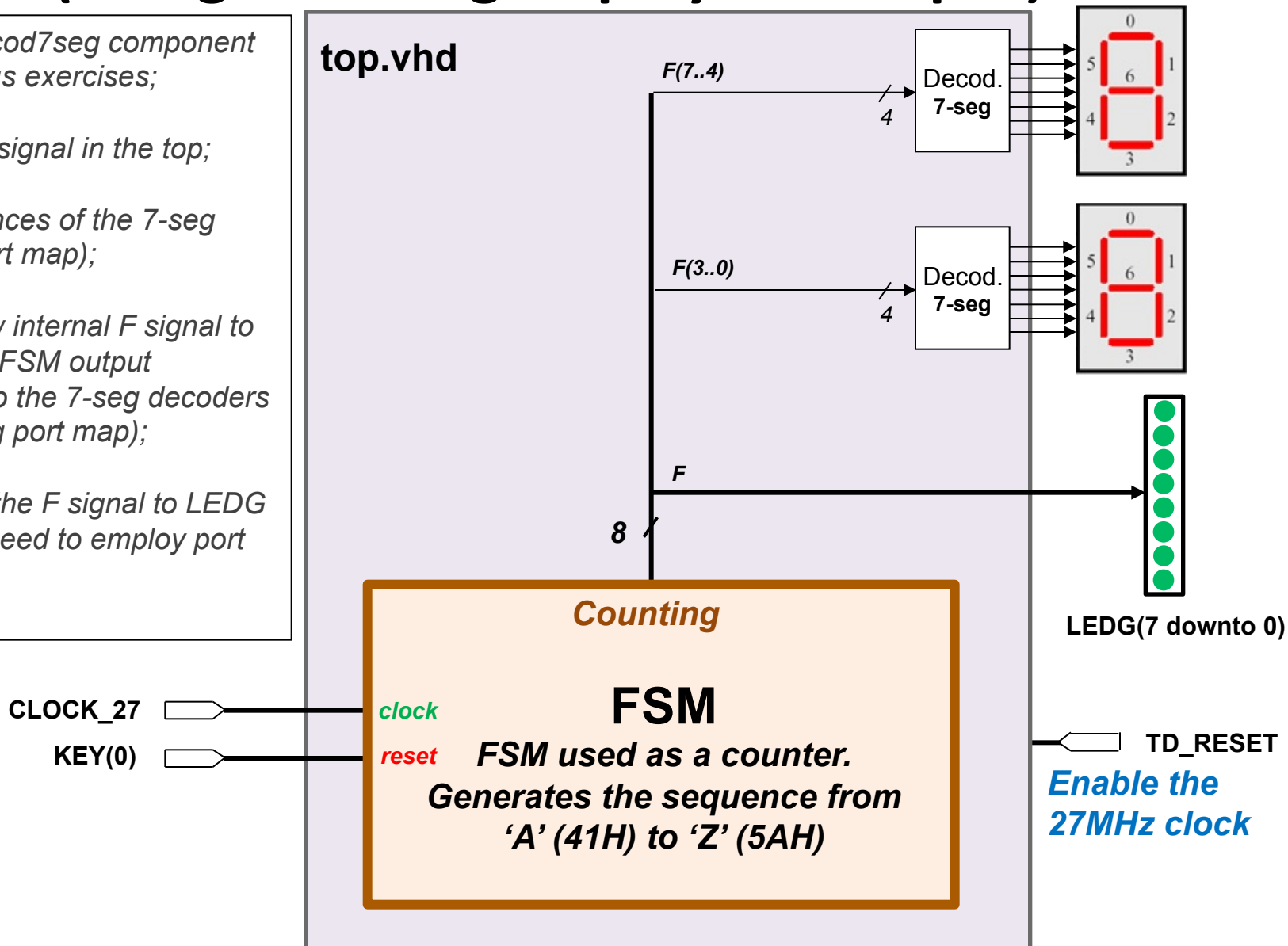
- **Describe an FSM in VHDL to generate the 'A' to 'Z' ASCII characters, shown the values (in binary) in the green LEDs (LEDG).**
- **Define an FSM with asynchronous reset (use KEY(0) for the reset) to initialize a counter with the first value of the sequence ('A' = 41H).**
- **At each 27 MHz clock pulse (rising edge), the counter must be incremented, generating the next ASCII table character.**
- **The FSM must have a reduced number of states, just enough to increment the counter, and check if it reached the end ('Z' = 5AH).**
- **After reaching the last ASCII table char ('Z' = 5AH), the FSM must go back to the start of the sequence, generating again the 'A' value.**

# Block diagram of the circuit to be designed (FSM used as a counter)



# Block diagram of the circuit to be designed (Using the 7-seg displays as output)

1. Use the *Decod7seg* component from previous exercises;
2. Create an *F* signal in the top;
3. Use 2 instances of the 7-seg decoder (port map);
4. Use the new internal *F* signal to connect the FSM output (Counting) to the 7-seg decoders inputs (using port map);
5. To connect the *F* signal to LEDG there is no need to employ port map:  
*LEDG <= F*.



# Tip: **Top.vhd** with the FSM component and 27MHz clock (with no 7-seg displays)

```
entity Top is
  port (
    LEDG: out std_logic_vector(7 downto 0);
    KEY: in std_logic_vector(3 downto 0);
    TD_RESET: out std_logic;
    CLOCK_27: in std_logic
  );
end Top;
architecture top_beh of Top is
  component CountASCII -- This is the FSM component
  port (
    valorASCII: out std_logic_vector(7 downto 0);
    clock: in std_logic;
    reset: in std_logic
  );
begin
  TD_RESET <= '1';

  L0: CountASCII port map ( LEDG, CLOCK_27, KEY(0) );
end top_beh;
```

Place TD\_RESET in '1' to  
"turn on" the CLOCK\_27  
signal in the DE2 board.

Tip: **CountASCII.vhd** – piece of code for the delay generation (for 27 MHz clock input)

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;    -- To use the '+' operator.

process(clock, ... )                -- When using the 27MHz clock, this process will be
begin                               -- executed 27 million times per second.
    ... -- ASCII counter states go here (eg. Start, Inc, End).
    when D1 => -- State to initialise the delay
        delay <= ( others => '0' );
        CS <= D2;
    when D2 => -- State to generate the delay for showing data on LEDG
        delay <= delay + 1;    -- "delay" has been reset on D1.
        CS <= D3;
    when D3 => -- State to test if it reached the max. value.
        if delay >= x"800000" then -- 8,388,608 / 27,000,000 = 0.3 * 3 = 1 s.
            CS <= S1; -- If it's reached the max. value, then exit the delay loop
            -- and go back to the counting ASCII FSM.
        else
            CS <= D4; -- Stay in the loop counting the delay.
        end if;
    when D4 => -- State to continue the delay counting.
        CS <= D2; -- This loop will generate the delay to allow
        -- showing the data on LEDG.
```

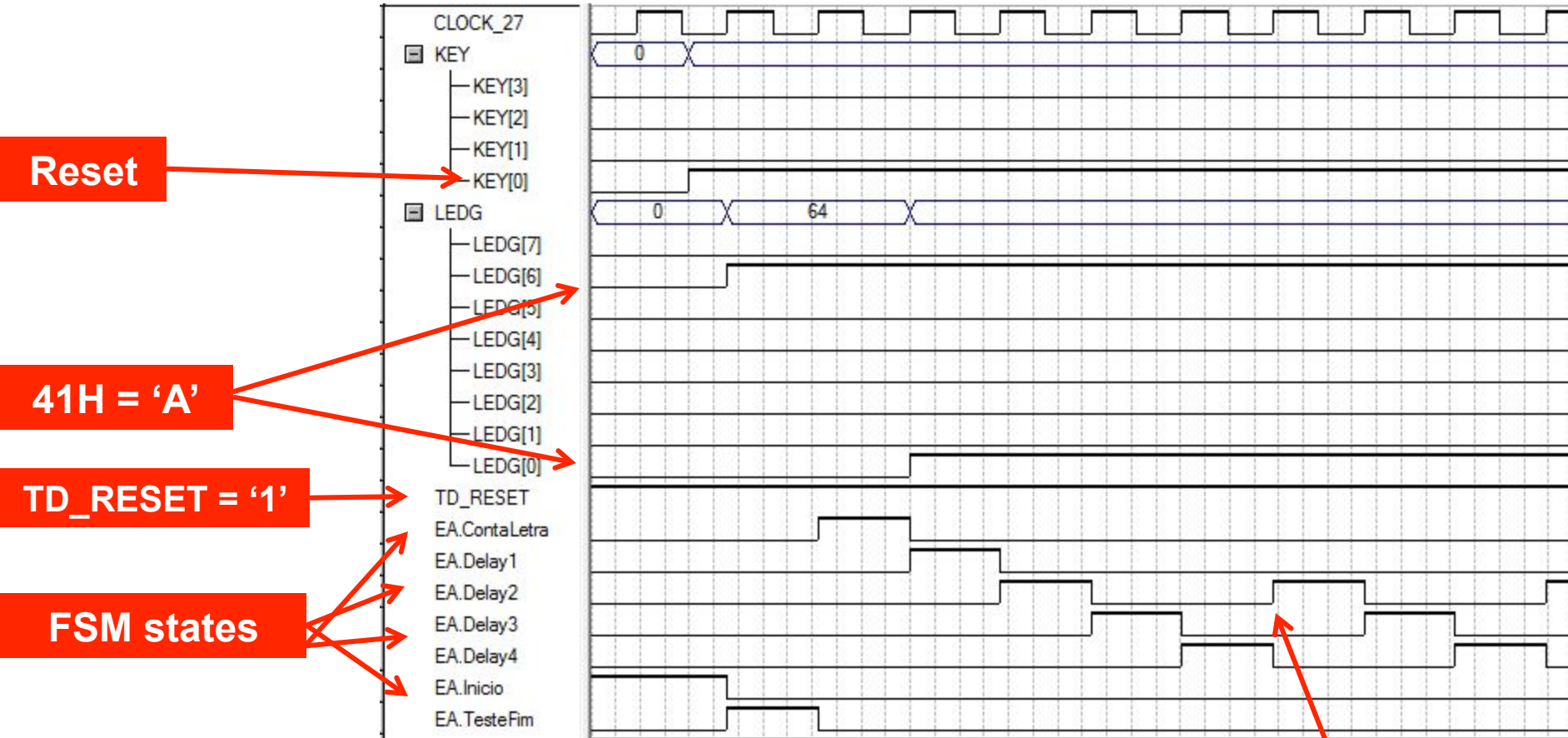
# Quartus II simulation

Type	Message
	Info: Using vector source file "C:/tmp/ContaASCII/ContaASCII.vwf"
[-]	Info: Option to preserve fewer signal transitions to reduce memory requirements is enabled
	Info: Simulation has been partitioned into sub-simulations according to the maximum transition count determined by the eng...
	Info: Simulation partitioned into 131 sub-simulations
	Info: Simulation coverage is 67.89 %
	Info: Number of transitions in simulation is 22484275007
[-]	Info: Quartus II Simulator was successful. 0 errors, 0 warnings
	Info: Peak virtual memory: 152 megabytes
	Info: Processing ended: Sun May 20 05:54:13 2012
	Info: Elapsed time: 19:26:26
	Info: Total CPU time (on all processors): 19:28:42

**Total time to perform a 20 seconds simulation in an i7 quad core (*hyper threading*, "8 cores"), 2.93GHz and 8 GB RAM was 19 hours and 28 minutes.**

Sistema	
Classificação:	4,4 Índice de Experiência do Windows
Processador:	Intel(R) Core(TM) i7 CPU 870 @ 2.93GHz 2.93 GHz
Memória instalada (RAM):	8,00 GB
Tipo de sistema:	Sistema Operacional de 64 Bits
Caneta e Toque:	Nenhuma Entrada à Caneta ou por Toque está disponível para este vídeo
Nome do computador, domínio e configurações de grupo de trabalho	
Nome do computador:	Bezerra
Nome completo do computador:	Bezerra
Descrição do computador:	Bezerra Desktop
Grupo de trabalho:	GRUPO

# Quartus II simulation



The delay states are repeated each 3 clock pulses (D2 in the delay tip slide).

# Quartus II simulation

Showing the counting in LEDG, at each pulse in the EA.ContaLetra state

