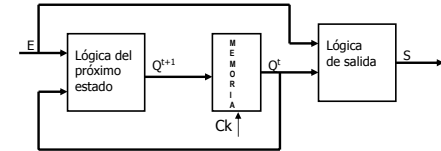


MAQUINA DE ESTADO FINITO (FSM)

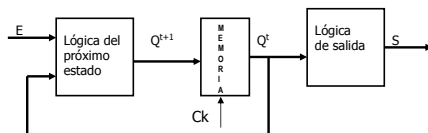
Autómata finito

Modelo de Mealy



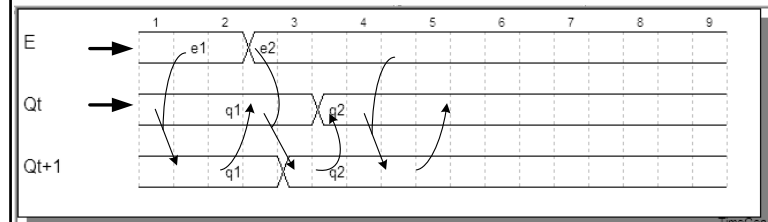
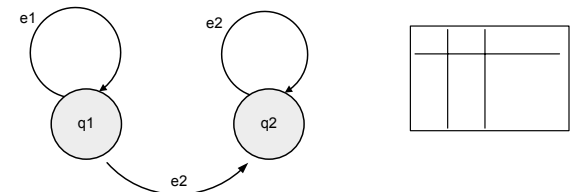
$$Q^{t+1} = f(E, Q^t) \quad S = g(E, Q^t)$$

Modelo de Moore



$$Q_{t+1} = f(E, Q^t) \quad S = g(E, Q^t)$$

TRANSICIÓN DE ESTADOS



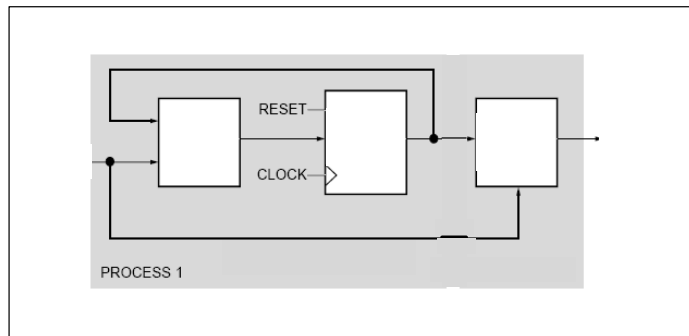
En el ambiente ISE tenemos distintas alternativas para describir una FSM:

- Con el código VHDL
- Con el editor de máquina de estado
- Con captura de esquemas

Implementación con Xilinx Synthesis Technology (XST)

Esta herramienta reconoce (y entonces optimiza) tres tipos (formatos) de descripciones (sincrónicas):

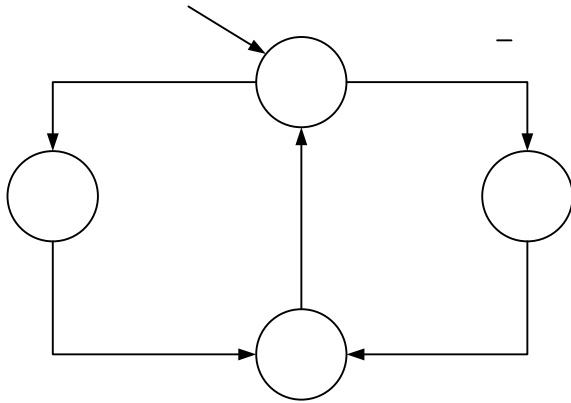
- Descripción en un sólo proceso
- Descripción en dos procesos
- Descripción en tres procesos



```
entity
.....
end entity

Architecture ...
- Definición de señales...
Definición de una señal tipo enumerada para representar el "estado"
(ejemplo: est1, est2, etc)

begin
process (clk, reset)
begin
- Llevar al sistema a condiciones iniciales con "reset" (estados y salidas)
- Con el flanco activo del reloj hacer (clk'event and clk='1'):
Case estado is
when est1
--- } próximos estados
--- }
--- } salidas
--- }
when est2
---
---
end process
end architecture
```



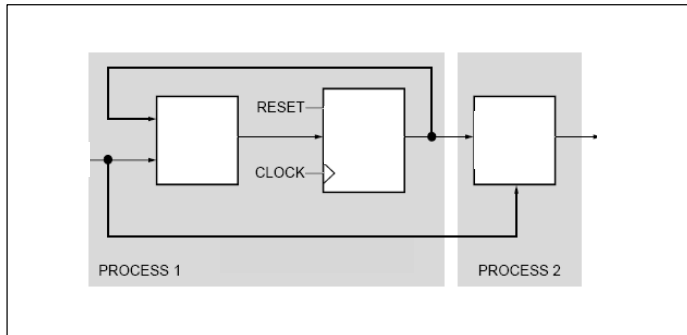
```

library IEEE;
use IEEE.std_logic_1164.all;

entity fsm is
  port (
    clk, reset, x1 : IN std_logic;
    outp : OUT std_logic);
end entity;

architecture beh1 of fsm is
  type state_type is (s1,s2,s3,s4);
  signal state: state_type;
begin
  process (clk, reset)
  begin
    if (reset = '1') then
      state <= s1;
      outp <= '1';
    elsif (clk='1' and clk'event) then
      case state is
        when s1 =>
          if x1='1' then
            state <= s2;
            outp <= '1';
          else
            state <= s3;
            outp <= '0';
          end if;
        when s2 => state <= s4; outp <= '0';
        when s3 => state <= s4; outp <= '0';
        when s4 => state <= s1; outp <= '1';
      end case;
    end if;
  end process;
end beh1;

```



s2

x1
0

outp

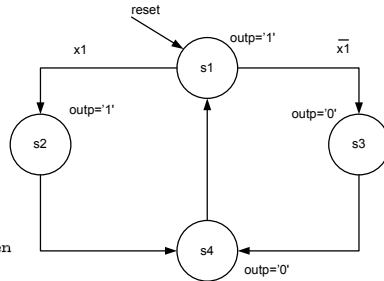
Puede ser
reemplazado por
sentencias de
asignación
concurrentes

```

library IEEE;
use IEEE.std_logic_1164.all;

entity fsm is
  port (
    clk, reset, x1 : IN std_logic;
    outp : OUT std_logic);
end entity;
architecture beh1 of fsm is
  type state_type is (s1,s2,s3,s4);
  signal state: state_type;
begin
  process1: process (clk, reset)
  begin
    if (reset = '1') then
      state <= s1;
    elsif (clk='1' and clk'Event) then
      case state is
        when s1 =>
          if x1='1' then
            state <= s2;
          else
            state <= s3;
          end if;
        when s2 => state <= s4;
        when s3 => state <= s4;
        when s4 => state <= s1;
      end case;
    end if;
  end process process1;

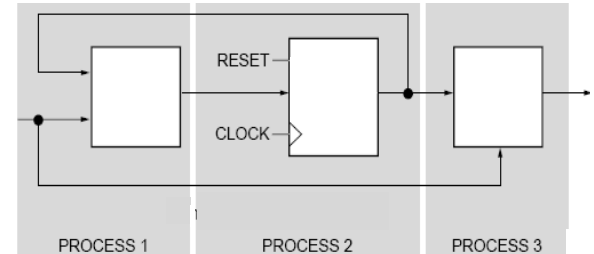
```



```

  process2 : process (state)
  begin
    case state is
      when s1 => outp <= '1';
      when s2 => outp <= '1';
      when s3 => outp <= '0';
      when s4 => outp <= '0';
    end case;
  end process process2;
end beh1;

```



```

entity
....
end entity

Architecture ...
- Definición de señales...
  Definición de dos señales tipo enumerada para representar el "estado" y
  el "proximo_estado"
  (ejemplo: est1, est2, etc)

```

```

begin
Proceso_1: porcess (clk, reset)
begin

```

```

- Llevar al sistema a condiciones iniciales con "reset" (estados)
- Con el flanco activo del reloj hacer (clk'event and clk='1'):

```

```

  estado <= proximo_estado

```

```

end process Proceso_1

```

```

Proceso_2: process (estado, entrada1, entrada2, ...)

```

```

  Case estado is

```

```

    when est1 =>

```

```

    --- } próximos estados (ej.: proximo_estado <= est1)
    --- }

```

```

    when est2 =>

```

```

    --- } próximos estados
    --- }

```

```

end process: Proceso_2

```

```

Proceso_3: porcess (estado)

```

```

  Case estado is

```

```

    when est1 => (salidas)

```

```

    when est2 => (salidas)

```

```

    ---
    ---

```

```

end process Proceso_3

```

```

end architecture

```

```

library IEEE;
use IEEE.std_logic_1164.all;

```

```

entity fsm is
  port (
    clk, reset, x1 : IN std_logic;
    outp : OUT std_logic);
end entity;

```

```

architecture beh1 of fsm is
  type state_type is (s1,s2,s3,s4);
  signal state, next_state: state_type;
begin

```

```

  process1: process (clk, reset)
  begin
    if (reset = '1') then
      state <= s1;
    elsif (clk = '1' and clk'Event) then
      state <= next_state;
    end if;
  end process process1;

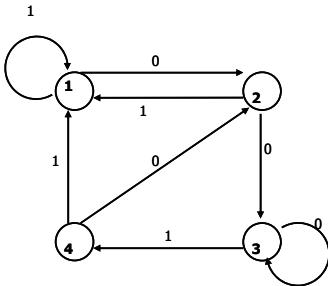
```

```

  process2 : process (state, x1)
  begin
    case state is
      when s1 =>
        if x1='1' then
          next_state <= s2;
        else
          next_state <= s3;
        end if;
      when s2 => next_state <= s4;
      when s3 => next_state <= s4;
      when s4 => next_state <= s1;
    end case;
  end process process2;
  process3 : process (state)
  begin
    case state is
      when s1 => outp <= '1';
      when s2 => outp <= '1';
      when s3 => outp <= '0';
      when s4 => outp <= '0';
    end case;
  end process process3;
end beh1;

```

Detectar la secuencia 0-0-1 (por Moore)



Con VHDL Declaración de la *entidad*

```
-- !!!! MAQUINA DE MOORE !!!!

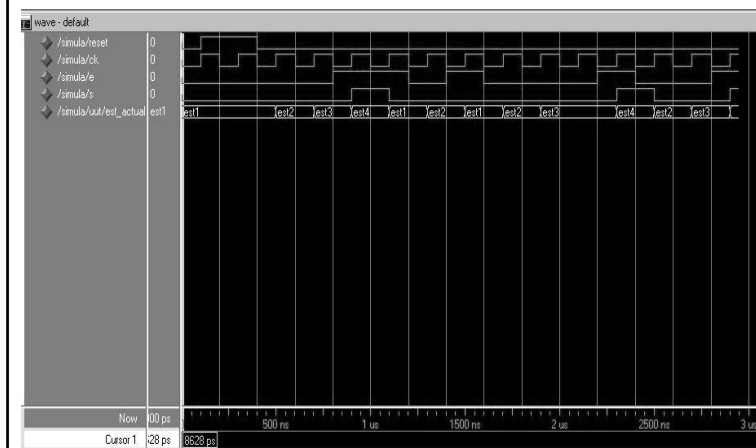
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Maq_sinc is
  Port ( reset : in std_logic;
        e : in std_logic;
        ck : in std_logic;
        s : out std_logic);
end Maq_sinc;
```

Con VHDL (*contin.*) Declaración de la *arquitectura*

```
architecture Behavioral of Maq_sinc is
  type estados is (est1, est2, est3, est4);
  signal est_actual: estados:= est1;
begin
  process (reset, ck)
  begin
    if reset = '1' then est_actual<= est1 ;
    elsif ck='1' and ck'event then
      case est_actual is
        when est1 => if e='1' then est_actual <= est1; else est_actual <= est2;
        end if;
        when est2 => if e='1' then est_actual <= est1; else est_actual <= est3;
        end if;
        when est3 => if e='1' then est_actual <= est4; else est_actual <= est3;
        end if;
        when est4 => if e='1' then est_actual <= est1; else est_actual <= est2;
        end if;
      end case;
    end process;
    process (est_actual)
    begin
      case est_actual is
        when est1 => s<='0';when est2 => s<='0';when est3 => s<='0';when est4 => s<='1';
        end case;
      end process;
    end Behavioral;
```

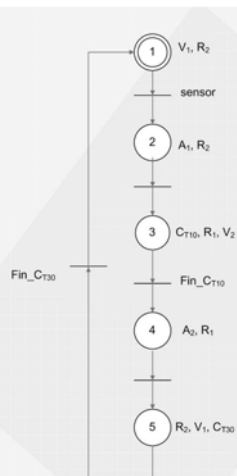
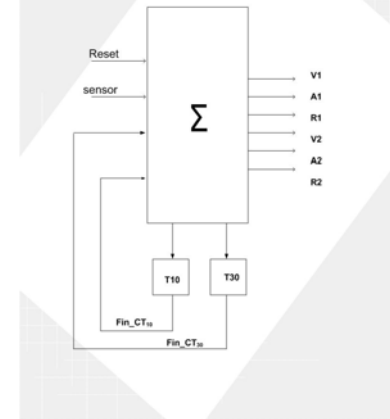
Resultado de la simulación



SEMÁFORO

- En la intersección de una *carretera* y un *camino vecinal* se instala un semáforo con el siguiente comportamiento:
 - En el estado inicial indica verde para la carretera y rojo en el camino
 - Existen sensores (sensor) que detectan la presencia de vehículos en el camino, cuando esto ocurre se habilita el tránsito por el camino durante 10 segundos.
 - Transcurrido los 10 segundos, se pone verde la carretera y durante 20 segundos no se atiende la señal del sensor.
 - Se dispone de una señal de reloj de frecuencia de 1Hz.
 - La evolución de los semáforos es la estándar (verde, amarillo, rojo)

Caja negra para modelar según una Red de Petri



Red de Petri

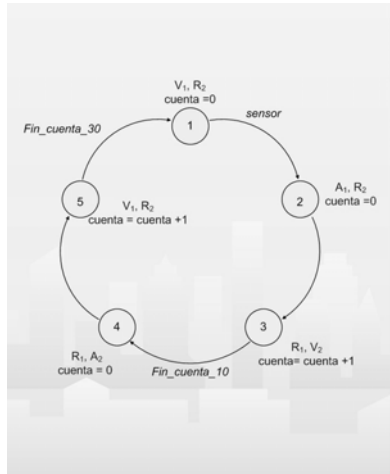


DIAGRAMA DE ESTADOS

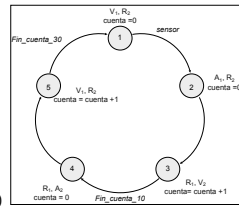
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

```
entity FSM is
    Port ( sensor : in std_logic;
          reset : in std_logic;
          clk : in std_logic;
          sem_carre : out std_logic_vector(2 downto 0);
          sem_camino : out std_logic_vector(2 downto 0));
end FSM;
```

```
architecture Behavioral of FSM is
    type estado is (inicial, carre_amari, cami_verde, cami_amari,espera);
    constant verde :std_logic_vector (2 downto 0):= "001";
    constant amarillo :std_logic_vector (2 downto 0):= "010";
    constant rojo :std_logic_vector (2 downto 0):= "100";
    signal estado_actual : estado:=inicial;
    signal reset_cuenta : boolean:= false;
    signal fin_cuenta_10, fin_cuenta_20 : boolean;
    signal cuenta : integer range 0 to 63;
    BEGIN
```

-- Definimos la máquina de estados

```
MAQUINA:
process (clk, reset)
begin
    if reset ='1' then
        estado_actual <= inicial;
    elsif clk='1' and clk'event then
        CASE estado_actual IS
            WHEN inicial =>
                if sensor ='1' then
                    estado_actual <=carre_amari;
                end if;
            WHEN carre_amari =>
                estado_actual <= cami_verde;
            WHEN cami_verde =>
                if fin_cuenta_10 then
                    estado_actual <= cami_amari;
                end if;
            WHEN cami_amari =>
                estado_actual <= espera;
            WHEN espera =>
                if fin_cuenta_20 then
                    estado_actual <= inicial;
                end if;
            END CASE ;
        end if;
    end process MAQUINA;
```



--(1)

--(2)

--(2)

--(3)

--(3)

--(4)

--(4)

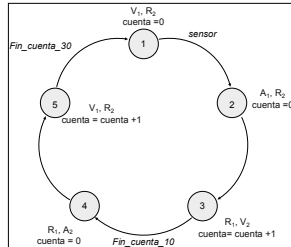
--(5)

--(5)

--(1)

SALIDA:

```
process (estado_actual)
begin
    CASE estado_actual is
        WHEN inicial => --(1)
            sem_carre <= verde;
            sem_camino <= rojo;
            reset_cuenta <= true;
        WHEN carre_amari => --(2)
            sem_carre <= amarillo;
            sem_camino <= rojo;
            reset_cuenta <= true;
        WHEN cami_verde => --(3)
            sem_carre <= rojo;
            sem_camino <= verde;
            reset_cuenta <= false;
        WHEN cami_amari => --(4)
            sem_carre <= rojo;
            sem_camino <= amarillo;
            reset_cuenta <= true;
        WHEN espera => --(5)
            sem_carre <= verde;
            sem_camino <= rojo;
            reset_cuenta <= false;
    END CASE;
end process salida;
```



-- Definición del contador

```
CONTADOR:
process (clk)
begin
    if clk='1' and clk'event then
        if reset_cuenta then cuenta <= 0;
        else cuenta <= cuenta + 1;
        end if;
    end if;
end process contador;
```

```
-- Detección de las finalizaciones de tiempos 10s y 20s
fin_cuenta_10 <= true WHEN cuenta = 9 ELSE false;
fin_cuenta_20 <= true WHEN cuenta = 19 ELSE false;
```

end Behavioral;

