Universidad de Alcalá

Departamento de Electrónica

Modelado de Sistemas Computacionales

Grado en Ingeniería de Computadores

Ejercicios de máquinas de estados.

Se desea diseñar un contador ascendente sincrono que siga la secuencia 2, 3, 5, 8, 2, 3,.... mediante una maquina de estados. El contador dispone de una entrada de *reset* sincrona activa a nivel bajo, RST, que inicia la salida a valor 2, **Y[3:0]**=0010 binario, y una entrada de *clock enable* activa a nivel bajo, CE, que permite los cambios de estado cuando se activa y los inhibe en caso contrario. El *reset* no se llevara a cabo si no esta activada la entrada CE. En la Figura 4.1 se muestra el diagrama del contador a disenar.

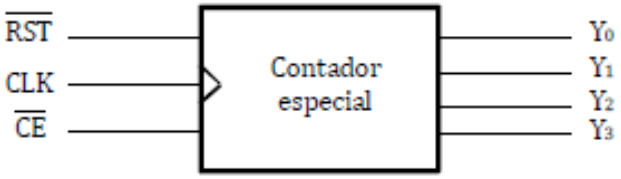


Figura 4.1. Contador a diseñar.

¿Qué tipo de maquina de estados, *Mealy* o *Moore*, seria necesario para implementar el sistema escrito?

La maquina es tipo Moore

Diseñe el diagrama de estados del contador como maquina de Moore siguiendo la nomenclatura de la Figura 4.2.

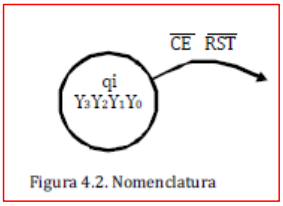
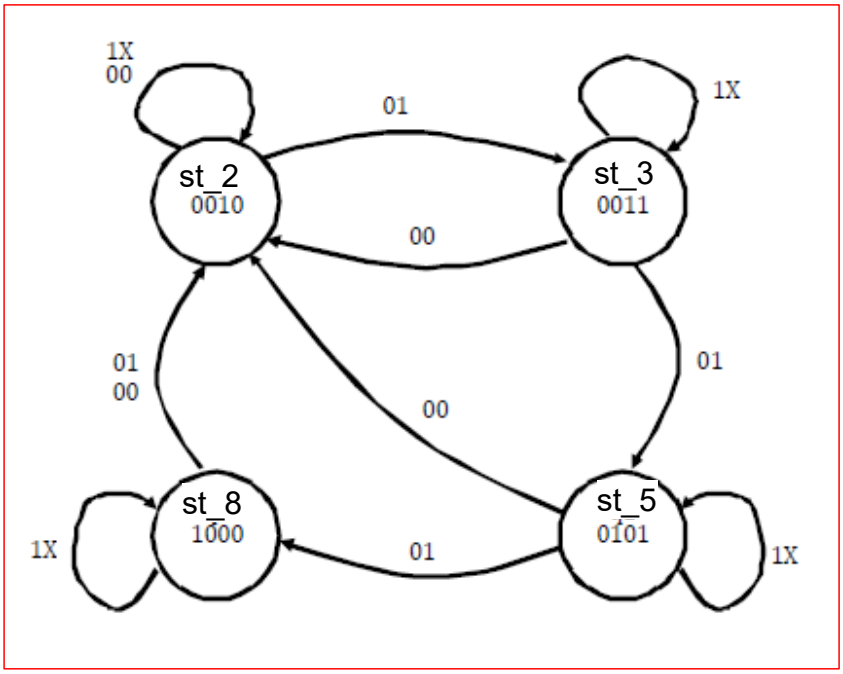


Figura 4.2. Nomenclatura



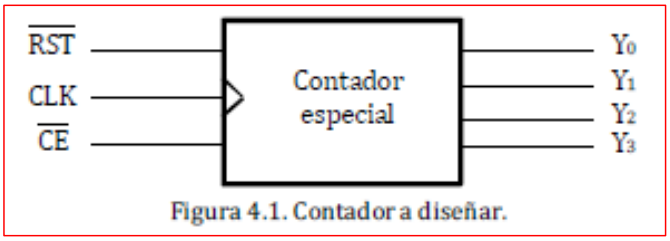
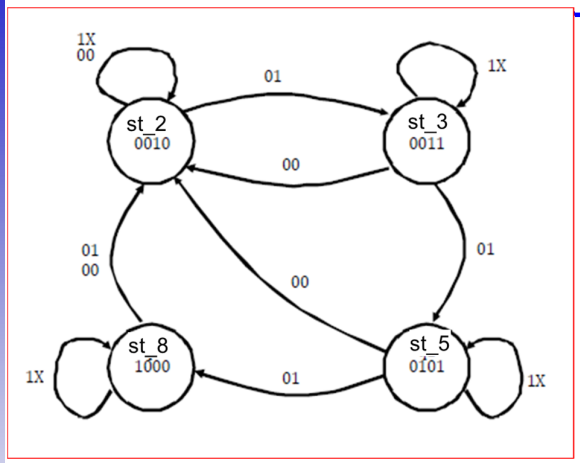
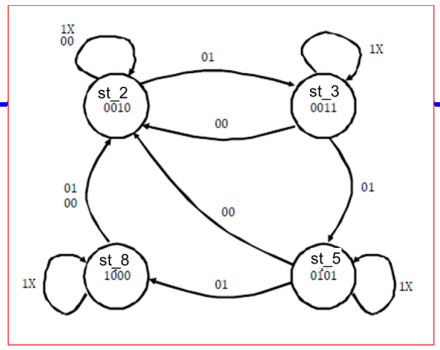


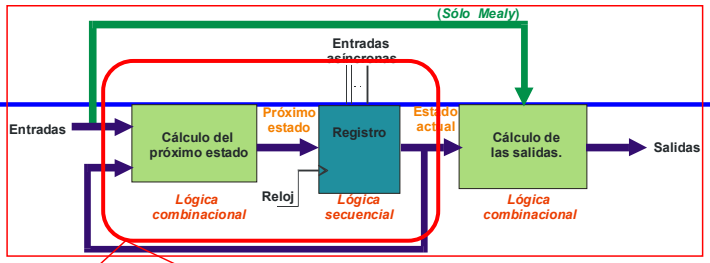
Figura 4.1. Contador a diseñar.

```
library ieee;
use ieee.std_logic_1164.all;


entity cnt_28 is
  port ( CLK : in  std_logic;
         RST : in  std_logic;
         CE  : in  std_logic;
         Y   : out std_logic_vector(3 downto 0));
end cnt_28;
```



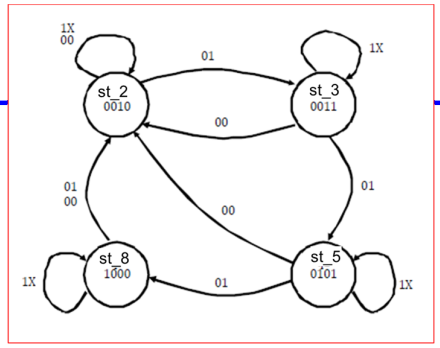

```
architecture rtl of cnt_28 is
  type tipo_estado is (std_2, std_3, std_4, std_5, std_6, std_7, std_8 );
  signal estado : tipo_estado;
```

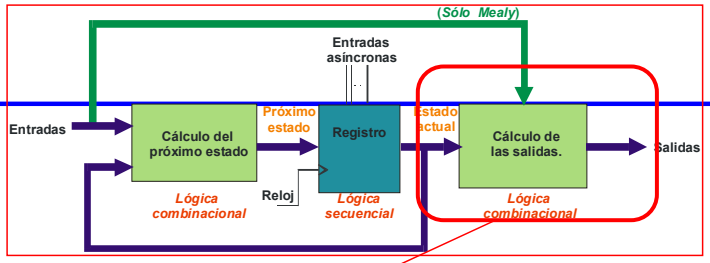






```
process (clk)
  begin  -- process
    if clk'event and clk = '1' then
      case estado is
        when std_2 =>
          if CE = '0' then
            if RST = '1' then
              estado <= std_3;
            end if;
          end if;
        when std_3 =>
          if CE = '0' then
            if RST = '0' then
              estado <= std_2;
            else
              estado <= std_4;
            end if;
          end if;
        when std_4 =>
          if CE = '0' then
            if RST = '0' then
              estado <= std_2;
            else
              estado <= std_5;
            end if;
          end if;
```

```
        when std_5 =>
          if CE = '0' then
            if RST = '0' then
              estado <= std_2;
            else
              estado <= std_6;
            end if;
          end if;
        when std_6 =>
          if CE = '0' then
            if RST = '0' then
              estado <= std_2;
            else
              estado <= std_7;
            end if;
          end if;
        when std_7 =>
          if CE = '0' then
            if RST = '0' then
              estado <= std_2;
            else
              estado <= std_8;
            end if;
          end if;
        when std_8 =>
          if CE = '0' then
            estado   <= std_2;
          end if;
      end case;
    end if;
  end process;
```






```
process (estado)
  begin  -- process
    case estado is
      when std_2 =>
        Y <= x"2";
      when std_3 =>
        Y <= x"3";
      when std_4 =>
        Y <= x"4";
      when std_5 =>
        Y <= x"5";
      when std_6 =>
        Y <= x"6";
      when std_7 =>
        Y <= x"7";
      when std_8 =>
        Y <= x"8";
    end case;
  end process;

end rtl;
```



```
library ieee;
use ieee.std_logic_1164.all;
entity cnt_28_tb is

end cnt_28_tb;

architecture sim of cnt_28_tb is

  signal CLK_i : std_logic := '0';
  signal RST_i : std_logic := '0';
  signal CE_i  : std_logic := '0';
  signal Y_i   : std_logic_vector(3 downto 0);
begin  -- sim

  DUT : entity work.cnt_28
    port map (
      CLK => CLK_i,
      RST => RST_i,
      CE  => CE_i,
      Y   => Y_i);

  CLK_i <= not CLK_i after 5 ns;

  process
  begin  -- process
    wait for 21 ns;
    wait until CLK_i = '0';
    RST_i <= '1';
    wait for 71 ns;
    wait until CLK_i = '0';
    CE_i  <= '1';
    wait for 22 ns;
    report "fin de laa simulción" severity failure;
  end process;

end sim;
```

Se desea diseñar un sistema de gestión del pago con tarjeta en una gasolinera cuyo diagrama se muestra en la Figura 4.1. Su funcionamiento sería el siguiente:

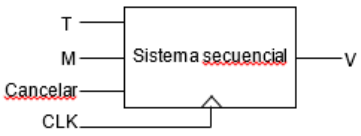


Figura 4.1. Diagrama de bloques

- Inicialmente el sistema estará en estado de reposo esperando que el cliente introduzca la tarjeta válida. Una vez introducida una tarjeta el sistema comprobará si es válida o no, obteniendo esa información de un dispositivo externo que genera una señal **T** a valor 1 cuando la tarjeta sea válida y 0 cuando no lo sea. Si no es válida, el sistema permanecerá en estado de reposo.
- Si la tarjeta es válida, pasará a un nuevo estado en el que, si el usuario aprieta la manilla de la manguera de la gasolina se genera una señal M a valor 1 para iniciar el repostaje. Cuando deje de apretar, M=0, finalizará el repostaje, volviendo al estado de reposo.
- Cuando se inicia el repostaje el sistema genera una señal de salida V activa a nivel alto que abre la válvula de la gasolina.
- Una vez la tarjeta ha sido validada es irrelevante si el usuario la retira o no, o si se modifica el estado de la entrada de validación.
- Hasta que la tarjeta se valide no comenzará el repostaje, independientemente de si se aprieta la manilla de la manguera de la gasolina o no.
- Hay un botón de cancelación (Cancelar) que devuelve el sistema al estado de reposo inmediatamente.

Dibuje el grafo de un autómata de Moore que se comporte según la descripción anterior siguiendo la nomenclatura de la Figura 4.2.

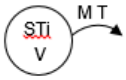
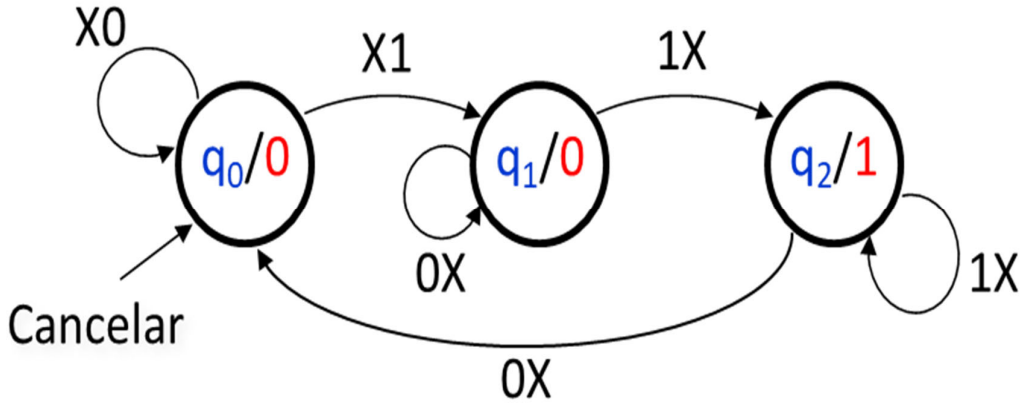


Figura 4.2. Nomenclatura.



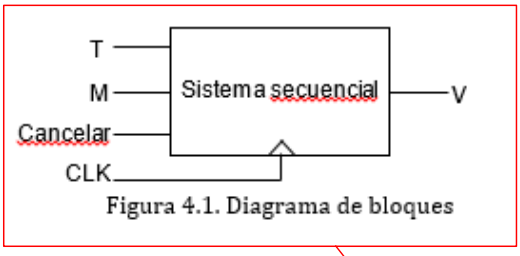
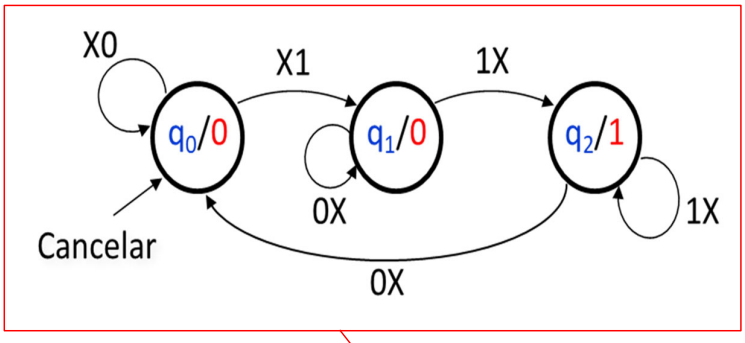
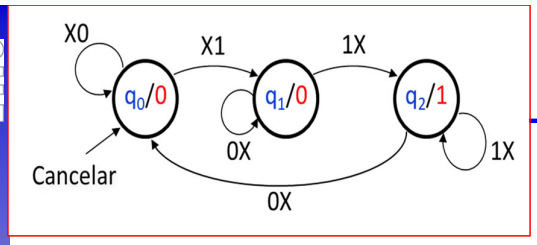


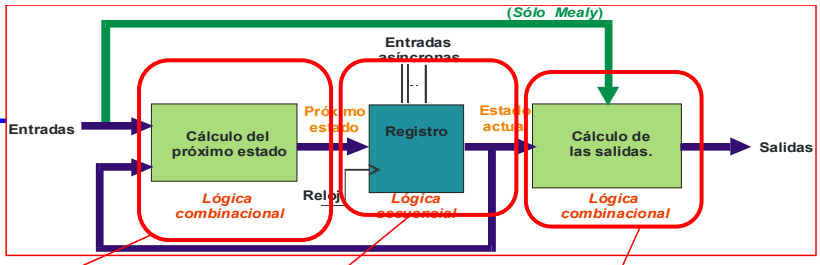
Figura 4.1. Diagrama de bloques

```
library ieee;
use ieee.std_logic_1164.all;


entity cnt_gasolina is
  port ( CLK      : in  std_logic;
         CANCELAR : in  std_logic;
         M        : in  std_logic;
         T        : in  std_logic;
         V        : out std_logic);
end cnt_gasolina;
```


```
architecture rtl of cnt_gasolina is
  type tipo_estado is (q0, q1, q2 );
  signal estado, prox_estado : tipo_estado;

begin
```






```
process (estado, T, M)
 begin
   case estado is
     when Q0 =>
       if T = '1' then
         prox_estado <= Q1;
       else
         prox_estado <= Q0;
       end if;
     when Q1 =>
       if M = '1' then
         prox_estado <= Q2;
       else
         prox_estado <= Q1;
       end if;
     when Q2 =>
       if M = '0' then
         prox_estado <= Q0;
       else
         prox_estado <= Q2;
       end if;
   end case;
 end process;
```

```
process (CLK, CANCELAR)
  begin
    if CANCELAR = '1' then
      estado <= Q0;
    elsif clk'event and clk = '1' then
      estado <= prox_estado;
    end if;
  end process;
```

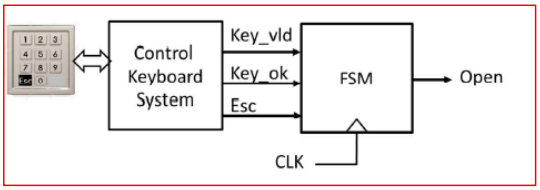
```
V <= '1' when estado = Q2 else '0';

end rtl;
```



```
library ieee;
use ieee.std_logic_1164.all;

entity cnt_gasolina_tb is

end cnt_gasolina_tb;

architecture sim of cnt_gasolina_tb is

  signal CLK_i      : std_logic := '0';
  signal CANCELAR_i : std_logic := '1';
  signal M_i        : std_logic := '0';
  signal T_i        : std_logic := '0';
  signal V_i        : std_logic;

begin  -- sim

  DUT : entity work.cnt_gasolina
    port map (
      CLK      => CLK_i,
      CANCELAR => CANCELAR_i,
      M        => M_i,
      T        => T_i,
      V        => V_i);

  CLK_i      <= not CLK_i after 5 ns;
  CANCELAR_i <= '0'after 123 ns;
```

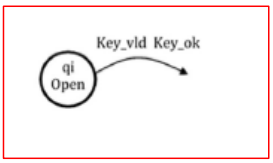
```
 process
  begin  -- process
    wait for 223 ns;
    wait until CLK_i = '0';
    M_i <= '1';
    wait for 23 ns;
    wait until CLK_i = '0';
    M_i <= '0';
    wait for 23 ns;
    wait until CLK_i = '0';
    T_i <= '1';
    wait for 33 ns;
    wait until CLK_i = '0';
    M_i <= '1';
    wait for 23 ns;
    wait until CLK_i = '0';
    T_i <= '0';
    wait for 33 ns;
    wait until CLK_i = '0';
    M_i <= '0';
    wait for 33 ns;
    wait until CLK_i = '0';
    M_i <= '1';
    wait for 23 ns;

    report "fin de laa simulción" severity failure;
  end process;
end sim;
```

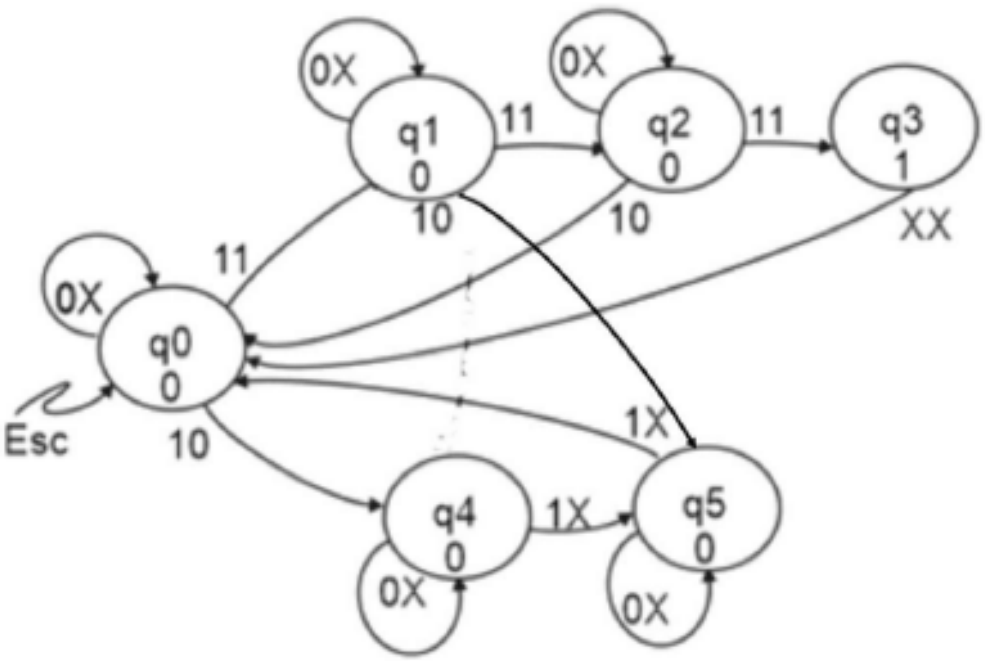
La Figura 4.1 muestra un sistema electrónico de control de acceso formado por un módulo de decodificación de teclado y clave (Control Keyboard System) y una máquina de estados (FSM). Esta última proporciona una señal Open (activa a nivel alto) que permite abrir una puerta siempre y cuando la clave introducida sea correcta.
El funcionamiento del sistema es el siguiente:

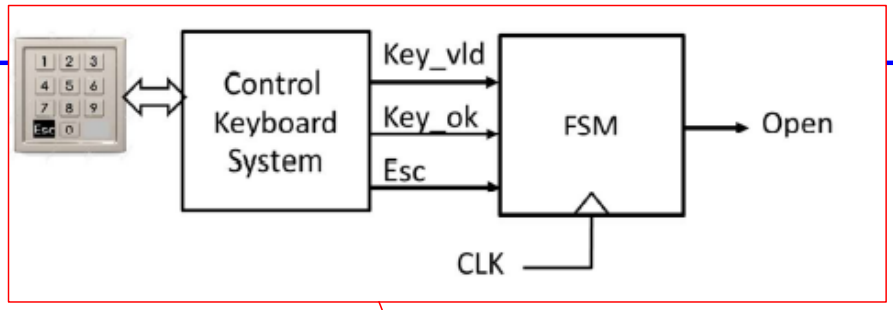


1. La clave consta de tres dígitos BCD y se encuentra almacenada en el módulo Control Keyboard System.
2. El módulo Control Keyboard System, que no hay que diseñar en esta cuestión, se encarga de decodificar la tecla pulsada y compararla con la clave en él almacenada.
3. La salida Key_vld del módulo Control Keyboard System se pone a nivel alto durante un periodo de la señal CLK cada vez que se pulsa una tecla numérica.
4. La salida Key_ok del módulo Control Keyboard System se pone a nivel alto durante un periodo de la señal CLK cada vez que se ha pulsado una tecla y esta coincide con el correspondiente dígito de la clave. Esta entrada sólo se va a activar si lo hace también Key_vld.
5. La puerta se abre (Open =1) cuando se hayan introducidos tres dígitos y estos coincidan con la clave. Hasta que no se hayan introducido los tres dígitos, la salida Open permanecerá a 0.
6. Si tras introducir el tercer dígito alguno de ellos no ha sido válido, se vuelve a esperar la introducción de una nueva clave.
7. Si tras introducir el tercer dígito la clave es correcta, la salida Open se activará durante un periodo de la señal CLK, volviéndose a esperar la introducción de una nueva clave.
8. Mientras se está abriendo la puerta (Open=1) nunca se va a pulsar una tecla.
9. Cuando se pulsa la tecla Esc el módulo Control Keyboard System activará su salida Esc (nivel alto), y se da por cancelado el proceso de introducción de la clave, volviéndose al estado de espera de la introducción de una nueva. Esta señal se considera asíncrona.

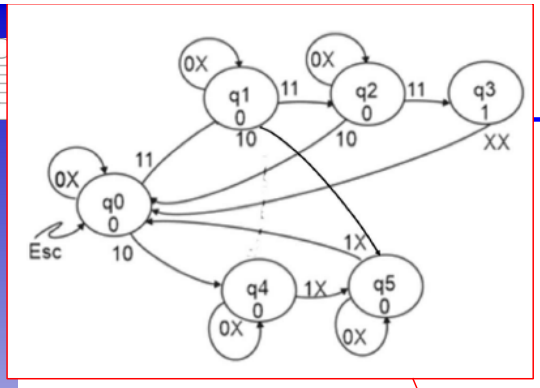
Dibujar el diagrama de estados de la máquina que controla la activación de la señal Open siguiendo la nomenclatura indicada.



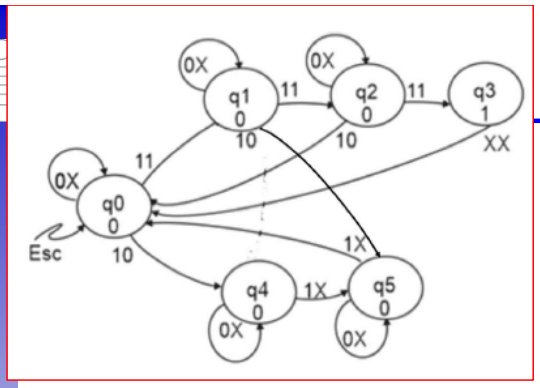


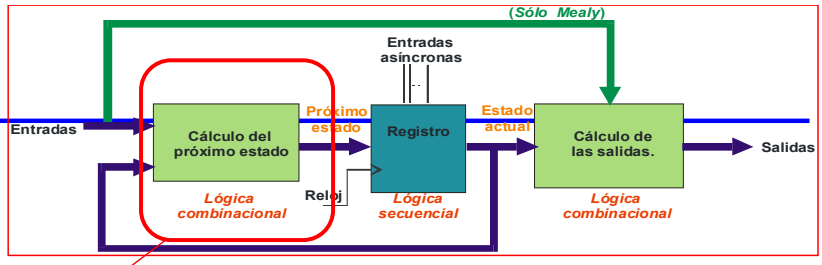



```
library ieee;
use ieee.std_logic_1164.all;

entity cnt_clave is
  port ( CLK     : in  std_logic;
         ESC     : in  std_logic;
         KEY_VLD : in  std_logic;
         KEY_OK  : in  std_logic;
         OPEN_C  : out std_logic);
end cnt_clave;
```

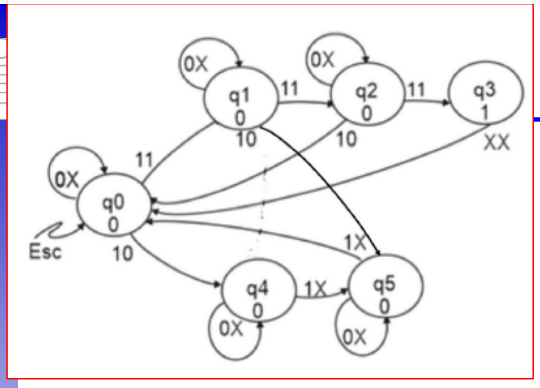

```
architecture rtl of cnt_clave is
  type tipo_estado is (q0, q1, q2, q3, q4, q5 );
  signal estado, prox_estado : tipo_estado;
  signal keys                : std_logic_vector(1 downto 0);
begin

 keys <= KEY_VLD&KEY_OK;
```

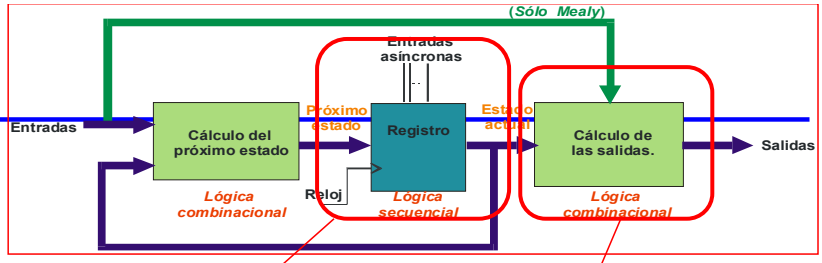





```
process (estado,keys )
 begin
   case estado is
     when Q0 =>
       if keys = "11" then
         prox_estado <= Q1;
       elsif keys = "10" then
         prox_estado <= Q4;
       else
         prox_estado <= Q0;
       end if;
     when Q1 =>
       if keys = "11" then
         prox_estado <= Q2;
       elsif keys = "10" then
         prox_estado <= Q5;
       else
         prox_estado <= Q1;
       end if;
     when Q2 =>
       if keys = "11" then
         prox_estado <= Q3;
       elsif keys = "10" then
         prox_estado <= Q0;
       else
         prox_estado <= Q2;
       end if;
```

```
when Q3 =>
        prox_estado   <= Q0;
      when Q4 =>
        if (keys = "11")or (keys = "10") then
          prox_estado <= Q5;
        else
          prox_estado <= Q4;
        end if;
      when Q5 =>
        if (keys = "11")or (keys = "10") then
          prox_estado <= Q0;
        else
          prox_estado <= Q5;
        end if;
   end case;

 end process;
```



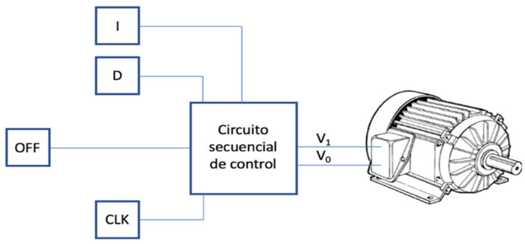

```
process (CLK, ESC)
  begin
    if ESC = '1' then
      estado <= Q0;
    elsif clk'event and clk = '1' then
      estado <= prox_estado;
    end if;
  end process;
```

```
OPEN_C <= '1' when estado = Q3 else '0';
```

```vhdl
library ieee;
use ieee.std_logic_1164.all;
entity cnt_clave_tb is
end cnt_clave_tb;

architecture sim of cnt_clave_tb is

  signal CLK_i      : std_logic := '0';
  signal ESC_i      : std_logic := '1';
  signal KEY_VLD_i  : std_logic := '0';
  signal KEY_OK_i   : std_logic := '0';
  signal OPEN_C_i   : std_logic;

begin  -- sim

  DUT : entity work.cnt_clave
    port map (
      CLK     => CLK_i,
      ESC     => ESC_i,
      KEY_VLD => KEY_VLD_i,
      KEY_OK  => KEY_OK_i,
      OPEN_C  => OPEN_C_i);

  CLK_i      <= not CLK_i after 5 ns;
  ESC_i <= '0'after 123 ns;

  process
  begin  -- process
    wait for 223 ns;
    --tecla ok
    wait until CLK_i = '0';
    KEY_VLD_i <= '1';
    KEY_OK_i  <= '1';
    wait until CLK_i = '0';
    KEY_VLD_i <= '0';
    KEY_OK_i  <= '0';
    wait for 23 ns;
    wait until CLK_i = '0';
    KEY_VLD_i <= '1';
    KEY_OK_i  <= '1';
    wait until CLK_i = '0';
    KEY_VLD_i <= '0';
    KEY_OK_i  <= '0';
    wait for 23 ns;
    wait until CLK_i = '0';
    KEY_VLD_i <= '1';
    KEY_OK_i  <= '1';
    wait until CLK_i = '0';
    KEY_VLD_i <= '0';
    KEY_OK_i  <= '0';
    wait until CLK_i = '0';
     -- tecla NOK
    wait until CLK_i = '0';
    KEY_VLD_i <= '1';
    KEY_OK_i  <= '1';
    wait until CLK_i = '0';
    KEY_VLD_i <= '0';
    KEY_OK_i  <= '0';
    wait for 23 ns;
    wait until CLK_i = '0';
    KEY_VLD_i <= '1';
    KEY_OK_i  <= '0';
    wait until CLK_i = '0';
    KEY_VLD_i <= '0';
    KEY_OK_i  <= '0';
      wait for 23 ns;
    wait until CLK_i = '0';
    KEY_VLD_i <= '1';
    KEY_OK_i  <= '1';
    wait until CLK_i = '0';
    KEY_VLD_i <= '0';
    KEY_OK_i  <= '0';
    wait for 23 ns;
    wait until CLK_i = '0';
    report "fin de laa simulción" severity failure;
  end process;
end sim;
```

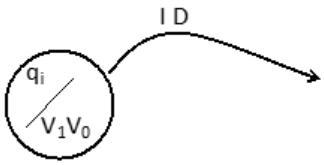
En la Figura 4.1, se muestra el sistema de control de la velocidad de un motor. El circuito de control, que debe diseñarse en este ejercicio, recibe la información de tres pulsadores: **I**, **D** y **OFF**, **activos a nivel alto**, cuyo funcionamiento se describe más adelante. En función de la información recogida en estos pulsadores, se generará la consigna de velocidad del motor mediante dos señales $\mathbf{V_1}$ y $\mathbf{V_0}$ configuradas tal y como se muestra en la Tabla 4.1.

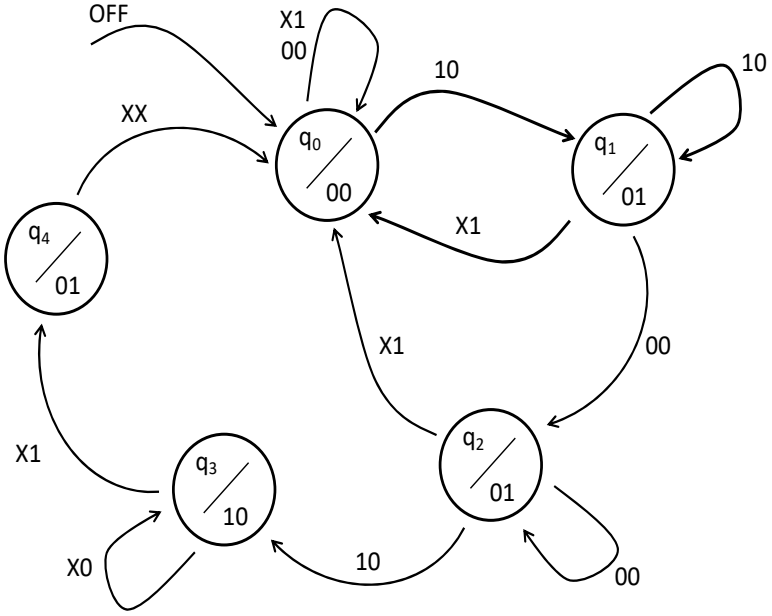
V_1 V_0	Estado del motor
00	Parado
01	Velocidad intermedia
10	Velocidad nominal
11	NO se utiliza

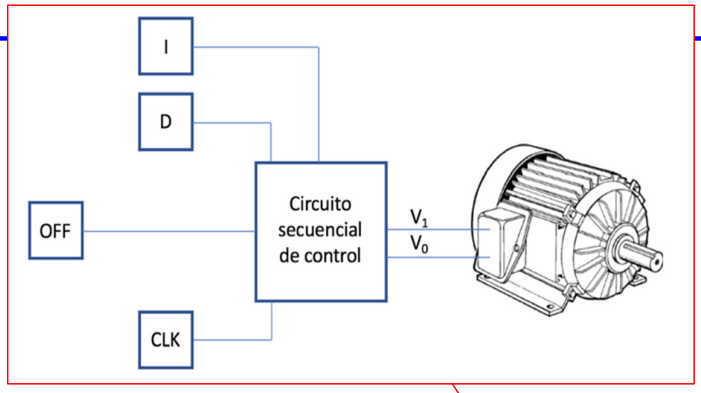


Los requisitos que debe garantizar el circuito de control son:

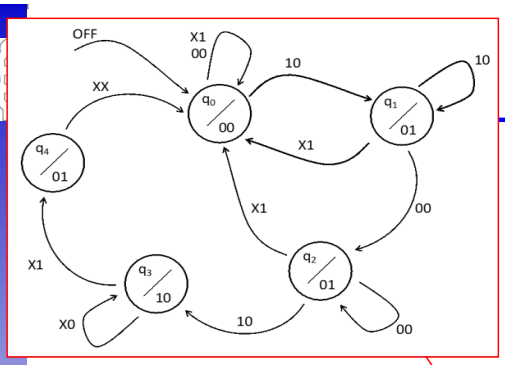
- La evolución del sistema está gobernada por la señal de reloj, **CLK,** en todos los casos en los que no se indique explícitamente lo contrario.
- Al conectar la alimentación del sistema, el motor estará parado y el circuito de control quedará preparado para comenzar a recibir las órdenes generadas por los botones. La activación de la señal **OFF** llevará el sistema, inmediatamente, a la misma situación de parada.
- En caso de activarse los pulsadores **I** y **D** simultáneamente, la activación del botón **D** es prioritaria.
- Con el motor parado, la primera pulsación del botón **I** provocará que el motor gire a su velocidad intermedia.
- Si el motor está girando a velocidad intermedia, una nueva pulsación del botón **I** llevará al motor a girar a su velocidad nominal. El diseño del sistema debe impedir que una pulsación prolongada del botón **I** lleve el motor desde la situación de parado hasta su velocidad nominal directamente. Para alcanzar la velocidad nominal estando el motor parado es necesario garantizar que el botón **I** se haya presionado, soltado y vuelto a presionar.
- La pulsación del botón **D** provocará la parada del motor. Si el motor está girando a su velocidad nominal, la parada se hará en dos pasos, haciendo que el motor gire a velocidad intermedia durante un ciclo de la señal **CLK** para pararse en el ciclo siguiente. Durante el tiempo que dura el proceso de parada, se ignorará la información de los botones **I** y **D**.



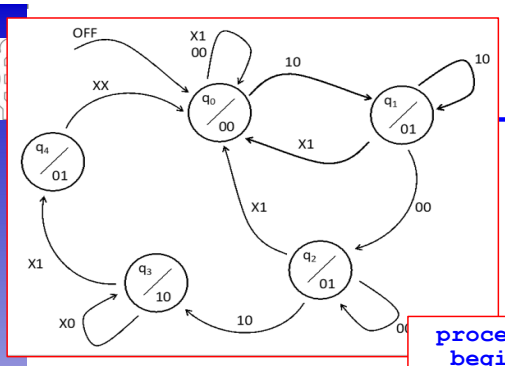


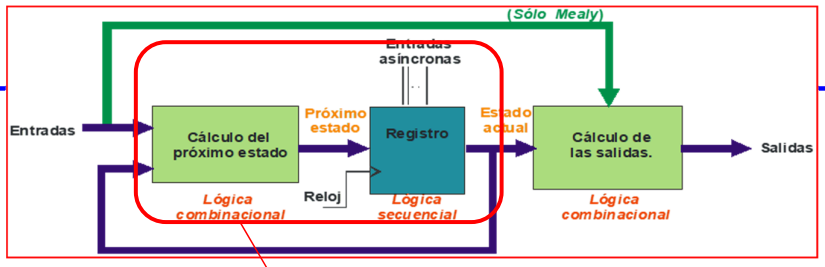



```
ibrary ieee;
use ieee.std_logic_1164.all;

entity cnt_motor is
  port ( CLK : in  std_logic;
         OFF : in  std_logic;
         I   : in  std_logic;
         D   : in  std_logic;
         V1  : out std_logic;
         V0  : out std_logic);
end cnt_motor;
```

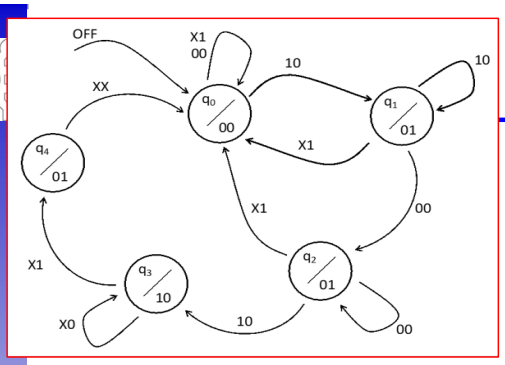


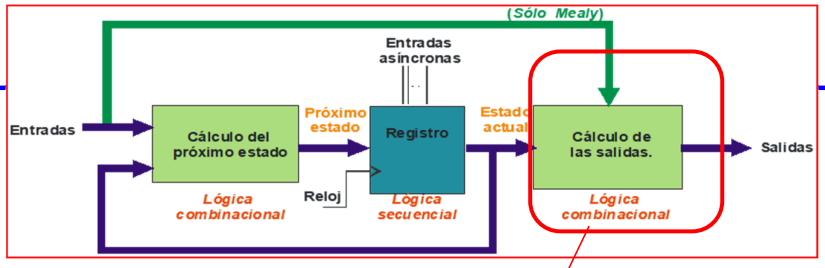

```
architecture rtl of cnt_motor is
  type tipo_estado is (q0, q1, q2, q3, q4 );
  signal estado   : tipo_estado;
  signal entradas : std_logic_vector(1 downto 0);
begin

  entradas <= I&D;
```




```
process (CLK, OFF)
 begin
   if OFF = '1' then
     estado       <= Q0;
   elsif clk'event and clk = '1' then
     case estado is
       when Q0 =>
         if entradas = "10" then
           estado <= Q1;
         end if;
       when Q1 =>
         if (entradas = "11")or (entradas = "01") then
           estado <= Q0;
         elsif entradas = "00" then
           estado <= Q2;
         end if;
       when Q2 =>
         if (entradas = "11")or (entradas = "01") then
           estado <= Q0;
         elsif entradas = "10" then
           estado <= Q3;
         end if;
       when Q3 =>
         if (entradas = "11")or (entradas = "01") then
           estado <= Q4;
         end if;
       when Q4 =>
         estado   <= Q0;
     end case;
   end if;
 end process;
```






```
  V0 <= '0' when (estado = Q3)or(estado = Q0) else '1';

  V1 <= '1' when estado = Q3 else '0';

end rtl;
```

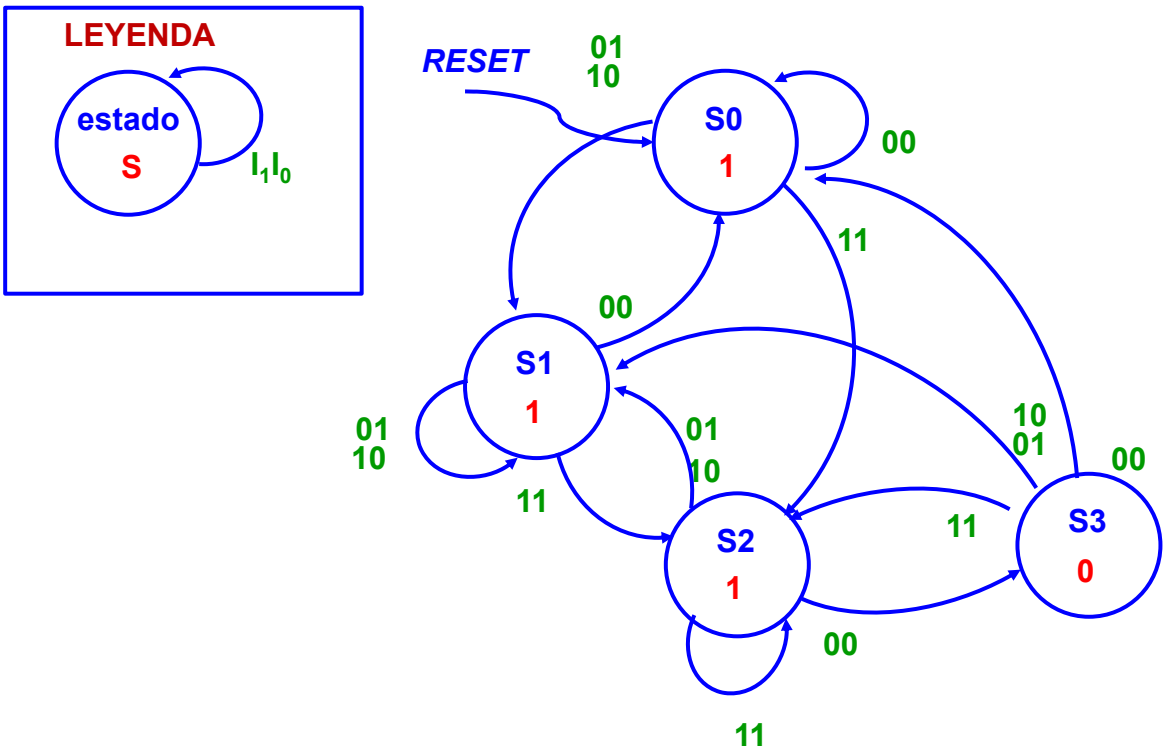


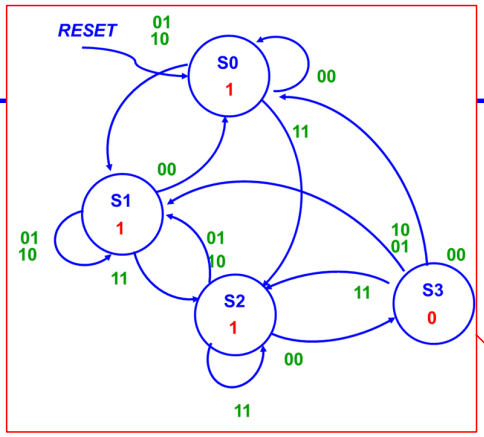
```
library ieee;
use ieee.std_logic_1164.all;
entity cnt_motor_tb is

end cnt_motor_tb;

architecture sim of cnt_motor_tb is

  signal CLK_i : std_logic := '0';
  signal OFF_i : std_logic := '1';
  signal I_i   : std_logic := '0';
  signal D_i   : std_logic := '0';
  signal V1_i  : std_logic;
  signal V0_i  : std_logic;

begin  -- sim

  DUT : entity work.cnt_motor
    port map (
      CLK => CLK_i,
      OFF => OFF_i,
      I   => I_i,
      D   => D_i,
      V1  => V1_i,
      V0  => V0_i);

  CLK_i <= not CLK_i after 5 ns;

  process
  begin  -- process
    wait for 123 ns;
    OFF_i <= '0';
    wait for 123 ns;
    wait until CLK_i = '0';
    I_i <= '1';
    wait for 23 ns;
    wait until CLK_i = '0';
    I_i <= '0';
    wait for 23 ns;
    wait until CLK_i = '0';
    I_i <= '1';
```

```
    wait for 23 ns;
    wait until CLK_i = '0';
    I_i <= '0';
    wait for 23 ns;
    wait until CLK_i = '0';
    D_i <= '1';
    wait until CLK_i = '0';
    I_i <= '1';
    D_i <= '0';
    wait for 23 ns;
    wait until CLK_i = '0';
    I_i <= '1';
    wait for 23 ns;
    wait until CLK_i = '0';
    I_i <= '0';
    wait for 23 ns;
    wait until CLK_i = '0';
    I_i <= '1';
    wait until CLK_i = '0';
    I_i <= '0';
    wait for 23 ns;
    wait until CLK_i = '0';
    I_i <= '1';
    D_i <= '1';
    wait until CLK_i = '0';
    I_i <= '0';
    D_i <= '0';
     wait until CLK_i = '0';
    I_i <= '1';
    wait for 23 ns;
    wait until CLK_i = '0';
    I_i <= '0';
    wait for 23 ns;
     OFF_i <= '1';
    wait for 123 ns;
    report "fin de laa simulción" severity failure;
  end process;

end sim;
```

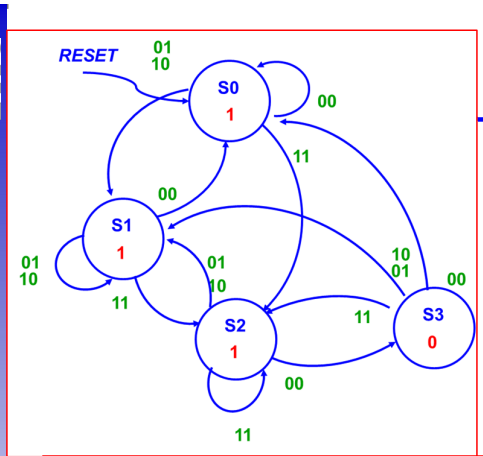
Un determinado sistema de control necesita detectar el momento en el que se producen **dos flancos de bajada simultáneos en sendas señales de entrada I_1 e I_0**. Por la propia naturaleza de estas señales, queda garantizado que sus cambios de nivel están sincronizados con los flancos de subida de la señal de reloj del sistema (***CLK***). Realice el diseño de un sistema secuencial síncrono que lleve a cabo esta detección de flancos simultáneos indefinidamente activando una salida (***S***) **a nivel bajo** durante un único periodo de la señal CLK tras los flancos. El circuito final debe disponer de una señal de reinicio asíncrona (***RESET***), activa a **nivel bajo**, que lleve la máquina a su estado inicial.

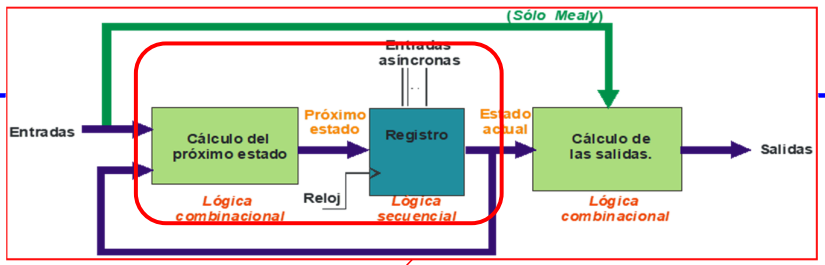
1. Obtenga en primer lugar el grafo correspondiente a la máquina de estados del circuito descrito en el enunciado, especificando claramente la notación utilizada para los estados, entradas y salidas.





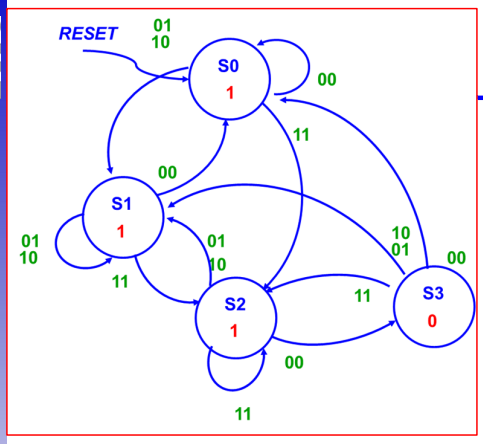

```
library ieee;
use ieee.std_logic_1164.all;

entity cnt_flancos is
  port ( CLK    : in  std_logic;
         RESET  : in  std_logic;
         I      : in  std_logic_vector(1 downto 0);
         S      : out std_logic);
end cnt_flancos;

architecture rtl of cnt_flancos is
  type tipo_estado is (S0, S1, S2, S3 );
  signal estado   : tipo_estado;

begin
```

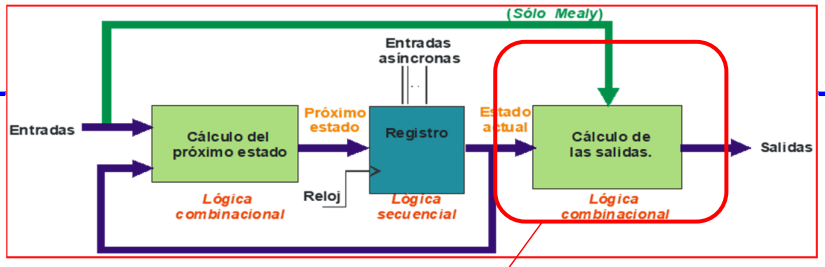






```
process (CLK, RESET)
  begin
    if RESET = '0' then
      estado          <= S0;
    elsif clk'event and clk = '1' then
      case estado is
        when S0     =>
          if (I = "10")or (I = "01") then
            estado   <= S1;
            elsif I = "11" then
              estado <= S2;
            end if;
            when S1 =>
            if (I = "11") then
              estado <= S2;
            elsif I = "00" then
              estado <= S0;
            end if;
            when S2 =>
            if (I = "10")or (I = "01") then
              estado <= S1;
            elsif I = "00" then
              estado <= S3;
            end if;
```

```
            when S3 =>
              if (I = "10")or (I = "01") then
                estado <= S1;
              elsif I = "00" then
                estado <= S0;
              else
                estado <= S2;
              end if;
          end case;
          end if;
      end process;
```






```
S <= '0' when (estado = S3) else '1';

 end rtl;
```



```
library ieee;
use ieee.std_logic_1164.all;


entity cnt_flancos_tb is

end cnt_flancos_tb;

architecture sim of cnt_flancos_tb is


signal CLK_i    : std_logic:='0';
signal RESET_i : std_logic:='0';
signal I_i      : std_logic_vector(1 downto 0):="00";
signal S_i      : std_logic;

begin  -- sim

  DUT:entity work.cnt_flancos
    port map (
        CLK    => CLK_i,
        RESET => RESET_i,
        I      => I_i,
        S      => S_i);


  CLK_i <= not CLK_i after 5 ns;
```

```
process
  begin  -- process
    wait for 123 ns;
    RESET_i <= '1';
    wait for 123 ns;
    wait until CLK_i = '0';
    I_i <= "10";
    wait for 23 ns;
    wait until CLK_i = '0';
     I_i <= "11";
    wait for 23 ns;
    wait until CLK_i = '0';
     I_i <= "10";
    wait for 23 ns;
    wait until CLK_i = '0';
    I_i <= "10";
    wait for 23 ns;
    wait until CLK_i = '0';
     I_i <= "00";
    wait until CLK_i = '0';
     I_i <= "11";
    wait for 23 ns;
    wait until CLK_i = '0';
     I_i <= "00";
    wait for 23 ns;
    wait until CLK_i = '0';
     I_i <= "11";
    wait for 23 ns;
    wait until CLK_i = '0';
    I_i <= "10";
    wait until CLK_i = '0';
    wait for 23 ns;
     RESET_i <= '0';
    wait for 123 ns;
    report "fin de la simulción" severity failure;
  end process;


end sim;
```

Se ha realizado el diseño de un detector de pulsos en una línea de datos, mediante una máquina de estados. Su funcionamiento se muestra en la figura 4.1 con una secuencia de entrada de ejemplo, y es el siguiente: el sistema detecta cuándo en la entrada ***E*** se ha recibido un pulso de cualquier duración, activándose a nivel alto la salida ***S*** durante un ciclo de *clk* una vez que E pasa a nivel bajo. *NOTA: El sistema no permite la llegada de dos pulsos consecutivos en un intervalo de tiempo inferior a un período de clk.*

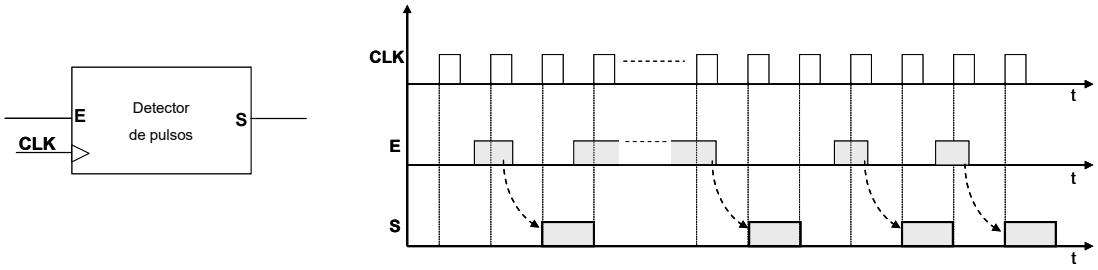
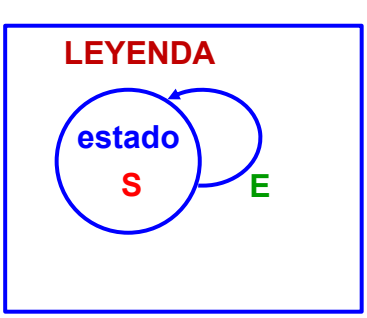
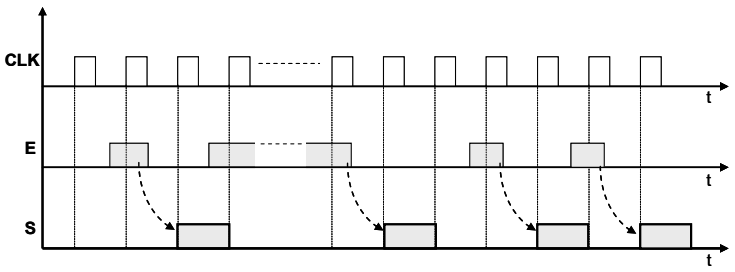
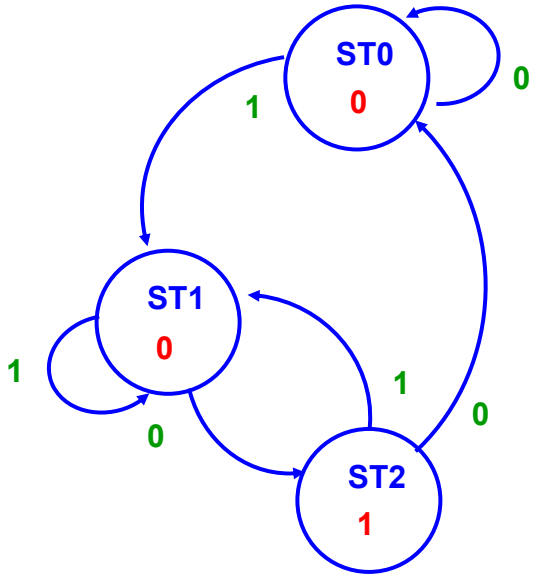


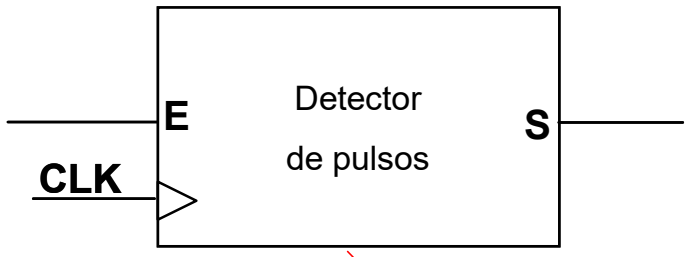
Figura 4.1. Máquina de estados y cronograma de funcionamiento

Obtenga el grafo correspondiente a la máquina de estados del detector de pulsos del enunciado, especificando claramente la notación utilizada para los estados, entradas y salidas.

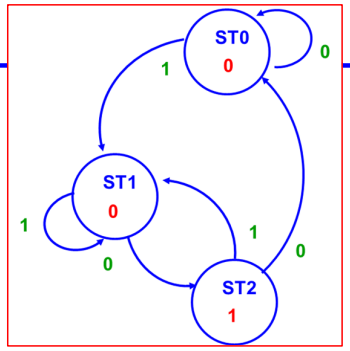


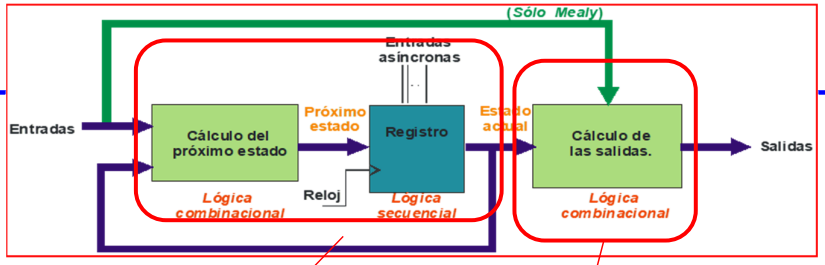







```
library ieee;
use ieee.std_logic_1164.all;

entity detect_pulsos is
  port ( CLK : in  std_logic;
         E   : in  std_logic;
         S   : out std_logic);
end detect_pulsos;

architecture rtl of detect_pulsos is
  type tipo_estado is (ST0, ST1, ST2 );
  signal estado : tipo_estado;
```




```
process (CLK)
 begin

   if clk'event and clk = '1' then
     case estado is
       when ST0 =>
         if E = '1' then
           estado <= ST1;
         end if;
       when ST1 =>
         if E = '0' then
           estado <= ST2;
         end if;
       when ST2  =>
         if E = '1' then
           estado <= ST1;
         else
           estado <= ST0;
         end if;
     end case;
   end if;
 end process;
```

```
S <= '1' when (estado = ST2) else '0';
```

```
ibrary ieee;
use ieee.std_logic_1164.all;


entity detect_pulsos_tb is

end detect_pulsos_tb;

architecture sim of detect_pulsos_tb is


  signal CLK_i : std_logic:='0';
  signal E_i   : std_logic:='0';
  signal S_i   : std_logic;

begin  -- sim

  DUT:entity work.detect_pulsos
    port map (
        CLK => CLK_i,
        E   => E_i,
        S   => S_i);


  CLK_i <= not CLK_i after 5 ns;
```

```
process
 begin  -- process
   wait for 123 ns;
   wait until CLK_i = '0';
   E_i <= '1';
   wait for 123 ns;
   wait until CLK_i = '0';
   E_i <= '0';
   wait for 23 ns;
   wait until CLK_i = '0';
   E_i <= '1';
   wait for 23 ns;
   wait until CLK_i = '0';
   E_i <= '0';
   wait for 23 ns;
   wait until CLK_i = '0';
   E_i <= '1';
    wait for 323 ns;
   wait until CLK_i = '0';
   E_i <= '0';
   wait for 223 ns;

   report "fin de la simulción" severity failure;
 end process;


end sim;
```


En muchas ocasiones es necesario disponer de un circuito digital (figura 4.1.a) que genere una señal ***E2x*** que tenga el doble de frecuencia que la una entrada ***E***, siempre y cuando esta última tenga un ciclo de trabajo del 50%. Obviamente, este sistema secuencial siempre trabajará con una señal de reloj (***CLK***) de frecuencia mucho mayor que la frecuencia de la entrada ***E***. En la figura 4.1.b se muestra el cronograma de funcionamiento del circuito.

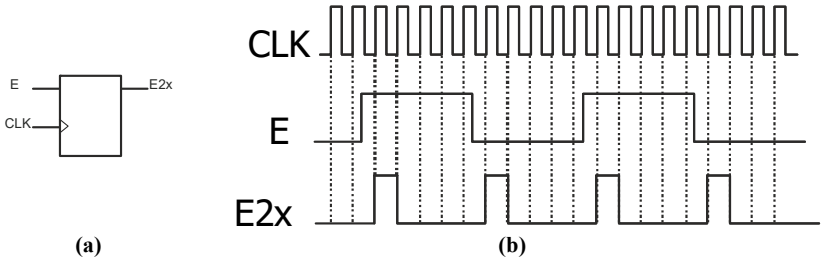
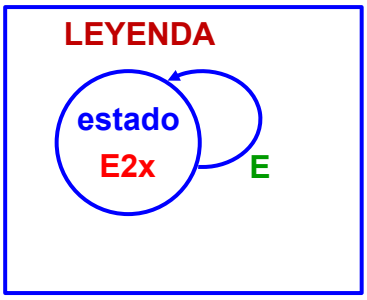
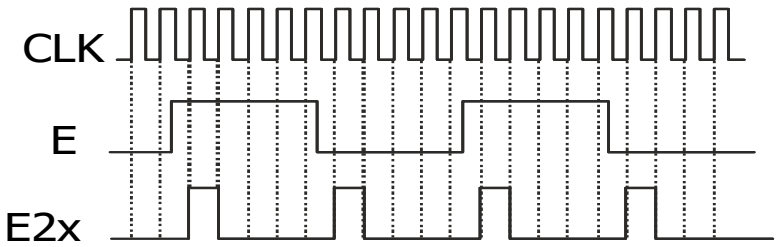
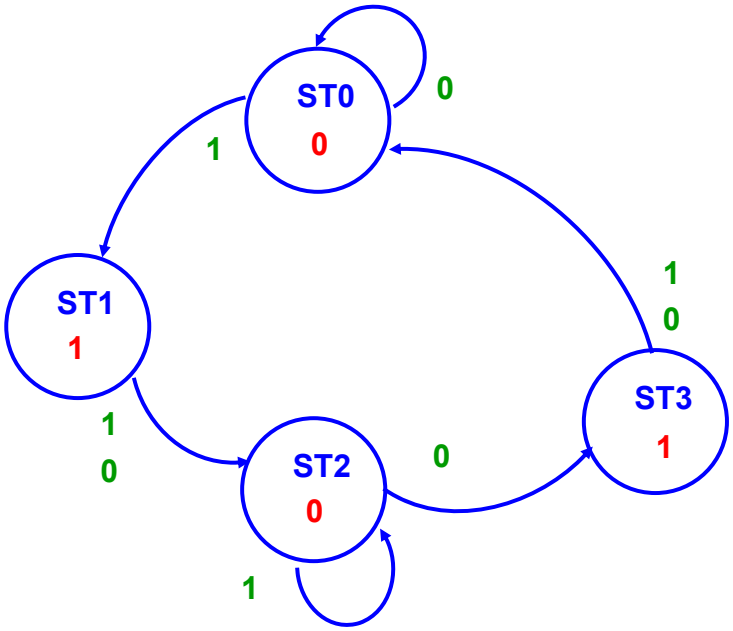


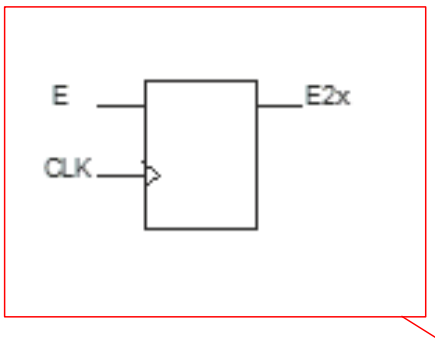
Figura 4.1. Circuito multiplicador de frecuencia. a) Terminales. b) Cronograma de funcionamiento.

Dibujar el grafo de funcionamiento del sistema secuencial

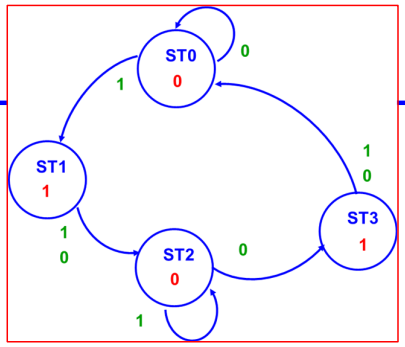


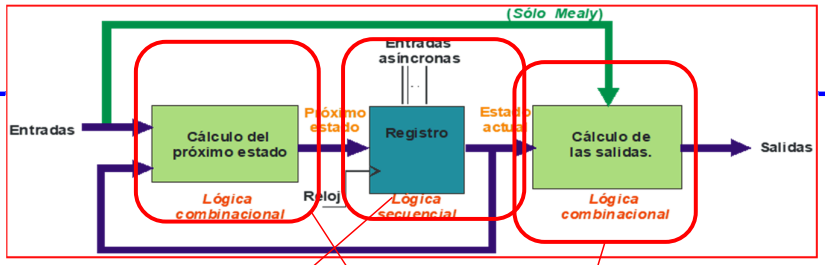







```
library ieee;
use ieee.std_logic_1164.all;
-----------------------------------------------
entity mult_frec is
  port ( e                    : in  std_logic;
         clk                  : in  std_logic;
         e2x                  : out
std_logic);
end mult_frec;
-----------------------------------------------
architecture rtl of mult_frec is
  type state is (q0, q1, q2, q3);
  signal pr_state, nx_state :     state;
begin
```






```
process (clk)
 begin
   if (clk'event and clk = '1') then
     pr_state <= nx_state;
   end if;
 end process;
```

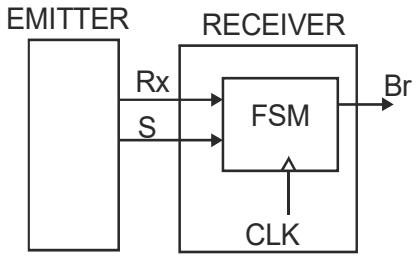
```
process (e, pr_state)
 begin
  case pr_state is
   when q0 =>
    e2x          <= '0';
    if (e = '1') then nx_state <= q1;
    else nx_state      <= q0;
    end if;
   when q1 =>
    e2x          <= '1';
    nx_state       <= q2;
   when q2 =>
    e2x          <= '0';
    if (e = '0') then nx_state <= q3;
    else nx_state      <= q2;
    end if;
   when q3 =>
    e2x          <= '1';
    nx_state       <= q0;
  end case;
 end process;
```



```
library ieee;
use ieee.std_logic_1164.all;

entity mult_frec_tb is

end mult_frec_tb;

architecture sim of mult_frec_tb is


  signal e_i   : std_logic;
  signal clk_i : std_logic:='0';
  signal e2x_i : std_logic;

begin  -- sim

  DUT : entity work.mult_frec
    port map (
      e   => e_i,
      clk => clk_i,
      e2x => e2x_i);
```

```
 CLK_i <= not CLK_i after 5 ns;

  process
  begin  -- process
    e_i <= '0';
    wait for 100 ns;
    wait until clk_i = '0';
    e_i <= '1';
    wait for 100 ns;
    wait until clk_i = '0';
  end process;


end sim;
```

Un sistema de transmisión de datos (ver Figura) está formado por un módulo emisor (*EMITTER*) y otro receptor (*RECEIVER*) utiliza un protocolo de comunicación síncrona en el que los datos se transmiten vía serie. El módulo receptor recibe todos los bits del dato por la línea ***RX*** y están sincronizados con la señal ***S***. Este módulo está formado por el bloque FSM que se corresponde con una máquina de estados finita.



El funcionamiento del sistema es el siguiente:

•Mientras no se transmita un bit del dato, la señal ***RX*** permanece a nivel alto.

•Si la señal ***RX*** permanece a nivel alto durante 2 activaciones de la señal ***S*** se considera la finalización de la transmisión de los bits del dato.

•Cada bit se transmite en dos campos: el primero siempre es 0 y el segundo corresponde con el valor del bit (0 o 1) (ver Figura 2.2).

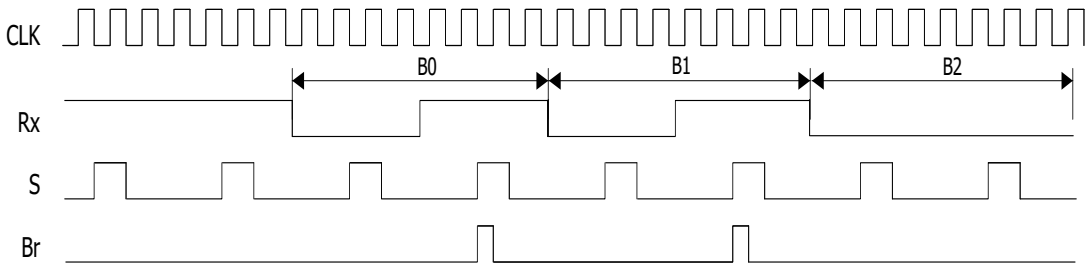
•Si cuando se transmite el primer campo de un bit la señal ***RX*** está a nivel alto, también, se considerara que ha finalizado la transmisión de los bits del dato.

•La señal ***S*** se genera continuamente, independiente de si se está transmitiendo un bit del dato o no. Esta señal se pone a nivel alto durante un periodo de la señal de ***CLK*** (ver Figura 2.2).

•El primer bit transmitido es el menos significativo (LSB), siendo irrelevante el número de bits del dato transmitido.

•La salida ***Br*** representa el valor del bit transmitido, permaneciendo a nivel bajo mientras no se esté transmitiendo ningún bit.

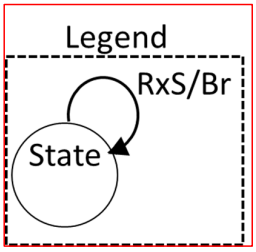
Así en la Figura, cuando se recibe un bit correspondiente a un **1** como en el caso de los bits B0 y B1, en primer lugar, la señal ***Rx*** pasa a nivel bajo durante un tiempo y a continuación a nivel alto. Así mismo cuando se recibe un bit correspondiente a un **0** como en el caso del bit B2 la señal ***Rx*** estará a nivel bajo.

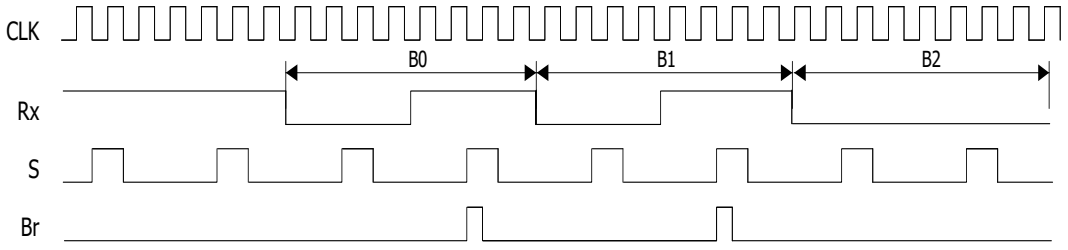


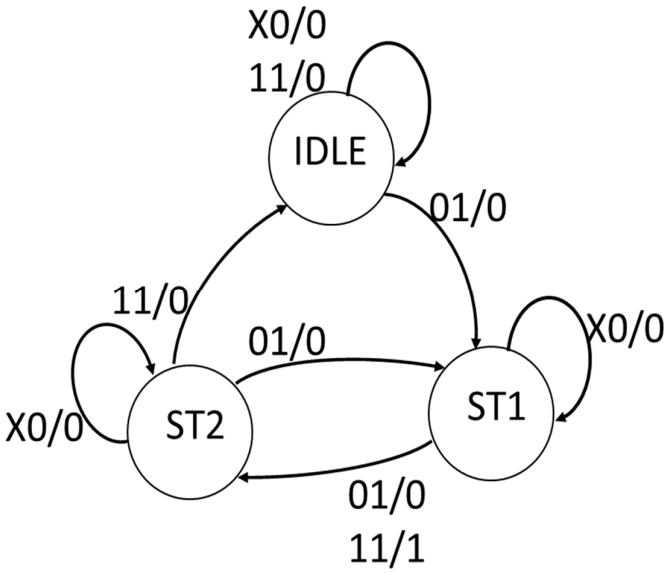
¿Qué tipo de maquina de estados, *Mealy* o *Moore*, seria necesario para implementar el sistema escrito?

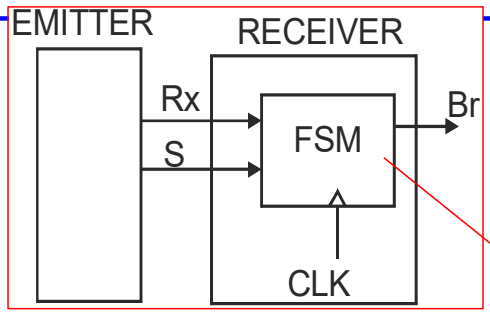
La maquina es tipo Mealy

Diseñe el grafo de la máquina de estados..

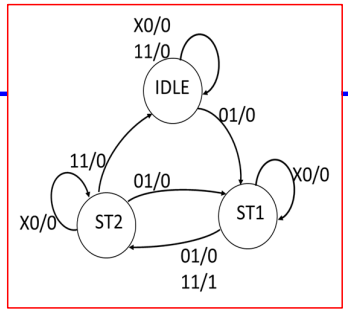


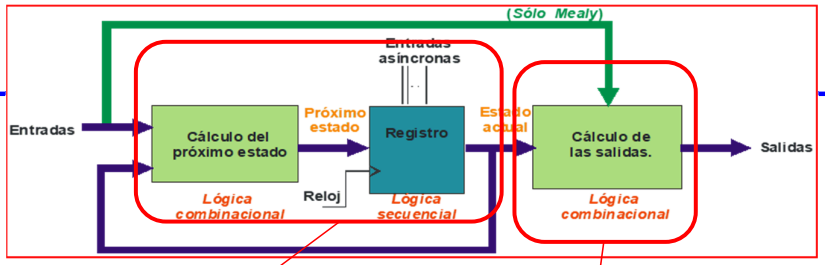







```
library ieee;
use ieee.std_logic_1164.all;

entity fsm is
  port (
    clk : in  std_logic;
    rst : in  std_logic;
    rx  : in  std_logic;
    s   : in  std_logic;
    Br  : out std_logic);
end fsm;

architecture rtl of fsm is

  type fsm_type is (idle, st1, st2);
  signal estado : fsm_type;
```






```
process (clk, rst)
 begin
   if rst = '1' then
     estado          <= idle;
   elsif clk'event and clk = '1' then
     case estado is
       when idle =>
         if s = '1' and rx = '0' then
           estado    <= st1;
         end if;
       when st1    =>
         if s = '1' then
           estado    <= st2;
         end if;
       when st2    =>
         if s = '1' then
           if rx = '0' then
             estado <= st1;
           else
             estado <= idle;
           end if;
         end if;
     end case;
   end if;
 end process;
```

```
br <= '1' when estado = st1 and s = '1' and rx = '1' else '0';
```