Universidad de Alcalá

Departamento de Electrónica

Modelado de Sistemas Computacionales

Grado en Ingeniería de Computadores

Tema 5: Modelado y síntesis de subsistemas secuenciales en VHDL.

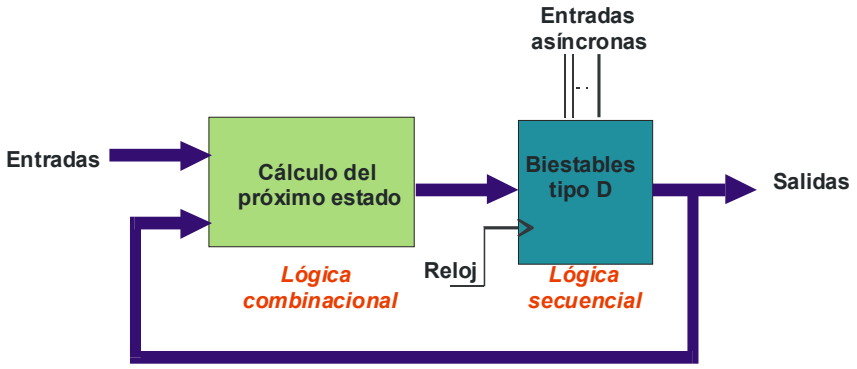
Circuitos secuenciales

- **Circuitos secuenciales.**
  - Las salidas, en cada instante, dependen sólo del valor actual de las entradas en ese mismo y instante y de la historia pasada.
  - Tienen memoria.
  - Son activos en uno, y sólo uno, de los flancos de la señal de sincronismo (reloj)

- **Tipos**:
  - Biestables
  - Contadores
  - Registros de desplazamiento
  - Maquinas de estado
  - "Memorias"

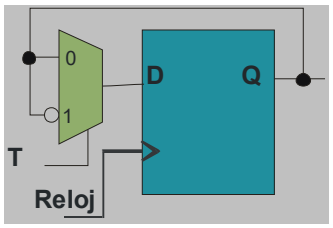
Estructura simplificada de un circuito secuencial.

- Valida para :
  - Biestables
  - Contadores
  - Registros de desplazamiento



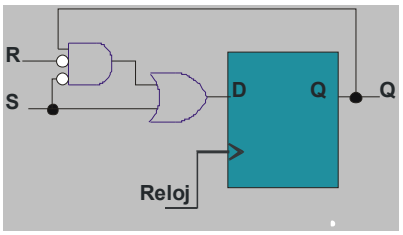
Algunos ejemplos.

Biestable T

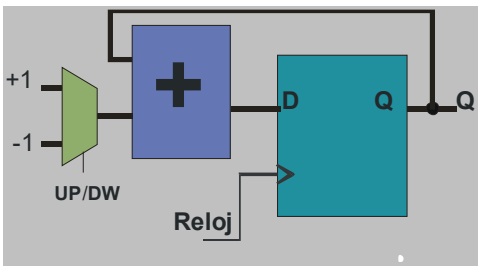


T	Q_{t+1}
0	Q_t
1	$\overline{Q}_t$

Biestable R-S



S	R	Q_{t+1}
0	0	Q_t
0	1	0
1	0	1
1	1	x



Contador ascendente/descendente

Tipos de entradas

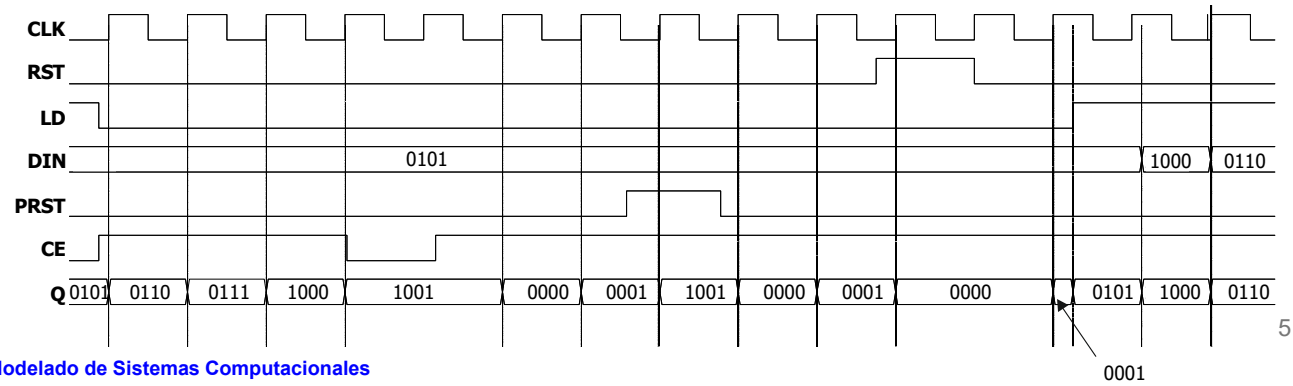
1. **Reloj**: señal que sincroniza los cambios de las salidas
2. **Asíncronas**: Son aquellas que provocan o pueden provocar un cambio en la salida del sistema secuencial en el instante en que se activan, independientemente del estado de la señal de reloj..
3. **Síncronas**: Son aquellas que provocan o pueden provocar un cambio en la salida del sistema secuencial sólo en presencia del flanco activo de la señal de reloj.

Sinc. Sinc. Sinc. Asinc. Reloj

RST	PRST	CE	LD	CLK	Q_{t+1}
X	X	X	1	X	Din
1	X	X	X	↑	0_x
0	1	X	X	↑	9_x
0	0	1	0	↑	Q_t+1
0	0	0	0	X	Q_t

Asinc.

```
entity cnt2 is
  port (
    RST, LD, PRST, CE : in  std_logic;
    CLK               : in  std_logic;
    Din               : in  std_logic_vector(3 downto 0);
    Q                 : out std_logic_vector(3 downto 0));
end cnt2;
```



- Los modelos VHDL de estos circuitos realizan mediante procesos.
- Se utiliza un sentencia ***if*** para modelar el funcionamiento secuencial.
- Esta sentencia ***if*** no tiene clausula ***else***.
- Los procesos deben incluir expresiones booleanas de detección de flanco.

```
(identificador'event and identificador='1')
(identificador'event and identificador='0')
```

→ Forma más común

```
not identificador'stable and identificador ='1'
not identificador'stable and identificador ='0'
```

→ Forma equivalentes

```
rising_edge(identificador)
falling_edge(identificador)
```

→ Funciones del paquete std_logic_1164

- Únicamente se puede detectar un flanco.
- Todos los elementos secuenciales deben ser sensibles al mismo flanco de reloj
- El número de señales de sincronismo (reloj) en un diseño debe ser pequeño.

Plantilla.

Para VHDL-2008

```
process (reloj, {señales_asíncronas})        process (all) is
   {declaración de variables};
begin
  if (señales_asíncronas_activas) then
       {sentencias secuenciales}
  elsif (señales_asíncronas_activas) then
       {sentencias secuenciales}
  elsif (expresión_booleana_de_detección_flanco) then
       {sentencias secuenciales}
  end if;
end process;
```

Modelado del funcionamiento asíncrono

Modelado del funcionamiento síncrono

El circuito sólo pude conmutar debido a las señales asíncronas o la señal de reloj.

Las señales asíncronas tienen preferencia sobre la señal de reloj.

En la lista de sensibilidad sólo aparecen las señales asíncronas y la señal de reloj.

La activación de las señales asíncronas se evalúa antes que la señal de reloj.

Si no existen señales asíncronas la plantilla se transforma en:

```
process (reloj)
   {declaración de variables};
begin
  if (expresión_booleana_de_detección_flanco)
then
     {algoritmo de funcionamiento síncrono}
  end if;
end process;
```

Circuitos secuenciales

- En el modelado del funcionamiento asíncrono sólo se evalúan las señales asíncronas.
- Se evalúan empezando por la más prioritaria y se va descendiendo en prioridad.
- La última evaluación corresponde con el flanco activo de la señal de reloj

RST	PRS	CLK	Q_{t+1}
1	X	X	0
0	1	X	1
0	1	↑	D

```
process (CLK, RST, PRS)
  begin
    if RST = '1' then
      Q <= '0';
    elsif PRS = '1' then
      Q <= '1';
    elsif CLK'event and CLK = '1' then
      Q <= D;
    end if;
  end process;
```


Tema 5: Modelado y síntesis de subsistemas secuenciales en VHDL.

Circuitos secuenciales

- Antes del ***if*** principal no se debe realizar ninguna asignación, sea a señal o a variable.

Código Erróneo

```
process (LD, CLK)
 variable aux : std_logic_vector(1 downto 0);
 variable cnt : std_logic_vector(3 downto 0);
begin
   aux:= B&C;
  if LD = '1' then
    cnt:= (others => '0');
    S1 <= '0';
    S2 <= '1';
  elsif CLK'event and CLK = '1' then
    if CE = '1' then
      S1 <= B;
      cnt := cnt(2 downto 0)&cnt(3);
      if aux = "00" then
        s2 <= '0';
        cnt := x"9";
      end if;
     end if;
  end if;
   Q1 <= cnt;
end proces
```

Código Correcto

```
process (LD, CLK)
 variable aux : std_logic_vector(1 downto 0);
 variable cnt : std_logic_vector(3 downto 0);
begin
  if LD = '1' then
    cnt:= (others => '0');
    Q1 := (others => '0');
    S1 <= '0';
    S2 <= '1';
  elsif CLK'event and CLK = '1' then
    if CE = '1' then
      S1 <= B;
      cnt := cnt(2 downto 0)&cnt(3);
      aux:= B&C;
      if aux = "00" then
        s2 <= '0';
        cnt := x"9";
      end if;
     end if;
     Q1 <= cnt;
  end if;
end process;
```

A la señal se le debe asignar una expresión con variables que se han generado en el ***if*** principal

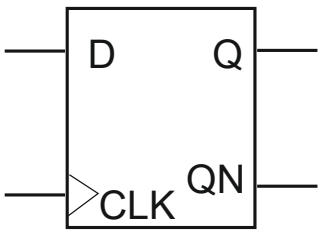
Circuitos secuenciales

- Un circuito secuencial puede generar una o más de una salida, las cuales se van a modelan utilizando el mismo proceso
- En un proceso secuencial toda asignación a señal siempre genera un biestable.
- Todas aquellas señales a las que se les asigne un valor en el modelado del funcionamiento asíncrono, también se les debe asignar un valor en el modelado del funcionamiento síncrono.
- No se usan variables para generar registros

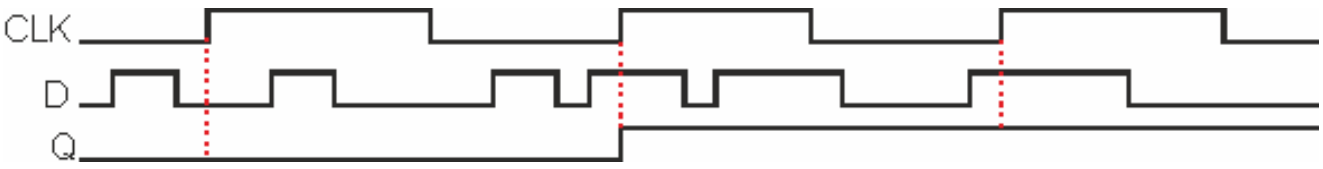
```
process (clk, a, b)
  begin
    if a = '0' then
      s0   <= "1010";
      s1   <= "0000";
    elsif b = "1100" then
      s0   <= "0110";
    elsif clk'event and clk = '1' then
      s0   <= s0+3;
      if c = '1' then
        s1 <= s1+1;
      else
        s1 <= s1-1;
      end if;
    end if;
  end proces
```


Biestables

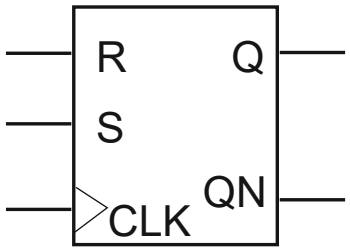
- Biestable D



CLK	D	Q_{t+1}	QN_{t+1}
↓	0	0	1
↓	1	1	0
0	X	Q_t	QN_t
1	X	Q_t	QN_t

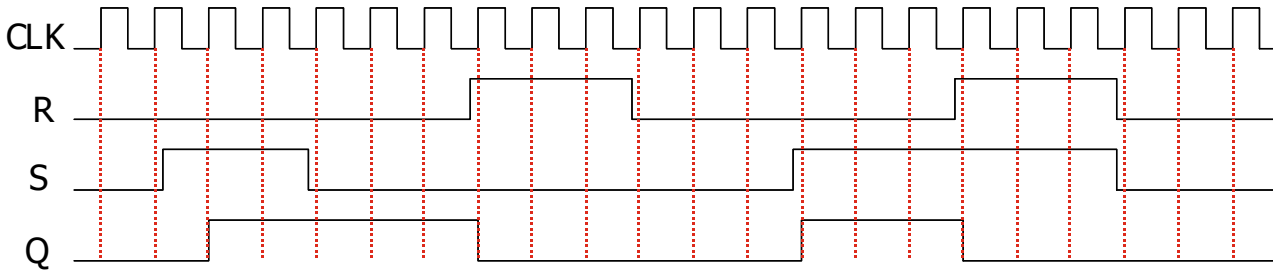


Biestable R-S

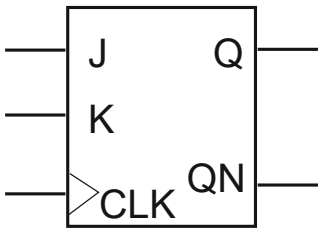


Supuesto **R** prioritaria

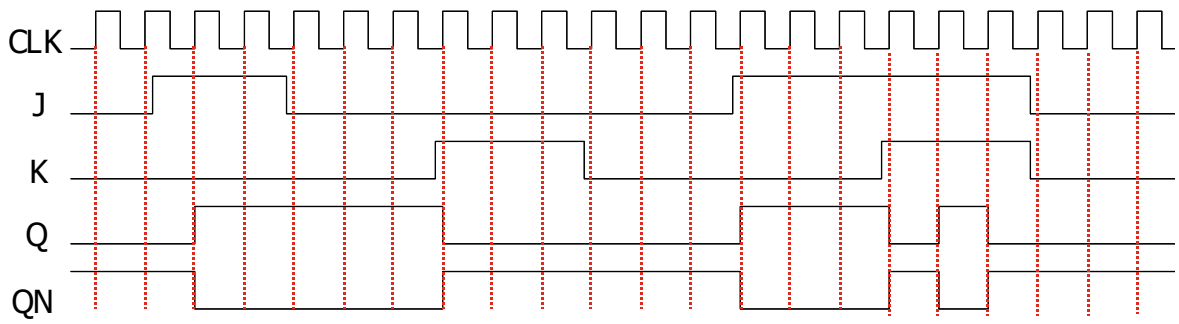
R	S	CLK	Q_{t+1}	QN_{t+1}
0	0	↓	Q_t	QN_t
1	0	↓	0	1
0	1	↓	1	0
1	1	↓	0	1
X	X	0	Q_t	QN_t
X	X	1	Q_t	QN_t



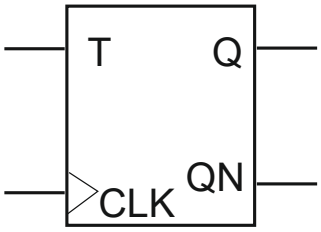
Biestable J-K



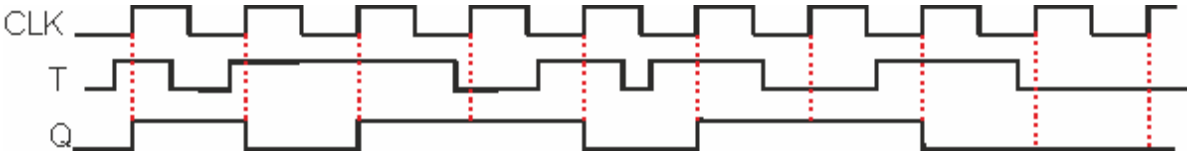
J	K	CLK	Q_{t+1}	QN_{t+1}
0	0	↓	Q_t	QN_t
0	1	↓	0	1
1	0	↓	1	0
1	1	↓	QN_t	Q_t
X	X	0	Q_t	QN_t
X	X	1	Q_t	QN_t



Biestable T



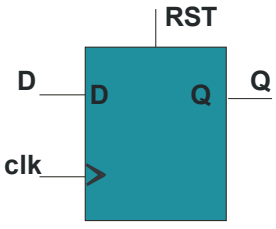
T	CLK	Q_{t+1}	QN_{t+1}
0	↑	Q_t	QN_t
1	↑	QN_t	Q_t
X	0	Q_t	QN_t
X	1	Q_t	QN_t



Biestable tipo D.

Con señal de inicialización asíncrona

RST	CLK	Q_{t+1}
1	X	0
0	↑	D



Para VHDL-2008

```
process (CLK, RST)
begin
 if RST = '1' then
  Q <= '0';
 elsif CLK'event and CLK = '1' then
  Q <= D;
 end if;
end process;
```

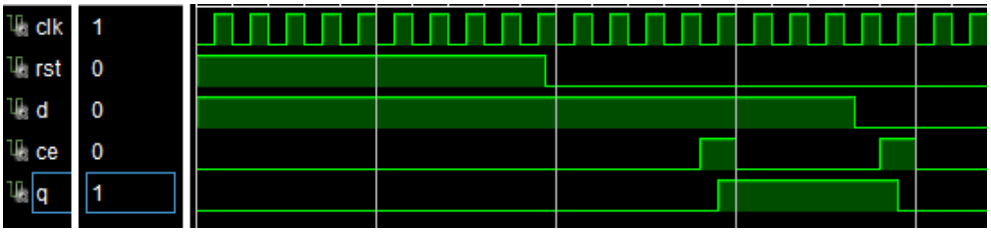
```
process (all)
begin
 if RST = '1' then
  Q <= (others=>'0');
 elsif CLK'event and CLK = '1' then
  Q <= D;
 end if;
end process;
```

El modelado de registros es idéntico al de los biestables D salvo por el hecho de que la entrada y salida de datos serán vectores.

- Biestable tipo D.
  - Con señal de inicialización asíncrona.
  - Con señal de habilitación (*Clock Enable*).

RST	CE	CLK	Q_{t+1}
1	X	X	0
0	1	↑	D
0	0	X	Q_t

```
process (CLK, RST)
  begin
    if RST = '1' then
      Q   <= '0';
    elsif CLK'event and CLK = '1' then
      if CE = '1' then
        Q <= D;
      end if;
    end if;
  end process;
```



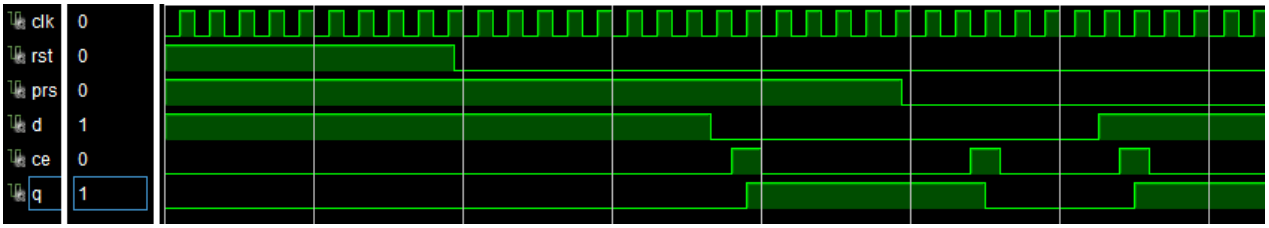
¡Este código no es sintetizable!

```
process (CLK, RST)
  begin
    if RST = '1' then
      Q   <= '0';
    elsif (CLK'event and CLK = '1')and (Ce='1') then
        Q <= D;
    end if;
  end process;
```


- Biestable tipo D.
  - Con señal de reset asíncrona.
  - Con señal de preset síncrona
  - Con señal de Clock Enable.

RST	CE	PRS	CLK	Q_{t+1}
1	X	X	X	0
0	1	1	↑	1
0	1	0	↑	D
0	0	X	X_t	Q_t

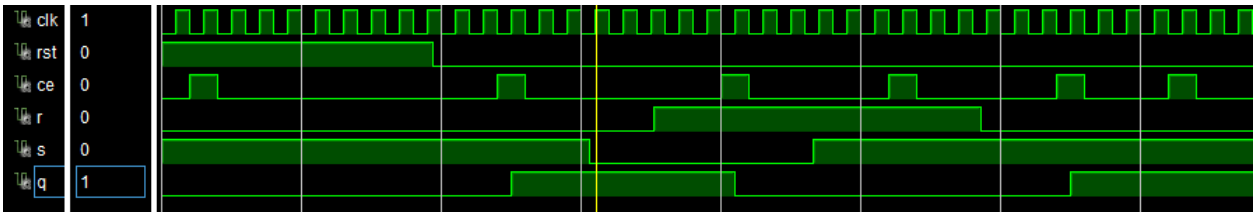
```
process (CLK, RST)
 begin
   if RST = '1' then
     Q     <= '0';
   elsif CLK'event and CLK = '1' then
     if CE = '1' then
       if PRS = '1' then
         Q <= '1';
       else
         Q <= D;
       end if;
     end if;
   end if;
 end process;
```



- Biestable tipo R-S.

RST	CE	R	S	CLK	Q_{t+1}
1	X	X	X	X	0
0	1	1	0	↑	0
0	1	0	1	↑	1
0	0	0	0	X	Q_t

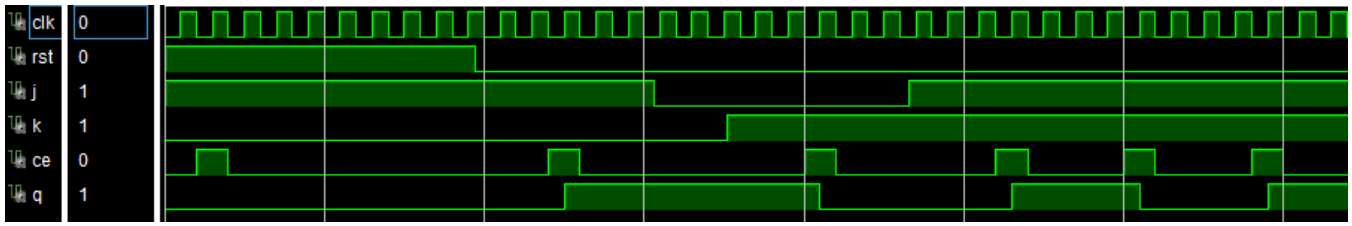
```
process (CLK, RST)
  begin
    if RST = '1' then
      Q     <= '0';
    elsif CLK'event and CLK = '1' then
      if CE = '1' then
        if R = '1' then
          Q <= '0';
        elsif S = '1' then
          Q <= '1';
        end if;
      end if;
    end if;
  end process;
```



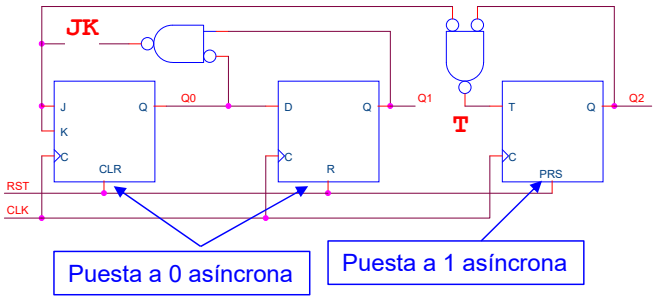
- Biestable tipo J-K.

RST	CE	J	K	CLK	Q_{t+1}
1	X	X	X	X	0
0	1	1	0	↑	1
0	1	0	1	↑	0
0	1	1	1	↑	Not(Q_t)
0	0	0	0	X	Q_t

```
process (CLK, RST)
  begin
    if RST = '1' then
      Q     <= '0';
    elsif CLK'event and CLK = '1' then
      if CE = '1' then
        if J = '1' and K = '0' then
          Q <= '1';
        elsif J = '0' and K = '1' then
          Q <= '0';
        elsif J = '1' and K = '1' then
          Q <= not Q;
        end if;
      end if;
    end if;
  end process;
```



- Otro ejemplo.



Cuando en un diseño se deben modelar varios biestables que comparten las mismas señales asíncronas y de sincronismo se pueden modelar utilizando un mismo proceso

```
process (clk, rst)
  begin  -- process
    if rst = '0' then
      q0   <= '0';
      q1   <= '0';
      q2   <= '1';
    elsif clk'event and clk = '1' then
      if (not q0 nand q1) = '1' then
        q0 <= not q0;
      end if;
      q1   <= q0;
      if (not (not q0 nand q1) nand not q2)= '1' then
        q2 <= not q2;
      end if;
    end if;
  end process;
```

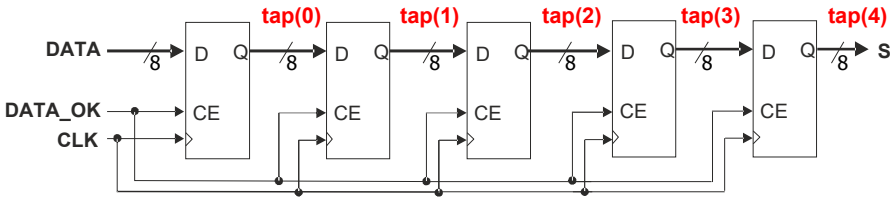
```
T<=(not (not q0 nand q1)) nand (not q2);
JK<=(not q0) nand q1;

process (clk, rst)
  begin  -- process
    if rst = '0' then
      q0   <= '0';
      q1   <= '0';
      q2   <= '1';
    elsif clk'event and clk = '1' then
      if JK = '1' then
        q0 <= not q0;
      end if;
      q1   <= q0;
      if T= '1' then
        q2 <= not q2;
      end if;
    end if;
  end process;
```


- Registros de desplazamiento.
  - Se utilizan vectores.
  - Se utiliza la concatenación (&) para realizar los desplazamientos.

Función	RST	SELEC	CLK	Q_{t+1}			
				Q[3]	Q[2]	Q[1]	Q[0]
Inicialización a 0	0	X	X	0	0	0	0
Mantener valor	1	0 0	X	Q[3]	Q[2]	Q[1]	Q[0]
Desplazamiento a derechas	1	0 1	↑	SRI	Q[3]	Q[2]	Q[1]
Desplazamiento a izquierdas	1	1 0	↑	Q[2]	Q[1]	Q[0]	SLI
Carga paralela	1	1 1	↑	D[3]	D[2]	D[1]	D[0]

```
process(CLK, RST)
    variable selec : std_logic_vector(1 downto 0);
  begin
    if (RST = '0') then
      Q<= (others => '0');
    elsif(CLK'event and CLK = '1') then
       selec := S1 & S0;
      case selec is
        when "01" =>
          Q <= sri & Q(3 downto 1);
        when "10" =>
          Q <= Q(2 downto 0) & sli;
        when "11" =>
          Q <= D;
        when others =>
          null;
      end case;
    end if;
  end process;
```

- Registros de desplazamiento con vectores.




```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity shf_array is
  port( CLK  : in  std_logic;
        RST  : in  std_logic;
        CE   : in  std_logic;
        DATA : in  std_logic_vector( 7 downto 0);
        S    : out std_logic_vector( 7 downto 0));
end shf_array;

architecture rtl of shf_array is

  type tap_type is array (4 downto 0) of std_logic_vector (7 downto 0);
  signal tap : tap_type;

begin

  process(clk, rst)
  begin
    if(rst = '1') then
      tap   <= (others => (others => '0'));
    elsif(clk'event and clk = '1') then
      if(CE = '1') then
        tap <= tap(3 downto 0)& DATA;
      end if;
    end if;
  end process;

  S <= tap(4);
end rtl;
```

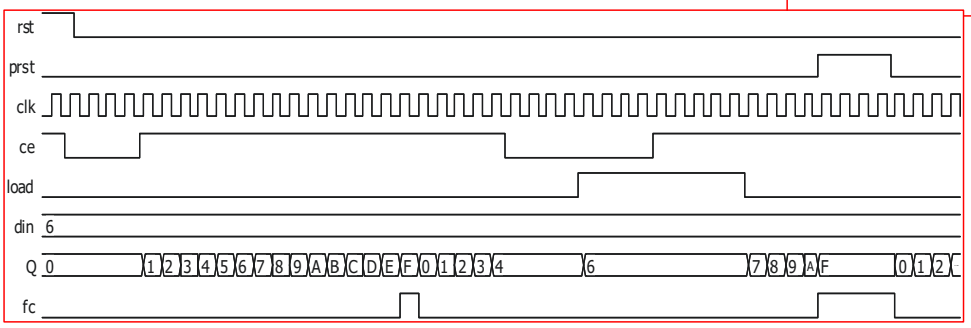

- Contadores.
  - En la descripción del algoritmo de funcionamiento síncrono aparecen operaciones de suma y/o resta.
  - El valor de cuenta se mantiene en una:
    - Señal: Si el valor desea ser consultado fuera del proceso
    - Variable: Si el valor no se desea consultar fuera del proceso
  - Los objetos que almacenan el valor de la cuenta pueden ser de tipo:
    - ***Integer***:
      - Cuando lo que interesa es conocer solo los valores que toma el contador.
      - Cuando el Número de valores es grande.
    - ***Unsigned***:
      - cuando se desea acceder a los bits de la cuenta.
      - Cuando la secuencia va de 0 a 2^n-1. Donde ***n*** es el número de bits del contador.

Contador binario

RST	PRST	CE	LOAD	CLK	Q_{t+1}
1	X	X	X	X	0_x
0	1	X	X	X	F_x
0	0	X	1	↑	Din
0	0	1	0	↑	Q_t+1
0	0	0	0	X	Q_t

No es necesario detectar que Q ha alcanzado el valor "1111"

```
signal Q   : unsigned(3 downto 0);
signal rst, prst, load : std_logic;
signal ce, clk, fc     : std_logic;
signal din:std_logic_vector (3 downto0);
...
  process (clk, rst, prst)
  begin
    if (rst = '1') then
      Q   <= (others => '0');
    elsif (prst = '1') then
      Q   <= (others => '1');
    elsif (clk'event and clk = '1')then
      if load = '1' then
        Q <= unsigned(din);
      elsif (ce = '1') then
        Q <= Q+1;
      end if;
    end if;
  end process;
  fc <= '1' when q = x"f" else '0';
```



Contador binario

RST	PRST	CE	LOAD	CLK	Q_{t+1}
1	X	X	X	X	0_x
0	1	X	X	X	F_x
0	0	X	1	X	Din
0	0	1	0	↑	Q_t+1
0	0	0	0	X	Q_t

```
signal Q   : unsigned(3 downto 0);
signal rst, prst, load : std_logic;
signal ce, clk, fc     : std_logic;
signal din:std_logic_vector (3 downto0);
...
  process (clk, rst, prst,load,din)
  begin
    if (rst = '1') then
      Q   <= (others => '0');
    elsif (prst = '1') then
      Q   <= (others => '1');
    elsif load = '1' then
        Q <= unsigned(din);
    elsif (clk'event and clk = '1')then
     if (ce = '1') then
        Q <= Q+1;
      end if;
    end if;
  end process;
  fc <= '1' when q = x"f" else '0';
```

❑Contadores: variación del fin de cuenta

Contador BCD ascendente/descendente

RST	PRST	LOAD	CE	UD	CLK	Q_{t+1}
1	X	X	X	X	X	0_x
0	1	X	X	X	X	9_x
0	0	1	X	X	X	D
0	0	0	1	1	↑	Q_t+1
0	0	0	1	0	↑	Q_t-1
0	0	0	0	X	X	Q_t

El paquete *numeric_std* permite comparar *unsigned* con *integer*.

```
signal Q  : unsigned(3 downto 0);
signal rst, prst, ld : std_logic;
signal ce, clk, ud     : std_logic;
signal D:std_logic_vector (3 downto0);
...
 process (CLK, RST, PRST, LD, D)
begin
 if RST = '1' then
   Q       <= (others => '0');
 elsif PRST = '1' then
   Q       <= "1001";
 elsif LD = '1' then
   Q       <= unsigned(D);
 elsif CLK'event and CLK = '1' then
   if CE = '1' then
     if UD = '1' then
       if Q = 9 then
         Q <= (others => '0');
       else
         Q <= Q+1;
        end if;
     else
       if Q = 0 then
         Q <= x"9";
       else
         Q <= Q-1;
       end if;
     end if;
   end if;
 end if;
end process;
```


Contadores

Tema 5: Modelado y síntesis de subsistemas secuenciales en VHDL.

❑Contadores: variación del fin de cuenta

Contador BCD ascendente/descendente

RST	PRST	LOAD	CE	UD	CLK	Q_{t+1}
1	X	X	X	X	X	0_x
0	1	X	X	X	X	9_x
0	0	1	X	X	X	D
0	0	0	1	1	↑	Q_t+1
0	0	0	1	0	↑	Q_t-1
0	0	0	0	X	X	Q_t


```
signal Q  : unsigned(3 downto 0);
signal rst, prst, ld : std_logic;
signal ce, clk, ud     : std_logic;
signal D:std_logic_vector (3 downto0);
...
 process (CLK, RST, PRST, LD, D)
begin
 if RST = '1' then
   Q       <= (others => '0');
 elsif PRST = '1' then
   Q       <= "1001";
 elsif LD = '1' then
   Q       <= unsigned(D);
 elsif CLK'event and CLK = '1' then
   if CE = '1' then
     if UD = '1' then
           Q <= Q+1;
       if Q = 9 then
         Q <= (others => '0');
        end if;
     else
           Q <= Q-1;
       if Q = 0 then
         Q <= x"9";
       end if;
     end if;
   end if;
 end if;
end process;
```


Modelado de Sistemas Computacionales

Contador BCD de 2 dígitos

```
signal Unid,Dec : unsigned(3 downto 0);

...

process(clk, rst)
begin
    if rst='1' then
       Unid<=(others=>'0');
       Dec <=(others=>'0');
    elsif CLK'event and CLK = '1' then
          Unid<=Unid+1;
       if Unid=9 then
          Unid<=(others=>'0');
          Dec<=Dec+1;
         if Dec=9 then
             Dec<=(others=>'0');
         end if;
       end if;
   end if;
end process;
```


Contadores: divisores de frecuencia

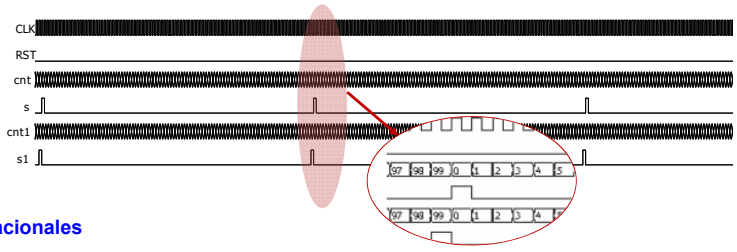
Es recomendable utilizar una constante que almacene el valor máximo de la cuenta

Se dimensiona el vector en función del valor máximo de la cuenta

```
constant K   : integer := 100;
signal    cnt1: unsigned (6 downto 0);
...
process (CLK, RST)
begin
 if RST = '1' then
   cnt1   <= (others => '0');
 elsif CLK'event and CLK = '1' then
   if cnt1 = K1-1 then
     cnt1   <= (others => '0');;
   else
     cnt1 <= cnt1+1;
   end if;
 end if;
end process;

 s1 <= '1' when cnt1 = K-1 else '0';
```

Contadores: divisores de frecuencia

```
constant K   : integer := 100;
signal    cnt1: unsigned (6 downto 0);
...
process (CLK, RST)
begin
 if RST = '1' then
   cnt1   <= (others => '0');
   s1 <= '0';
 elsif CLK'event and CLK = '1' then

   if cnt1 = K1-1 then
     cnt1 <= 0;
     s1 <= '1';
   else
     cnt1 <= cnt1+1;
     s1 <= '0';
   end if;
 end if;
end process;
```



Contadores: divisores de frecuencia

Utilización de enteros (*integer*)

Se acota el rango de valores

```
constant K   : integer := 100;
signal    cnt1:integer range 0 to K-1;
...
process (CLK, RST)
begin
 if RST = '1' then
   cnt1   <= 0;
 elsif CLK'event and CLK = '1' then
   if cnt1 = K1-1 then
     cnt1   <= 0;;
   else
     cnt1 <= cnt1+1;
   end if;
 end if;
end process;

 s1 <= '1' when cnt1 = K-1 else '0';
```

Interconexión de circuitos secuenciales

- Diseño asíncrono: coexisten varias señales de sincronismo.

- Los diseños asíncronos son menos fiables/estables:
  - El valor de los retados determina si el sistema funciona o no.
  - Se necesitan más señales de reloj.

Tema 5: Modelado y síntesis de subsistemas secuenciales en VHDL.

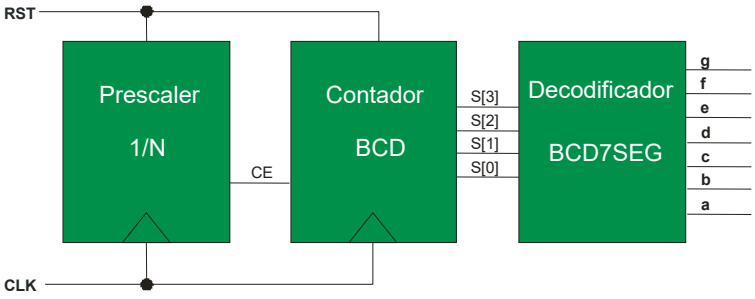
Interconexión de circuitos secuenciales

- Diseño síncrono: la señal de sincronismo es común.

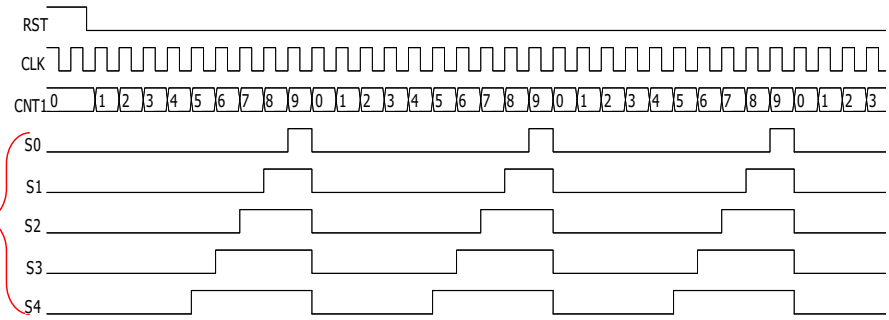
- Los diseños síncronos son más fiables:
  - Los eventos están sincronizados con flancos de reloj que se producen en intervalos perfectamente definidos.
  - Las salidas de una etapa disponen de un ciclo de reloj para propagarse

Interconexión de circuitos secuenciales

N=10

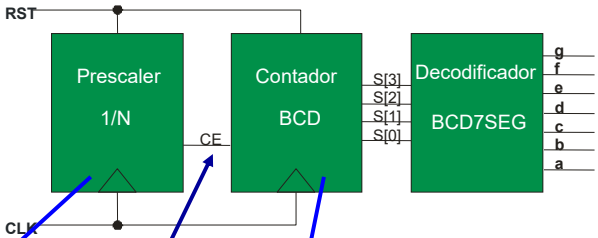


CE=?



Tema 5: Modelado y síntesis de subsistemas secuenciales en VHDL.

Interconexión de circuitos secuenciales



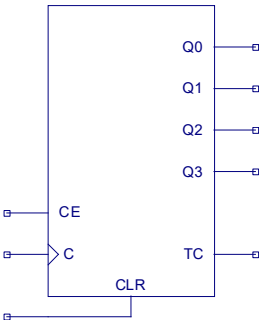

```
signal   Q   : unsigned(3 downto 0);
signal   cnt : unsigned(3 downto 0);
...
  process (CLK, RST)
  begin  -- process
    if RST = '1' then
      cnt   <= (others => '0');
    elsif CLK'event and CLK = '1' then
      if cnt = 9 then
        cnt <= 0;
      else
        cnt <= cnt+1;
      end if;
    end if;
  end process;
```

```
process (CLK, RST)
  begin
    if RST = '1' then
      Q     <= (others => '0');
    elsif CLK'event and CLK = '1' then
      if cnt = 9 then
        if Q = 9 then
          Q <= (others => '0');
        else
          Q <= Q+1;
        end if;
      end if;
    end if;
  end process;
```

Asíncrona

Síncrona

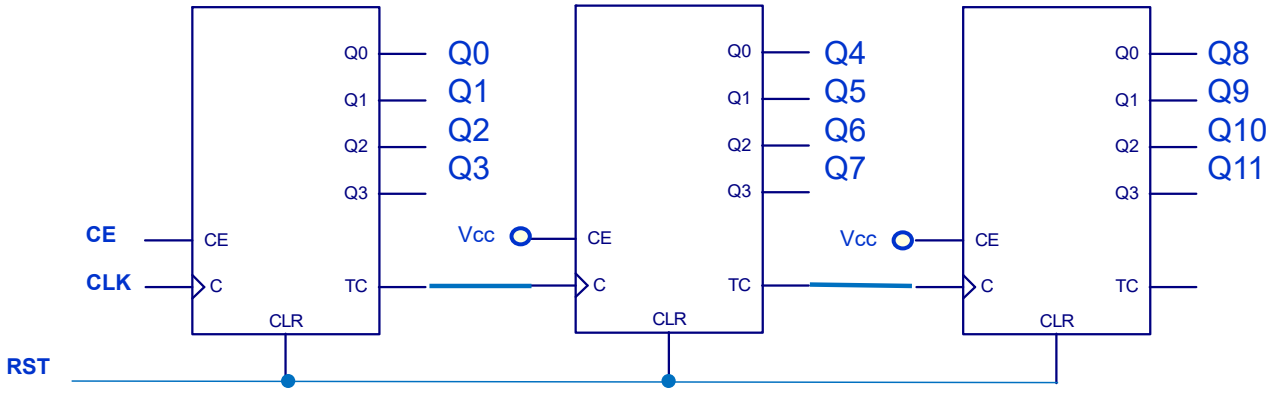
CLR	CE	C	$Q3_{t+1}$	$Q2_{t+1}$	$Q1_{t+1}$	$Q0_{t+1}$
1	X	X	0	0	0	0
0	0	X	$Q3_t$	$Q2_t$	$Q1_t$	$Q0_t$
0	1	↑	cuenta ascendente			



TC = Q3•Q2•Q1•Q0

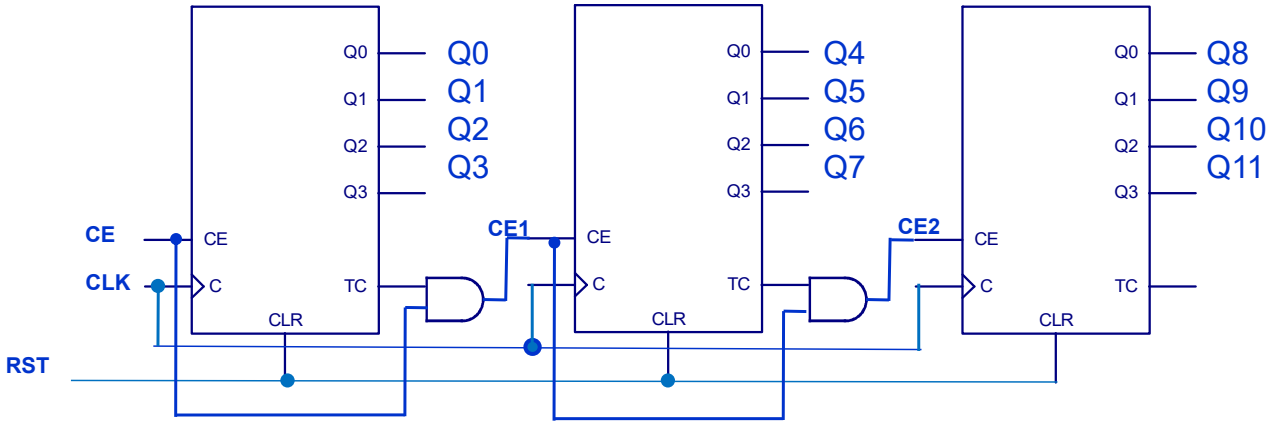
¿Cómo consigo un contador de 12 bits?

☹ Diseño muy malo



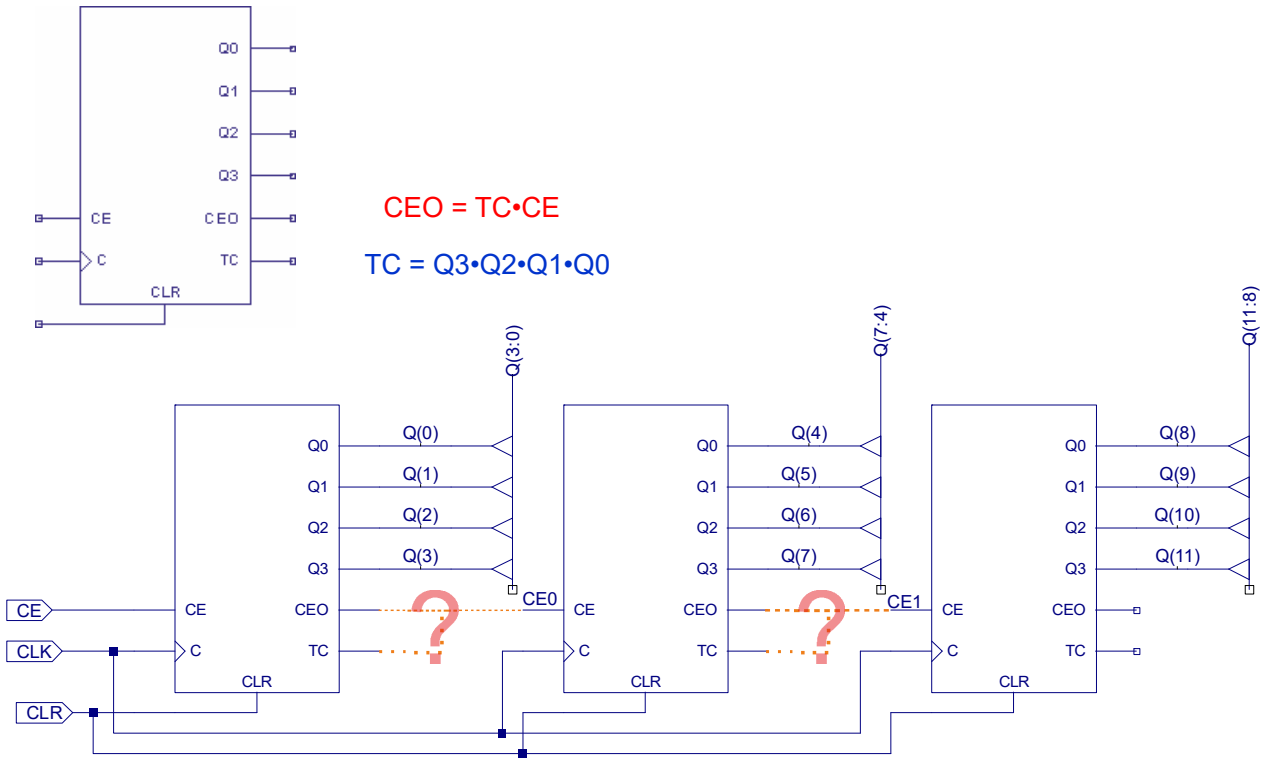
☹ Y no funciona

Interconexión de circuitos secuenciales

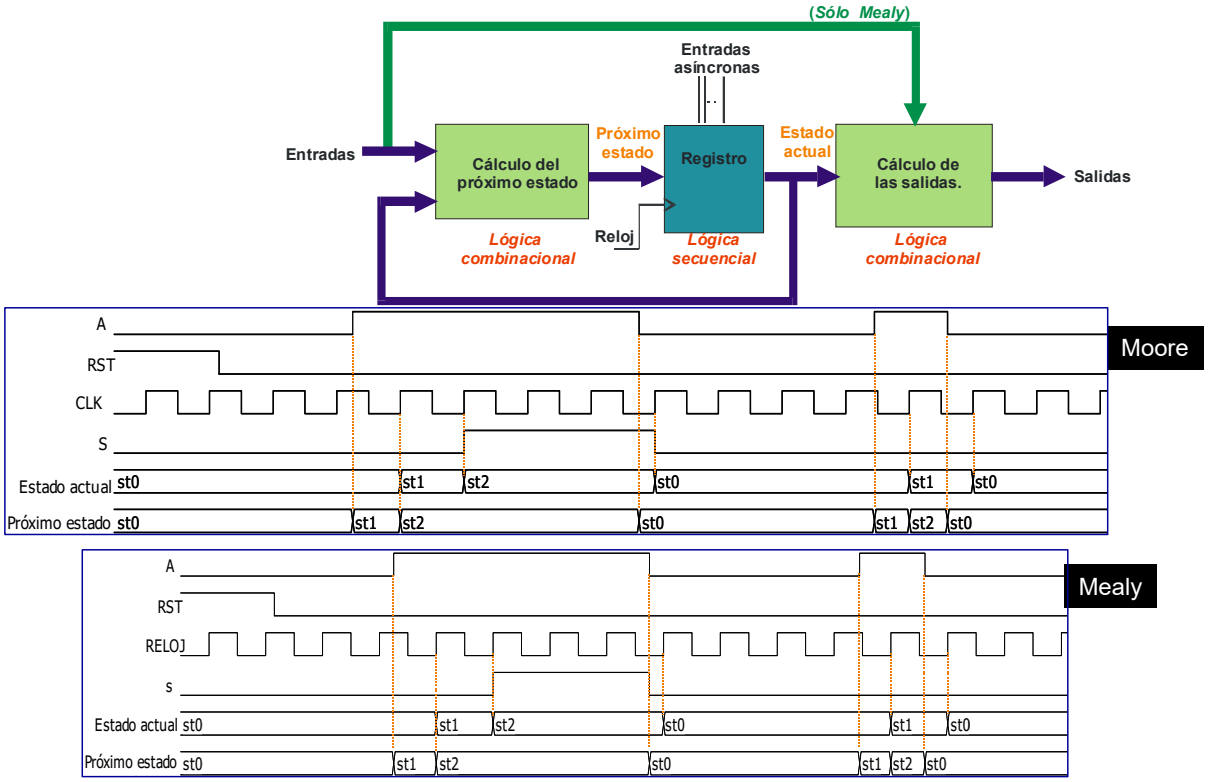


☺ Buen diseño

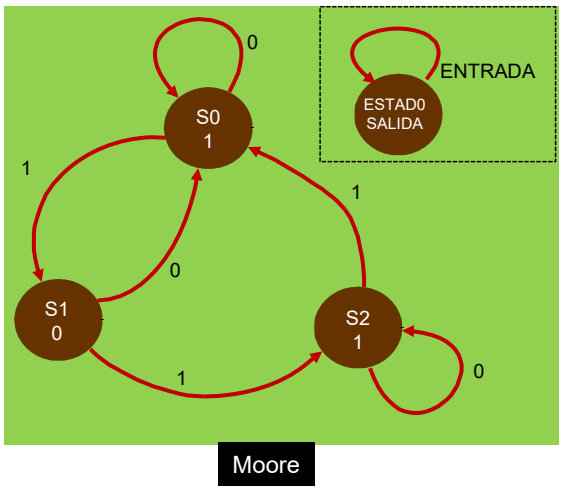
Interconexión de circuitos secuenciales

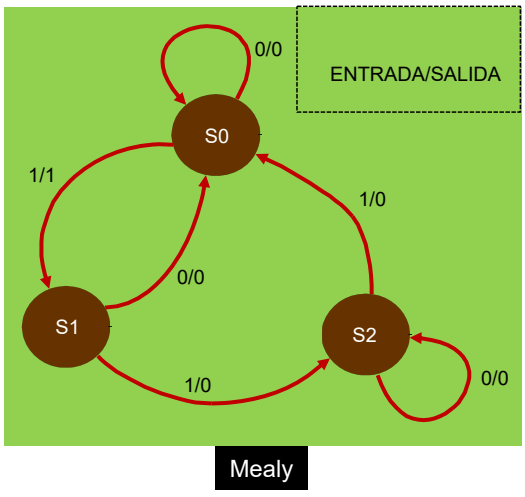


- Maquinas de estados.



- El funcionamiento viene reflejado con un grafo.

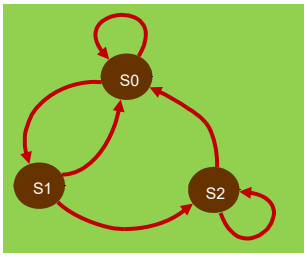




Maquinas de estados

- Hay distintas formas de modelado dependiendo de los procesos empleados para representar los distintos bloques.
- El estado de la máquina se representa mediante un tipo de datos enumerado.

```
type identificador _tipo is ({lista de estadodos});
signal {identificadore_señales}: identificador _tipo;
```

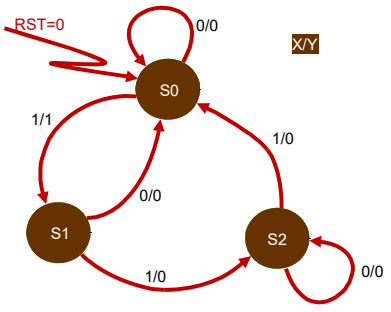



```
type type_FSM is (S0, S1, S2);
signal std_act, prox_std :type_FSM;
```

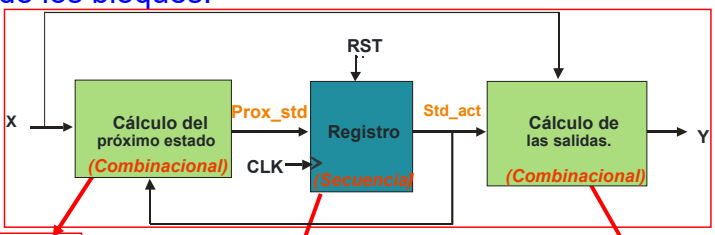

Tema 5: Modelado y síntesis de subsistemas secuenciales en VHDL.

Maquinas de estados

- Modelado con un proceso para cada uno de los bloques.




```
type FSM is (S0, S1, S2);
signal std_act, prox_std :FSM;
```




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

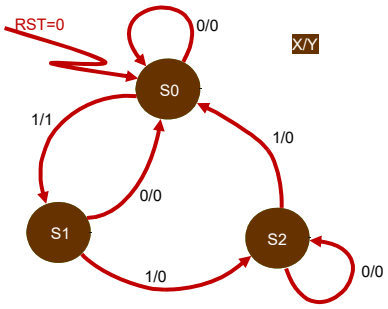
```
process(clk,rst)
 begin
  if (rst='0') then
   std_act <= S0;
  elsif (clk'event and clk='1') then
   std_act <= prox_std;
  end if;
end process;
```

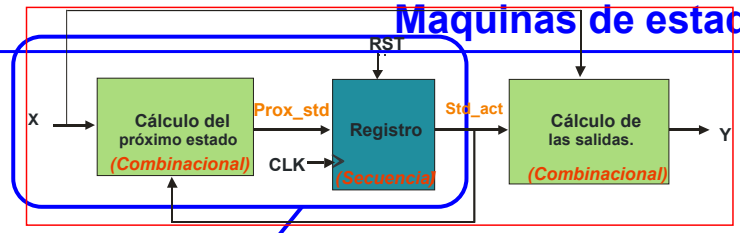
```
process(X,std_act)
 begin
  case std_act is
   when S0 =>
    if (X='1') then
     Y<='1';
    else
     Y<='0';
    end if;
   when S1 =>
     Y<='0';
   when S2 =>
     Y<='0';
  end case;
end process;
```

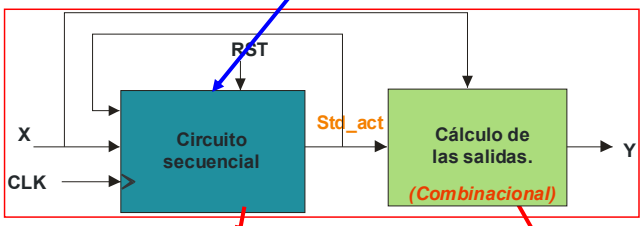
```
Y <='1'when (X='1'and std_act=S0) else '0';
```

Maquinas de estados

❑Modelado con dos procesos








```
type FSM is (S0, S1, S2);
signal std_act:FSM;
```

```
process(clk,rst)
 begin
  if (rst='0') then
   std_act <= S0;
  elsif (clk'event and clk='1') then
    case std_act is
   when S0 =>
    if (X= '1') then
     std_act <= S1;
    end if;
   when S1 =>
    if (X= '0') then
     std_act <= S0;
   when S2 =>
    if (X= '0') then
     std_act <= S0;
    end if;
   end case;  end if;
end process;
```

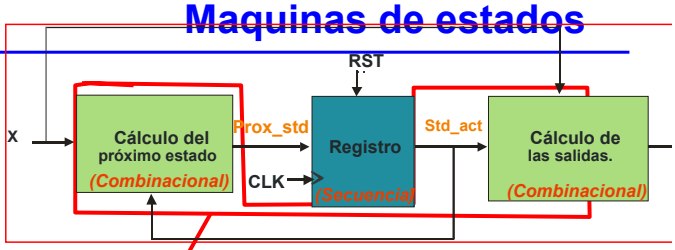
```
process(X,std_act)
 begin
  case std_act is
   when S0 =>
    if (X='1') then
     Y<='1';
    else
     Y<='0';
    end if;
   when S1 =>
     Y<='0';
   when S2 =>
     Y<='0';
  end case;
end process;
```

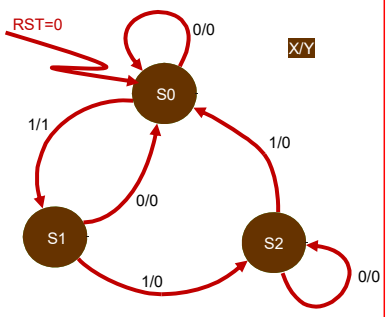
```
Y <='1'when (X='1'and std_act=S0) else '0';
```

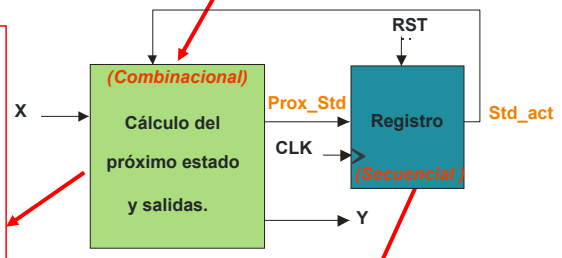

Maquinas de estados

❑Modelado con dos procesos:

- Secuencial :almacena el estado actual de la máquina
- Combinacional: calcula el próximo estado y la salida.








```
type FSM is (S0, S1, S2);
signal std_act, prox_std :FSM;
```

```
process(ENT,std_act)
  begin
    case std_act is
      when S0 =>
        if (X='1') then
           prox_std <= S1;
           Y <='1';
        else
           prox_std <= S0;
           Y <='0';
        end if;
      when S1 =>
         Y <='0';
        if (X='0') then
           prox_std <= S0;
        else
           prox_std <= S1;
        end if;
       when S2 =>
         Y <='0';
        if (X='0') then
           prox_std <= S2;
        else
           prox_std <= S0;
        end if;
    end case;
  end process;
```

```
process(clk,rst)
 begin
  if (rst='0') then
   std_act <= S0;
  elsif (clk'event and clk='1') then
   std_act <= prox_std;
  end if;
end process;
```

- En VHDL las memorias se modelan como arrays:
  - Es preciso declarar un tipo array para dar soporte a la memoria.
  - El índice del array debe ser un entero
    - Uso de funciones de conversión en el bus de direcciones.
  - El tipo de memoria a inferir se controla mediante atributos
    - Específicos de cada sintetizador.
  - Las operaciones de lectura y escritura se realizan de forma simultánea:
    - Un proceso para cada operación.
    - La escritura siempre es síncrona, la lectura depende.

TIPO DE ACCESO	RAM DISTRIBUIDAS	BLOCK RAMs
LECTURA	ASÍNCRONA	SÍNCRONA
ESCRITURA	SÍNCRONA	SÍNCRONA

47


```
type identificador_array is array (K-1downto 0) of std_logic_vector (N-1 downto 0);
signal mem_ident : identificador_array;
```

```
process(CLK)
 begin
   if(CLK'event and CLK = '1') then
     if  WE = '1'  then
         mem(to_integer(unsigned(ADDR_IN))) <= DIN;
     end if;
   end if;
 end process;
```

Escritura

```
 process(CLK,RST)
 begin
 if (RST = '1') then
         DOUT <= (others => '0');
   elsif(CLK'event and CLK = '1') then
         DOUT <= mem(to_integer(unsigned(ADDR_OUT)));
   end if;
 end process;
```

Lectura

Memoria RAM

```
architecture rtl of ram_mem is
  type mem_type is array (255 downto 0) of std_logic_vector (7 downto 0);
  signal mem : mem_type;

begin

 process(CLK)
  begin
    if(CLK'event and CLK = '1') then
      if  WE = '1'  then
          mem(to_integer(unsigned(ADDR_IN))) <= DIN;
      end if;
    end if;
  end process;

  process(CLK)
  begin
    if(CLK'event and CLK = '1') then
          DOUT <= mem(to_integer(unsigned(ADDR_OUT)));
    end if;
  end process;

end rtl;
```


Memoria ROM

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

entity seno is
  port (
    ADDR : in  std_logic_vector(7 downto 0);
    CLK  : in  std_logic;
    DOUT : out std_logic_vector(7 downto 0) );
end seno;

architecture rtl of seno is

  type cell is array (0 to 255) of std_logic_vector(7 downto 0);
  constant memoria  : cell := (
  0  => x"00",
  1  => x"00",
  2  => x"00",
  3  => x"00",
  4  => x"00",
 ....
  253  => x"00",
  254  => x"00",
  255  => x"00");
```

```
begin

  process (clk)
  begin
    if clk'event and clk = '1' then
      DOUT<=memoria(to_integer(unsigned(addr)));
    end if;
  end process;

end rtl;
```