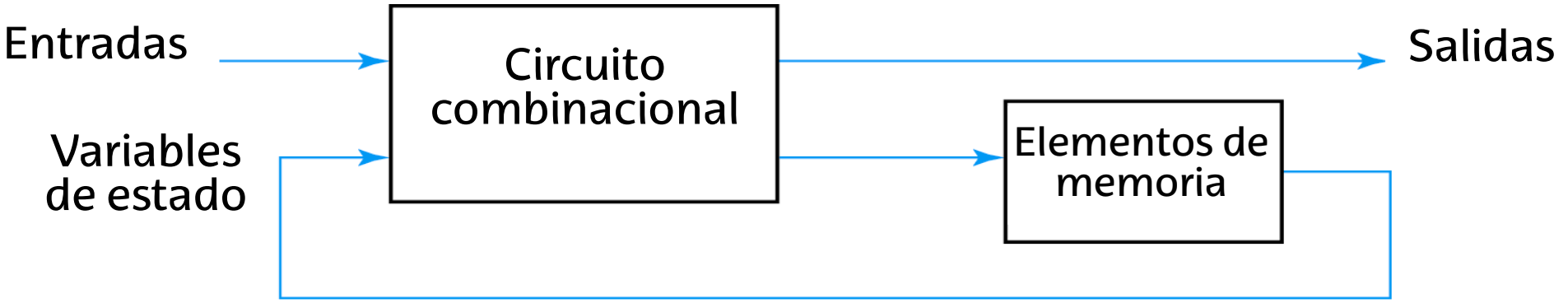
Tema 4:
Bloques secuenciales

Circuito secuencial

- Mediante los circuitos combinacionales no es posible **mantener valores binarios** en el tiempo o producir secuencias binarias.
- Esas funciones son las que desarrollan los **circuitos secuenciales**, que son la base de las **memorias y los microprocesadores**.
- En los circuitos secuenciales, la **salida** depende del **valor actual y de los valores anteriores de las entradas**.

Circuito secuencial: estado

- Estos circuitos generan una función de las entradas actuales que es reenviada como nueva entrada del circuito➔Realimentación.
- La señal realimentada se llama señal de estado, y a cada combinación actual de valores se le llama estado del circuito.

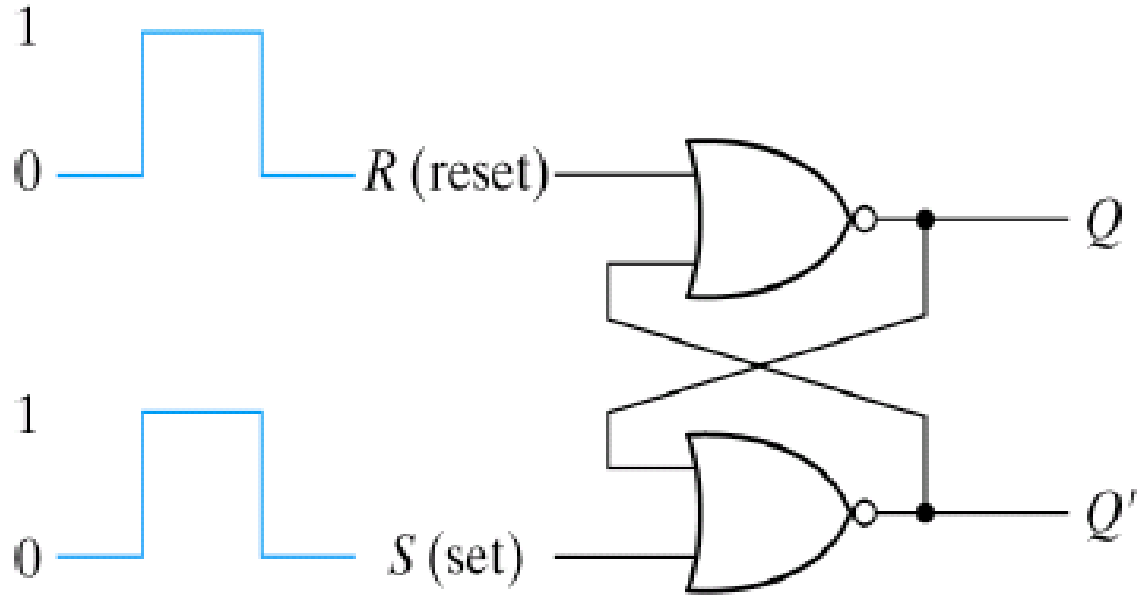


Circuito secuencial: estado

- Cada vez que el estado del circuito cambie, la salida tendrá nuevos valores aunque el valor de la entrada sea el mismo.
- La variable de estado debe permanecer aunque la entrada cambie, pues de lo contrario se perdería el estado del circuito➔Elementos de memoria.

Biestable asíncrono (latch) S-R

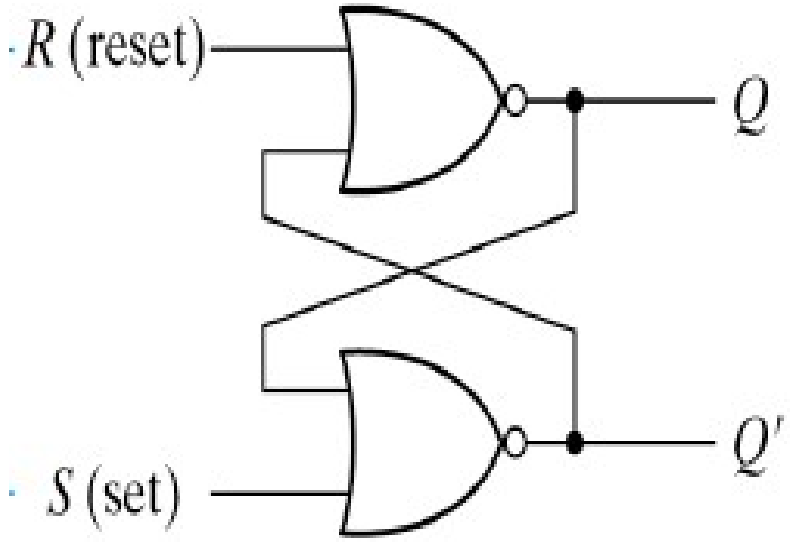
- El latch S-R es un elemento de memoria, capaz de mantener el valor de un bit indefinidamente.
- Un 1 en S cambia el valor de la salida a 1, y un 1 en R cambia el valor de la salida a 0.



S	R	Q	Q'	
1	0	1	0	
0	0	1	0	*Después de que S =1*
0	1	0	1	
0	0	0	1	*Después de que R =1*
1	1	0	0	

Biestable asíncrono (latch)S-R con puertas NOR

- Aunque las entradas cambien a 0, el valor de la salida (estado) se mantiene hasta que vuelva a cambiar a 1 alguna entrada.
- Si las dos entradas valen 1, aparece un valor de estado que no se puede mantener➔Estado inestable.

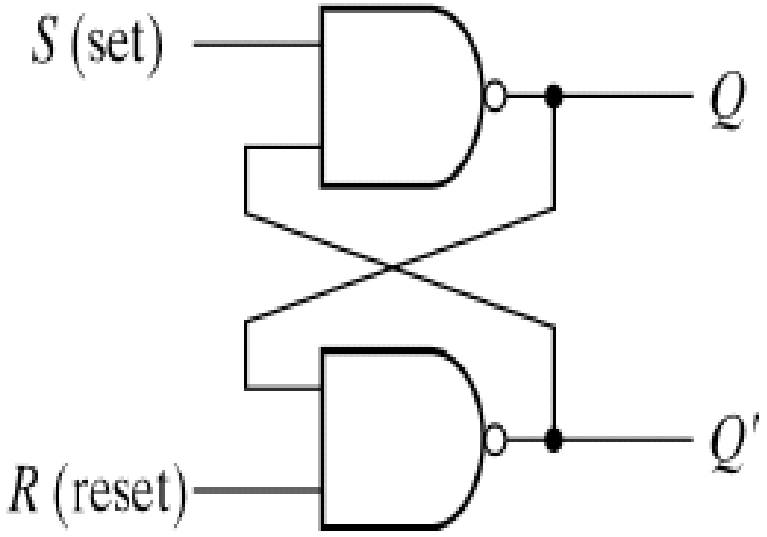


Q	S	R	Q*	Q*'
0	0	0	0	1
0	0	1	0	1
0	1	0	1	0
0	1	1	0	0
1	0	0	1	0
1	0	1	0	1
1	1	0	1	0
1	1	1	0	0

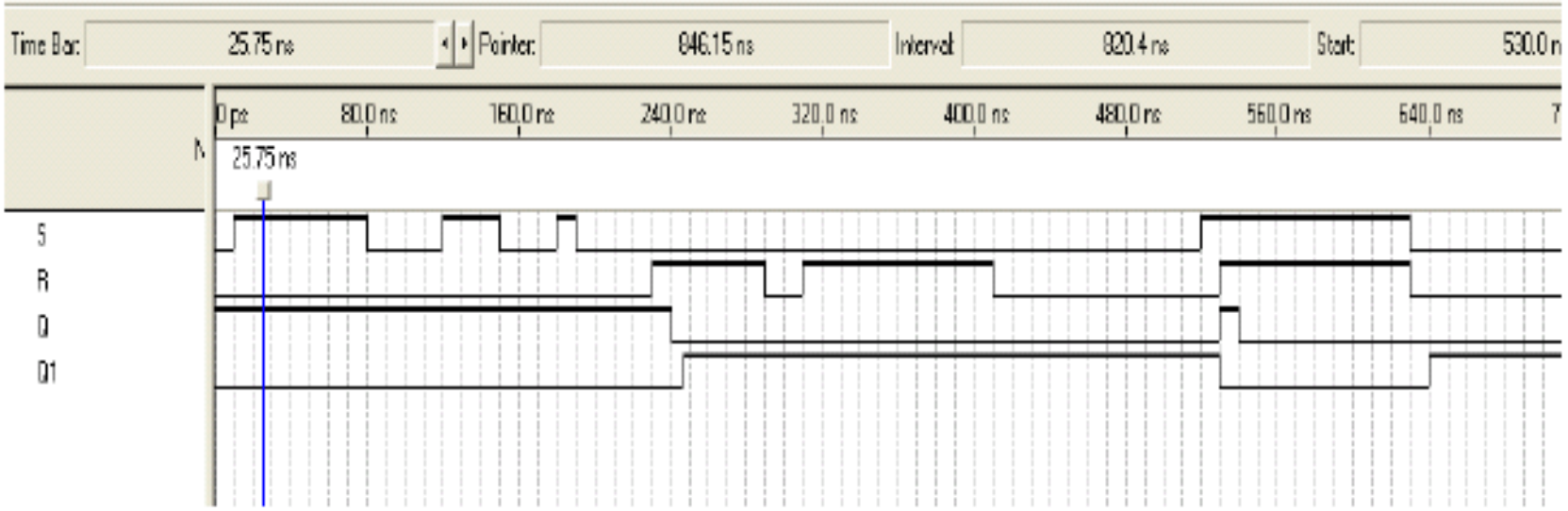
Q: valor actual del estado
Q*: nuevo valor de estado

Biestable asíncrono (latch) S-R con puertas NAND

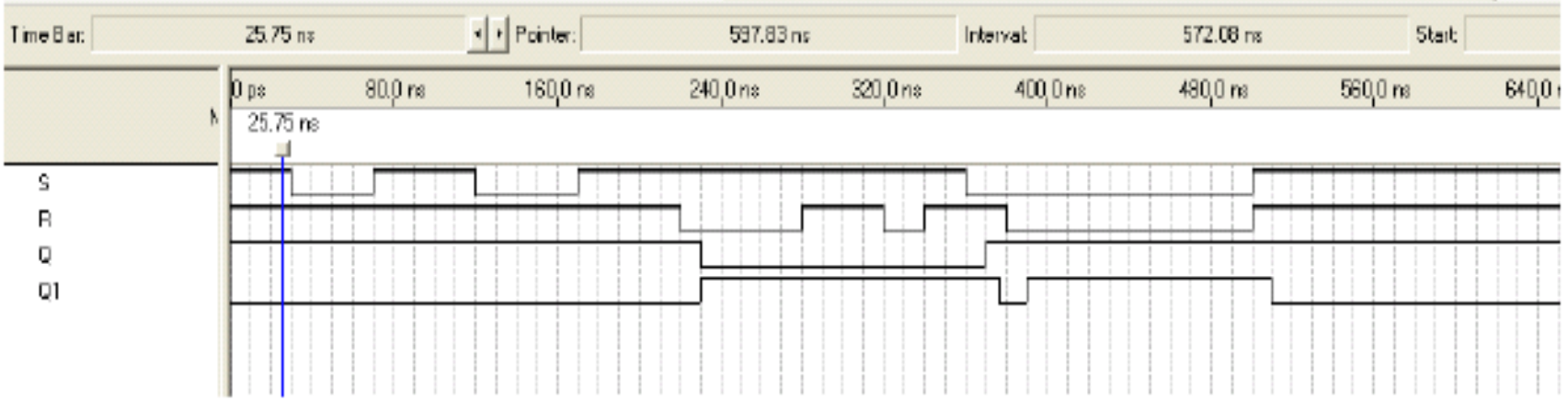
- El latch S-R se puede implementar con puertas NAND.
- Este circuito funciona con entradas a nivel bajo➔S y R se activan con 0s.
- El estado inestable aparece con dos 0s en las entradas.



Q	S	R	Q*	Q*'
0	0	0	1	1
0	0	1	1	0
0	1	0	0	1
0	1	1	0	1
1	0	0	1	1
1	0	1	1	0
1	1	0	0	1
1	1	1	1	0



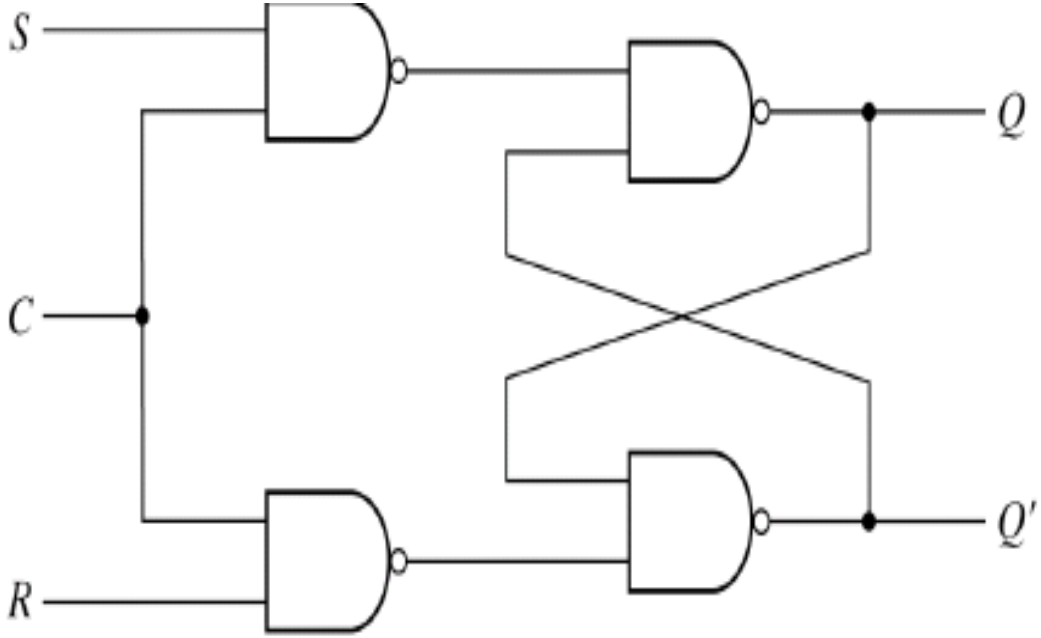
Cronograma de latch SR con puertas NOR



Cronograma de latch SR con puertas NAND

Biestable asíncrono (latch) S-R con entrada de control

- Los cambios de estado se pueden suceder en cascada.
- Podemos evitarlo limitándolos en el tiempo.
- Sólo puede producirse cambio de estado mientras la entrada de control sea 1.



C	S	R	Q*	Q*'
0	X	X	Q	Q'
1	0	0	Q	Q'
1	0	1	0	1
1	1	0	1	0
1	1	1	1	1

Biestable asíncrono (latch) S-R

SIN ENTRADA DE CONTROL

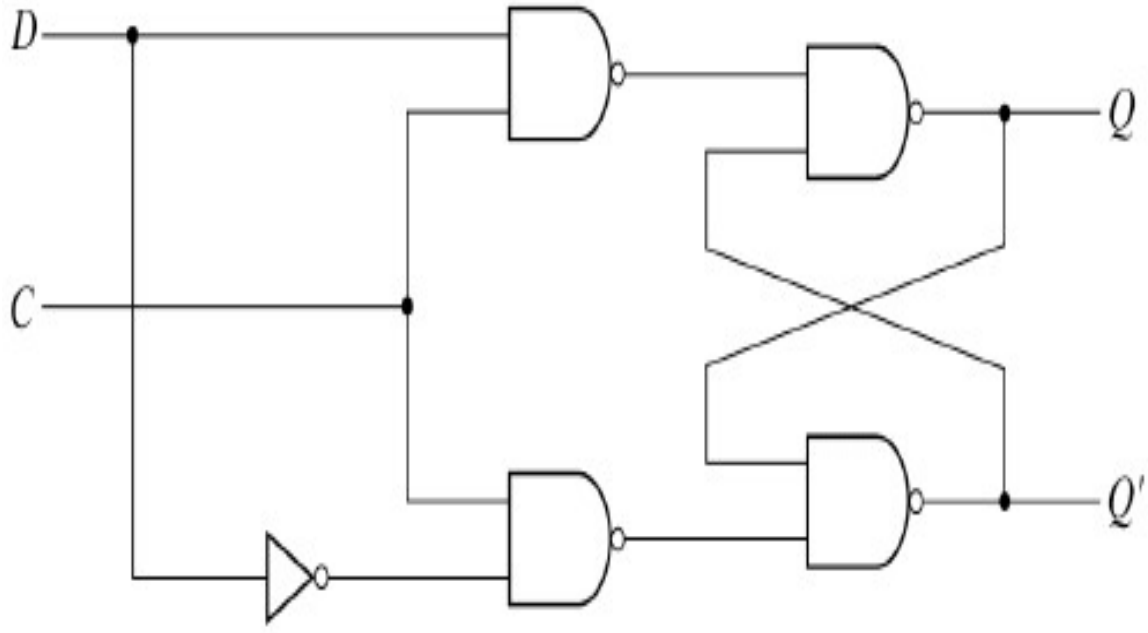
```
library ieee;
use ieee.std_logic_1164.all;

ENTITY simSR IS
PORT ( S,R: IN std_logic;
Q, Q1: OUT std_logic);
END simSR;

ARCHITECTURE a OF simSR IS
SIGNAL B, B1: std_logic;

BEGIN
b1<=S NOR b;
b<=R NOR b1;
Q<=B;
Q1<=B1;
END a;
```

CON ENTRADA DE CONTROL

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

ENTITY simSRcont IS
PORT ( S, R, C: IN std_logic;
Q, Q1: OUT std_logic);
END simSRcont;

ARCHITECTURE a OF simSRcont IS
SIGNAL B, B1: std_logic;

BEGIN
b <=(S NAND C) NAND b1;
b1 <=(R NAND C) NAND b;
Q <=B;
Q1 <=B1;
END a;
```

Biestable asíncrono (latch) D con entrada de control

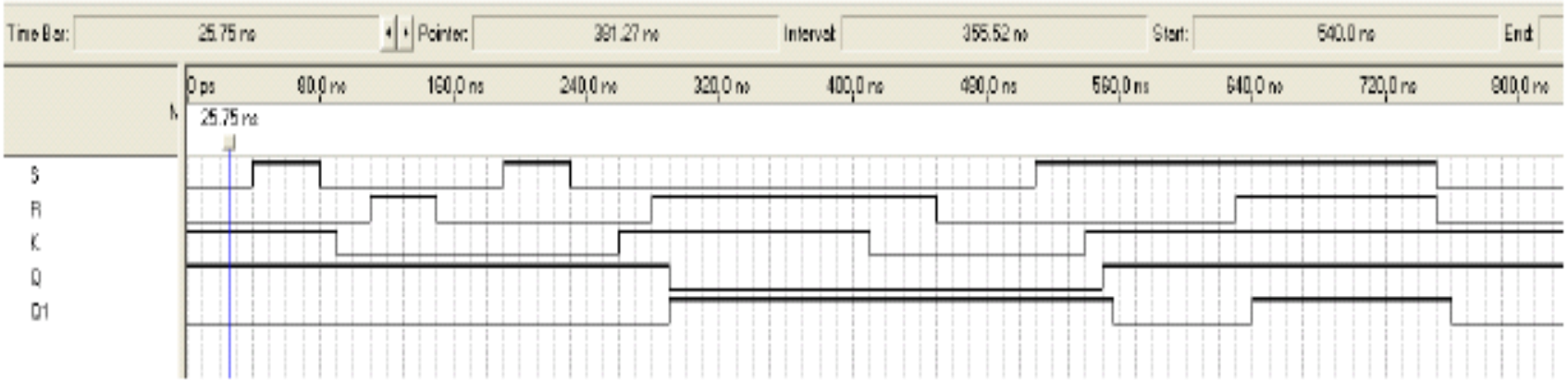
- En este diseño no aparece el estado inestable.
- El valor de la entrada se repite en la salida, salvo si la entrada de control está a 0➔Memoria del último estado.



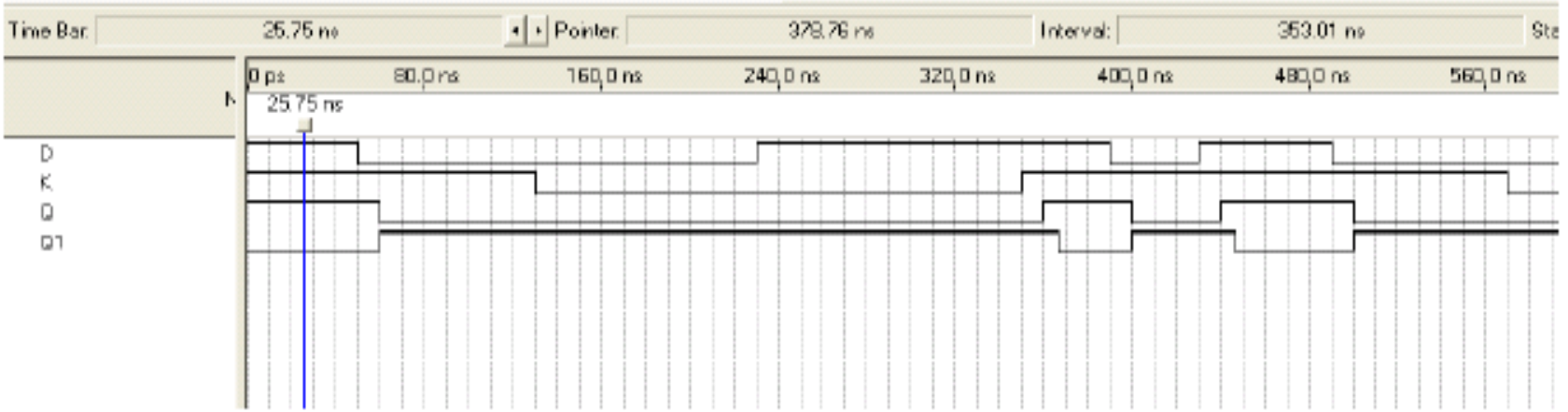
C	D	Q*	Q*'
0	X	Q	Q'
1	0	0	1
1	1	1	0

Biestable asíncrono (latch) D con entrada de control

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

ENTITY simDcont IS
PORT ( D, C: IN std_logic;
        Q, Q1: OUT std_logic);
END simDcont;

ARCHITECTURE a OF simDcont IS
SIGNAL B, B1: std_logic;

BEGIN

PROCESS (D, C)
BEGIN

IF C='0' THEN
      B<=B;
      B1<=B1;
ELSE
    B<=D;
    B1<= NOT D;
END IF;

END PROCESS;

Q <=B;
Q1 <=B1;

END a;
```



Cronograma de latch SR con entrada de control



Cronograma de latch D con entrada de control

Activación por nivel y por flanco

- La señal de control (activación por nivel) define un intervalo de tiempo en el que se puede cambiar el estado.
- Durante el intervalo de activación del elemento de memoria, los cambios de estado se pueden dar en cascada.

Activación por nivel alto

Activación por nivel y por flanco

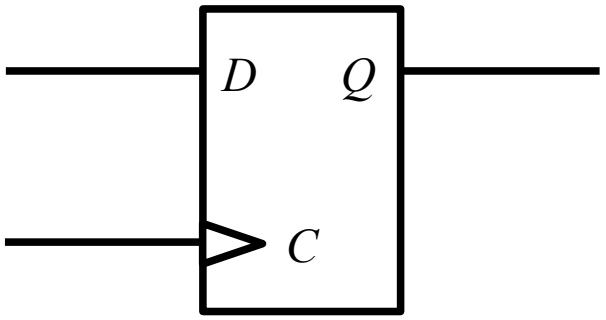
- Se puede reducir este intervalo a sólo el necesario para que una señal cambie de valor➔Activación por flanco.
- Si la señal de activación es de frecuencia constante, los cambios de estado estarán sincronizados a esa frecuencia➔Reloj.

Activación por flanco de subida (flanco positivo)

Activación por flanco de bajada (flanco negativo)

Biestable síncrono (flip-flop) D

- Es el elemento de memoria equivalente al cerrojo D con entrada de control, pero activado por flanco.
- A cada flanco del reloj la entrada aparece en la salida.



C	D	Q*	Q*'
X	X	Q	Q'
↑	0	0	1
↑	1	1	0

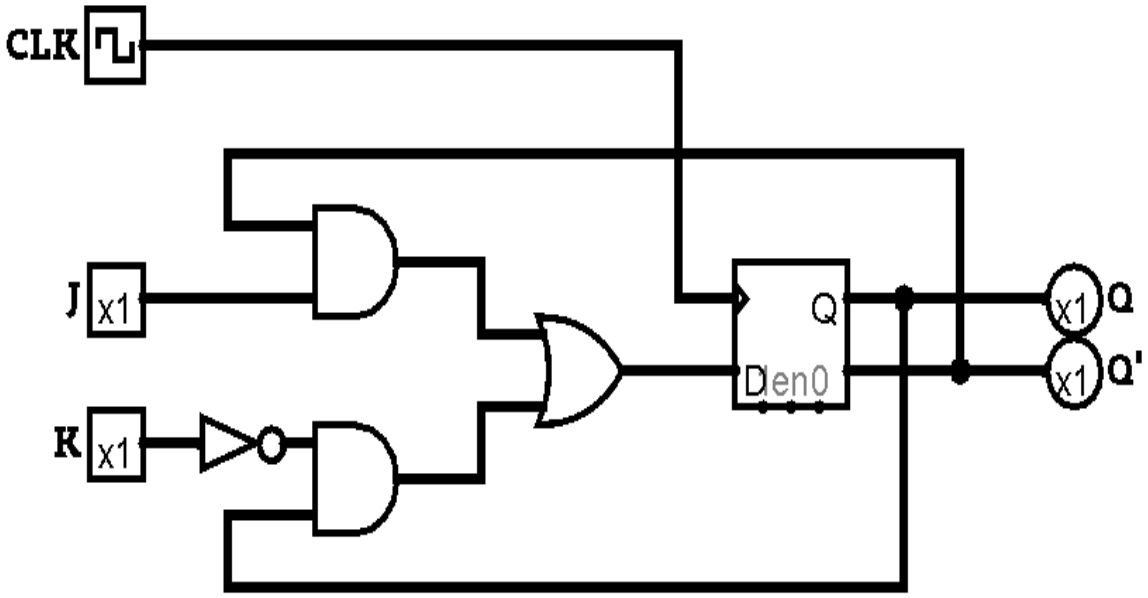
↑: cuando C cambia de 0 a 1

Biestable síncrono (flip-flop) D

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

ENTITY simFFD IS
PORT ( D, CLK: IN std_logic;
        Q, Q1: OUT std_logic);
END simFFD;

ARCHITECTURE a OF simFFD IS

BEGIN

PROCESS (D, CLK)


BEGIN

IF (CLK'EVENT AND CLK='1') THEN
      Q<=D;
      Q1<=NOT D;
END IF;

END PROCESS;

END a;
```

Biestable síncrono (flip-flop) JK

- Es una modificación del flip-flop D con dos entradas.
- Semejante al SR, pero si se activan las dos entradas, la salida cambia.

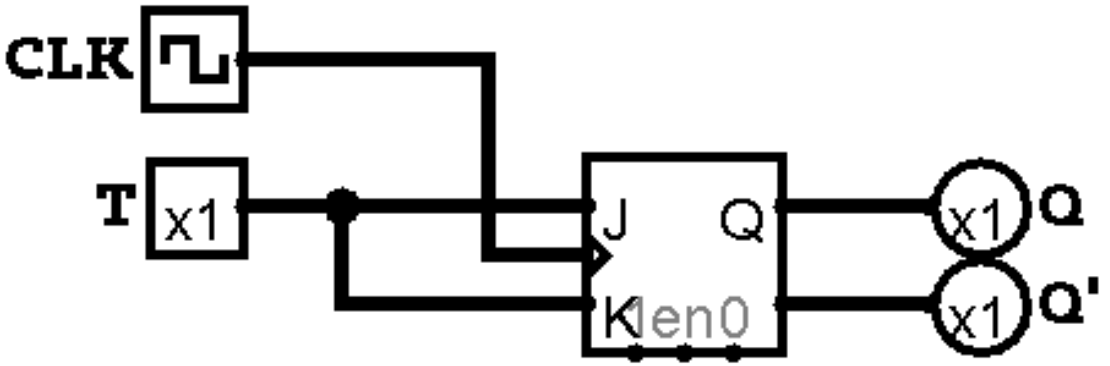


C	J	K	Q*	Q*'
X	X	X	Q	Q'
↑	0	0	Q	Q'
↑	0	1	0	1
↑	1	0	1	0
↑	1	1	Q'	Q

$$D = J \cdot \bar{Q} + \bar{K} \cdot Q$$

Biestable síncrono (flip-flop) T

- Modificación del JK con una entrada.
- Mientras la entrada T sea 1, la salida cambia a cada flanco del reloj.
- Se aplica en contadores, ya que el cambio de una variable es la cuenta de una cifra binaria.



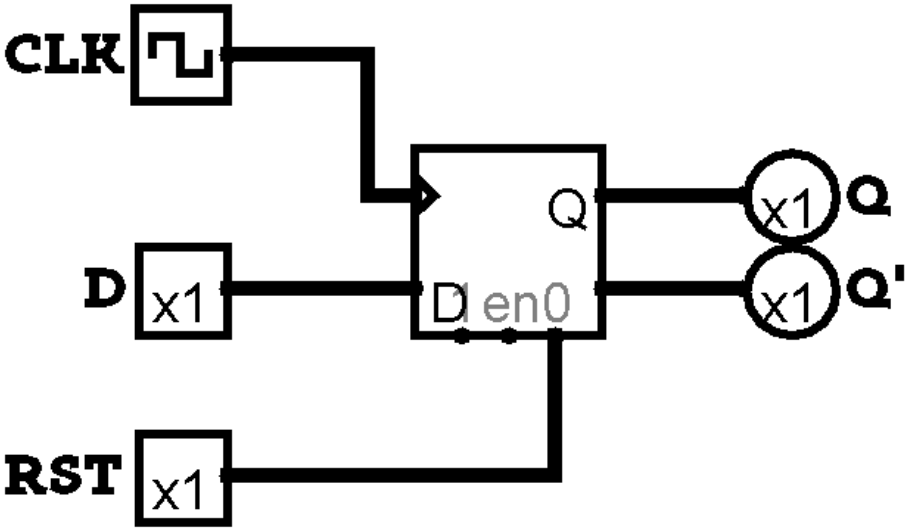
C	T	Q*	Q*'
X	X	Q	Q'
↑	0	Q	Q'
↑	1	Q'	Q

Biestable síncrono (flip-flop) T

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

ENTITY simFFT IS
PORT ( T, CLK: IN std_logic;
           Q, Q1: OUT std_logic);
END simFFT;

ARCHITECTURE a OF simFFT IS
SIGNAL B: std_logic;

BEGIN

PROCESS (T, CLK)

BEGIN

IF (CLK'EVENT AND CLK='1') THEN
    IF T='1' THEN B<=NOT B;
        END IF;
    END IF;
END PROCESS;

Q<= B;
Q1<= NOT B;

END a;
```

Entradas asíncronas

- Los flip-flop pueden incorporar el comportamiento del cerrojo➔El estado cambia cuando cambia la entrada.
- La entrada asíncrona R lleva al estado a 0, y la entrada asíncrona S lleva al estado a 1.
- Las entradas asíncronas tienen preferencia sobre las entradas de los flip-flop.

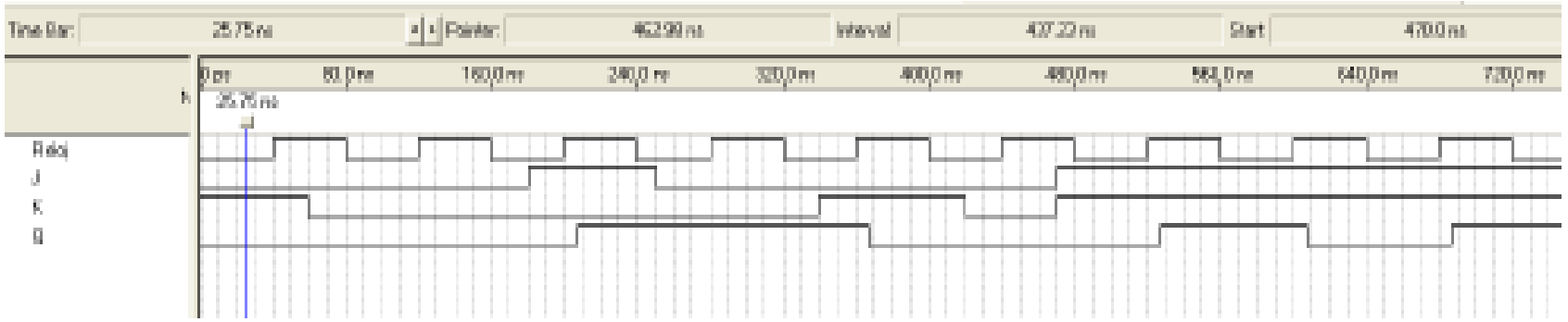


R	*C*	*D*	*Q**	*Q*'*
0	X	X	0	1
1	X	X	*Q*	*Q'*
1	↑	0	0	1
1	↑	1	1	0

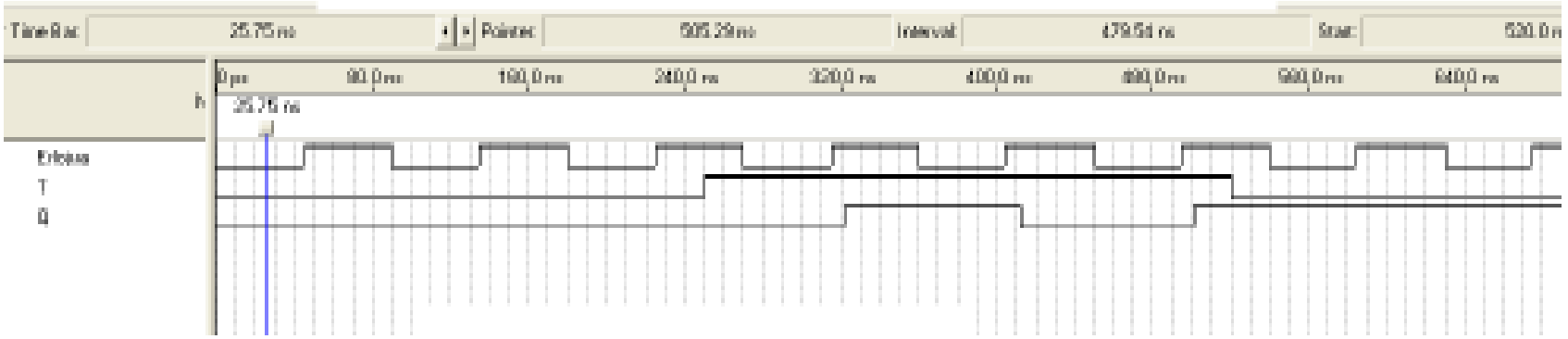
Biestable síncrono (flip-flop) D con entradas asíncronas

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

ENTITY simFFJK IS
PORT ( pre, clr, J, K, CLK: IN std_logic;
            Q, Q1: OUT std_logic);
END simFFJK;

ARCHITECTURE a OF simFFJK IS
SIGNAL D, B, B1: std_logic;

BEGIN

D<=(J AND B1) OR ((NOT K) AND B) ;
Q<= B;
Q1<= B1;
```

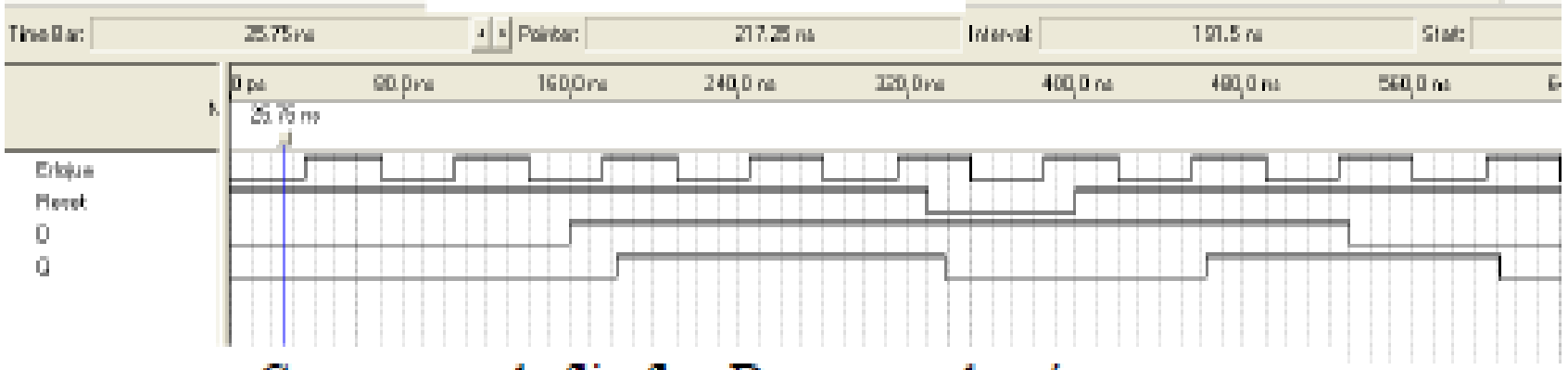
```
PROCESS (pre, clr, D, CLK)
BEGIN

IF pre='1' THEN
     B<='1';
     B1<='0';
ELSIF clr='1' THEN
     B<='0';
     B1<='1';
ELSIF (CLK'EVENT AND CLK='1')
  THEN
     B<=D;
     B1<=NOT D;
    END IF;

END PROCESS;

END a;
```



Cronograma de flip-flop JK activado por flanco



Cronograma de flip-flop T

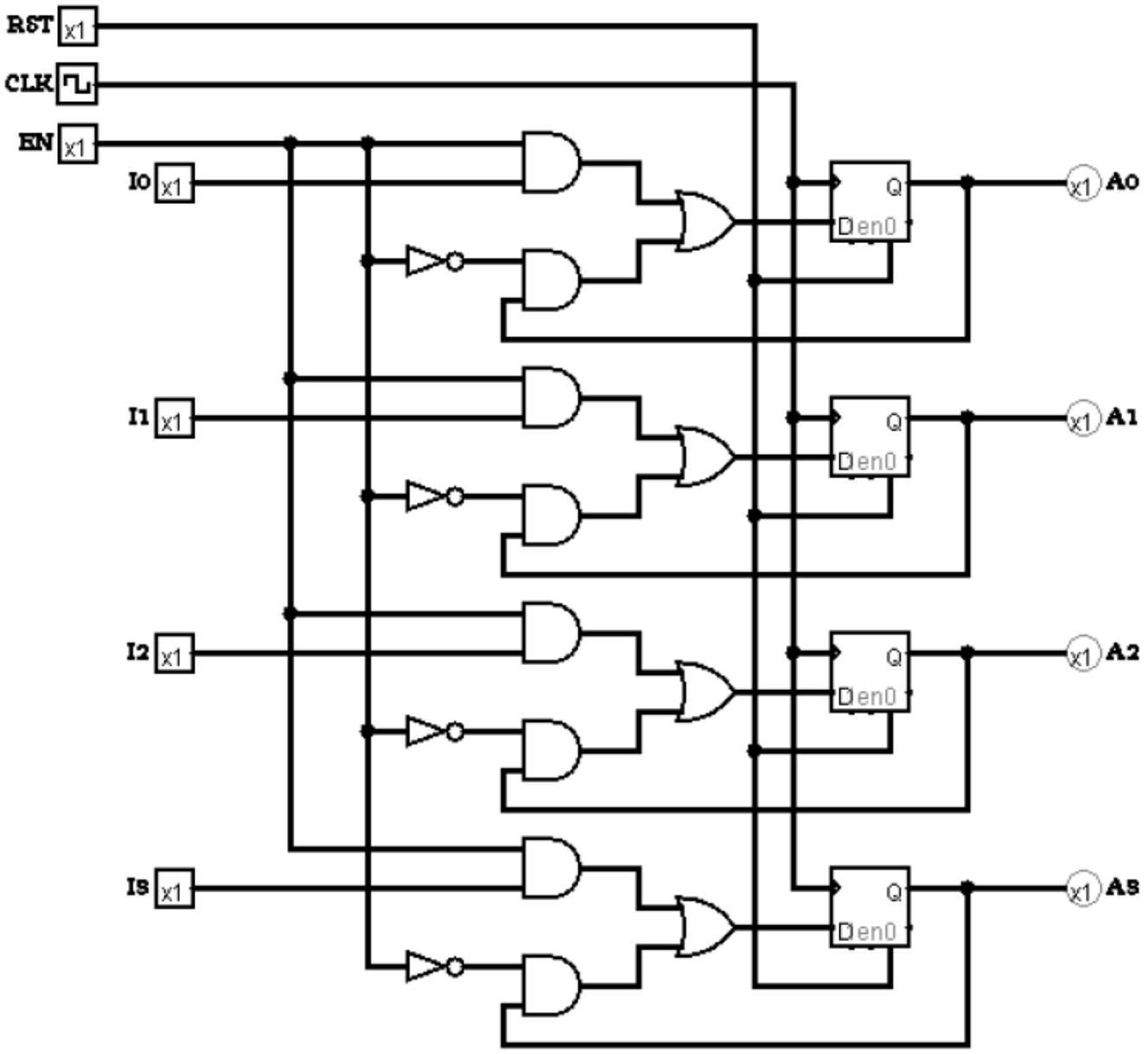


Cronograma de flip-flop D con entrada asíncrona reset

Registros y contadores

- Las aplicaciones más sencillas de los elementos de memoria son los registros y los contadores.
- Los registros permiten mantener el valor de un dato binario (registro de almacenamiento) y operar sobre él (registro de desplazamiento).
- Los contadores producen un dato numérico de una secuencia predeterminada, a cada flanco de reloj.

Registro de almacenamiento



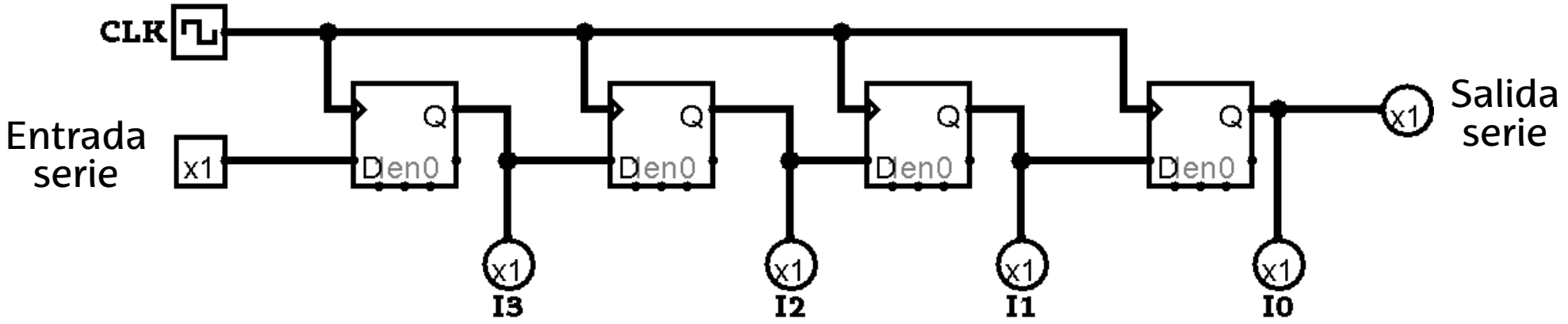
- Mantiene el dato binario de I_3-I_0 en la salida A_3-A_0, desde que aparezca el flanco de ***CLK*** hasta el nuevo flanco, si ***EN*** es 1.
- Mientras ***EN*** sea 0, no hay ningún cambio en A_3-A_0.
- Al activar ***RST***, el valor de A_3-A_0 vuelve a 0.

Registro de almacenamiento

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY registro IS
PORT(clk, en, rst: IN std_logic;
        I: IN std_logic_vector(3 downto 0);
        A:OUT std_logic_vector(3 downto 0));
END registro;

ARCHITECTURE a OF registro IS
signal D: STD_LOGIC_VECTOR (3 downto 0);

BEGIN
A<=D;

PROCESS (rst, clk)

BEGIN

IF rst = '1' then D<= (OTHERS => '0');
ELSIF (clk'EVENT AND clk='1') THEN
      IF en = '1' THEN D<=I;
          END IF;
      END IF;
END PROCESS;

end a;
```

Registro de desplazamiento

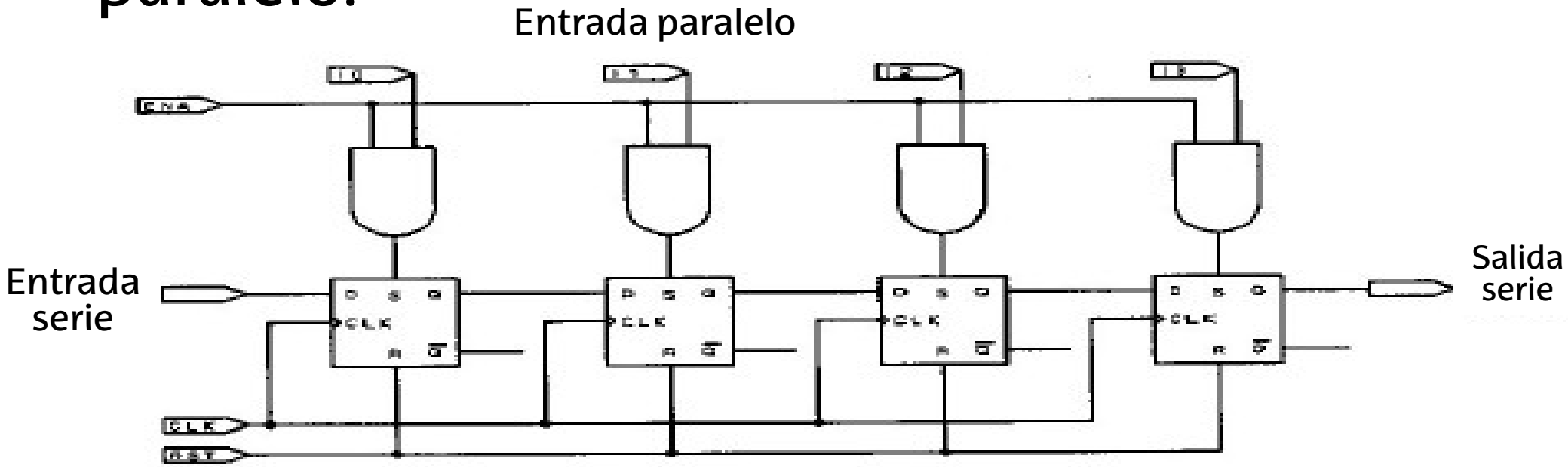
- El registro de desplazamiento modifica el dato introducido, escribiendo cada bit del dato en la siguiente posición.
- El dato puede ser introducido en formato serie (un bit cada período de reloj) o en formato paralelo (todos los bits se introducen a la vez).



Registro de desplazamiento a la derecha de 4 bit

Registro de desplazamiento

- Existen diseños con desplazamiento a la izquierda o incluso en los que se puede elegir el desplazamiento a la izquierda o a la derecha.
- En este diseño se puede elegir entrada serie o paralelo.

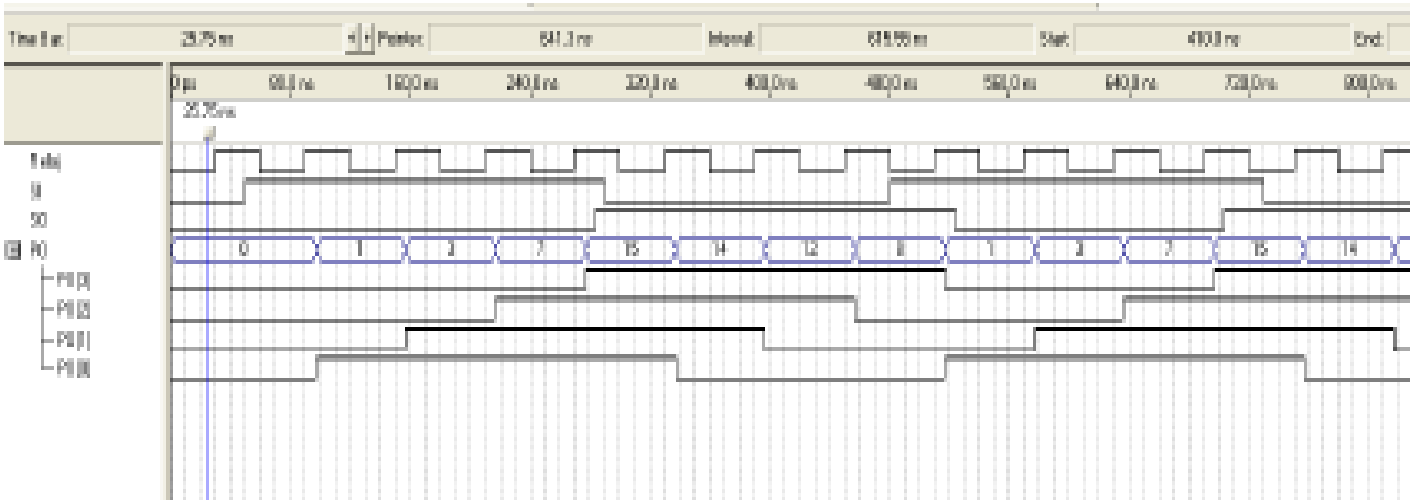


Registro de desplazamiento a la derecha de 4 bit con entrada serie y paralelo

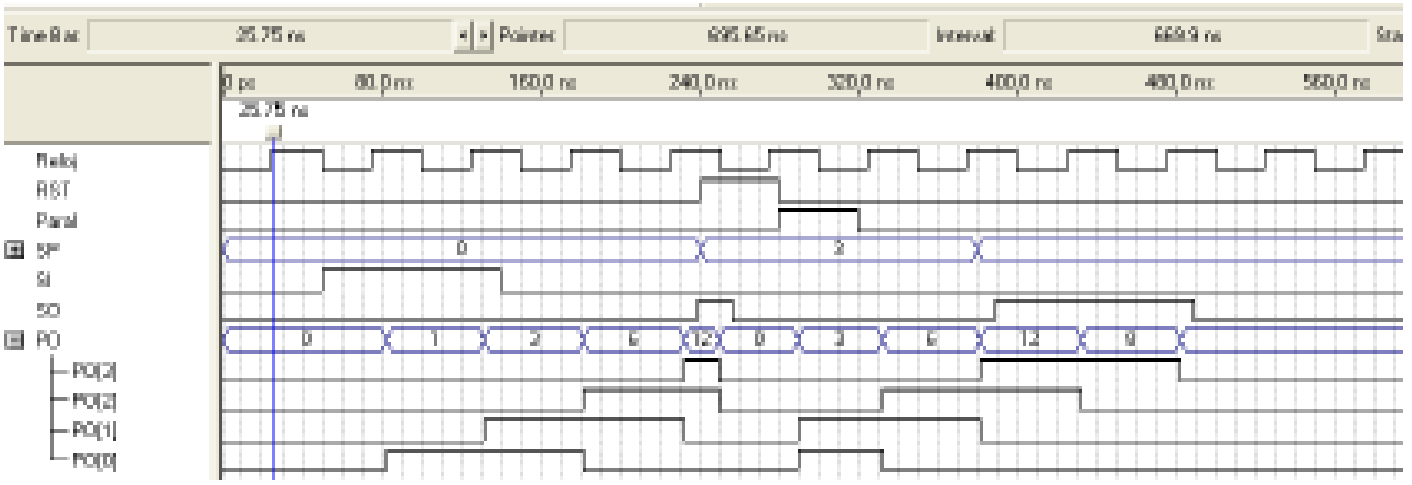
Registro de desplazamiento

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

ENTITY regdesp IS
PORT ( Es, CLK: IN std_logic;
            E: IN std_logic_vector (3 DOWNTO 0);
            K: IN std_logic_vector (1 DOWNTO 0);
            S: OUT std_logic_vector (3 DOWNTO 0));
END regdesp;

ARCHITECTURE a OF regdesp IS
SIGNAL B: std_logic_vector (3 DOWNTO 0);

BEGIN
PROCESS (K, E, CLK)
BEGIN
IF K="11" THEN B<=E;
ELSIF CLK'EVENT AND CLK='1' THEN
      CASE K IS
            WHEN "01"=> B(3)<=B(2);
                        B(2)<=B(1);
                        B(1)<=B(0);
                        B(0)<=Es;
            WHEN "10"=> B(3)<=Es;
                        B(2)<=B(3);
                        B(1)<=B(2);
                        B(0)<=B(1);
            WHEN OTHERS=> B<=B;
      END CASE;
END IF;
END PROCESS;
S<= B;
END a;
```

K	Funcionamiento
00	Almacenamiento
01	Desp. a la izquierda
10	Desp. a la derecha
11	Carga paralela



Cronograma de registro de desplazamiento a la derecha de 4 bit

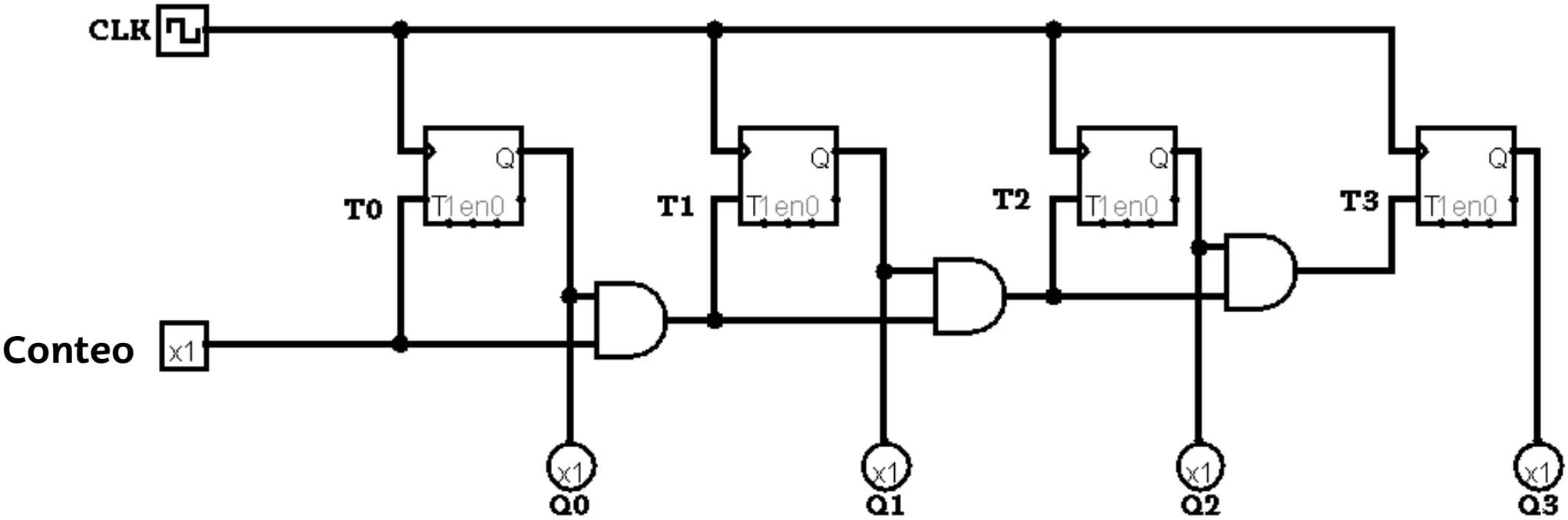


Cronograma de registro de desplazamiento a la derecha de 4 bit con entrada y salida serie y con entrada paralelo

Contadores

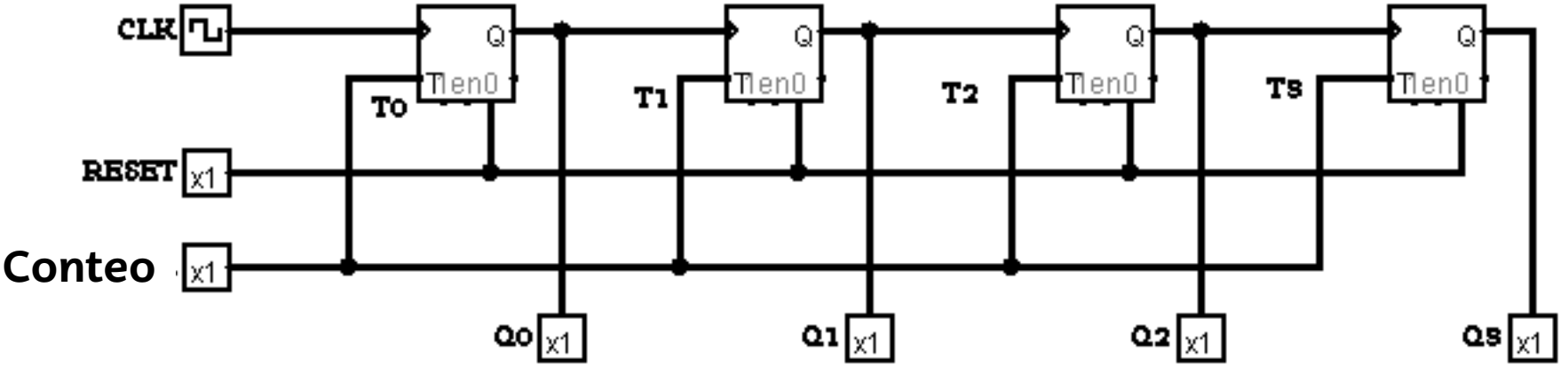
- Los contadores producen un dato binario que va cambiando a cada flanco de reloj, de una secuencia prefijada.
- La secuencia suele ser la binaria ascendente o descendente, pero puede ser también BCD.
- Los contadores se llaman síncronos si todos los bits del dato producido cambian al mismo tiempo.
- Si no ocurre así, se llaman asíncronos.

Contador síncrono



- Este contador produce, en Q_3 - Q_0, la secuencia binaria ascendente, a cada pulso de ***CLK***.
- Cada bit cambia cuando los anteriores son 1.
- La secuencia se detiene si ***Conteo*** vuelve a 0.

Contador asíncrono

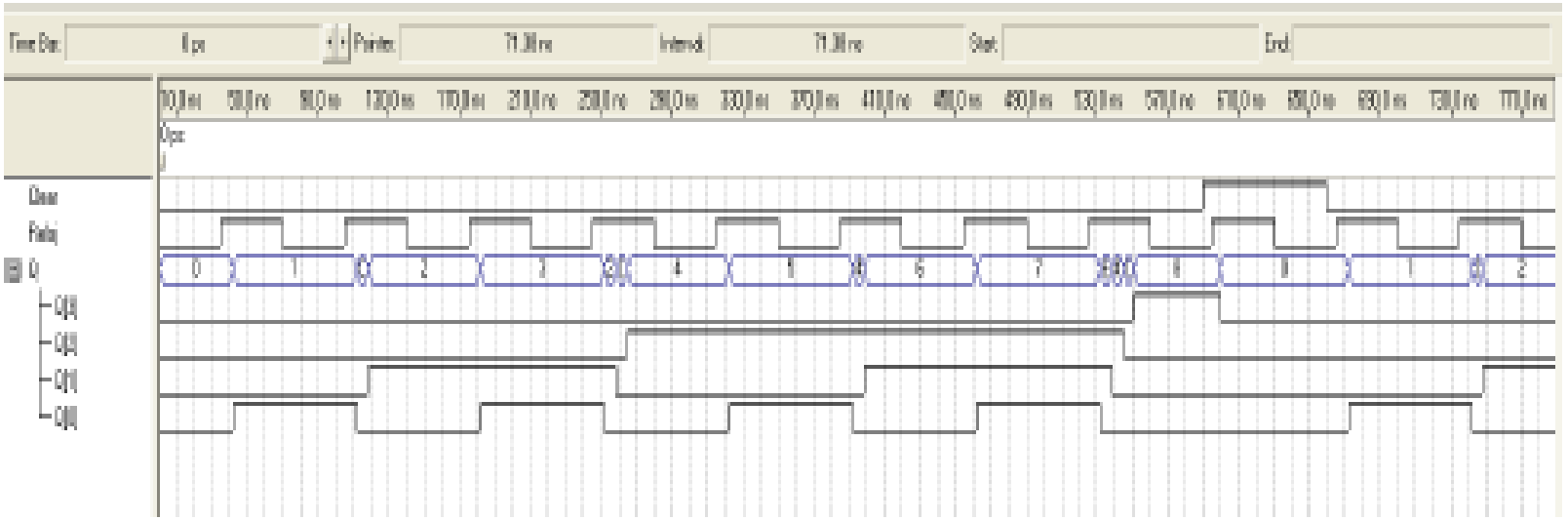


- Cada flip-flop tiene una señal de reloj diferente, por lo que cambia en diferente instante.
- La secuencia es la binaria descendente a cada flanco de ***CLK***, pero hay pequeños intervalos erróneos.
- Al activar ***RESET***, Q_3 - Q_0 vuelve a 0 sin esperar al flanco de ***CLK***.

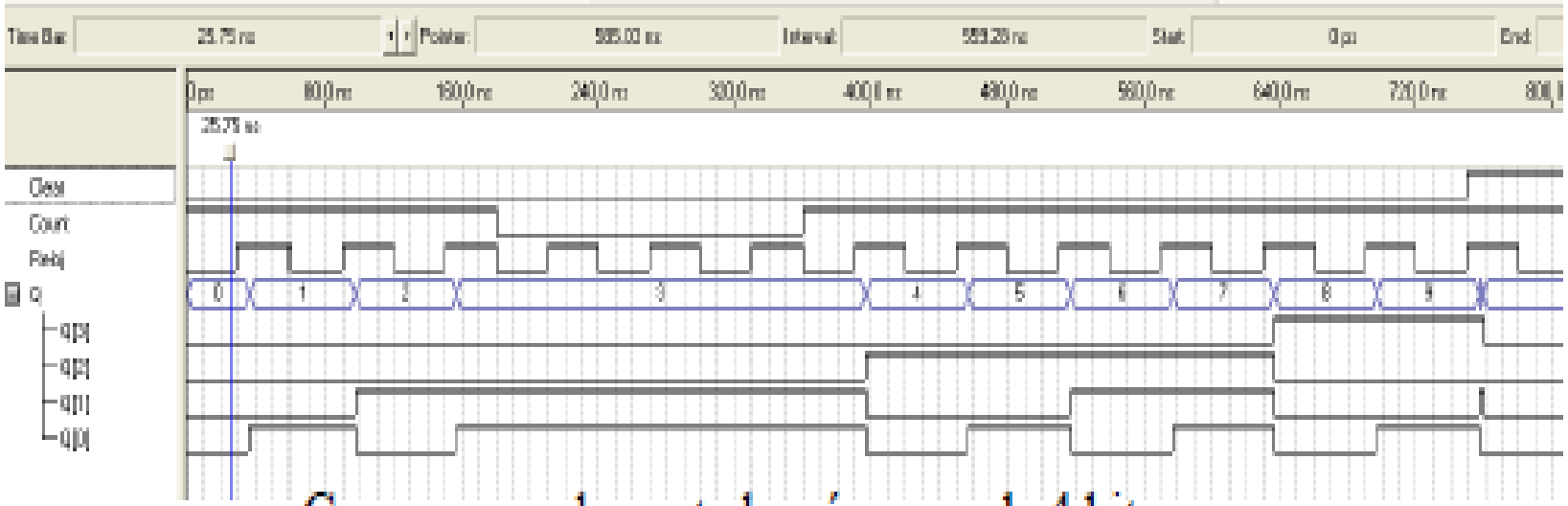
Contador

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

ENTITY conta IS PORT (
        CLK: IN std_logic;
        E: IN std_logic_vector (3 downto 0);
        K:IN std_logic_vector (1 downto 0);
        S:OUT std_logic_vector (3 downto 0)
);
END conta;

ARCHITECTURE a OF conta IS
signal B: std_logic_vector (3 downto 0);

BEGIN

S<=B;

PROCESS (E, K, CLK)
BEGIN

IF K = "11" THEN
      B <= E;
ELSIF (CLK'EVENT AND CLK = '1') THEN
    CASE K IS
       WHEN "01" => B<= B+1;
       WHEN "10" => B<= B-1;
       WHEN OTHERS => B<= B;
    END CASE;
END IF;
END PROCESS;

END a;
```

K	Funcionamiento
00	Almacenamiento
01	Ascendente
10	Descendente
11	Carga paralela



Cronograma de contador asíncrono de 4 bit



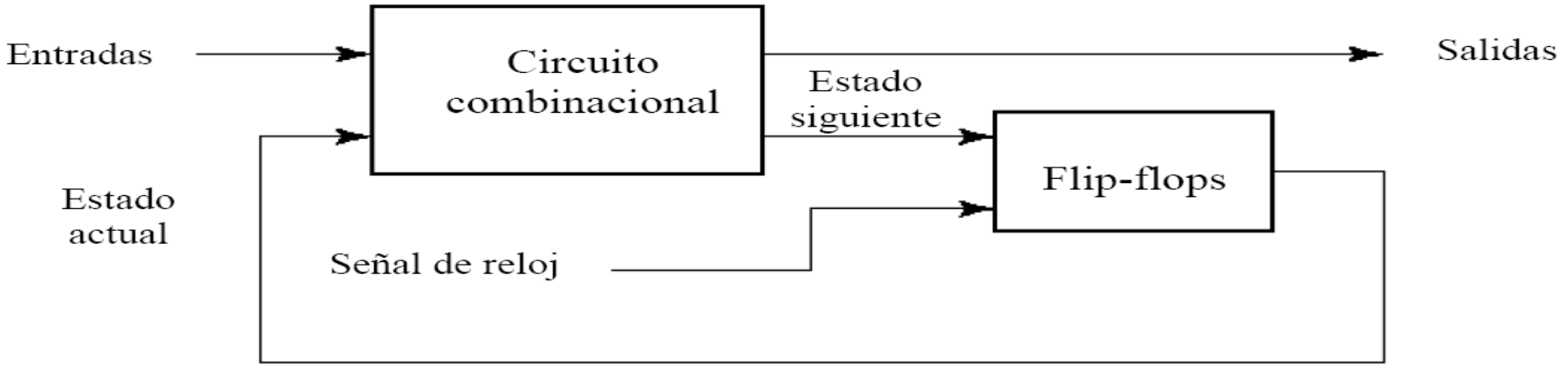
Cronograma de contador síncrono de 4 bit

Análisis y síntesis de sistemas secuenciales síncronos

- Los circuitos secuenciales cuyos elementos de memoria son flip-flops con la misma señal de reloj son los sistemas secuenciales síncronos.
- Análisis es el estudio de un sistema para describir su funcionamiento.
- Síntesis es el diseño de un circuito a partir de la descripción de su funcionamiento.

Sistema secuencial síncrono

- En estos sistemas, la salida es función del estado actual y de la entrada.
- Junto con la salida, se define el valor de estado siguiente, que se almacena en los flip-flops.
- El estado sólo puede cambiar cuando lo hace la señal de reloj.



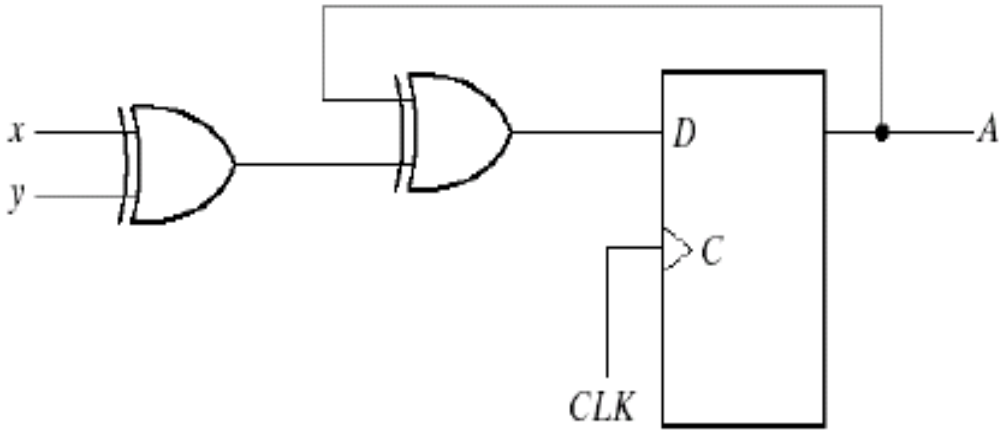
Cronograma de la señal de reloj

Análisis de sistemas secuenciales

- Para analizar un sistema secuencial, hay que conocer el valor de nuevo estado para cada combinación de valores de entrada y estado actual➔Ecuaciones de nuevo estado.

- De este modo, y con las ecuaciones de respuesta de los flip-flops, conoceremos los cambios de estado (transiciones) del sistema para cada posible valor de entrada.

Análisis de sistemas secuenciales



Estado actual	Entradas		Estado siguiente
Q	***X***	***Y***	***Q****
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Tabla de estados

Estado siguiente: $D=X \oplus Y \oplus Q$

Respuesta flip-flop: $Q^*=D$

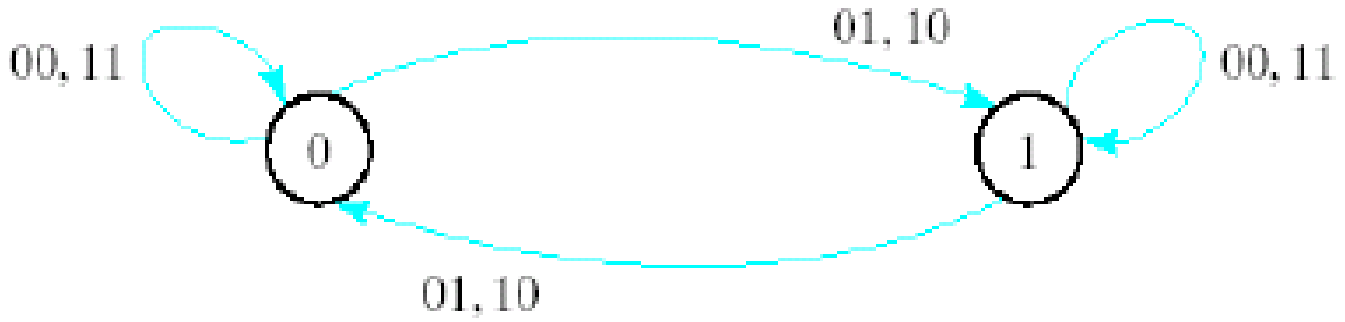
Salida: $A=Q$

Análisis de sistemas secuenciales

- Los diagramas de estado son representaciones gráficas de todas las posibles transiciones del sistema secuencial, con los correspondientes valores de salida➔Ecuaciones de salida.

- El diagrama de estado permite predecir el comportamiento del sistema ante cualquier sucesión de variables de entrada (conocido el estado inicial).

Análisis de sistemas secuenciales

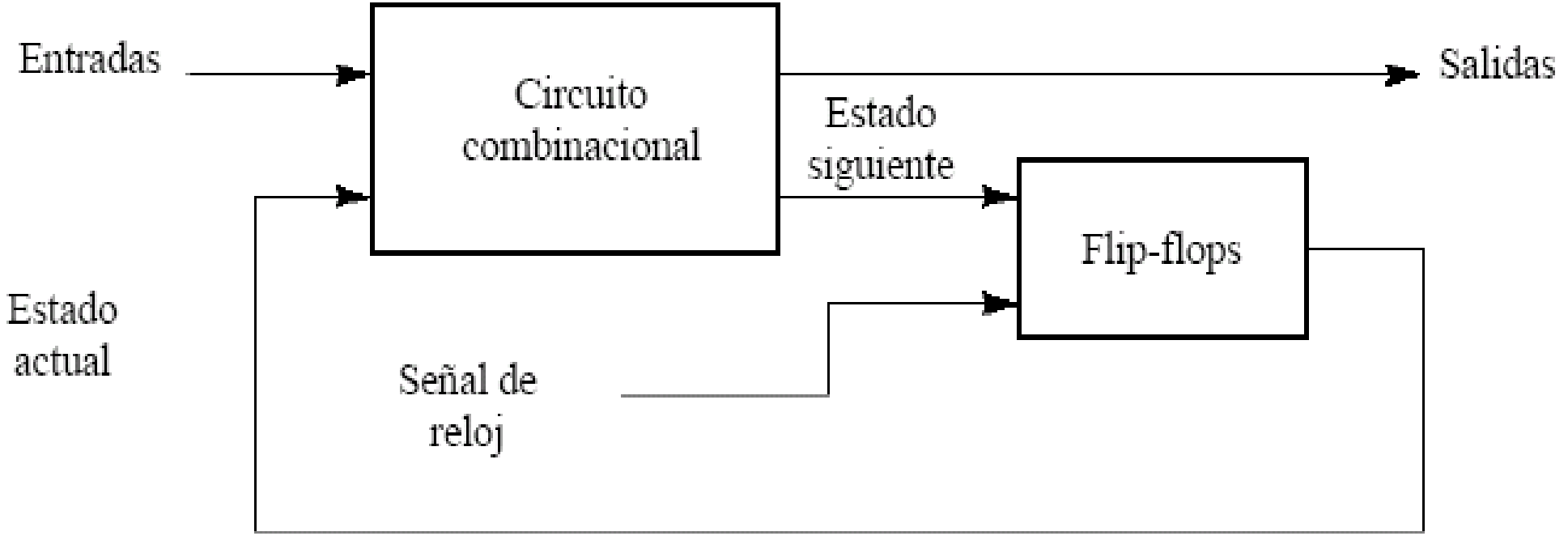


X	0	0	1	1	0	0	0
y	0	1	1	0	0	1	0
A	0	0	1	1	0	0	1
Estado actual	0	0	1	1	0	0	1

No importa cuándo cambia la entrada, los cambios de estado se realizan en el instante de flanco de reloj.

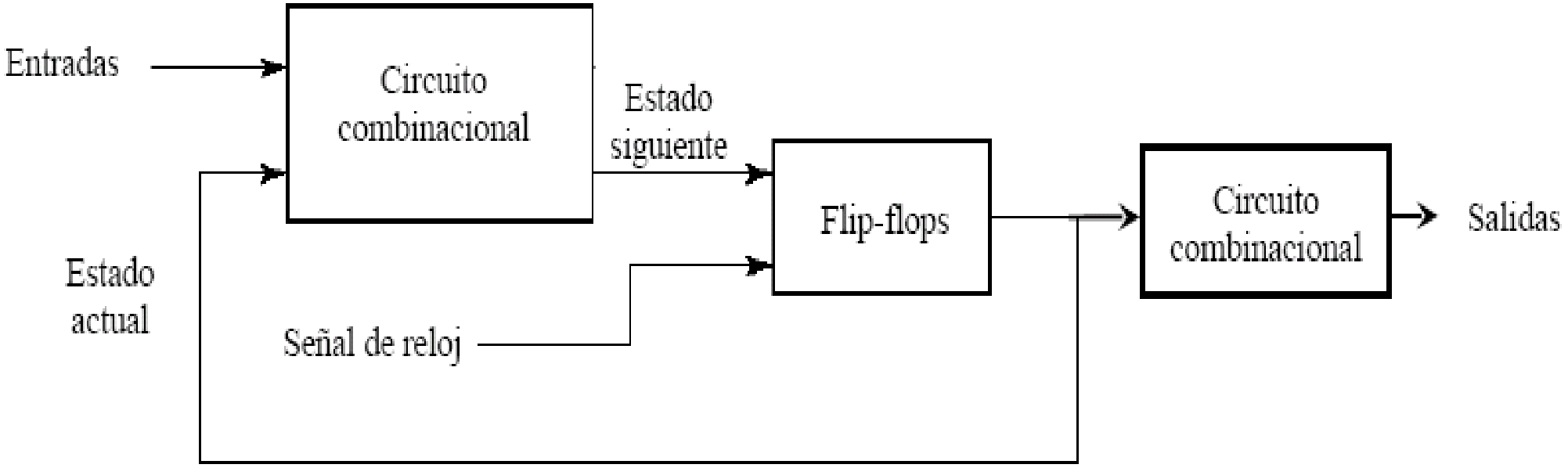
Diseño de sistemas secuenciales: Modelo Mealy

- Para diseñar el sistema, podemos considerar dos modelos para las salidas del sistema.
- Las funciones de salida dependen tanto del estado actual como de las variable de entrada➔Modelo Mealy.



Diseño de sistemas secuenciales: Modelo Moore

- Podemos simplificar el diseño, tomando las funciones de salida como dependientes sólo del estado actual➔Modelo Moore.
- De este modo, primero cambia el estado, y luego la salida.



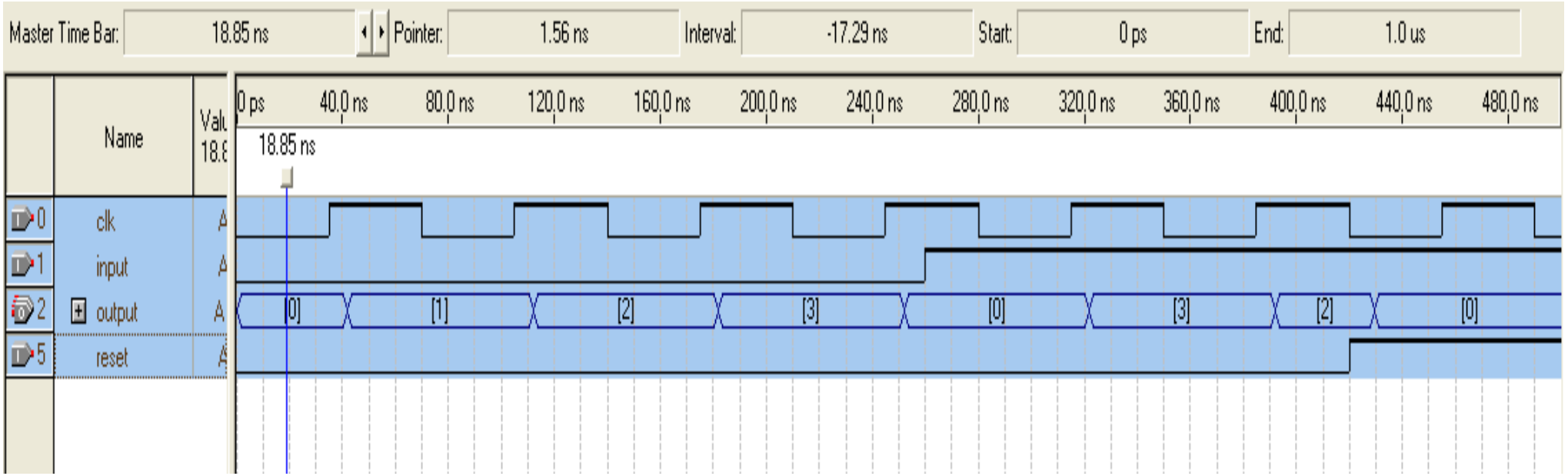
Diseño de sistemas secuenciales

- Para diseñar un sistema secuencial (supondremos modelo Moore), debemos elaborar un diagrama de estados a partir de la descripición del comportamiento del sistema.
- Debemos incluir en el diagrama todas las posibles transiciones, y considerar todos los estados necesarios para crear el comportamiento descrito.
- Dibujar un cronograma acorde con la descripción, suele ayudar a entender el comportamiento que preveemos para nuestro circuito.

Diseño de sistemas secuenciales

Descripción del comportamiento:

- Diseñar un circuito contador de dos bit que cuente en sentido ascendente cuando la señal X sea 0, y que cuente en sentido descendente cuando la señal X sea 1.



Diseño de sistemas secuenciales

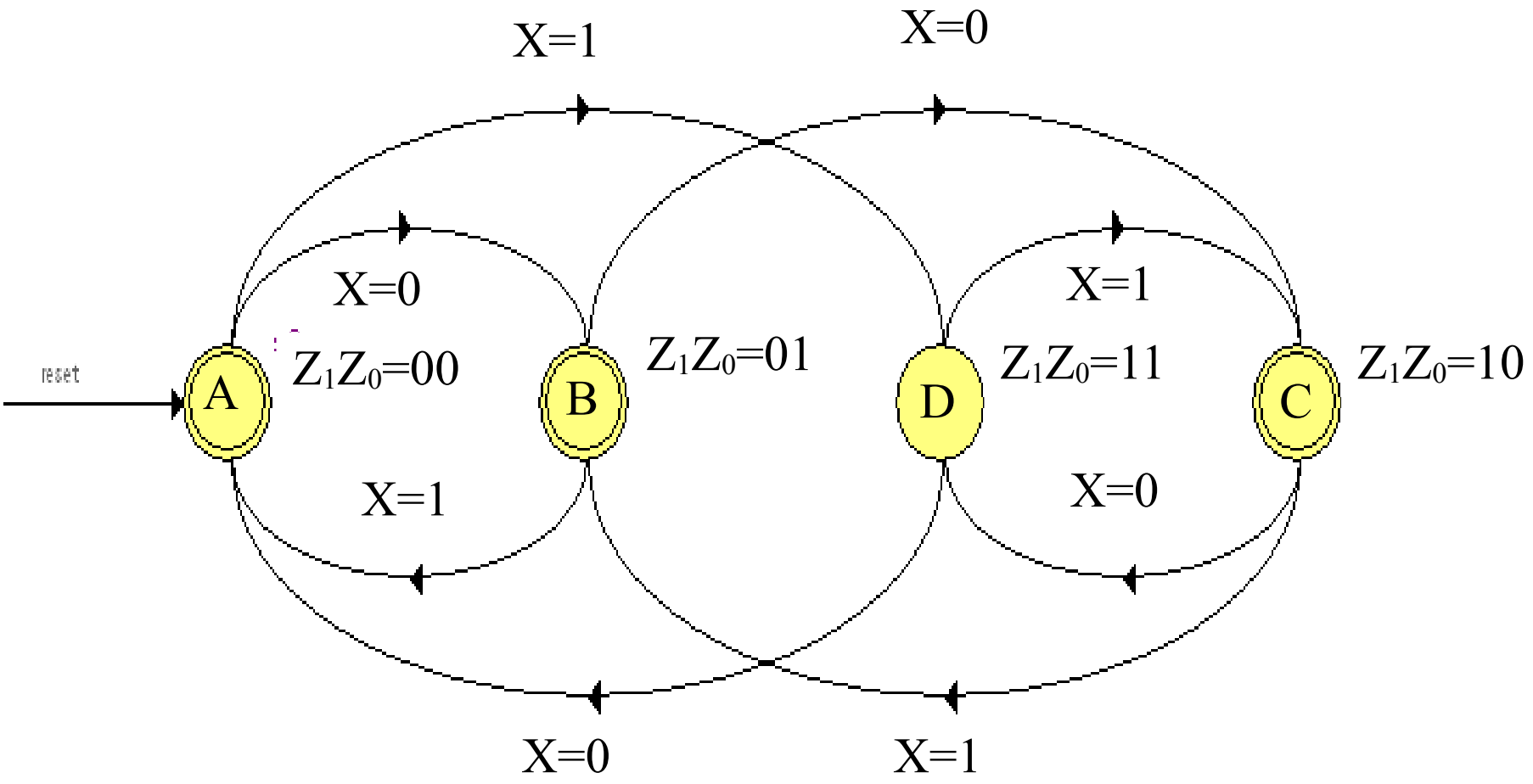


Diagrama de estados

Diseño de sistemas secuenciales

- Esta información la escribiremos en forma de tabla de estados, con una fila por cada estado y una columna por cada valor de entrada posible; escribiremos los valores de salida uno por cada fila (estado).

- Los estados se codifican en variables binarias (asignación de estados), que serán nuestras variables de estado.

Diseño de sistemas secuenciales

Estado actual	*X=0*	*X=1*	Z_1	Z_0
A	B	D	0	0
B	C	A	0	1
C	D	B	1	0
D	A	C	1	1

Tabla de estados

Estado actual	Q_1	Q_0
A	0	0
B	0	1
C	1	0
D	1	1

Asignación de estados

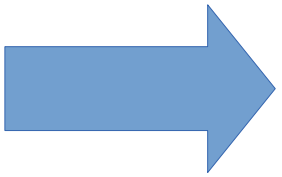
Q_1Q_0	*X=0*	*X=1*	Z_1	Z_0
00	01	11	0	0
01	10	00	0	1
10	11	01	1	0
11	00	10	1	1
	$Q_1^*Q_0^*$			

Diseño de sistemas secuenciales

- Los valores de nuevo estado (Q*) se hacen corresponder con las variables de entrada a los flip-flops (D, J-K) mediante sus ecuaciones.
- Los flip-flop D son más sencillos de usar, pero con los J-K se crean diseños más simplificados.
- Con estos datos, podemos escribir las ecuaciones de nuevo estado y de salida del sistema secuencial que cumple el funcionamiento descrito.

Diseño de sistemas secuenciales

C	J	K	Q*	Q*'
X	X	X	Q	Q'
↑	0	0	Q	Q'
↑	0	1	0	1
↑	1	0	1	0
↑	1	1	Q'	Q



Q	Q*	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

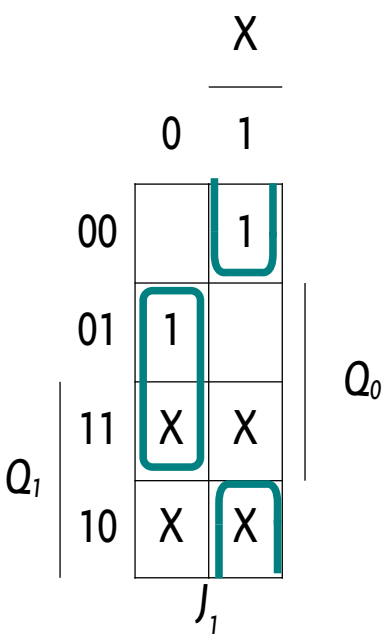
Flip-flop JK: tabla de la verdad y versión abreviada

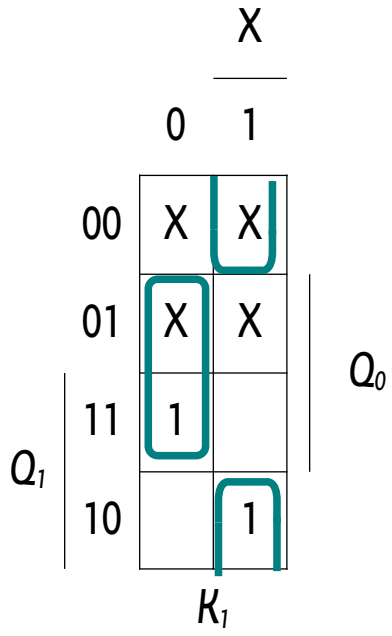
Q_1Q_0	X=0		X=1		Z_1	Z_0
00	0X	1X	1X	1X	0	0
01	1X	X1	0X	X1	0	1
10	X0	1X	X1	1X	1	0
11	X1	X1	X0	X1	1	1
	J_1K_1	J_0K_0				

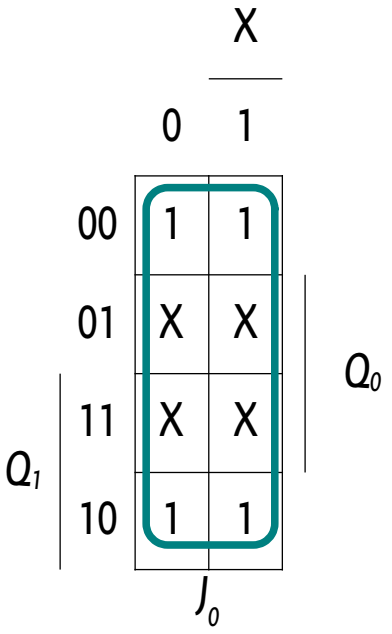
Diseño de sistemas secuenciales

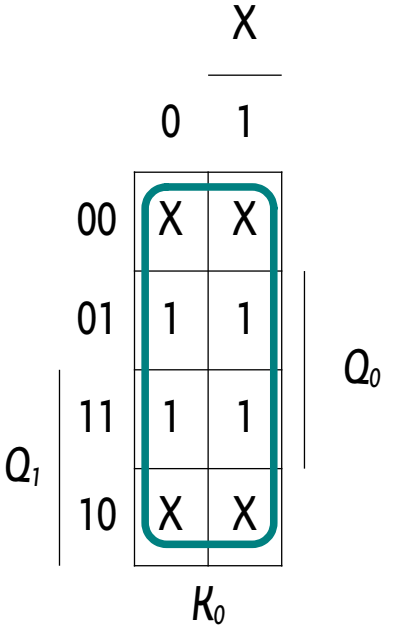
Q_1Q_0	Z_1	Z_0
00	0	0
01	0	1
10	1	0
11	1	1

$$Z_1 = Q_1$$
$$Z_0 = Q_0$$
$$J_1 = Q_0 \cdot \bar{X} + \bar{Q}_0 \cdot X = Q_0 \oplus X$$
$$K_1 = Q_0 \cdot \bar{X} + \bar{Q}_0 \cdot X = Q_0 \oplus X$$
$$J_0 = 1$$
$$K_0 = 1$$

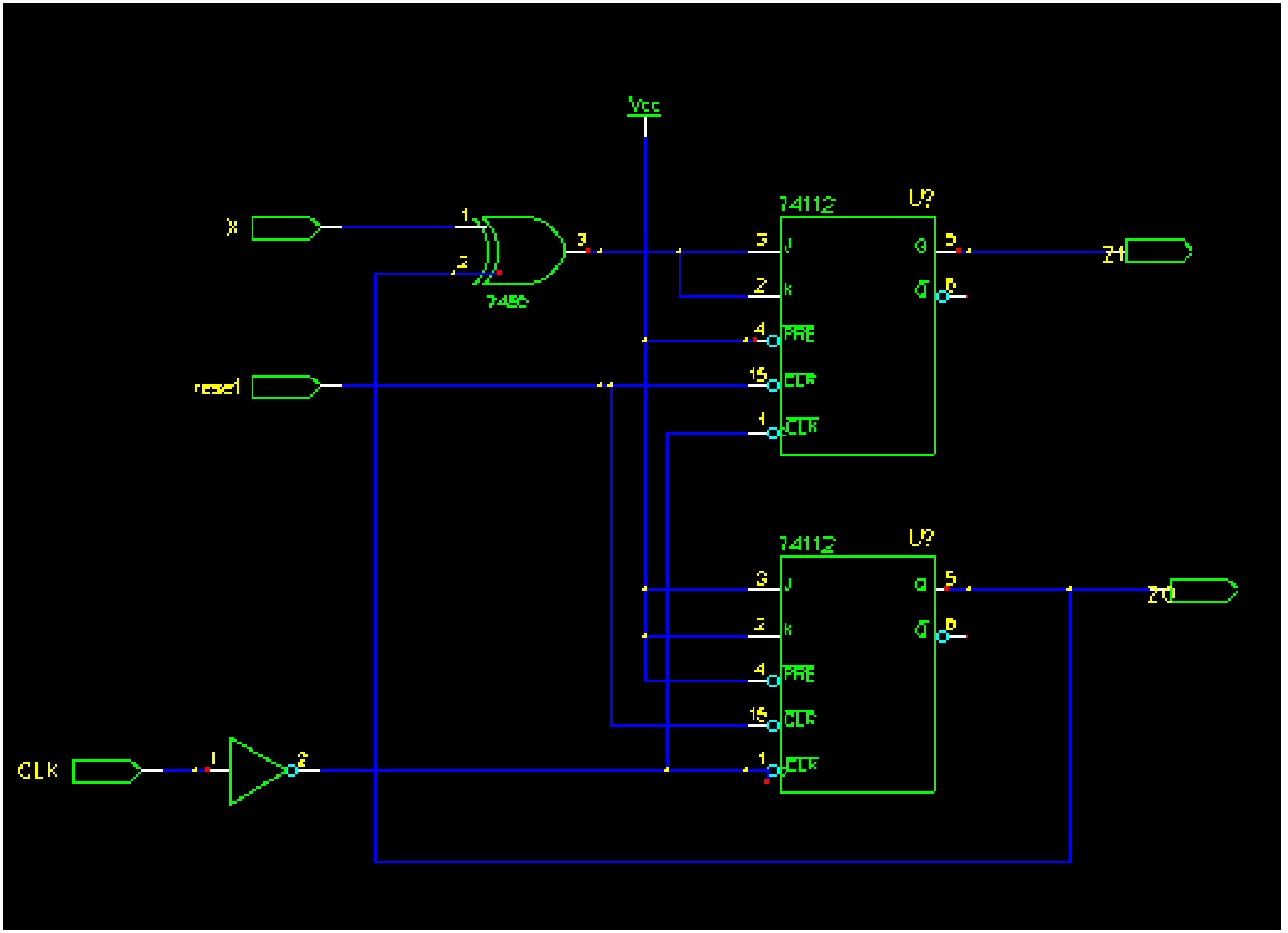








Diseño de sistemas secuenciales



$$Z_1 = Q_1$$
$$Z_0 = Q_0$$
$$J_1 = Q_0 \oplus X$$
$$K_1 = Q_0 \oplus X$$
$$J_0 = 1$$
$$K_0 = 1$$
$$\overline{CLR} = \overline{RESET}$$

Diseño de sistemas secuenciales

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY moore IS
PORT(
        clk:        IN   std_logic;
        x:          IN   std_logic;
        reset :     IN   std_logic;
        Z        :  OUT      std_logic_vector(1
        DOWNTO 0));
END moore;

ARCHITECTURE a OF moore IS

TYPE estado IS (s0, s1, s2, s3);
SIGNAL cuenta   : estado;


BEGIN
```

```
PROCESS (clk, reset)
BEGIN
IF reset = '1' THEN cuenta <= s0;
ELSIF (clk'EVENT AND clk='1') THEN

        CASE cuenta IS
            WHEN s0=>
                IF x = '0' THEN cuenta
                <= s1;
                        ELSE  cuenta  <=
                        s3;
                END IF;
            WHEN s1=>
                IF X = '0' THEN cuenta
                <= s2;
                        ELSE  cuenta  <=
                        s0;
                END IF;
```

Diseño de sistemas secuenciales

```
        WHEN s2=>
            IF x = '0' THEN cuenta <=
            s3;
                    ELSE cuenta <= s1;
            END IF;
        WHEN s3 =>
            IF x = '0' THEN cuenta <=
            s0;
                    ELSE cuenta <= s2;
            END IF;
        END CASE;

END IF;

END PROCESS;
```

```
PROCESS (cuenta)
BEGIN

CASE cuenta IS
        WHEN s0 =>z <= "00";
        WHEN s1 =>z <= "01";
        WHEN s2 =>z <= "10";
        WHEN s3 =>z <= "11";
END CASE;

END PROCESS;

END a;
```