Titulación: Ingeniería Informática

Asignatura: Fundamentos de Computadores

Bloque 3: Sistemas secuenciales

Tema 8: Elementos de memoria

José Ignacio Martínez Torre

Luis Rincón Córcoles

ÍNDICE

BIBLIOGRAFÍA

- Román Hermida, Ana Mº del Corral, Enric Pastor, Fermín Sánchez
  "Fundamentos de Computadores" , cap. 5
  Editorial Síntesis
- Thomas L. Floyd
  "Fundamentos de Sistemas Digitales", cap. 8
  Editorial Prentice Hall
- Daniel D. Gajski
  "Principios de Diseño Digital", cap. 6
  Editorial Prentice Hall
- M. Morris Mano
  "Diseño Digital", cap. 6
  Editorial Prentice Hall

INTRODUCCIÓN

- Los elementos de memoria básicos se llaman biestables, por que pueden estar en dos posibles estados: con la salida a ‘0’ o a ‘1’.
- Según las entradas de datos de que disponen podemos establecer varios tipos de biestables:
  - S-R: entradas de puesta a 1 (S, *set*) y puesta a 0 (R, *reset*)
  - J-K: entradas de puesta a 1 (J, *set*) y puesta a 0 (K, *reset*)
  - D: entrada de datos (D)
  - T: entrada de inversión o basculamiento (*toggle*)
- Los biestables pueden ser asíncronos o síncronos:
  - Asíncronos (*latch*): cuando cambia cualquiera de las entradas puede cambiar la salida.
  - Síncronos: la salida puede cambiar sólo cuando la señal de sincronismo lo permite.

INTRODUCCIÓN

- Además, dentro de los biestables síncronos se puede diferenciar entre:
  - Síncronos por nivel (*gated latch*): la salida puede cambiar cuando la señal de sincronismo está en un determinado nivel (alto o bajo).
  - Síncronos por flanco (*low/high edge-triggered flip-flop*): la salida puede cambiar cuando la señal de sincronismo pasa de un nivel a otro.
    - Cuando pasa de nivel bajo a alto => síncrono por flanco de subida.
    - Cuando pasa de nivel alto a bajo => síncrono por flanco de bajada.
- Los más utilizados son los **biestables síncronos por flanco**, y reciben el nombre de ***flip-flops***.
- Normalmente los biestables síncronos cuentan con entradas asíncronas que se utilizan para forzar un valor determinado en los mismos al margen del reloj (prevalecen sobre él).
  - Puesta a 0 asíncrona (*clear*, *reset*)
  - Puesta a 1 asíncrona (*preset*, *set*)

INTRODUCCIÓN

- Tabla de excitación de un biestable:
  - Muestra las entradas que hay que introducir en un biestable para gobernar sus transiciones entre estados.

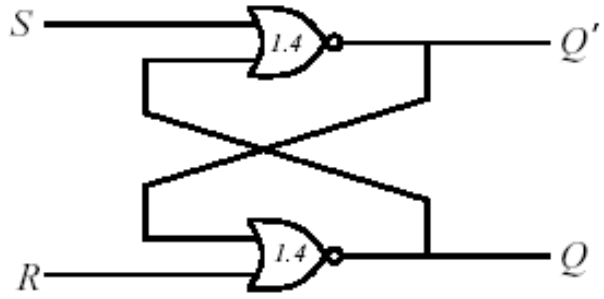
Estado actual	Estado siguiente	Entradas			
		X0	X1	…	Xn
Si	Sj	V0	V1	…	Vn
.	.		.		
.	.		.		
.	.		.		

ÍNDICE

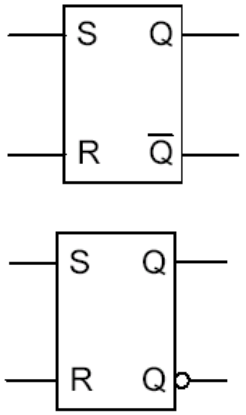
BIESTABLES ASÍNCRONOS

- El elemento de memoria más sencillo es el *latch* SR que tiene dos puertas NOR o NAND de dos entradas conectadas entre sí.
  - Es un biestable con un estado de *Set* (puesta a 1) y otro de *Reset* (puesta a 0).

Latch SR NOR Asíncrono:

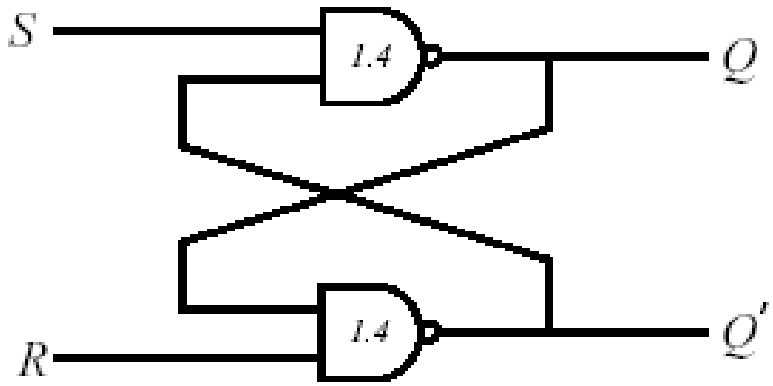


S	R	Q	Q (next)	Q'(next)	
0	0	0	0	1	(hold)
0	0	1	1	0	(hold)
0	1	X	0	1	(reset)
1	0	X	1	0	(set)
1	1	X	0	0	(?)



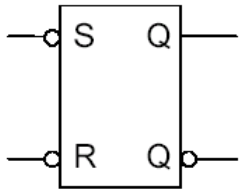
BIESTABLES ASÍNCRONOS

- ***Latch* SR NAND Asíncrono:**



S	R	Q	Q (next)	Q' (next)	
0	0	X	1	1	(?)
0	1	X	1	0	(set)
1	0	X	0	1	(reset)
1	1	0	0	1	(hold)
1	1	1	1	0	(hold)

El *latch* SR NAND opera con señales activas bajas

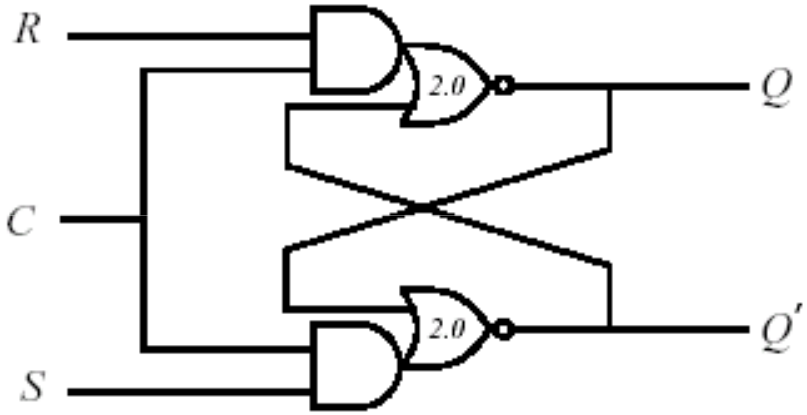


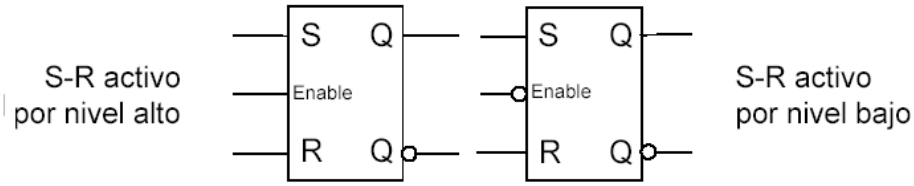
ÍNDICE

BIESTABLES SÍNCRONOS POR NIVEL

***Latch* SR NOR síncrono por nivel (*gated latch*):** añadiendo unas puertas AND y una señal de sincronismo C, se puede convertir el *latch* SR NOR asíncrono en uno síncrono por nivel.

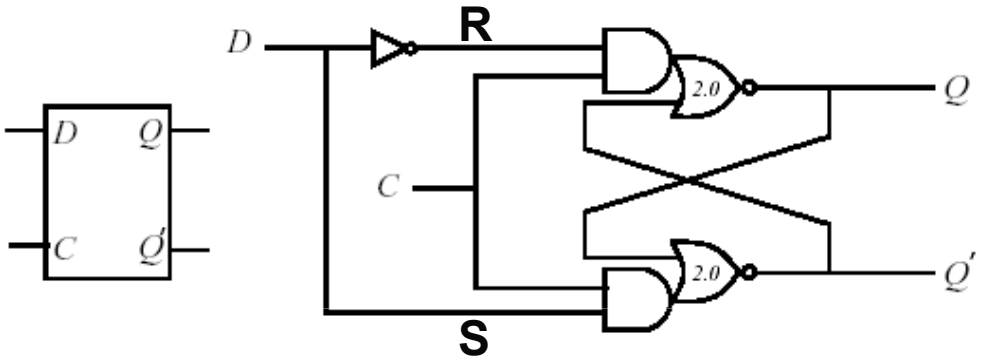
C	S	R	Q	Q (next)	
0	X	X	0	0	(inactive)
0	X	X	1	1	(inactive)
1	0	0	0	0	(hold)
1	0	0	1	1	(hold)
1	0	1	X	0	(reset)
1	1	0	X	1	(set)
1	1	1	X	NA	(?)





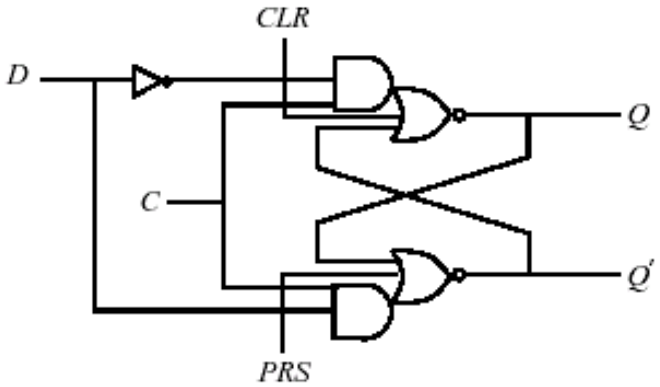
BIESTABLES SÍNCRONOS POR NIVEL

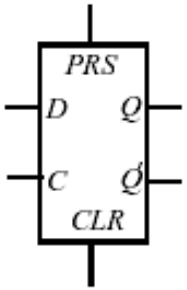
***Latch* D síncrono (*gated latch*):** añadiendo un inversor para que la entrada R y la entrada S nunca tengan el mismo valor se elimina el estado prohibido, y se obtiene un **biestable D**.



C	D	Q	Q (next)
0	X	0	0
0	X	1	1
1	0	X	0
1	1	X	1

***Latch* D síncrono (*gated latch*) con *Clear* y *Preset* asíncronos:** se pueden añadir entradas de *clear* y *preset* asíncronas conectándolas directamente el *latch* SR interno.





CLR	PRS	
0	0	ON
0	1	Preset (Q=1)
1	0	Clear (Q=0)
1	1	No permitido

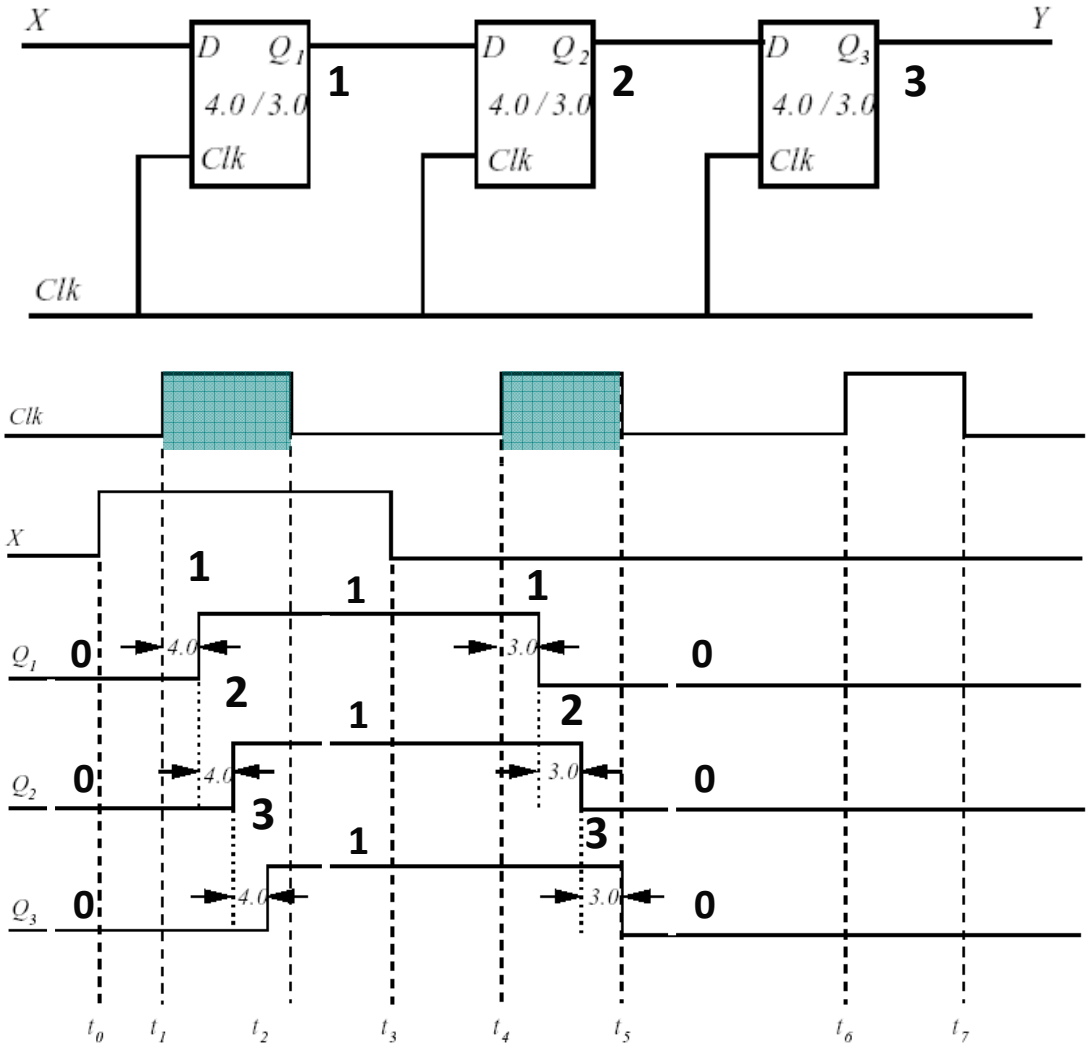
BIESTABLES SÍNCRONOS POR NIVEL

Comportamiento indeseado de los biestables disparados por nivel:

Desplazamiento de un valor alto X=1 por 3 *latches*.
Comportamiento deseado: 100, 010, 001(/2)

Comportamiento obtenido: 111, 000, 000
Se debe a que los *latches* son transparentes y cuando C=H los cambios en D se propagan

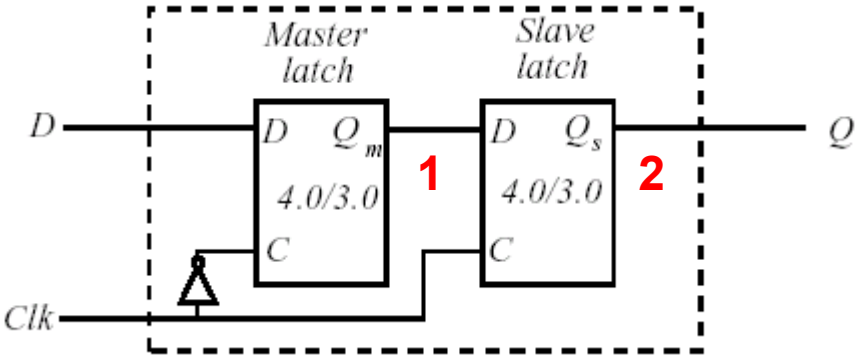
Solución:
***Flip-flop master-slave* o disparado por flanco**



ÍNDICE

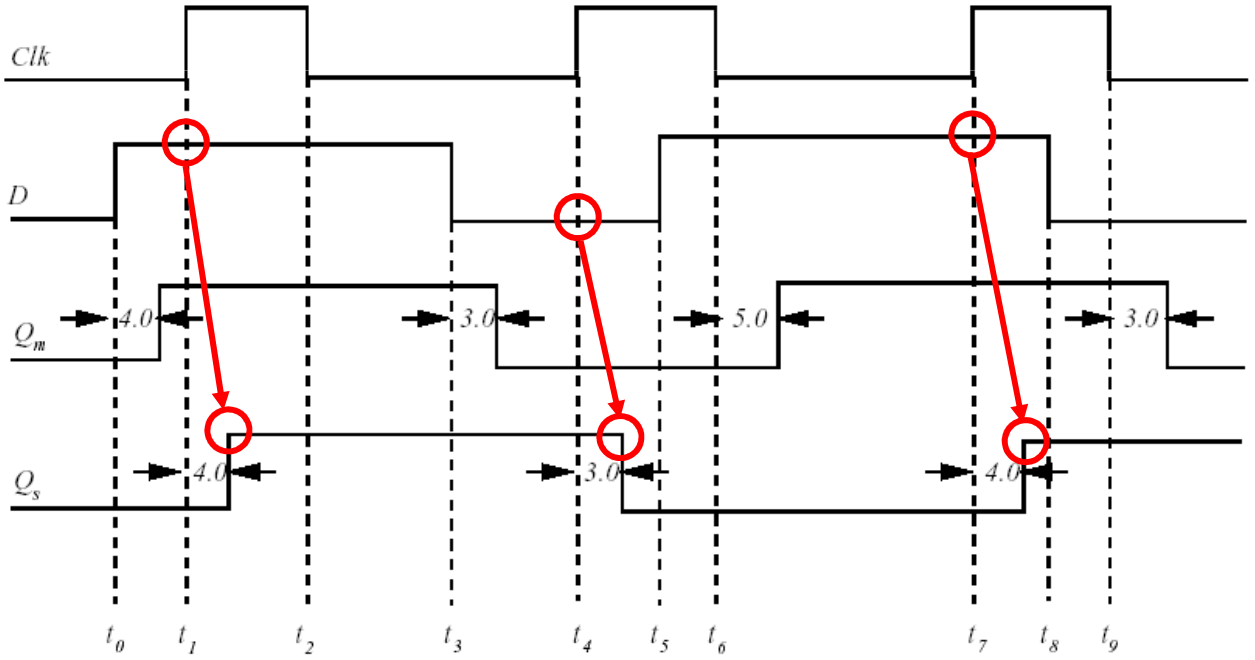
BIESTABLES SÍNCRONOS POR FLANCO

***Flip-flop* D *master-slave*:** utilizando dos biestables síncronos por nivel se puede construir un biestable síncrono por flanco.



Con Clk=0 el primer D captura y el segundo D no cambia
Con Clk=1 al revés

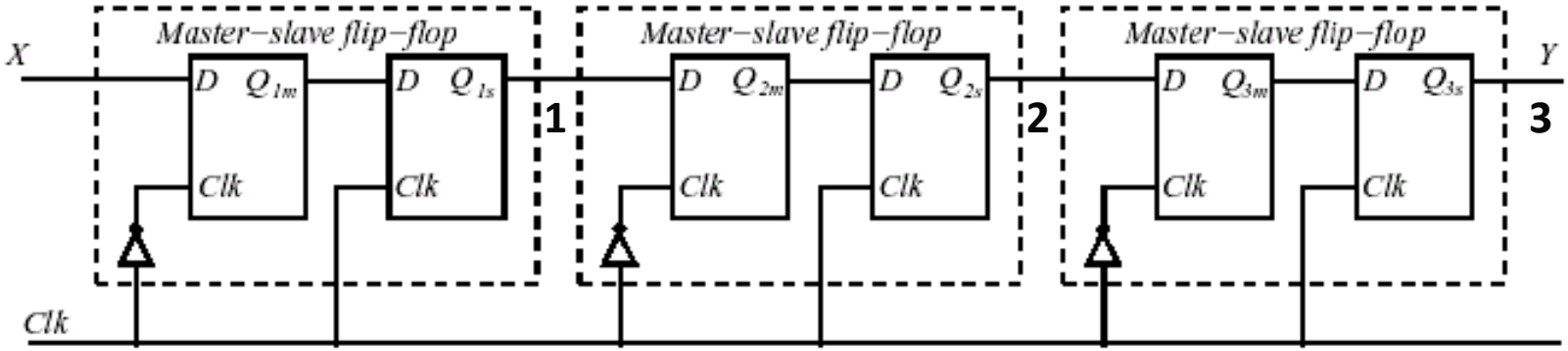
Este modo de operación implica que **se captura D en el flanco de subida del reloj Clk**



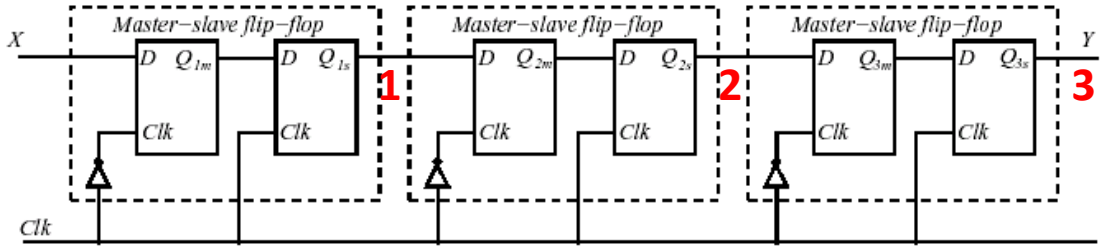
BIESTABLES SÍNCRONOS POR FLANCO

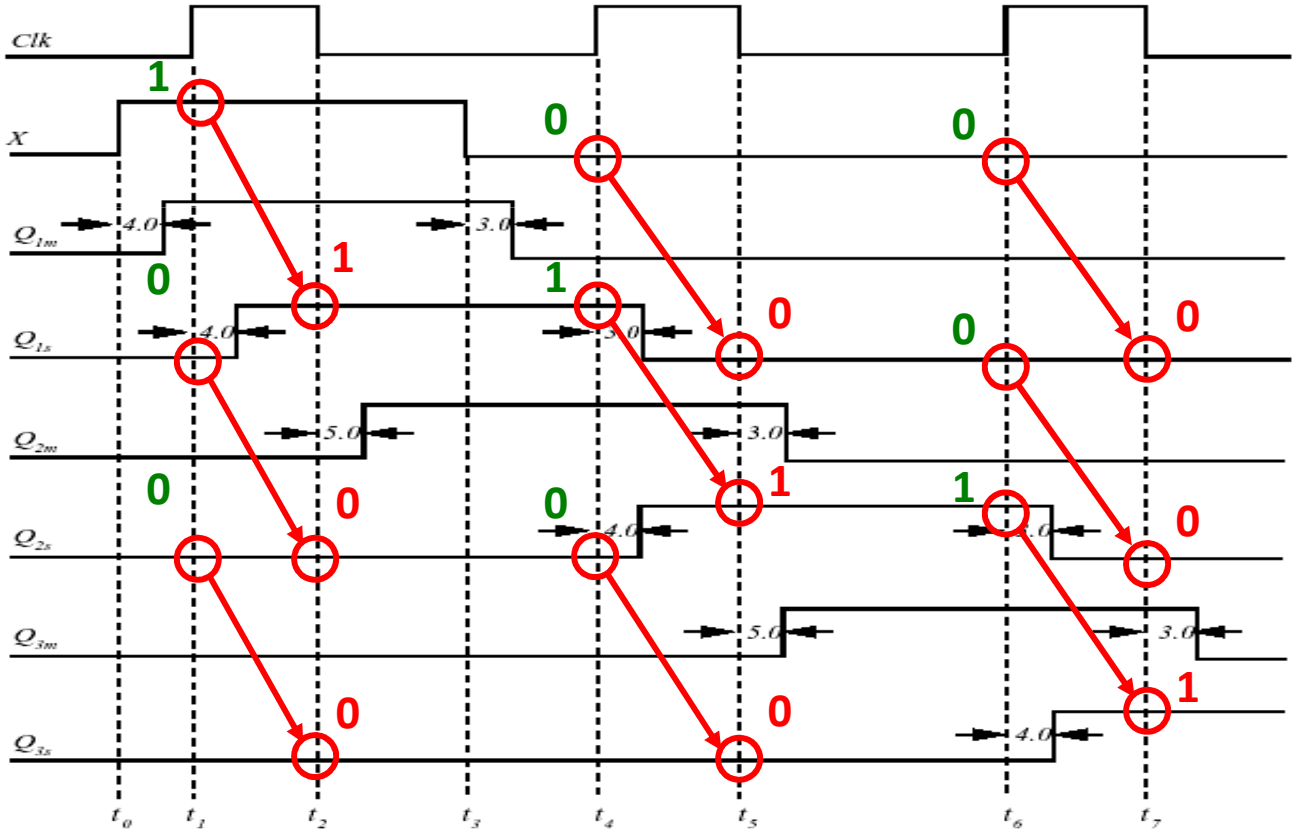
Comportamiento deseado de los biestables disparados por flanco:

Desplazamiento de un valor alto X=1 por 3 biestables *master-slave*.
Comportamiento deseado: 100, 010, 001(/2)



BIESTABLES SÍNCRONOS POR FLANCO





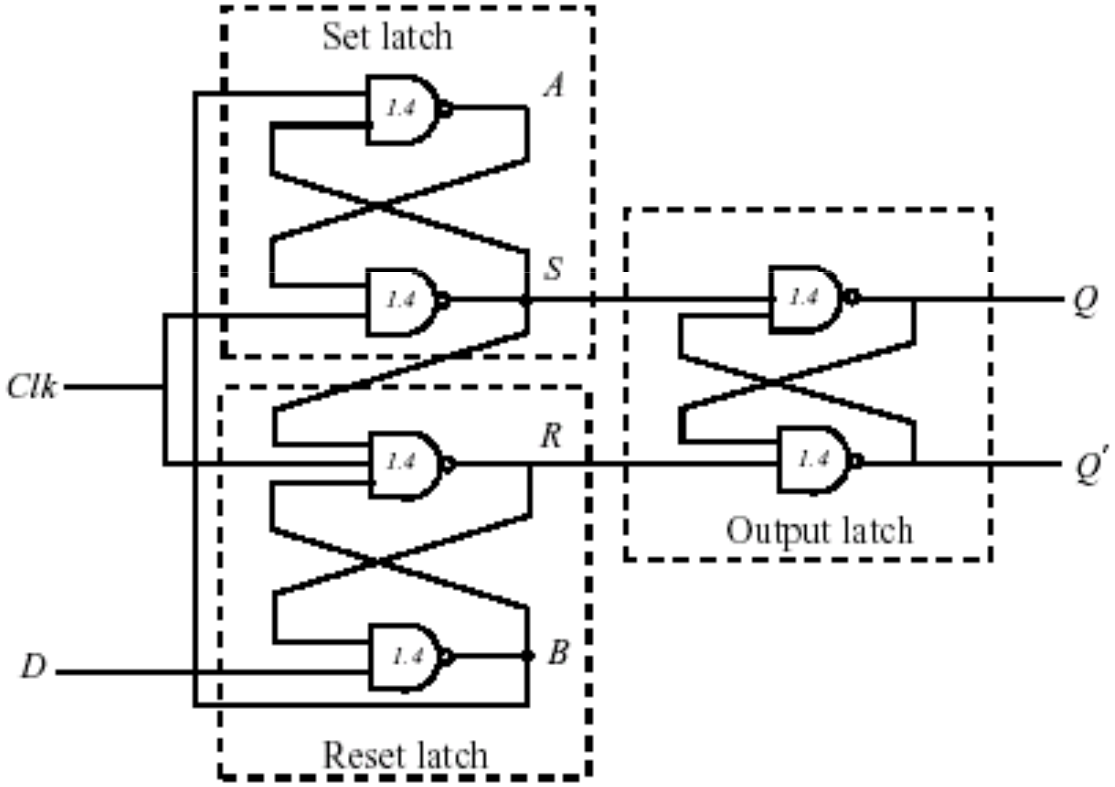
100, 010, 001

Desplazamiento correcto

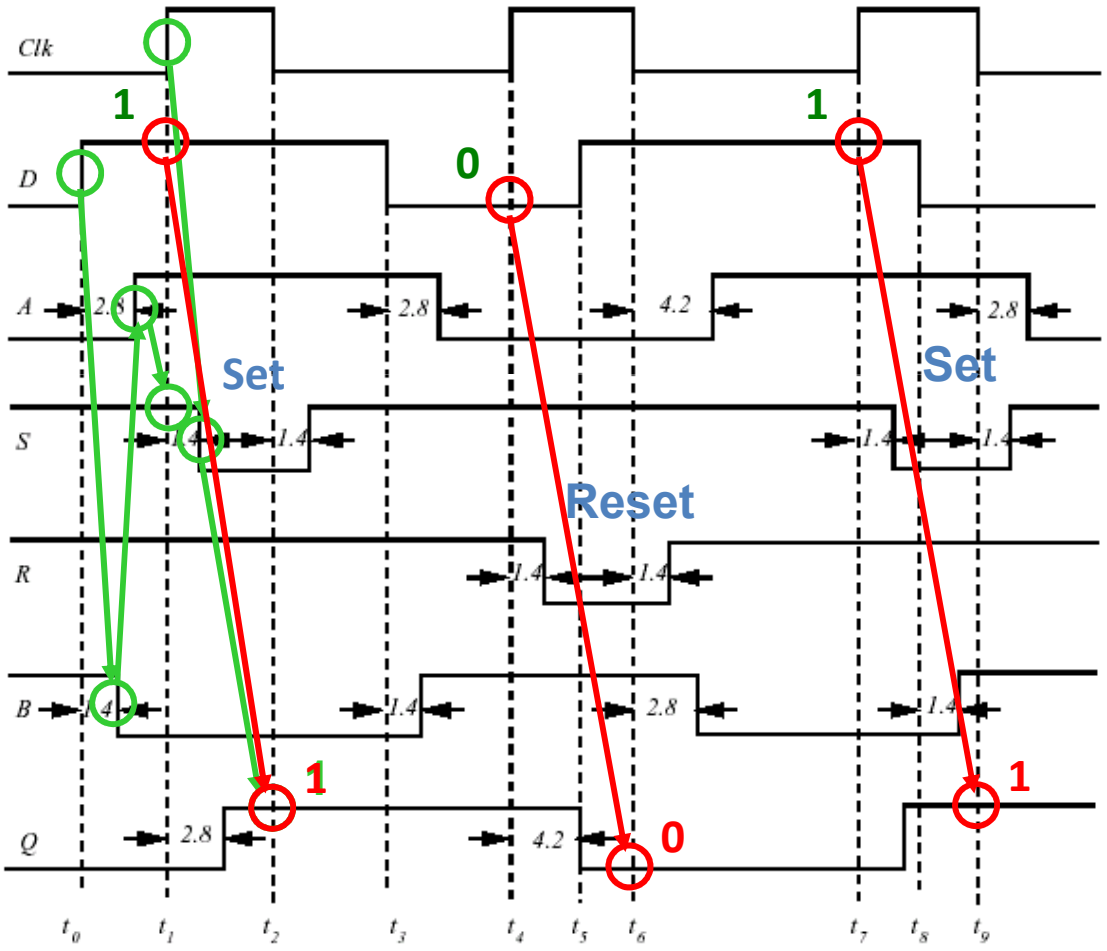
BIESTABLES SÍNCRONOS POR FLANCO

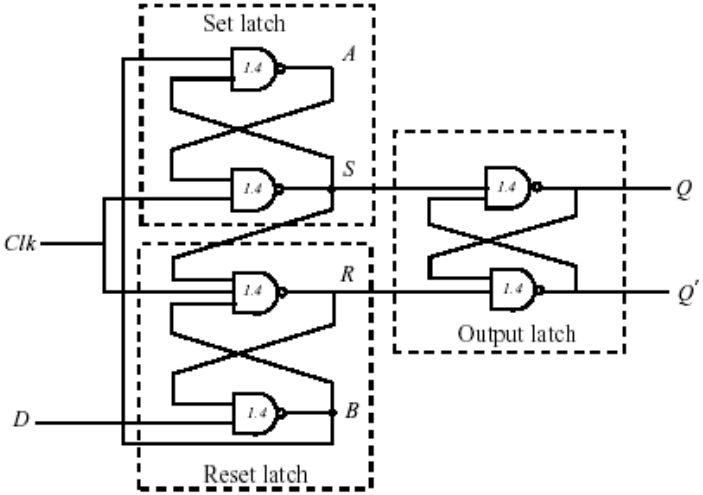
***Flip-flop* D disparado por flanco de subida del reloj:**

Consiste en añadir al *latch* convencional dos *latches* que aseguren que en S y en R nunca se da la condición prohibida.



BIESTABLES SÍNCRONOS POR FLANCO



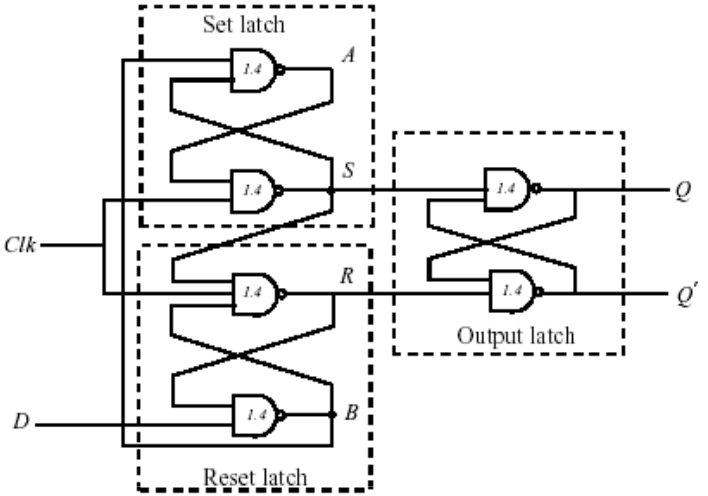


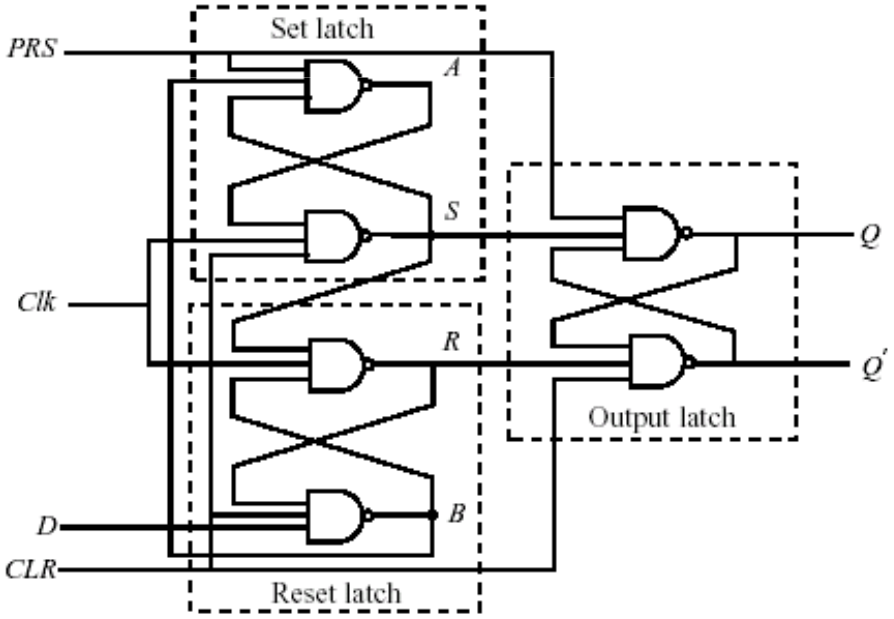
Este modo de operación implica que **se captura D en el flanco de subida del reloj Clk**

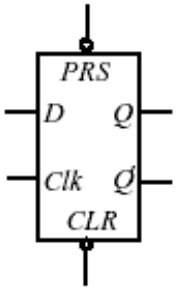
BIESTABLES SÍNCRONOS POR FLANCO

***Flip-flop* D disparado por flanco de subida del reloj con *Clear* y *Preset*:**

El *Flip-flop* D base + 2 entradas adicionales CLR y PRS en las NAND del *latch* de salida



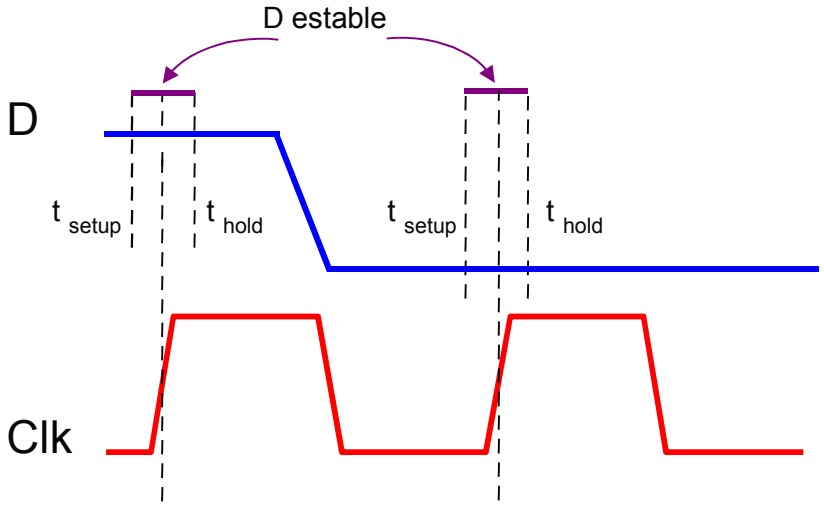




CLR	PRS	
0	0	No permitido
0	1	Clear (Q=0)
1	0	Preset (Q=1)
1	1	ON

PARÁMETROS DE LOS BIESTABLES

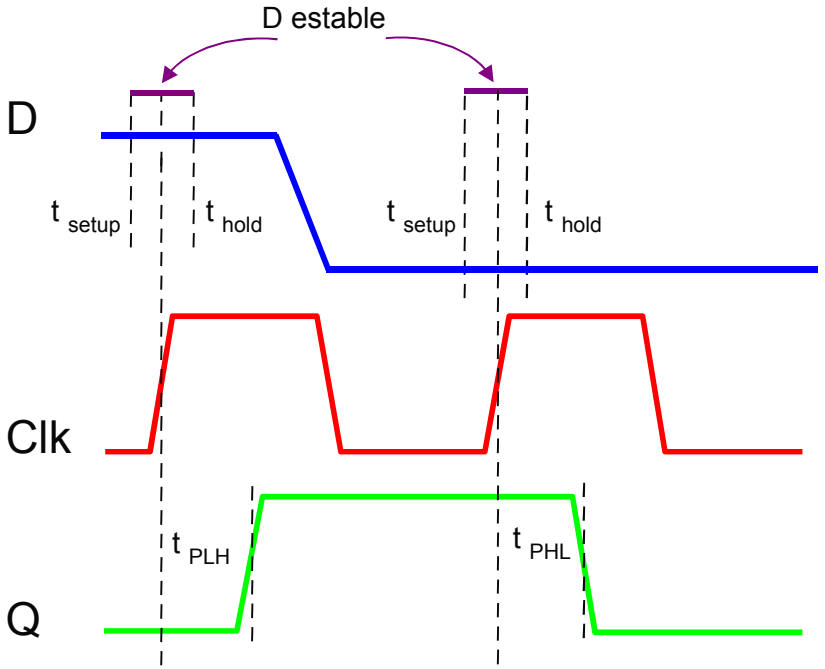
- Para que un *flip-flop* funcione correctamente, hay que respetar dos tiempos:
  - Tiempo de preestabilización (*setup time*): tiempo previo al flanco activo en el que las entradas deben permanecer estables.
  - Tiempo de mantenimiento (*hold time*): tiempo posterior al flanco activo en el que las entradas deben permanecer estables.



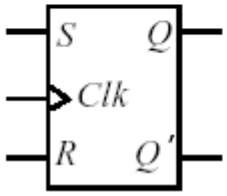
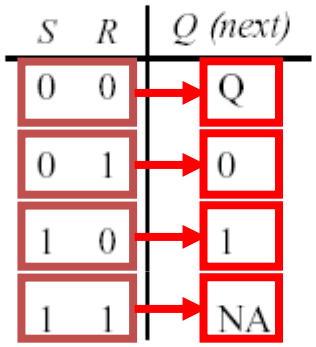
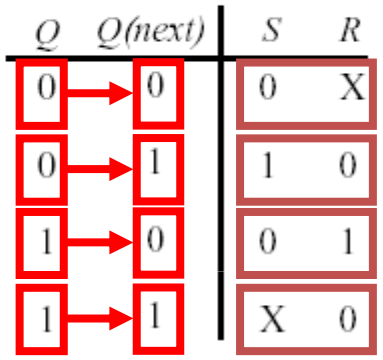
- Si no se respetan estos tiempos: metaestabilidad.
  - El *flip-flop* puede tomar un valor que ni es 1 ni es 0.
  - El *flip-flop* se recupera solo, pero no se sabe cuánto tiempo tarda en hacerlo.

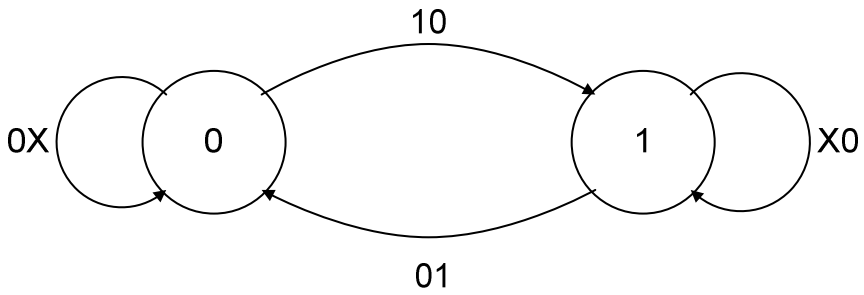
PARÁMETROS DE LOS BIESTABLES

- Otro tiempo importante es el retardo de propagación (*propagation delay*):
  - Es el tiempo transcurrido desde el flanco hasta que la salida del biestable presenta el nuevo estado.
  - El tiempo de cambio de valor alto a bajo (t_{PHL}) y el de bajo a alto (t_{PLH}) no tienen por qué ser iguales.
  - El retardo de propagación es siempre mayor que el tiempo de mantenimiento.



FLIP-FLOP SR

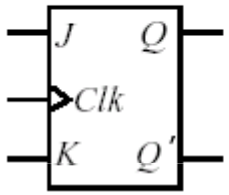
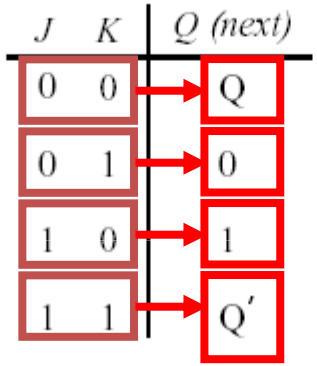
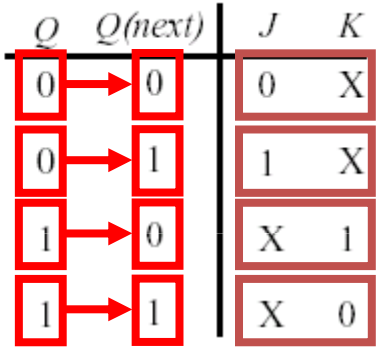
Flip–flop name	*Flip–flop symbol*	*Characteristic table*	*Characteristic equation*	*Excitation table*
SR			$Q\ (next) = S + R'Q$ $SR = 0$	



FLIP-FLOP SR: MODELO VHDL

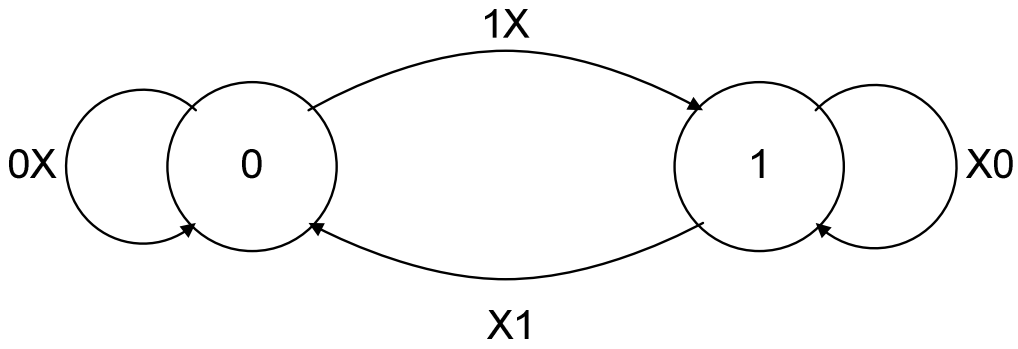
```
library ieee;
use ieee.std_logic_1164.all;

entity flipflopSR is
  port(clk, rst, S, R: in std_logic;
       Q: out std_logic);
end flipflopSR;

architecture funcional of flipflopSR is
begin
  process(clk, rst)
    begin
    if rst = '1' then
      Q <= '0';
    elsif rising_edge(clk) then
      if S = '1' then
        Q <= '1';
      elsif R = '1' then
        Q <= '0';
      end if;
    end if;
  end process;
end funcional;
```

FLIP-FLOP JK

Flip–flop name	*Flip–flop symbol*	*Characteristic table*	*Characteristic equation*	*Excitation table*
JK			$Q\ (next) = JQ' + K'Q$	

J	K	Q (next)
0	0	Q
0	1	0
1	0	1
1	1	Q'

Q	Q(next)	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

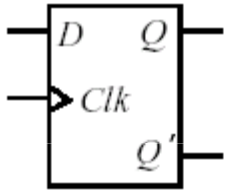
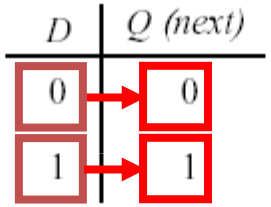
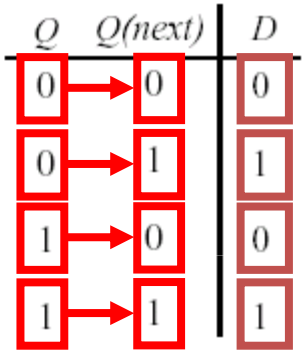


FLIP-FLOP JK: MODELO VHDL

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

entity flipflopJK is
  port(clk, rst, J, K: in std_logic;
       Q: out std_logic);
end flipflopJK;
```

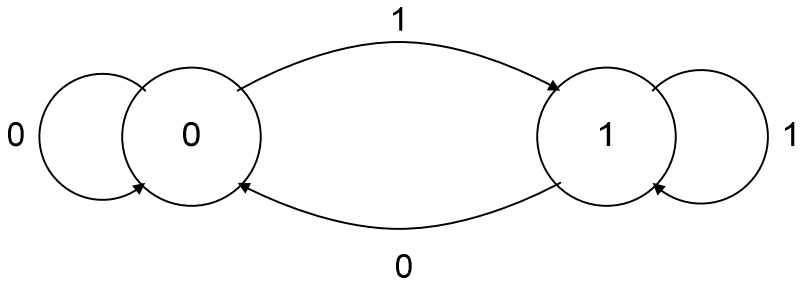
```
architecture funcional of flipflopJK is
signal JK: std_logic_vector(1 downto 0);
signal Q_aux: std_logic;
begin
  JK <= J&K;
  process(clk, rst)
    begin
    if rst = '1' then
      Q_aux <= '0';
    elsif rising_edge(clk) then
      case JK is
        when "00" => Q_aux <= Q_aux;
        when "01" => Q_aux <= '0';
        when "10" => Q_aux <= '1';
        when others => Q_aux <= not Q_aux;
      end case;
    end if;
  end process;
  Q <= Q_aux;
end funcional;
```

FLIP-FLOP D

Flip–flop name	*Flip–flop symbol*	*Characteristic table*	*Characteristic equation*	*Excitation table*
D			$Q(next) = D$	

D	*Q (next)*
0	0
1	1

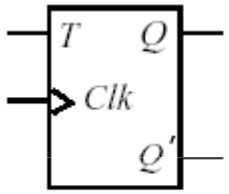
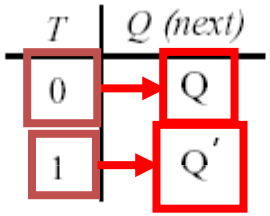
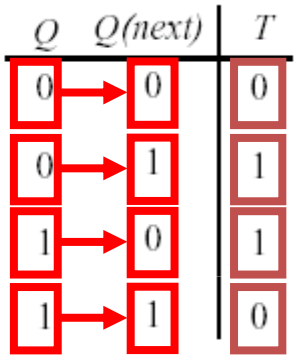
Q	*Q(next)*	*D*
0	0	0
0	1	1
1	0	0
1	1	1



FLIP-FLOP D: MODELO VHDL

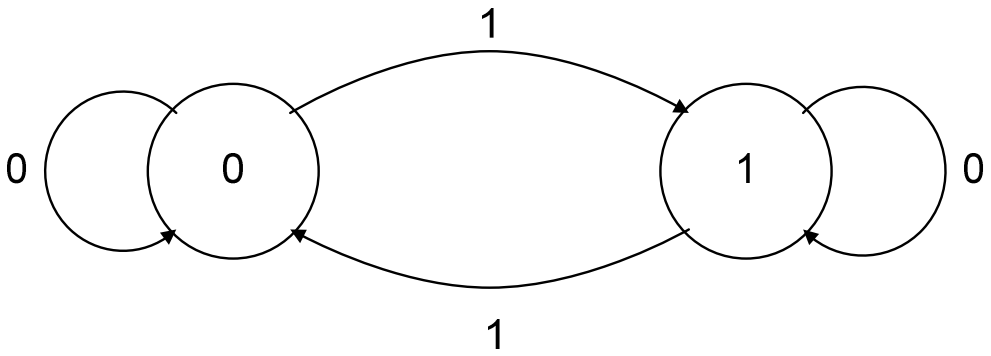
```
library ieee;
use ieee.std_logic_1164.all;

entity flipflopD is
  port(clk, rst, D: in std_logic;
       Q: out std_logic);
end flipflopD;

architecture funcional of flipflopD is
begin
  process(clk,rst)
  begin
    if rst = '1' then
      Q <= '0';
    elsif rising_edge(clk) then
      Q <= D;
    end if;
  end process;
end funcional;
```

FLIP-FLOP T

Flip–flop name	*Flip–flop symbol*	*Characteristic table*	*Characteristic equation*	*Excitation table*
T			$Q\,(next) = TQ' + T'Q$	

T	*Q (next)*
0	Q
1	Q'

Q	*Q(next)*	*T*
0	0	0
0	1	1
1	0	1
1	1	0



FLIP-FLOP T: MODELO VHDL

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

entity flipflopT is
  port(clk, rst, T: in std_logic;
       Q: out std_logic);
end flipflopT;

architecture funcional of flipflopT is
signal Q_aux: std_logic;
begin
  process(clk,rst)
  begin
    if rst = '1' then
      Q_aux <= '0';
    elsif rising_edge(clk) then
      if T = '1' then
        Q_aux <= not Q_aux;
      end if;
    end if;
  end process;
  Q <= Q_aux;
end funcional;
```

ÍNDICE

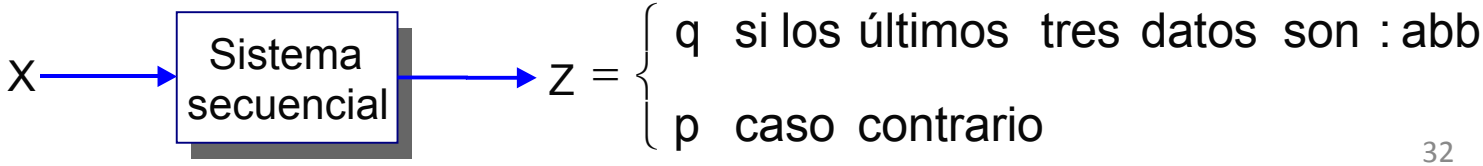
SÍNTESIS DE FSM CON *FLIP-FLOPS*

- Hemos visto ya la síntesis de máquinas secuenciales utilizando *flip-flops* de tipo D.
- Revisaremos la síntesis, pero empleando ahora *flip-flops* de otras clases.
- La dificultad estriba en la síntesis de la función de transición.
  - Con *flip-flops* D basta con conocer el estado próximo, y suministrar a la entrada D el valor del estado buscado.
  - Con otros *flip-flops* hay que conocer el estado actual y el próximo, y en función de ambos establecer el valor de las entradas.
  - En los *flip-flops* SR y JK será preciso calcular dos funciones por cada variable de estado.

- Usaremos el mismo ejemplo que en el tema anterior.
  - Materializaremos únicamente máquinas de Mealy.

$X \in \{a, b\}$

$Z \in \{p, q\}$



SÍNTESIS DE FSM CON *FLIP-FLOPS*

Diagrama de estados

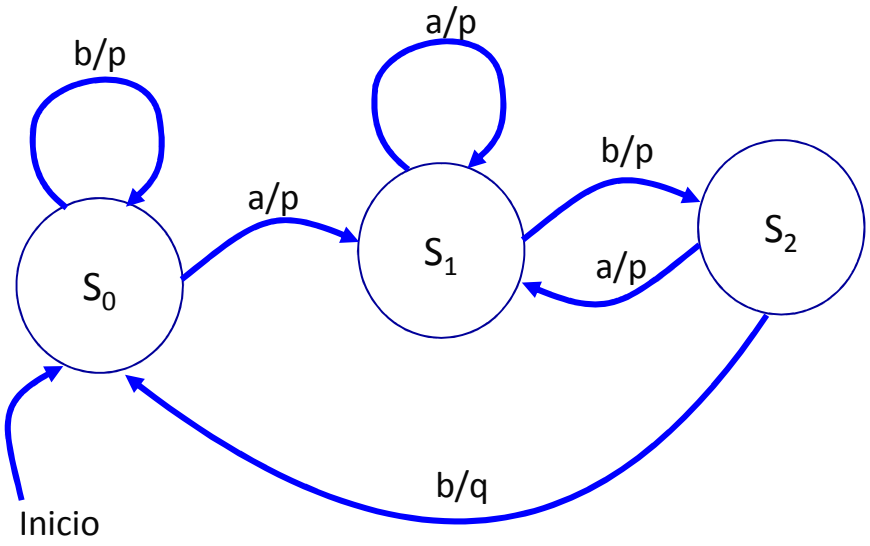


Tabla de estados y salidas

Estado actual	Entrada actual	
	A	**b**
S_0	S_1/p	S_0/p
S_1	S_1/p	S_2/p
S_2	S_1/p	S_0/q
	Estado siguiente / Salida	

Codificación binaria

Entrada		Salida		Estado		
X(t)	**X_0**	**Z(t)**	**Z_0**	**S(t)**	**Q_1**	**Q_0**
a	0	p	0	S_0	0	0
b	1	q	1	S_1	0	1
				S_2	1	0

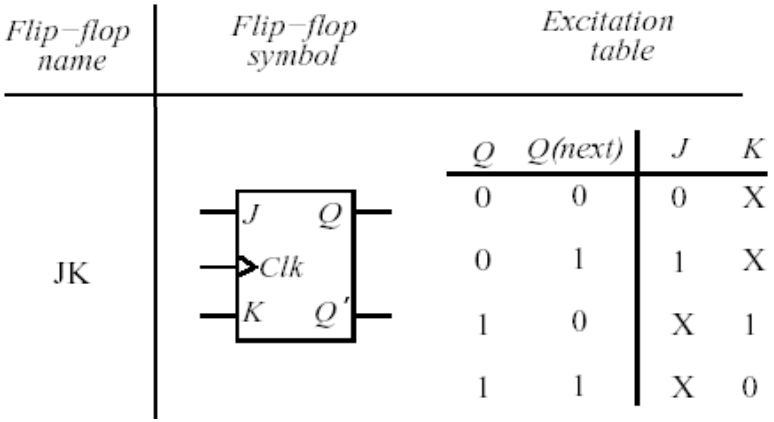
Tabla de estados y salidas

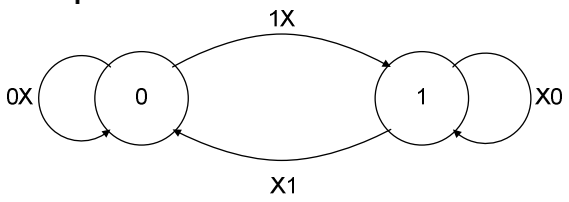
S(t)		X_0	
Q_1	**Q_0**	**0**	**1**
0	0	01/0	00/0
0	1	01/0	10/0
1	0	01/0	00/1

SÍNTESIS DE FSM CON *FLIP-FLOPS*: JK

- Si elegimos biestables JK, tenemos que crear una nueva tabla de excitación y salida.

S(t)		X_0	
Q_1	Q_0	0	1
0	0	01/0	00/0
0	1	01/0	10/0
1	0	01/0	00/1





S(t), X			S(t+1), Z			Entradas de excitación			
Q_1	Q_0	X	Q'_1	Q'_0	Z	J_1	K_1	J_0	K_0
0	0	0	0	1	0	0	X	1	X
0	0	1	0	0	0	0	X	0	X
0	1	0	0	1	0	0	X	X	0
0	1	1	1	0	0	1	X	X	1
1	0	0	0	1	0	X	1	1	X
1	0	1	0	0	1	X	1	0	X
1	1	0	X	X	X	X	X	X	X
1	1	1	X	X	X	X	X	X	X

SÍNTESIS DE FSM CON *FLIP-FLOPS*: JK

- Es preciso materializar J_1, K_1, J_0, K_0 y Z.

S(t), X			S(t+1), Z			Entradas de excitación			
Q_1	Q_0	X	Q'_1	Q'_0	Z	J_1	K_1	J_0	K_0
0	0	0	0	1	0	0	X	1	X
0	0	1	0	0	0	0	X	0	X
0	1	0	0	1	0	0	X	X	0
0	1	1	1	0	0	1	X	X	1
1	0	0	0	1	0	X	1	1	X
1	0	1	0	0	1	X	1	0	X
1	1	0	X	X	X	X	X	X	X
1	1	1	X	X	X	X	X	X	X

Función de transición

$$J_1(Q_1, Q_0, X_0) = X_0 \cdot Q_0$$

$$K_1(Q_1, Q_0, X_0) = 1$$

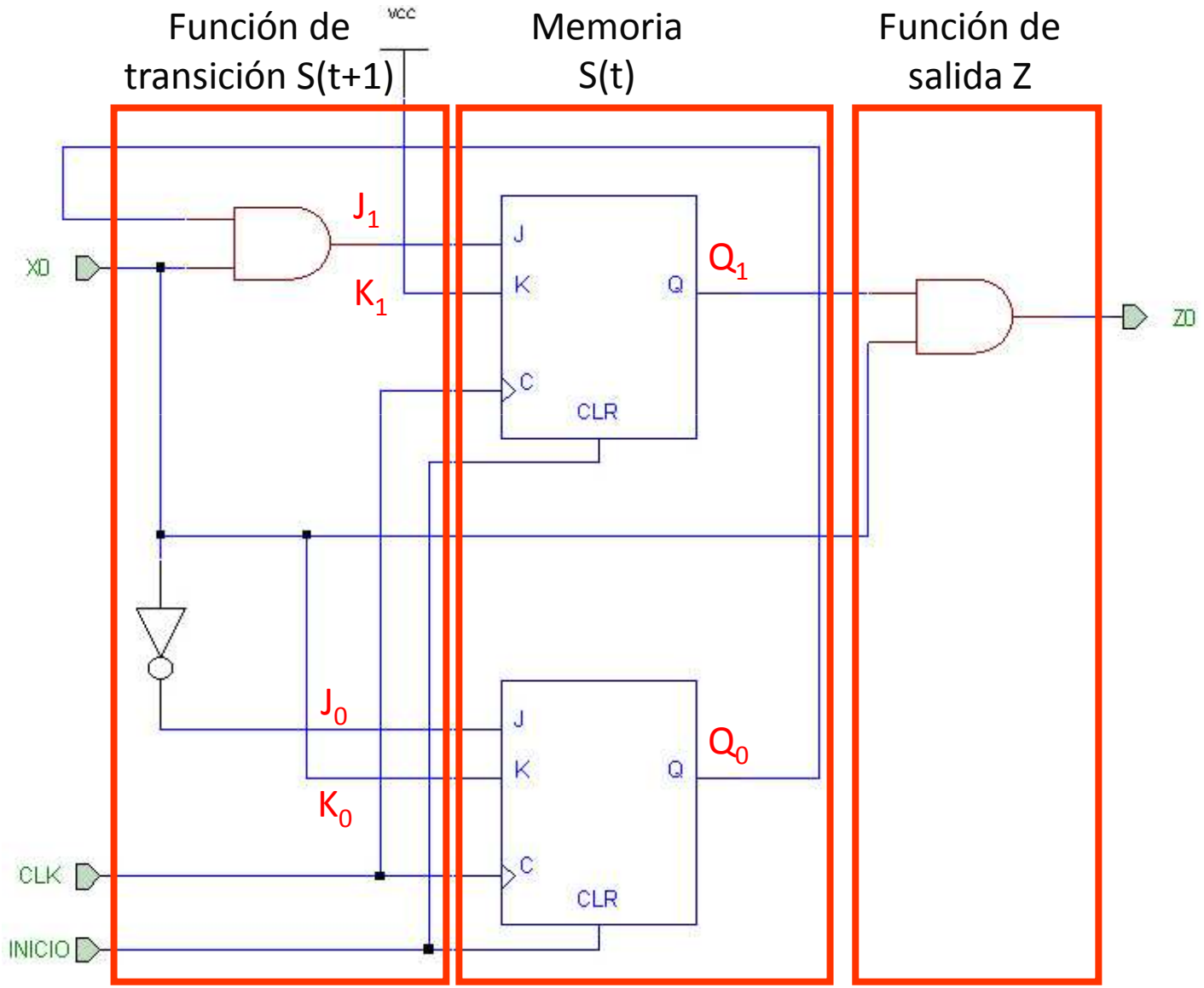
$$J_0(Q_1, Q_0, X_0) = \overline{X_0}$$

$$K_0(Q_1, Q_0, X_0) = X_0$$

Función de salida

$$Z_0(Q_1, Q_0, X_0) = X_0 \cdot Q_1$$

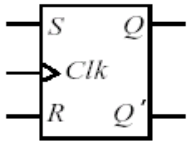
SÍNTESIS DE FSM CON *FLIP-FLOPS*: JK

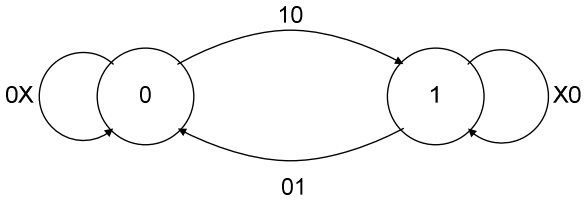


SÍNTESIS DE FSM CON *FLIP-FLOPS*: SR

- Si elegimos biestables SR, la tabla de excitación y salida cambia.

S(t)		X_0	
Q_1	Q_0	0	1
0	0	01/0	00/0
0	1	01/0	10/0
1	0	01/0	00/1

Flip–flop name	*Flip–flop symbol*	*Excitation table*			
		Q	*Q(next)*	*S*	*R*
SR		0	0	0	X
		0	1	1	0
		1	0	0	1
		1	1	X	0



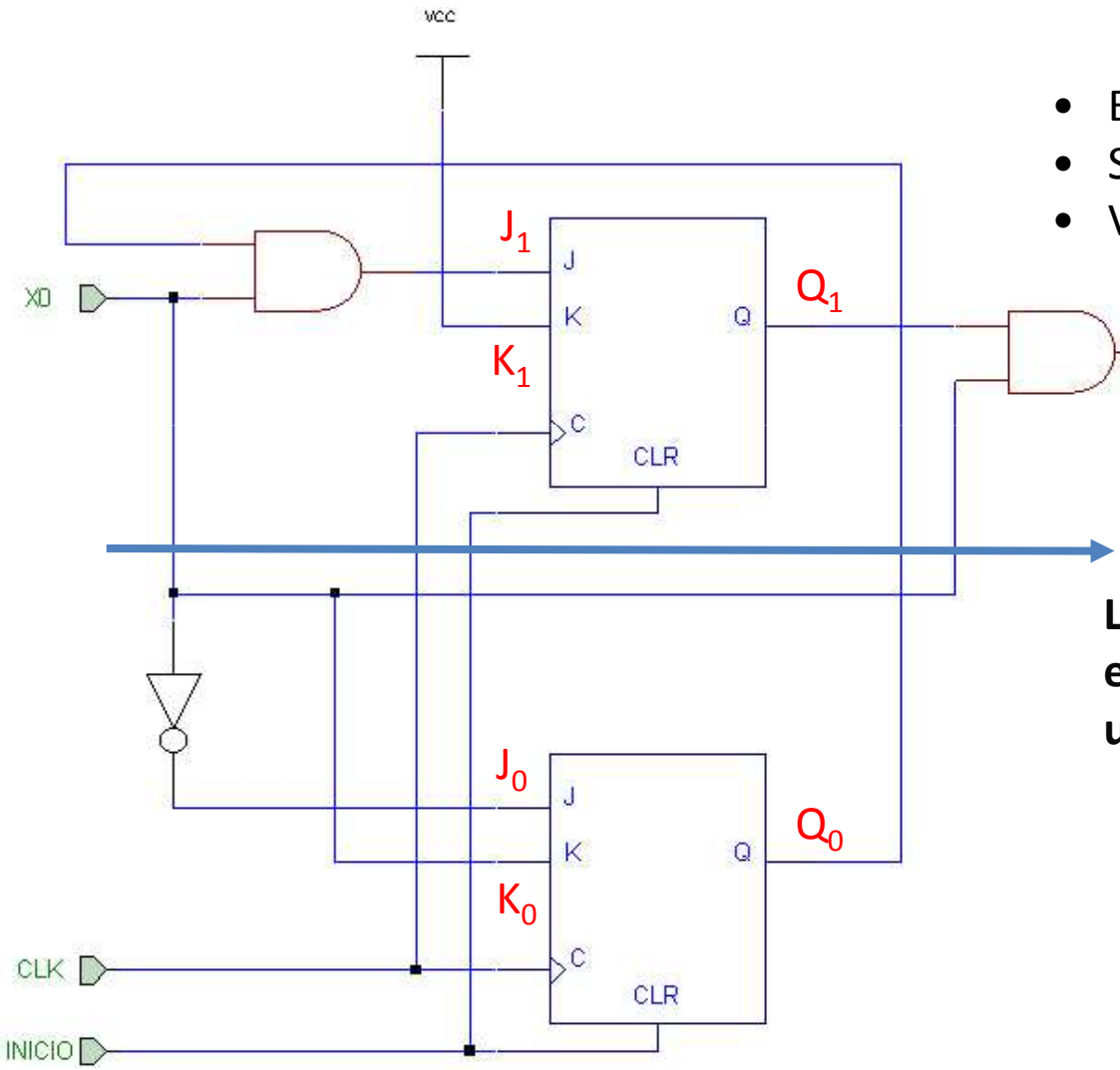
S(t), X			S(t+1), Z			Entradas de excitación			
Q_1	Q_0	X	Q'_1	Q'_0	Z	S_1	R_1	S_0	R_0
0	0	0	0	1	0	0	X	1	0
0	0	1	0	0	0	0	X	0	X
0	1	0	0	1	0	0	X	X	0
0	1	1	1	0	0	1	0	0	1
1	0	0	0	1	0	0	1	1	0
1	0	1	0	0	1	0	1	0	X
1	1	0	X	X	X	X	X	X	X
1	1	1	X	X	X	X	X	X	X

- Sería preciso materializar S_1, R_1, S_0, R_0 y Z.

ÍNDICE

ANÁLISIS DE FSM CON *FLIP-FLOPS*

- Lo que cambia es la generación de la tabla de excitación y salida.



- Entradas: $X_0 \in \{0,1\}$
- Salidas: $Z_0 \in \{0,1\}$
- Variables de estado: 2 $\{Q_1,Q_0\}$

La salida depende del estado y de la entrada: es una máquina de Mealy.

ANÁLISIS DE FSM: ECUACIONES Y TABLAS - MEALY

Función de transición

$$J_1(Q_1, Q_0, X_0) = X_0 \cdot Q_0$$
$$K_1(Q_1, Q_0, X_0) = 1$$
$$J_0(Q_1, Q_0, X_0) = \overline{X_0}$$
$$K_0(Q_1, Q_0, X_0) = X_0$$

Función de salida

$$Z_0(Q_1, Q_0, X_0) = X_0 \cdot Q_1$$

S(t), X			Entradas de excitación				S(t+1), Z		
Q_1	Q_0	X	J_1	K_1	J_0	K_0	Q'_1	Q'_0	Z
0	0	0	0	X	1	X	0	1	0
0	0	1	0	X	0	X	0	0	0
0	1	0	0	X	X	0	0	1	0
0	1	1	1	X	X	1	1	0	0
1	0	0	X	1	1	X	0	1	0
1	0	1	X	1	0	X	0	0	1
1	1	0	X	X	X	X	X	X	X
1	1	1	X	X	X	X	X	X	X

La salida depende del estado y de la entrada: es una máquina de Mealy.

S(t)		X_0	
Q_1	Q_0	0	1
0	0	01/0	00/0
0	1	01/0	10/0
1	0	01/0	00/1
1	1	01/0	00/1

ANÁLISIS DE FSM: DECODIFICACIÓN Y REPRESENTACIÓN FORMAL - MEALY

- Asignamos nombres y valores a las entradas, los estados y la salida.

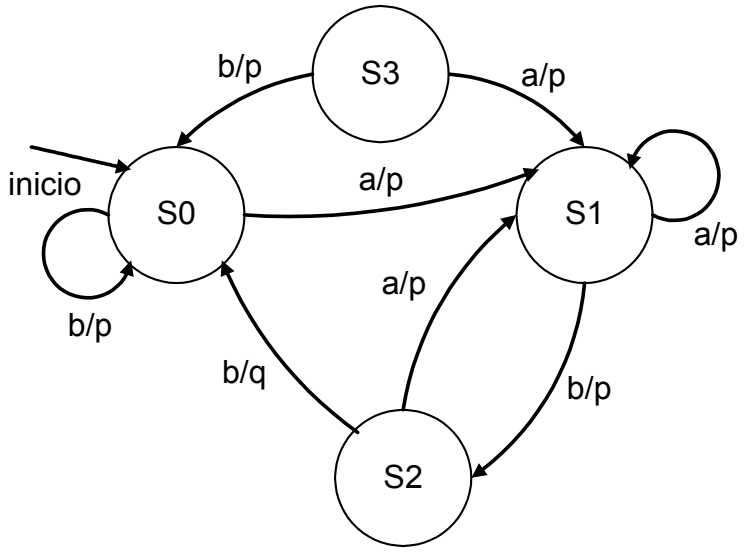
S(t)		X_0	
Q_1	Q_0	0	1
0	0	01/0	00/0
0	1	01/0	10/0
1	0	01/0	00/1
1	1	01/0	00/1

Entrada	
X_0	X(t)
0	a
1	b

Salida	
Z_0	Z(t)
0	p
1	q

Estado		
Q_1	Q_0	S(t)
0	0	S_0
0	1	S_1
1	0	S_2
1	1	S_3

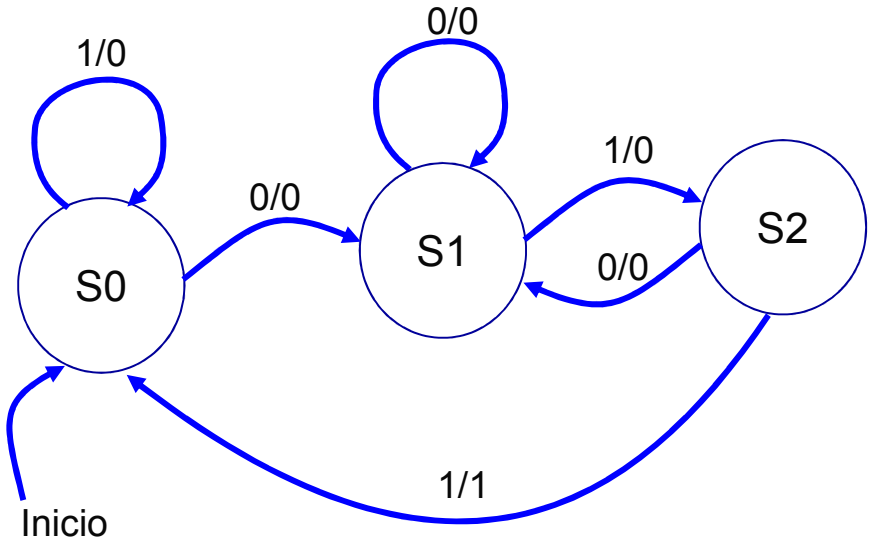
Estado actual	Entrada actual	
	a	b
S_0	S_1/p	S_0/p
S_1	S_1/p	S_2/p
S_2	S_1/p	S_0/q
S_3	S_1/p	S_0/q



ÍNDICE

DESCRIPCIÓN DE MÁQUINAS DE ESTADOS EN VHDL

- Las máquinas de estados se pueden describir fácilmente en VHDL de forma funcional.
- Se describen utilizando 2 procesos, uno que modela el cálculo del estado siguiente y de la salida, y otro proceso que modela la actualización del estado.
- Se puede crear un tipo de datos enumerado para los estados, o se puede utilizar un vector de bits.
- Vamos a ver un ejemplo con la siguiente máquina de estados:



DESCRIPCIÓN DE MÁQUINAS DE ESTADOS EN VHDL

- Ejemplo: entidad y proceso que modela el estado futuro y la salida.

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

entity fsm_funcional is
  port(clk,rst,x: in std_logic;
       z: out std_logic);
end fsm_funcional;
```

```
architecture funcional of fsm_funcional is
type estados is (S0, S1, S2);
signal ns, ps: estados; --next state y present state
begin
  --proximo estado y salida
  process(x,ps)
    begin
      case ps is
        when S0 => if x = '0' then ns <= S1; z <= '0';
                   else ns <= S0; z <= '0';
                   end if;
        when S1 => if x = '0' then ns <= S1; z <= '0';
                   else ns <= S2; z <= '0';
                   end if;
        when S2 => if x = '0' then ns <= S1; z <= '0';
                   else ns <= S0; z <= '1';
                   end if;
      end case;
    end process;
```

DESCRIPCIÓN DE MÁQUINAS DE ESTADOS EN VHDL

- Ejemplo (continuación): proceso que modela la actualización del estado.

```
--actualización del estado
    process(clk,rst)
      begin
        if rst = '1' then
          ps <= S0;
        elsif rising_edge(clk) then
          ps <= ns;
        end if;
      end process;

end funcional;
```

DESCRIPCIÓN DE MÁQUINAS DE ESTADOS EN VHDL

- Ejemplo (continuación): test-bench.

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

entity test_fsm_funcional is
end test_fsm_funcional;

architecture test of test_fsm_funcional is
signal clk: std_logic := '0';
signal rst,x,z: std_logic;
begin
  clk <= not clk after 5 ns;
  rst <= '0', '1' after 10 ns, '0' after 20 ns;
  x <= '1', '0' after 30 ns,'1' after 60 ns, '0' after 70 ns, '1' after 80 ns;
  inst: entity work.fsm_funcional port map(clk,rst,x,z);
end test;
```

DESCRIPCIÓN DE MÁQUINAS DE ESTADOS EN VHDL

- Ejemplo (continuación): resultado de la simulación.