

# Arquitectura de Computadoras

## Unidad 0 Lógica Secuencial

Edgardo Gho  
Carlos Rodríguez



UM 1082 LA2/3

8225

FERRANTI  
ULA 2C184E  
8214

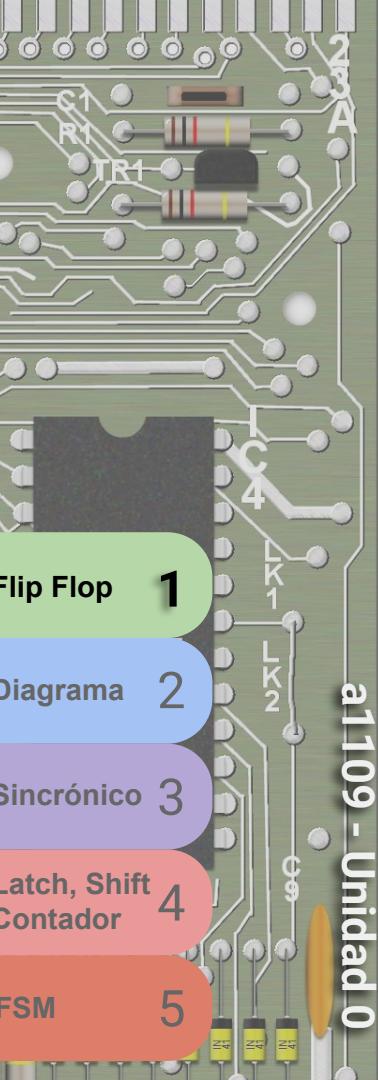
ZILLOG  
Z8400A PS  
Z80A CPU  
8220

SINCLAIR  
RESEARCH 8223Pg  
D2364C 649 © 1981

Toshiba  
TMM2016P  
2-EE4

<http://www.mainbyte.com/ls1000>

# Flip Flop RS



Flip Flop 1

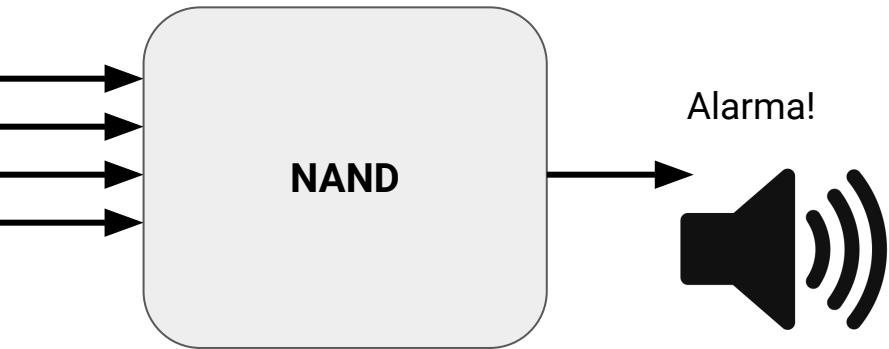
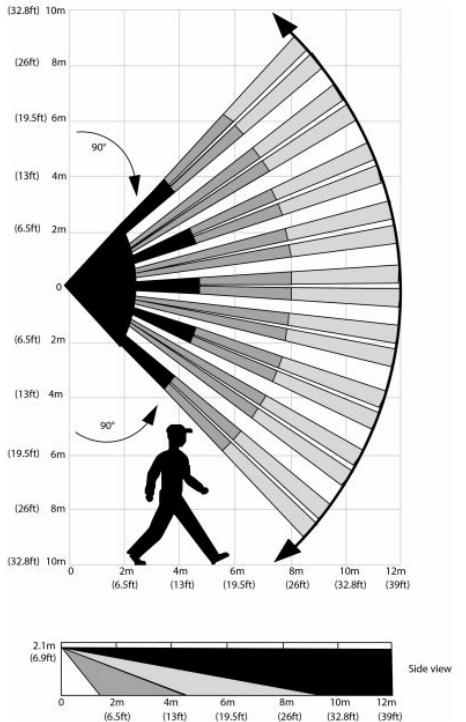
Diagrama 2

Sincrónico 3

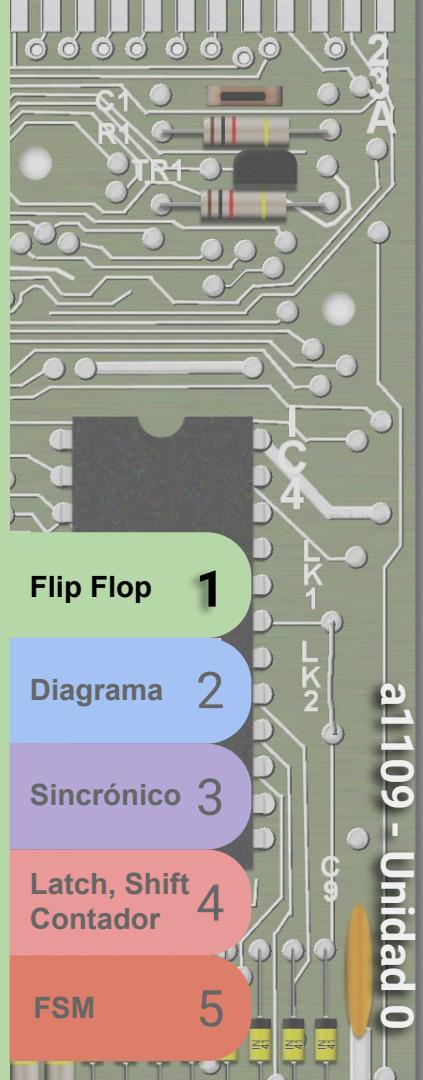
Latch, Shift  
Contador 4

FSM 5

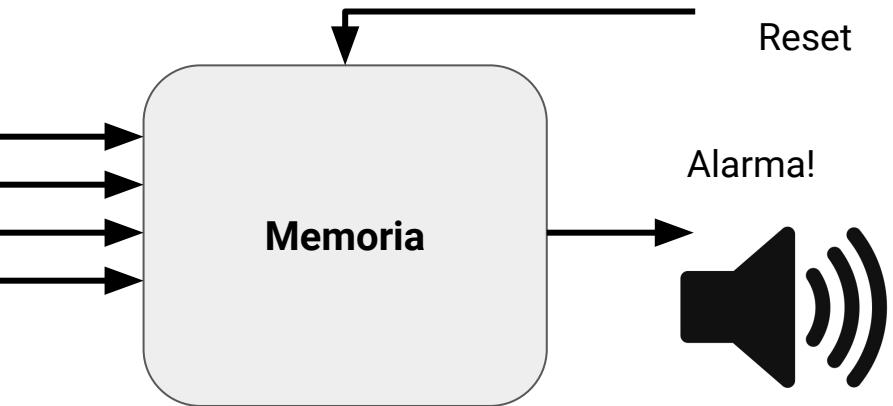
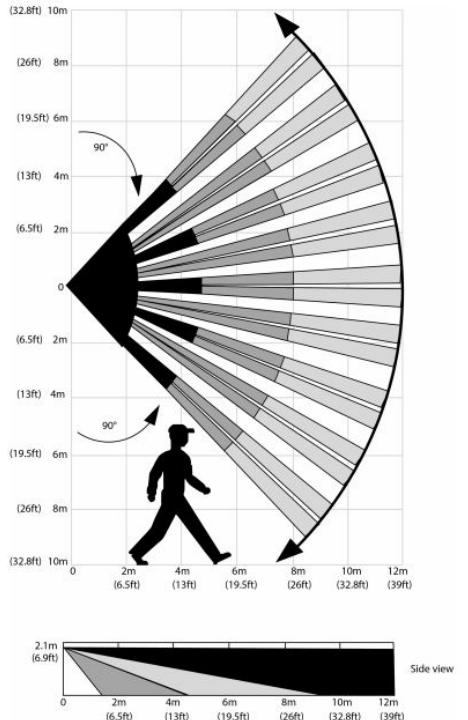
# Necesitamos memorizar una entrada



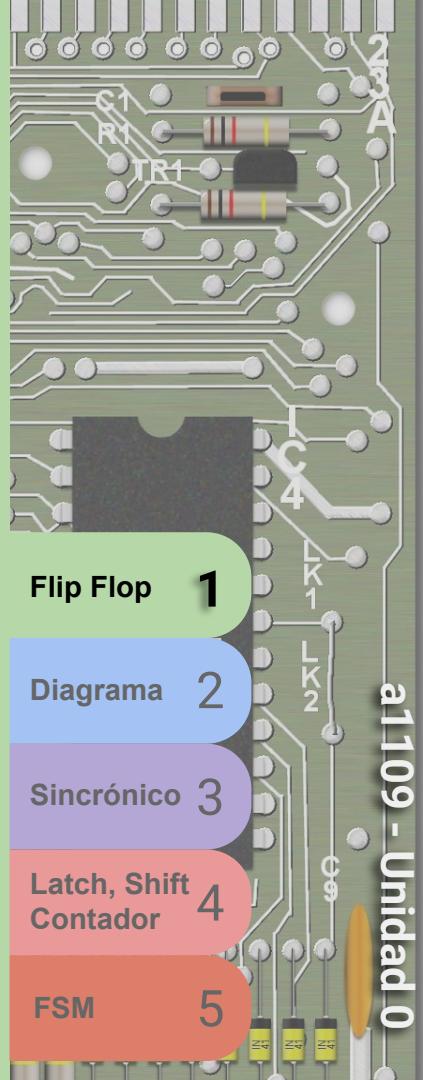
Un sensor de movimiento genera un uno cuando no detecta una persona, y un cero cuando detecta movimiento. Esto se debe a que si se corta el cable del sensor, nos interesa detectar esto como un movimiento. Por ende con una compuerta NAND, todos los sensores deben estar en uno para que la alarma esté apagada. En el momento que un sensor detecta una persona o se corta un cable se dispara la alarma. El problema es que si la persona se deja de mover, la NAND vuelve a generar un cero. Necesitamos poder memorizar el estado de alarma.



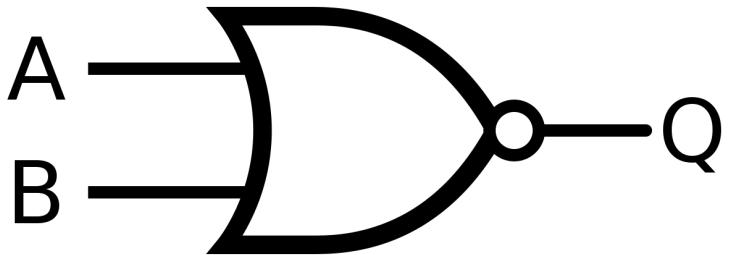
# Necesitamos memorizar una entrada



Pero también necesitamos una forma de cambiar la salida Alarma en el caso que necesitemos apagarla. Para eso usamos una entrada extra de reset que vuelve la salida alarma a cero. Esta entrada de Reset debe estar “protegida” ya que si el intruso la accede el sistema de alarma se vuelve inservible. Veamos un circuito que memorice una única entrada y permita restablecer (reset) la salida.

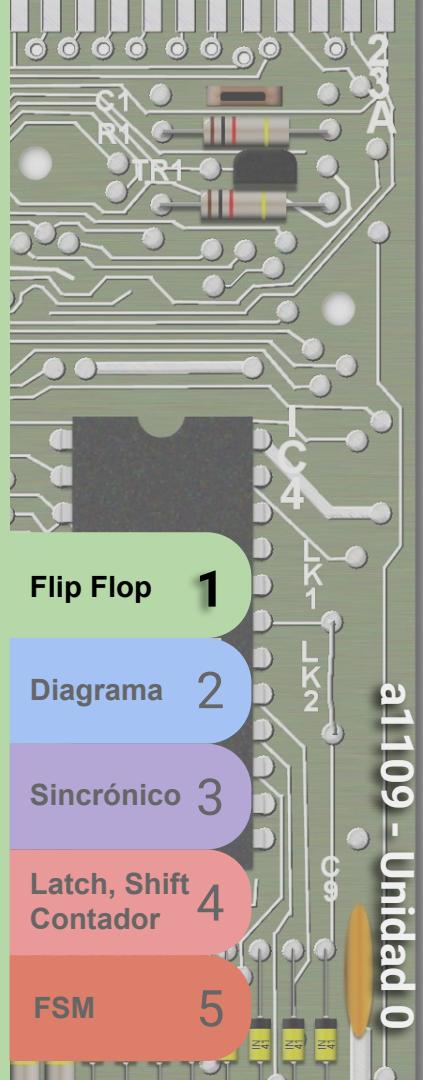


Repasemos la compuerta NOR. Esta es una OR negada, por ende solo produce 1 cuando sus entradas son ambas 0.

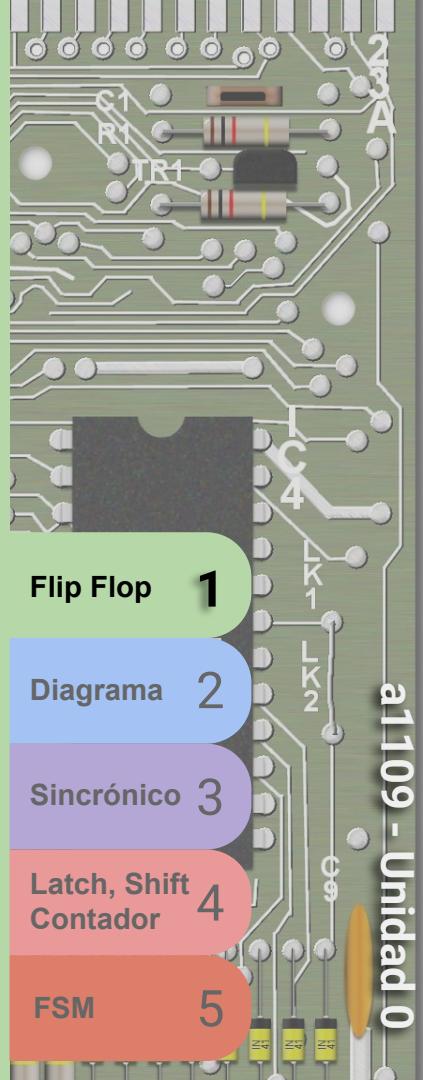
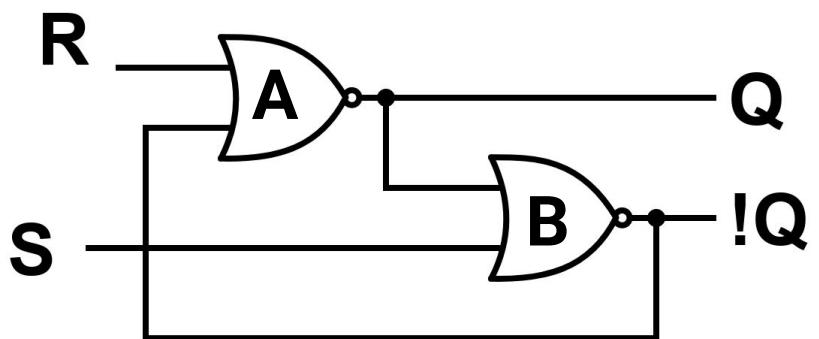
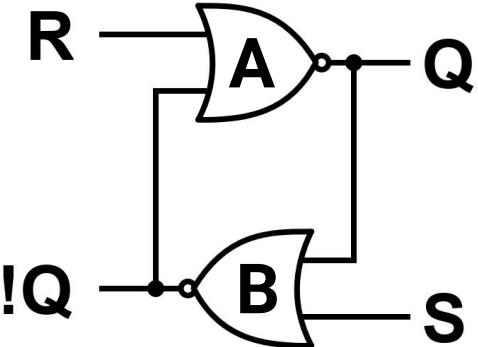
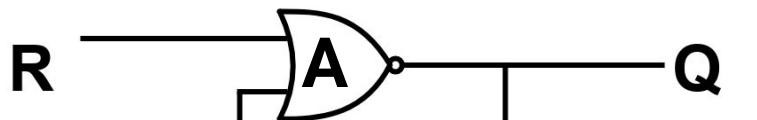


## Compuerta NOR

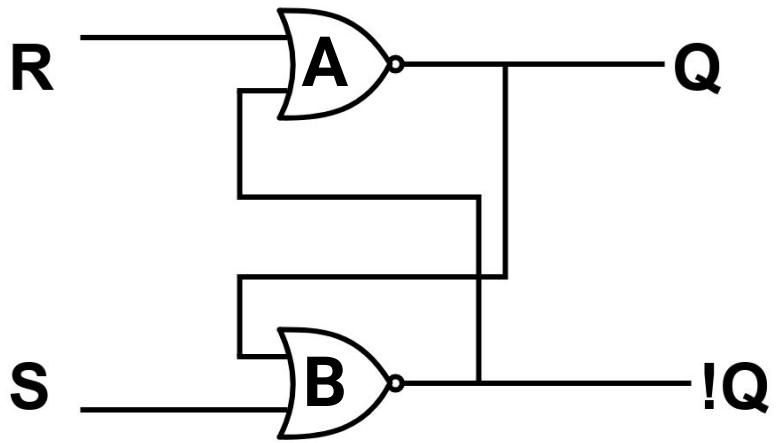
A	B	Q
0	0	1
0	1	0
1	0	0
1	1	0



# Flip Flop RS asincrónico



# Flip Flop RS asincrónico

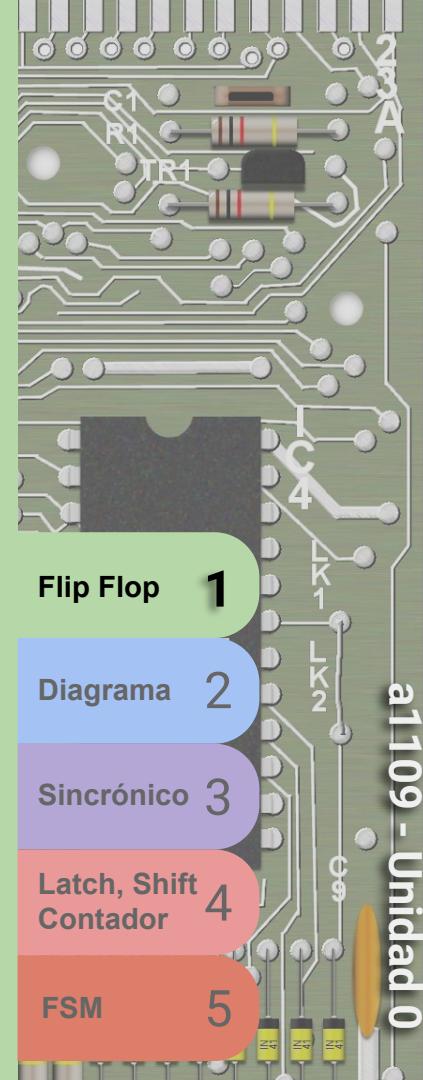


Este circuito posee 2 entradas (R y S) y 2 salidas (Q y !Q). La entrada S la llamamos entrada de Set (establecer) y la entrada R la llamamos de Reset (restablecer). La salida Q y !Q debieran ser complementarias.

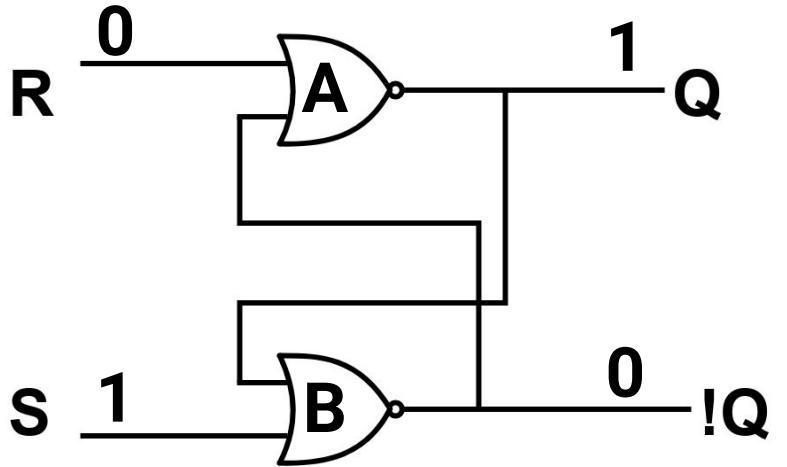
S	R	Q <sub>N</sub>
0	0	Q <sub>N-1</sub>
0	1	0
1	0	1
1	1	?

La tabla de verdad de este circuito posee valores bien definidos para los casos S=0,R=1 (Q=0) y S=1,R=0 (Q=1). Sin embargo en la combinación S=0,R=0 tenemos como valor Q<sub>N-1</sub>, o sea, el valor depende del valor que tenía Q un "instante de tiempo" antes. Aquí empezamos a tener que tener en cuenta el "tiempo". Por último cuando S=1, R=1 tenemos un valor prohibido.

**Nota: NOR=1 si y sólo si ambas entradas son 0.**



# Flip Flop RS asincrónico

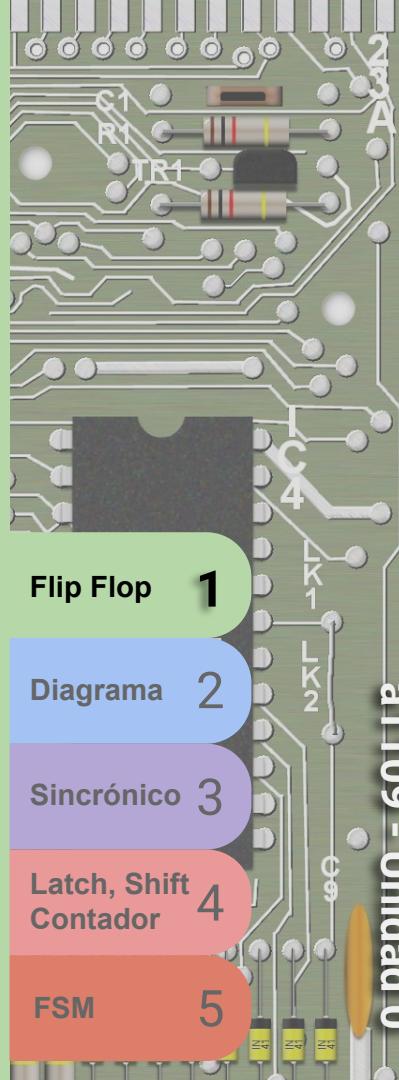


Para analizar el FF, damos valores a sus entradas. Vemos que cuando  $S=1, R=0$  la salida  $Q=1$ , por ende podemos empezar en este estado. Vemos que  $\bar{Q}=0$  ya que  $Q=1$ .

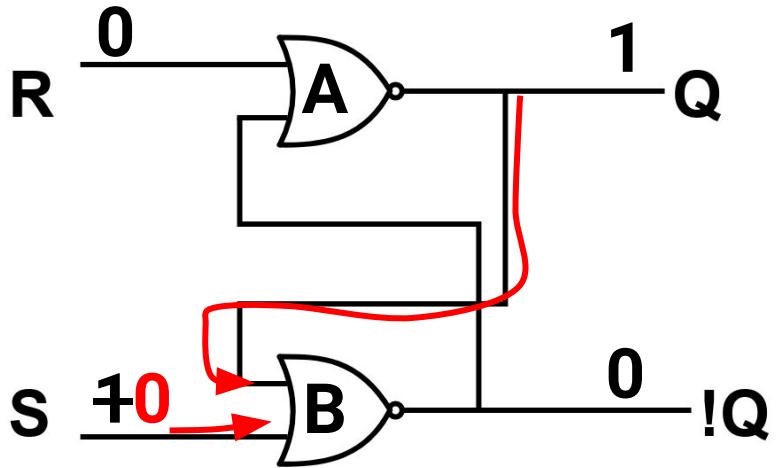
S	R	$Q_N$
0	0	$Q_{N-1}$
0	1	0
1	0	1
1	1	?

Ahora que estamos en un estado conocido podemos empezar a estudiar qué ocurre cuando se producen cambios en las entradas del FF.

Nota: NOR=1 si y sólo si ambas entradas son 0.



## Flip Flop RS asincrónico

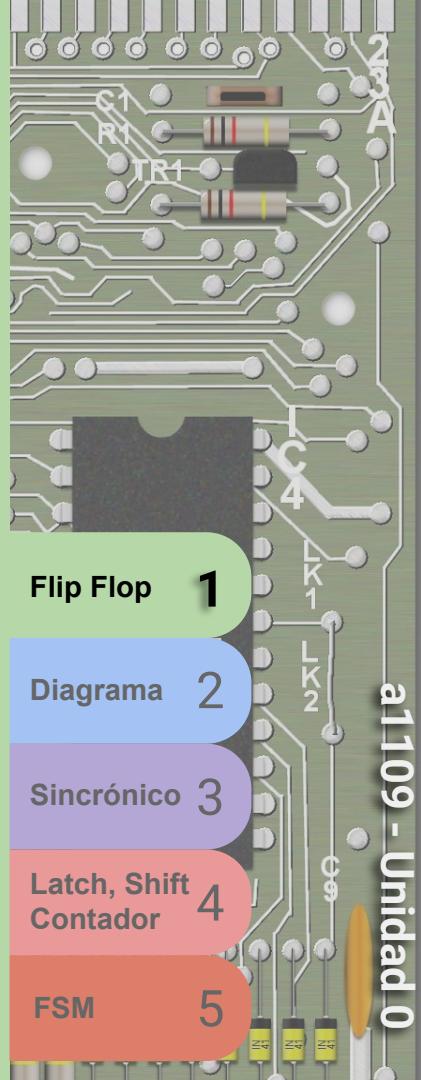


Cambiamos la entrada  $S=1$  a  $S=0$ . Vemos que previamente NOR B tenía como entrada 1 y 1 pero ahora tiene 1 y 0. La compuerta NOR solo vale 1 cuando ambas entradas son cero. Por ende no hay cambio en NOR B, se sigue manteniendo en cero.

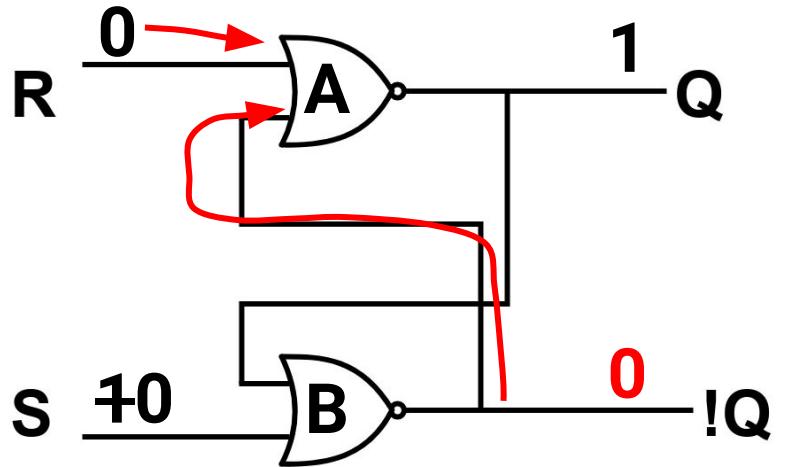
S	R	$Q_N$
0	0	$Q_{N-1}$
0	1	0
1	0	1
1	1	?

Si bien todavía no vimos las particulares de la tecnología de las compuertas, podemos asumir que pasa un pequeño tiempo desde que se produce un cambio en las entradas de una compuerta hasta que la misma produce la salida correspondiente.

Nota: NOR=1 si y sólo si ambas entradas son 0.



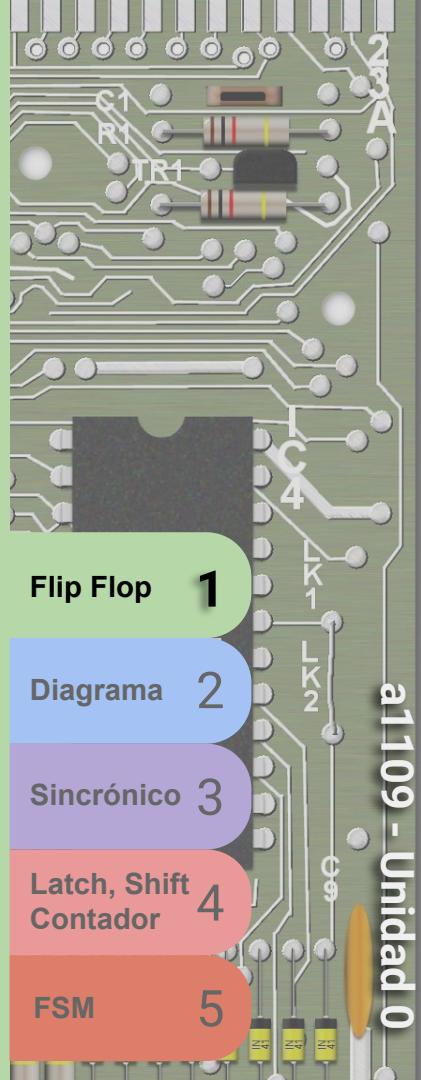
# Flip Flop RS asincrónico



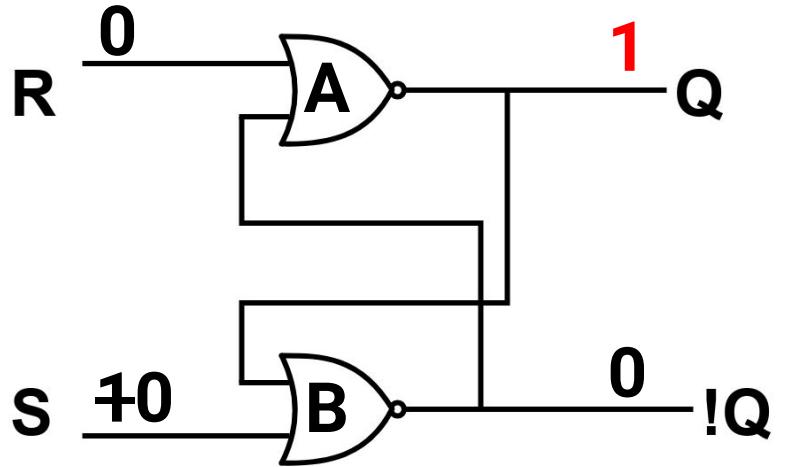
S	R	$Q_N$
0	0	$Q_{N-1}$
0	1	0
1	0	1
1	1	?

Nota: NOR=1 si y sólo si ambas entradas son 0.

Vemos que NOR B se mantiene en cero, y este valor es tomado como entrada en NOR A. Las entradas de NOR A son ambas cero, por ende la salida (después de un tiempo de propagación) se mantiene en 1.



# Flip Flop RS asincrónico

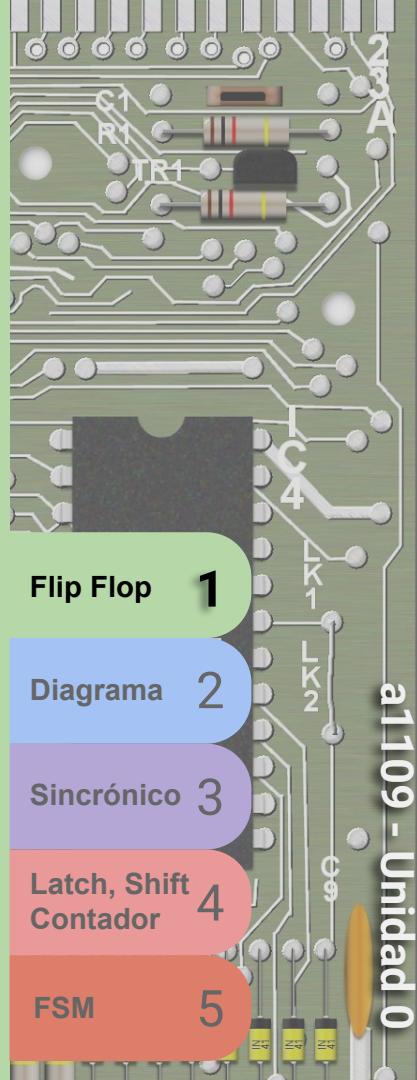


Luego la salida Q se mantiene en 1, que era el estado original, o sea el estado llamado  $Q_{N-1}$ .

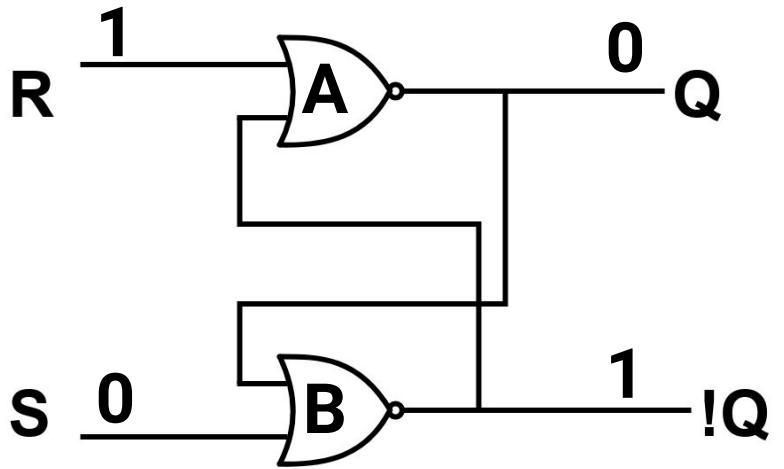
S	R	$Q_N$
0	0	$Q_{N-1}$
0	1	0
1	0	1
1	1	?

Vemos que cuando  $R=0, S=1$  y  $Q=1$  .. pasamos  $S=0$  y la salida Q se mantuvo en uno. Esta es la característica fundamental del flip flop... mantiene su estado cuando sus entradas son  $S=0, R=0$ . Veamos qué hubiese ocurrido si el FF comienza con  $R=1, S=0, Q=0$  y pasamos  $R=0$  a continuación...

Nota: NOR=1 si y sólo si ambas entradas son 0.



# Flip Flop RS asincrónico

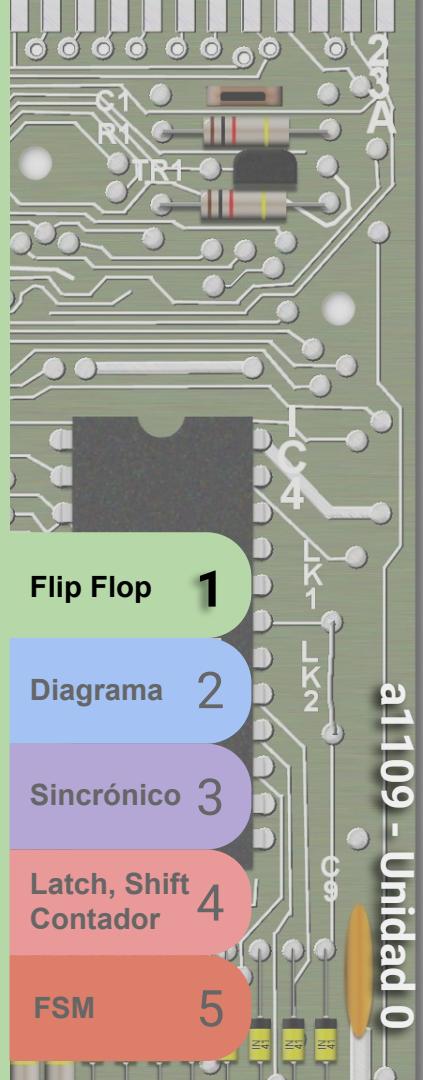


Damos ahora valores  $S=0$  y  $R=1$ , lo cual según la tabla de verdad define  $Q=0$  y  $\bar{Q}=1$ .

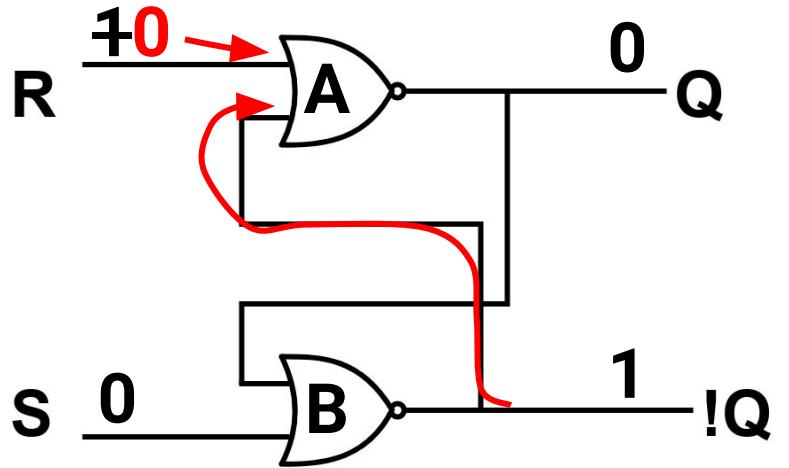
S	R	$Q_N$
0	0	$Q_{N-1}$
0	1	0
1	0	1
1	1	?

Notemos que en este estado tanto NOR A y NOR B mantienen un respectivas salidas. El FF se encuentran en lo que llamamos un estado estable (donde  $Q$  y  $\bar{Q}$  son complementarias) y el circuito mantiene sus salidas estables para un valor de entrada.

**Nota: NOR=1 si y sólo si ambas entradas son 0.**



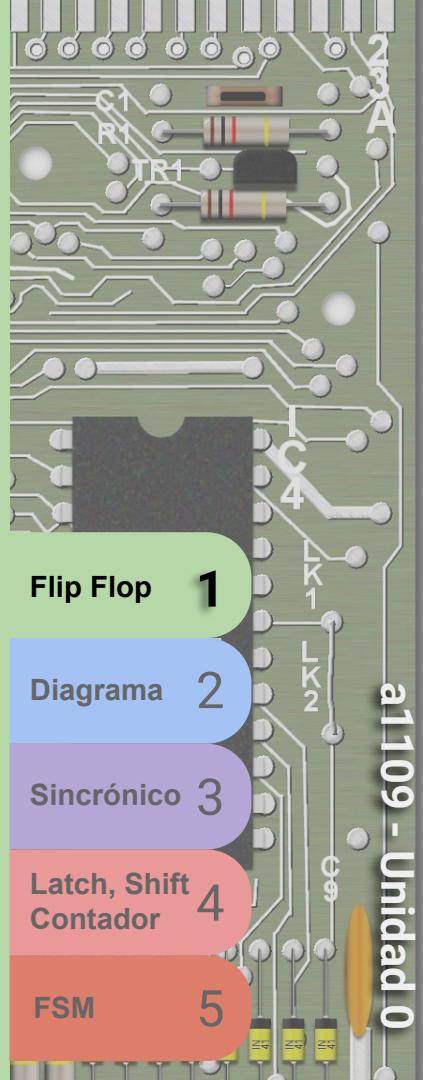
## Flip Flop RS asincrónico



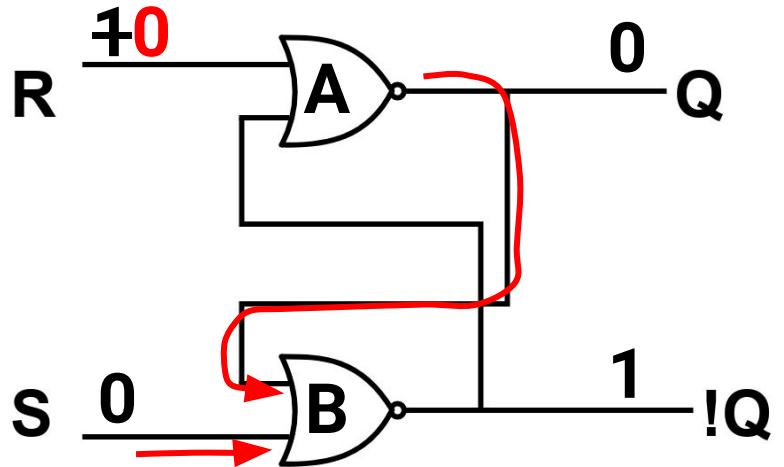
S	R	$Q_N$
0	0	$Q_{N-1}$
0	1	0
1	0	1
1	1	?

Nota: NOR=1 si y sólo si ambas entradas son 0.

Pasamos ahora  $R=0$ . Vemos que ahora NOR A tiene en sus entradas 0 y 1. El valor de salida de NOR A se mantiene en cero, pero esto ocurre luego de un tiempo de propagación.



## Flip Flop RS asincrónico

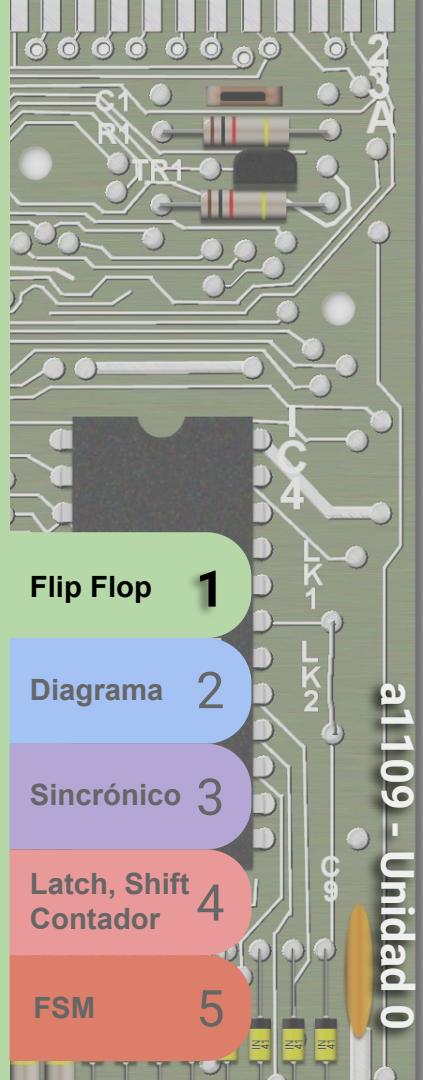


Vemos que las entradas de NOR B son 0 y 0, lo cual resulta en una salida 1 (luego de un tiempo de propagación).

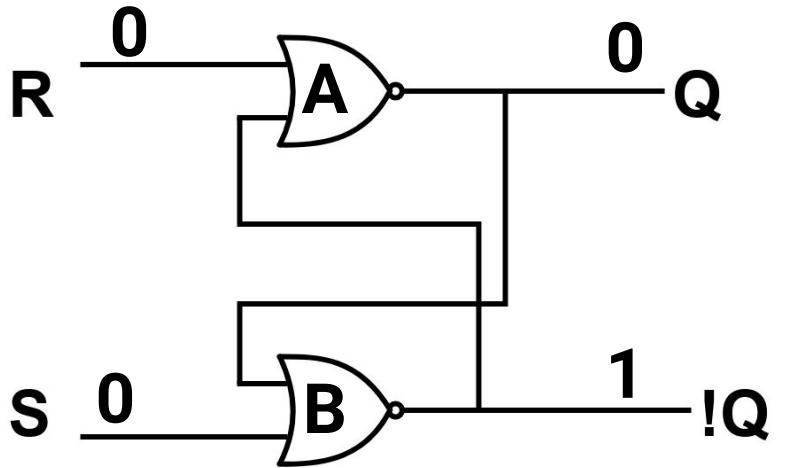
S	R	$Q_N$
0	0	$Q_{N-1}$
0	1	0
1	0	1
1	1	?

Nótese que tanto NOR A y NOR B mantuvieron sus respectivos valores luego del cambio de R=1 a R=0.

Nota: NOR=1 si y sólo si ambas entradas son 0.



## Flip Flop RS asincrónico

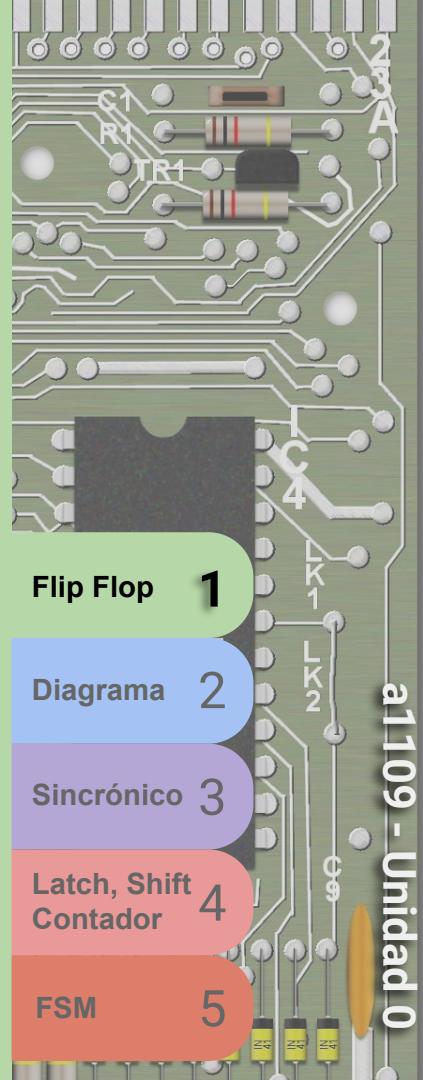


Hasta ahora hemos visto situaciones donde partimos de un estado conocido (el caso de SET o de RESET) y volvemos al estado en donde el FF memoriza. Sin embargo tenemos los casos donde el nuevo estado de la entrada cambia el valor memorizado.

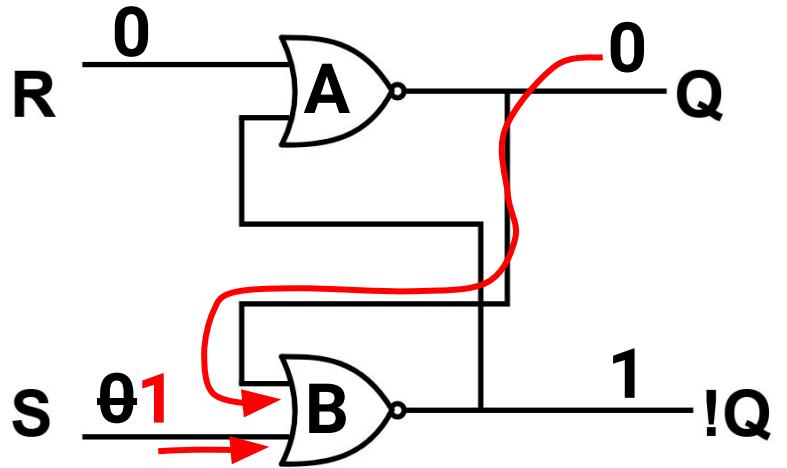
S	R	Q <sub>N</sub>
0	0	Q <sub>N-1</sub>
0	1	0
1	0	1
1	1	?

Partimos ahora en este estado donde R=0, S=0 y Q=0. Seguro llegamos a este estado porque en algún momento S=0 y R=1. Vemos que el valor memorizado es Q=0. Por ende para cambiarlo a 1 vamos a generar un 1 en la entrada S.

**Nota: NOR=1 si y sólo si ambas entradas son 0.**



## Flip Flop RS asincrónico

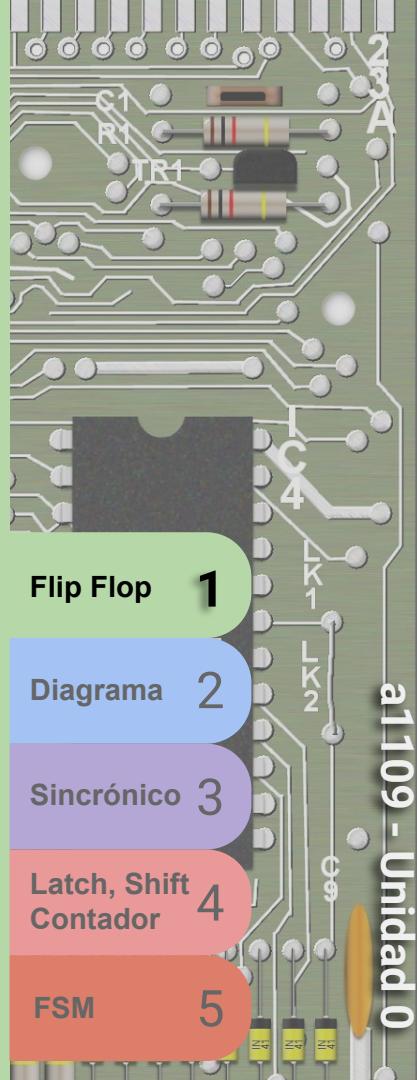


Cambiamos S=1. Vemos que ahora NOR B tiene en sus entradas 0 y 1. Esto quiere decir que 0 NOR 1 = 0.. Pero en este instante la salida de NOR B es 1.

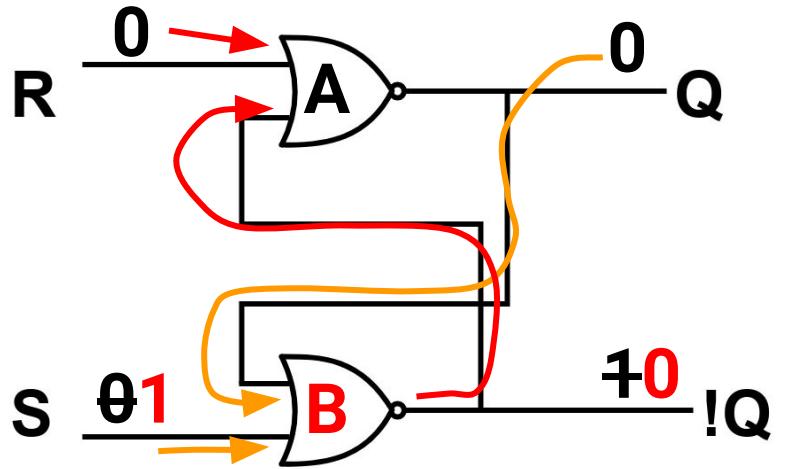
S	R	$Q_N$
0	0	$Q_{N-1}$
0	1	0
1	0	1
1	1	?

Aquí vemos la influencia del tiempo de propagación. La compuerta NOR B se encuentra produciendo un 1 correspondiente al momento cuando Q=0 y S=0... pero ahora S=1.. el valor de NOR B tiene que pasar de 1 a 0, pero eso no se produce instantáneamente.

Nota: NOR=1 si y sólo si ambas entradas son 0.



# Flip Flop RS asincrónico

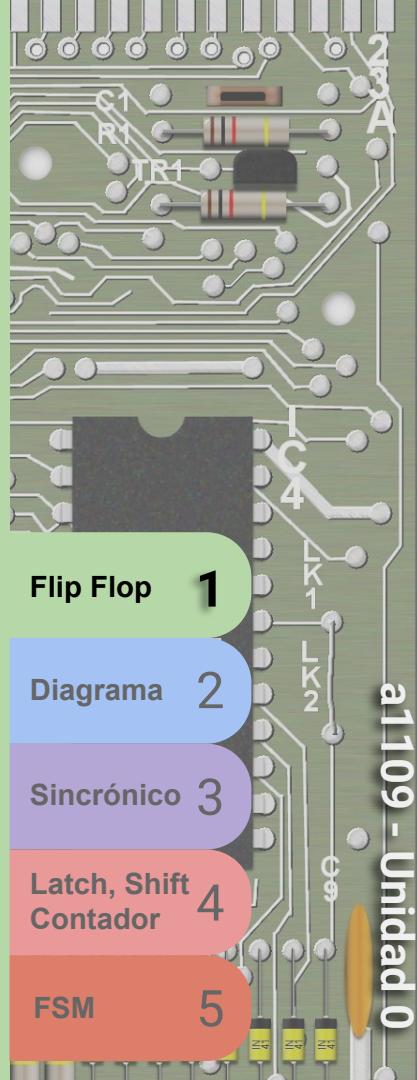


La compuerta NOR B cambia su salida a 0. Esto hace que  $\bar{Q}=0$ . Notemos que en este instante de tiempo  $Q=\bar{Q}$ ... lo cual no cumple con el requisito de que ambas salidas sean complementarias.

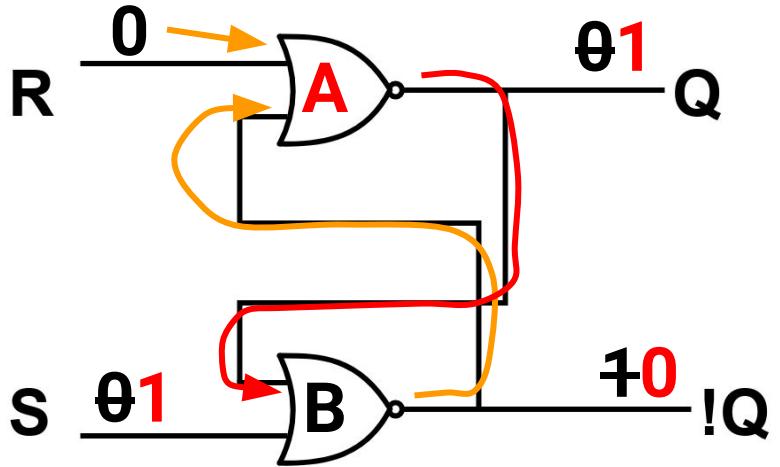
S	R	$Q_N$
0	0	$Q_{N-1}$
0	1	0
1	0	1
1	1	?

En este preciso instante, el FF se encuentra en lo que llamamos “Estado de transición” o inestable. Notemos que la salida de NOR B se encuentra en una de las entradas de NOR A. Esta compuerta ahora ve que sus entradas son 0 y 0 .. por ende luego de un tiempo de programación va a comutar a 1... pero en este instante de tiempo todavía mantiene  $Q=0$ .

**Nota: NOR=1 si y sólo si ambas entradas son 0.**



# Flip Flop RS asincrónico



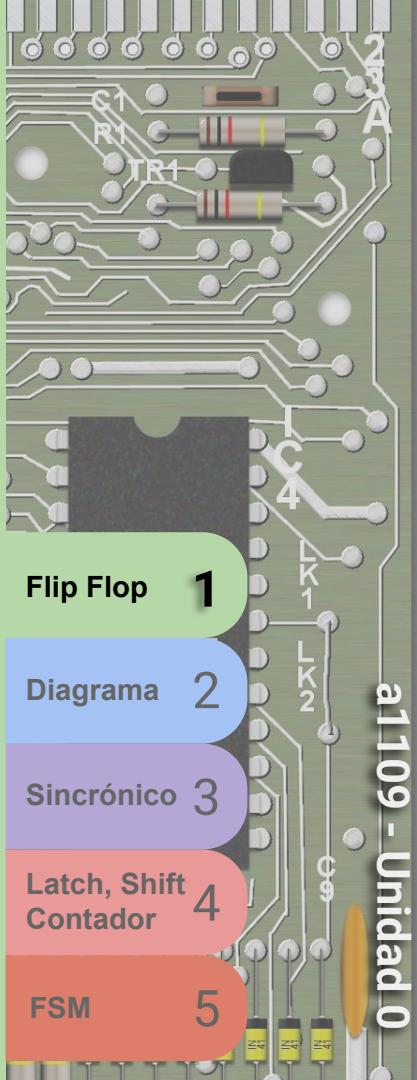
Ahora luego de un tiempo de propagación la compuerta NOR A convierte sus entradas 0 y 0 en un 1 de salida.

Vemos que ahora nuevamente el FF vuelve al estado "estable" ya que  $Q$  y  $\bar{Q}$  son complementarias.

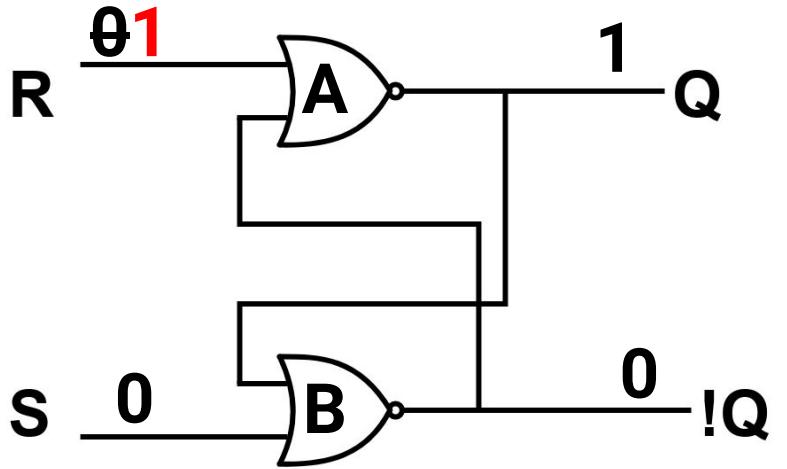
S	R	$Q_N$
0	0	$Q_{N-1}$
0	1	0
1	0	1
1	1	?

Vemos que ahora la salida de NOR A vuelve a estar conectada a la entrada de NOR B. Pero ahora NOR B tiene como entradas 1 y 1, lo que mantiene su salida en 0. Es aquí que termina de comutar y el cambio original de pasar  $S=0$  a  $S=1$  impacta finalmente sobre el FF. Previamente vimos el paso de  $S=0$  (cuando  $Q=1$ ) el cual se produce sin estados inestables de por medio.

**Nota: NOR=1 si y sólo si ambas entradas son 0.**



## Flip Flop RS asincrónico

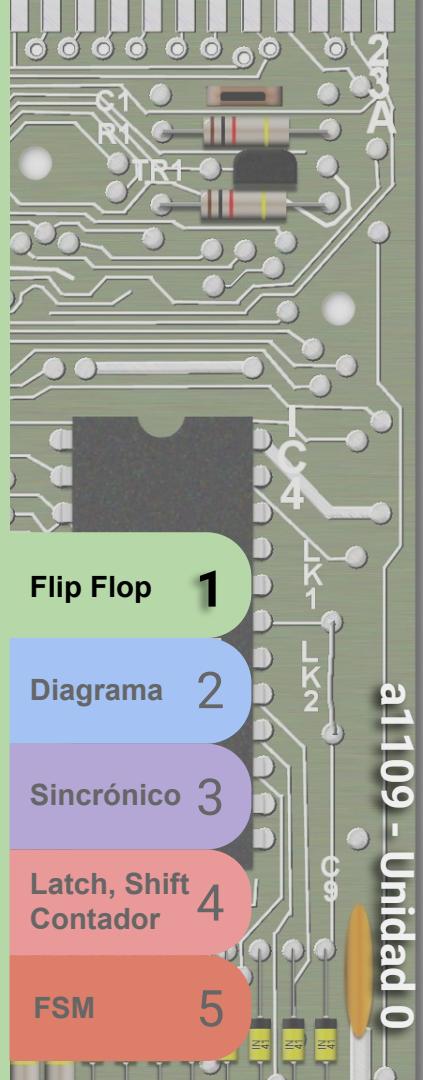


Por último veremos el caso pendiente donde la salida  $Q=1$  y cambiamos la entrada  $R$  para que restablezca el valor 0 a la salida.

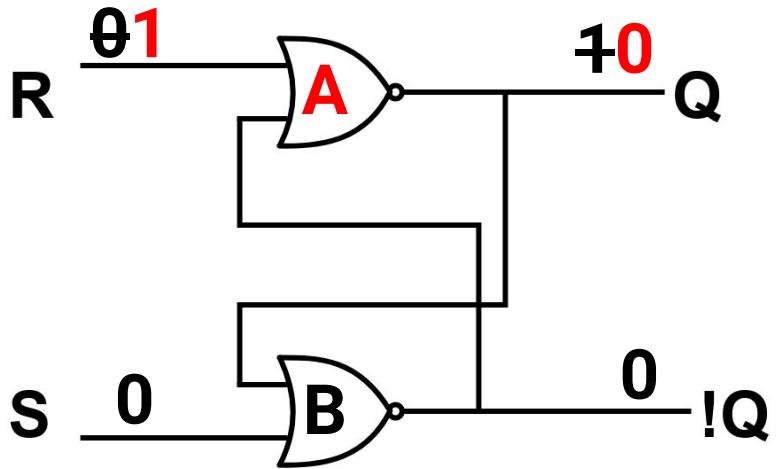
S	R	$Q_N$
0	0	$Q_{N-1}$
0	1	0
1	0	1
1	1	?

Ahora que  $R=1$ , la compuerta NOR A tiene en sus entradas un 0 y un 1, por ende su salida cambia a 0, pero esto requiere en tiempo, no es inmediato.

Nota: NOR=1 si y sólo si ambas entradas son 0.



# Flip Flop RS asincrónico

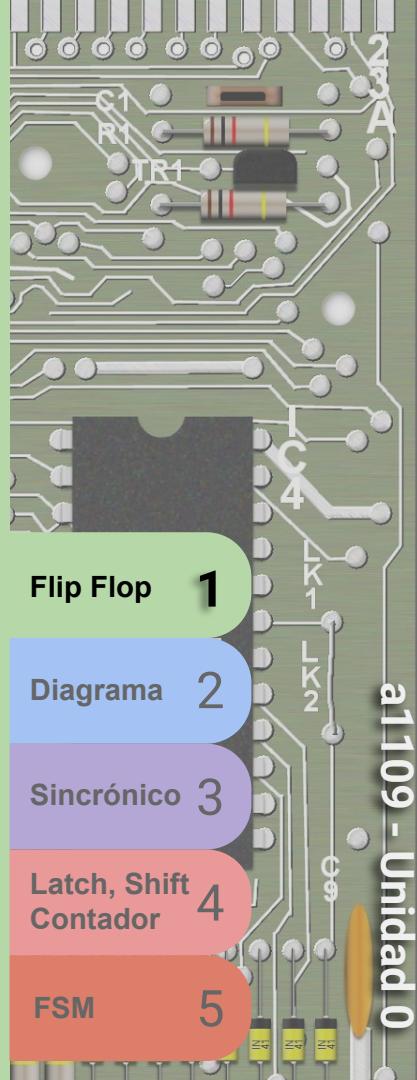


La salida Q ahora es 0. Vemos que  $Q = \neg Q$ , con lo cual el circuito pasa a estado inestable.

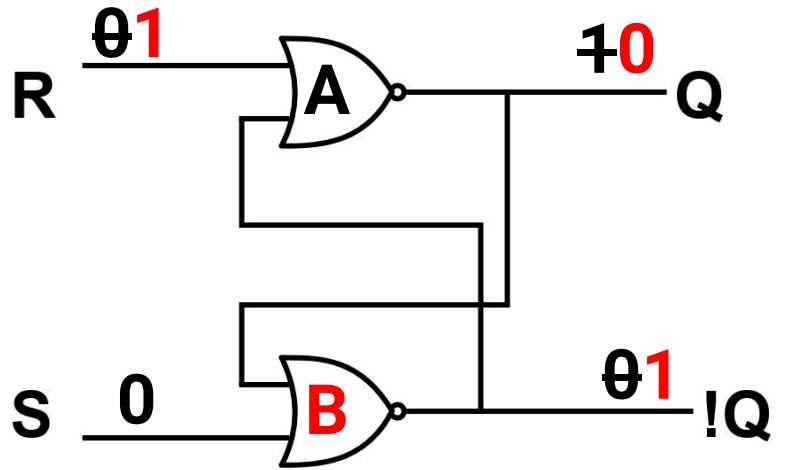
S	R	$Q_N$
0	0	$Q_{N-1}$
0	1	0
1	0	1
1	1	?

La salida de NOR A se conecta como entrada a NOR B. Vemos que ahora NOR B tiene como entradas 0 y 0, por ende luego de un tiempo de propagación va a comutar a 1. Sin embargo, si sacamos una foto en este preciso instante, el valor de NOR B sigue siendo 0 y por ende el estado del flip flop es de transición.

Nota: NOR=1 si y sólo si ambas entradas son 0.



# Flip Flop RS asincrónico

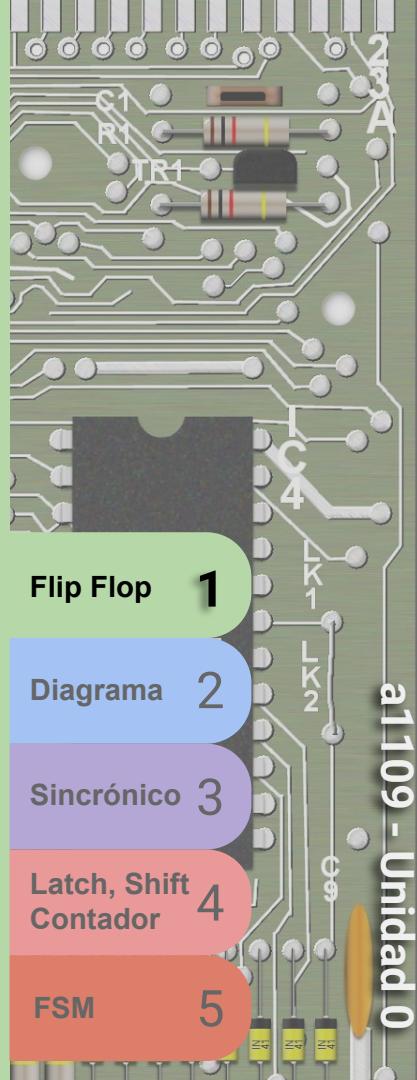


Luego de que NOR B conmuta el circuito ahora tiene  $!Q=1$ . Vemos que vuelve al estado estable donde Q y  $!Q$  son complementarias.

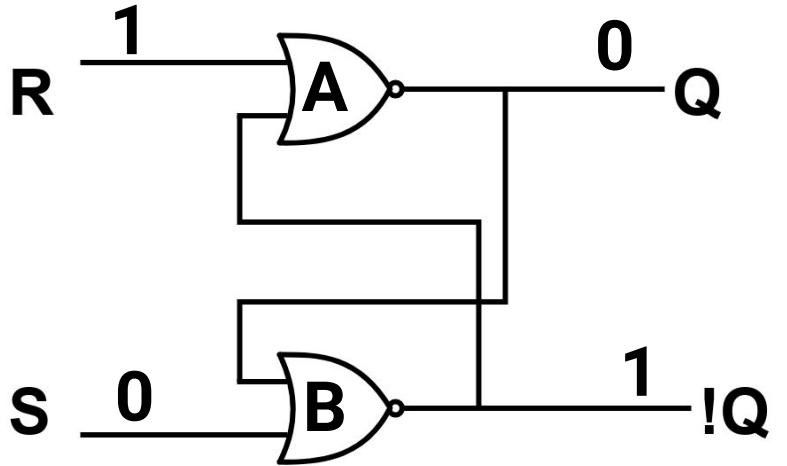
S	R	$Q_N$
0	0	$Q_{N-1}$
0	1	0
1	0	1
1	1	?

Si bien Q ya había cambiado un instante de tiempo anterior, el valor de  $!Q$  termina de conmutar en este instante. Es ahora que el circuito termina de conmutar. Pero vemos que la transición de  $Q=1$  a  $Q=0$  “se ve” más rápido que la transición de  $Q=0$  a  $Q=1$ . Veremos en la unidad siguiente que tomamos el peor de los casos para definir el tiempo que tarda en conmutar un circuito.

**Nota: NOR=1 si y sólo si ambas entradas son 0.**



# Flip Flop RS asincrónico



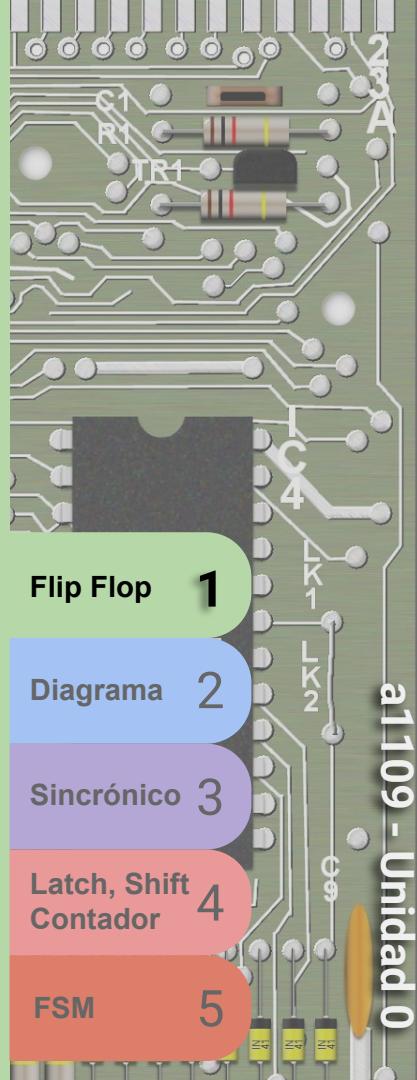
Nos queda ver qué ocurre cuando las entradas toman el valor  $R=1$  y  $S=1$  a la vez.

Tomamos como punto inicial  $R=1$ ,  $S=0$ , por ende  $Q=0$ . Ahora el siguiente estado será  $S=1$  de forma tal que  $R=1$  y  $S=1$ .

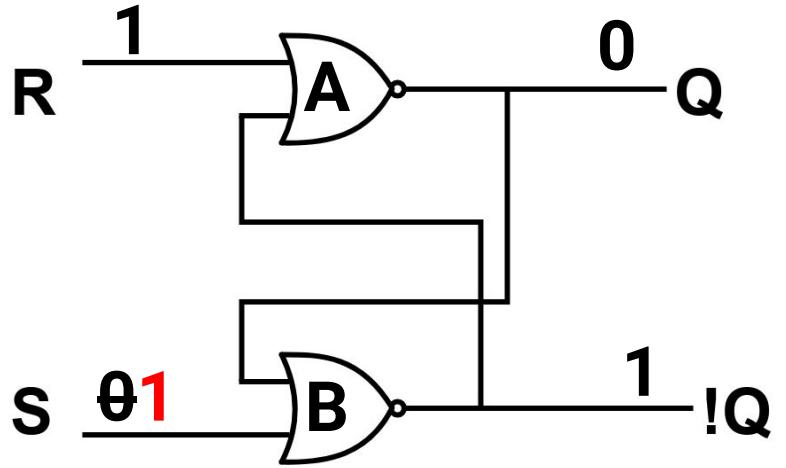
S	R	$Q_N$
0	0	$Q_{N-1}$
0	1	0
1	0	1
1	1	?

Vemos que en la tabla de verdad ese estado lo representamos como desconocido. En realidad se lo conoce como estado prohibido, ya que impide saber qué valor toma el flip flop. Tener un circuito que no podemos determinar cómo se va a comportar es indeseable, por eso se lo conoce como estado prohibido.

**Nota: NOR=1 si y sólo si ambas entradas son 0.**



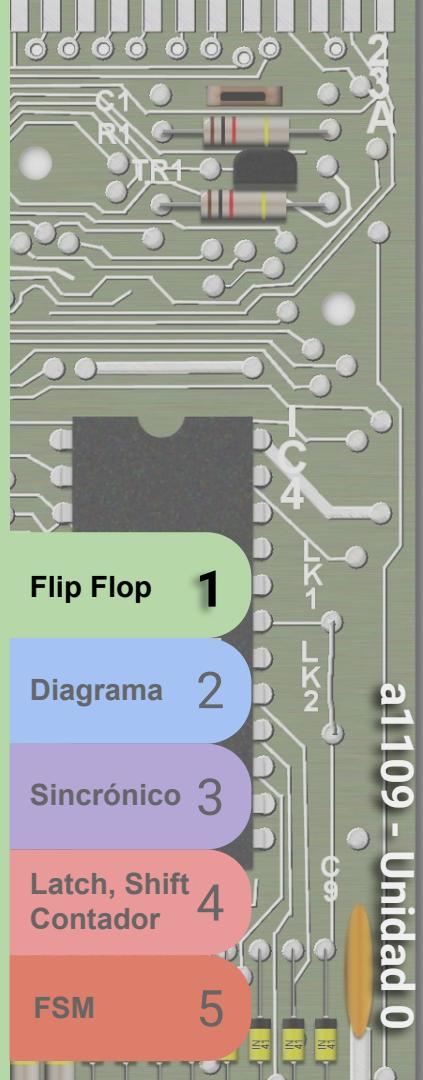
# Flip Flop RS asincrónico



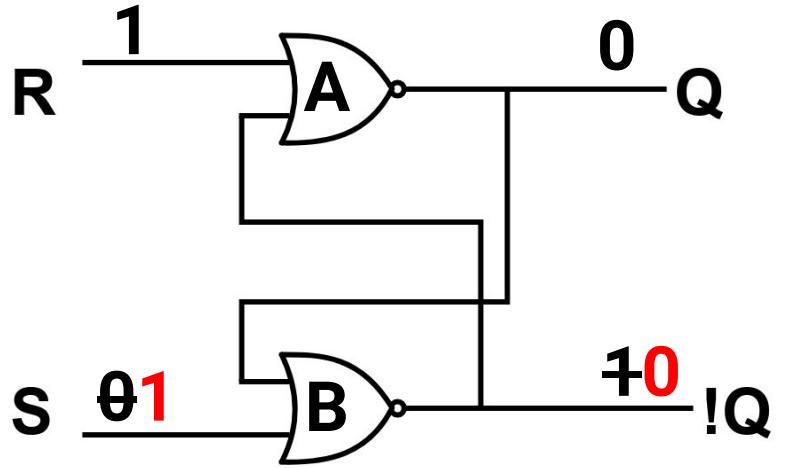
Ahora NOR B tiene como entradas 0 y 1, por ende comuta a 0 en su salida...

S	R	$Q_N$
0	0	$Q_{N-1}$
0	1	0
1	0	1
1	1	?

Nota: NOR=1 si y sólo si ambas entradas son 0.



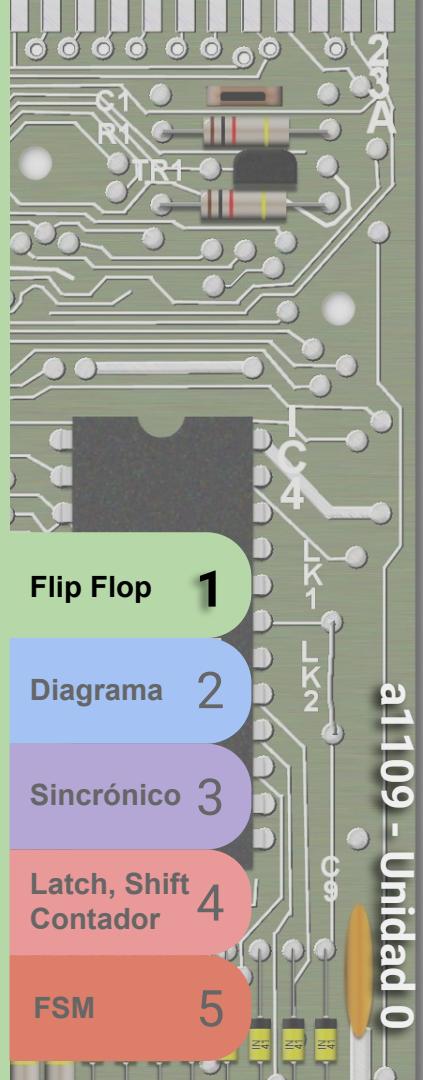
# Flip Flop RS asincrónico



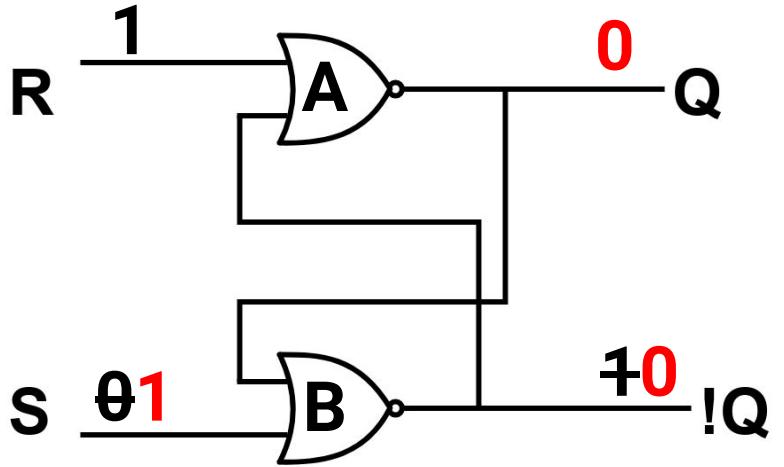
Vemos que  $\bar{Q}$  cambia a 0 y las entradas de NOR A ahora son 1 y 0.

S	R	$Q_N$
0	0	$Q_{N-1}$
0	1	0
1	0	1
1	1	?

Nota: NOR=1 si y sólo si ambas entradas son 0.



# Flip Flop RS asincrónico



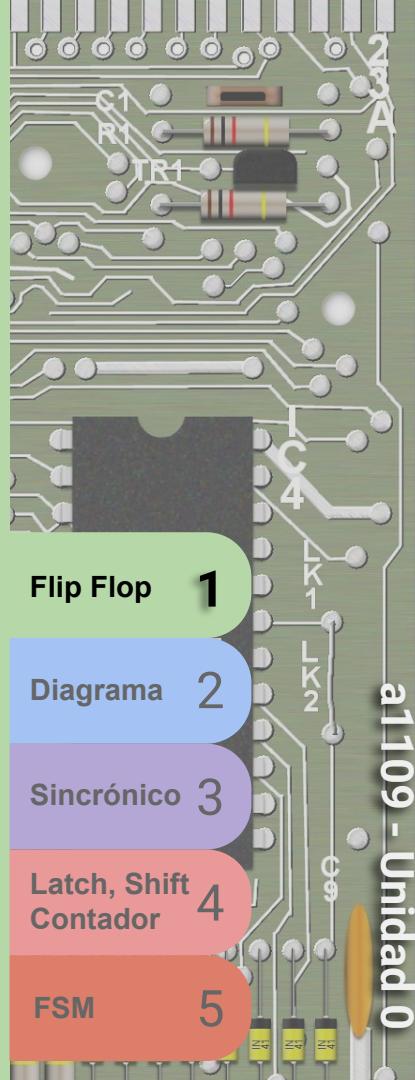
NOR A sigue manteniendo un 0 a su salida, y vemos que  $Q=\neq Q$  (estado inestable). En este punto el circuito queda en este estado inestable.

S	R	$Q_N$
0	0	$Q_{N-1}$
0	1	0
1	0	1
1	1	?

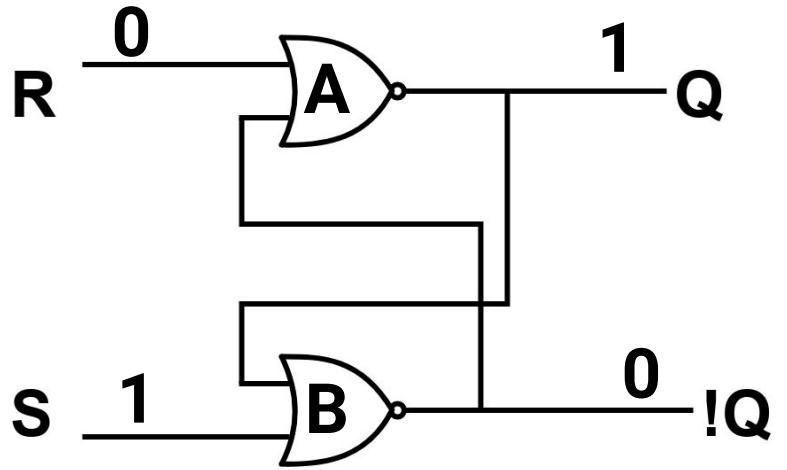
Este flip flop ahora perdió la capacidad de memorizar y mantiene sus salidas Q y  $\neq Q$  no complementarias.

Veamos ahora el otro caso donde  $S=1$ ,  $R=0$  y convertimos  $R=1$ ....

Nota: NOR=1 si y sólo si ambas entradas son 0.



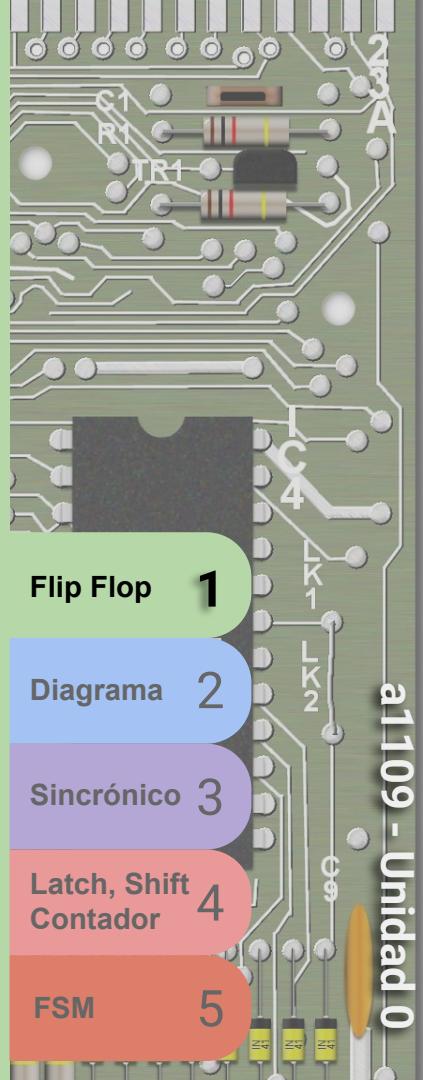
# Flip Flop RS asincrónico



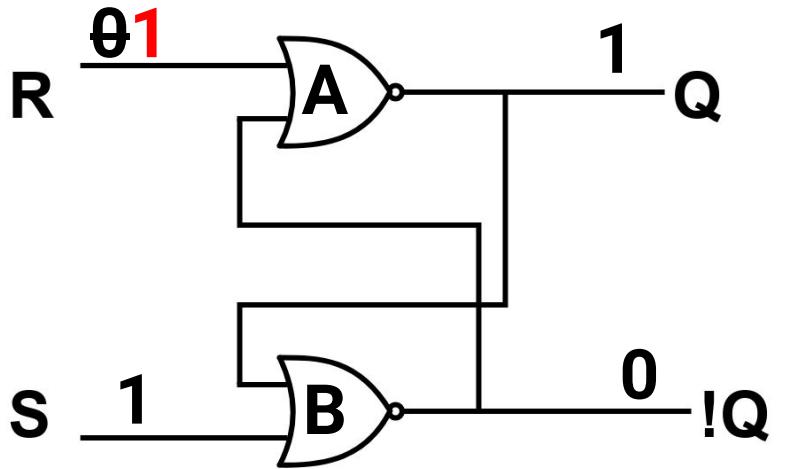
Ahora partimos del estado donde  $S=1$  y  $R=0$ . Vemos que  $Q=1$ .

S	R	$Q_N$
0	0	$Q_{N-1}$
0	1	0
1	0	1
1	1	?

Nota: NOR=1 si y sólo si ambas entradas son 0.



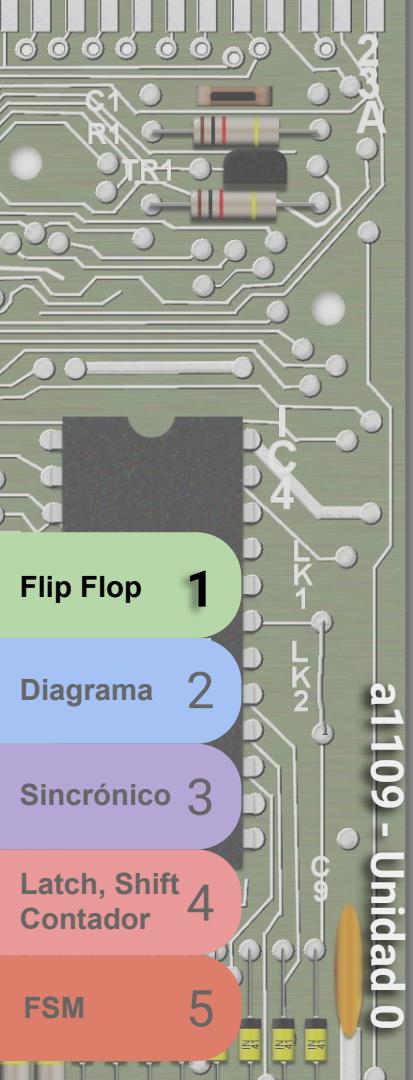
## Flip Flop RS asincrónico



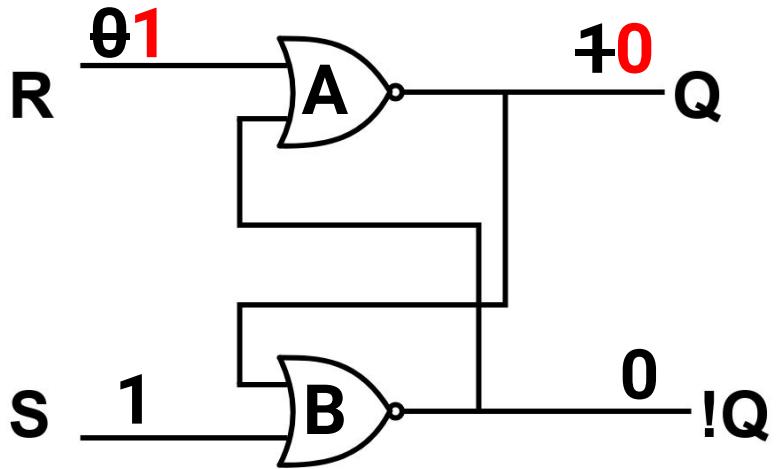
R pasa a ser 1.. Vemos que las entradas de NOR A son ahora 1 y 0, por ende la salida va a comutar a 0.

S	R	$Q_N$
0	0	$Q_{N-1}$
0	1	0
1	0	1
1	1	?

Nota: NOR=1 si y sólo si ambas entradas son 0.



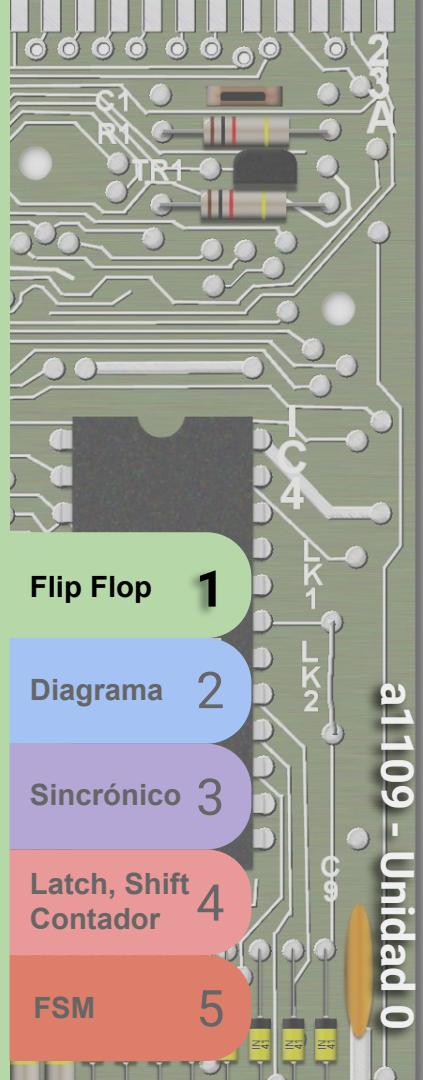
## Flip Flop RS asincrónico



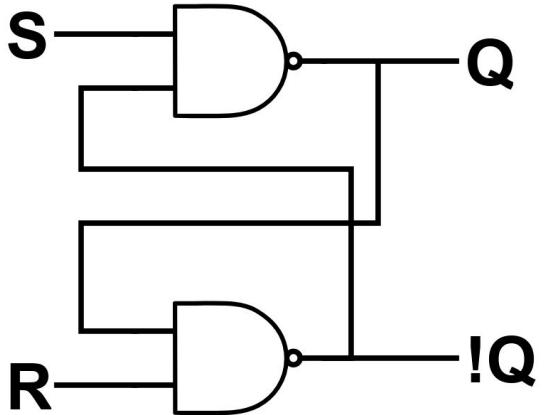
Vemos que ahora las entradas de NOR B son 1 y 0, por ende mantiene su salida en 0. Vemos que esto nuevamente fuerza a  $Q = !Q$ , rompiendo el requisito de que sean complementarias perdiendo la capacidad de memorizar.

S	R	$Q_N$
0	0	$Q_{N-1}$
0	1	0
1	0	1
1	1	?

Nota: NOR=1 si y sólo si ambas entradas son 0.



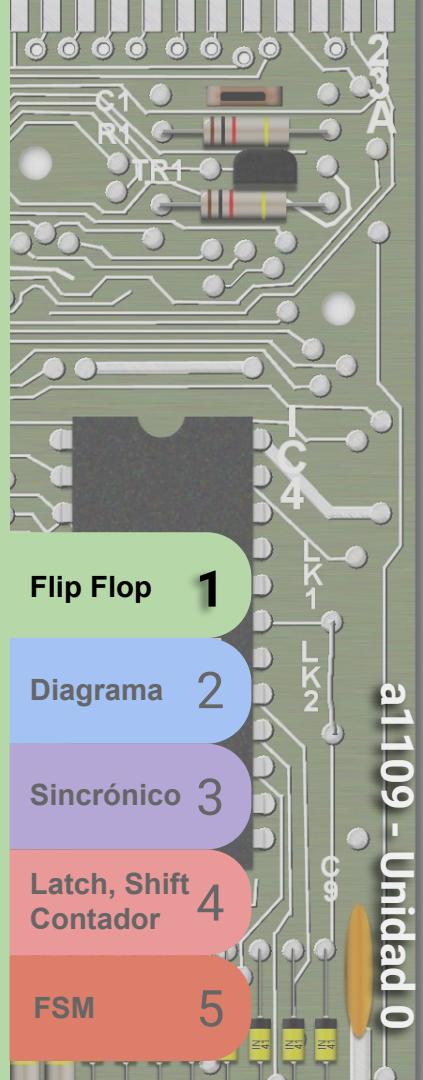
# Flip Flop RS asincrónico con NAND



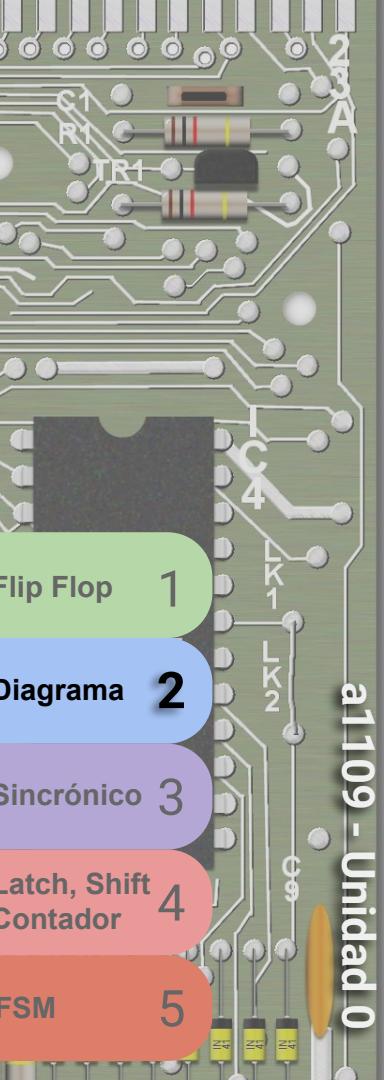
Podemos implementar un FF RS utilizando compuertas NAND. Este se comporta también como un circuito con memoria, pero la tabla de verdad ahora es distinta.

S	R	NANDQ <sub>N</sub>
0	0	?
0	1	1
1	0	0
1	1	Q <sub>N-1</sub>

Notemos que en esta implementación con Nand, la salida Q está conectada a la NAND que se encuentra conectada a S. Vemos que el estado que memoriza ahora es S=1 y R=1. Cuando queremos establecer hacemos S=0 y cuando queremos restablecer hacemos R=0.



# Diagramas de tiempo

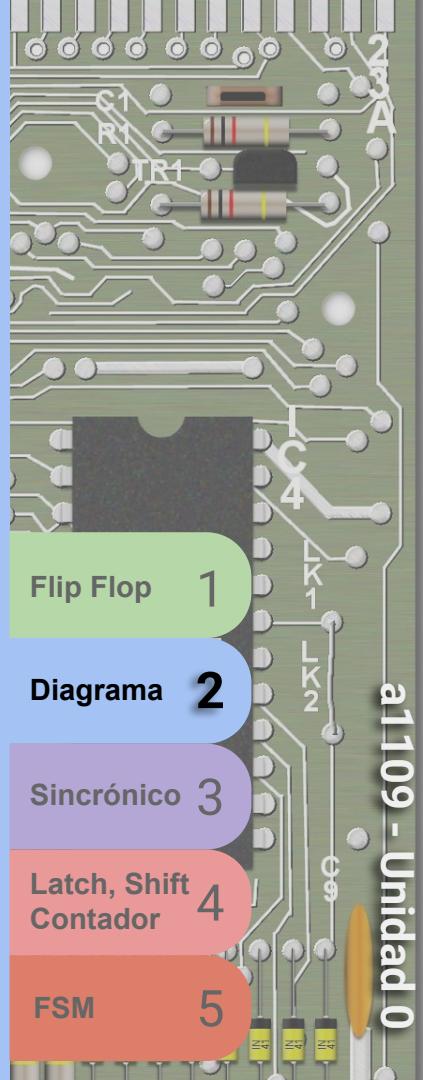


# Ver los gráficos en función del tiempo

En los circuitos combinacionales vistos previamente poseemos una tabla de verdad que relaciona las entradas con las salidas. Las compuertas con las que se implementan los circuitos son elementos físicos. Como tales tienen propiedades y se comportan de manera no ideal. Si bien para representar su función lógica los tratamos como ideales, al momento de implementar el circuito con compuertas reales pueden aparecer transiciones indeseadas (glitches) o podemos notar que existen tiempos de retardos no considerados originalmente.

Esto lo veremos en detalle en la unidad 1 de tecnología. Por ahora alcanza con saber que desde el momento en el que se produce un cambio en una entrada hasta que ese cambio impacta en la salida transcurre un pequeño tiempo de propagación.

En el caso de circuitos combinacionales, solo con ver la tabla de verdad podemos entender como funciona el circuito. Dado que no depende de valores anteriores, los diagramas no son otra cosa más que una representación de la tabla de verdad.



# Ver los gráficos en función del tiempo

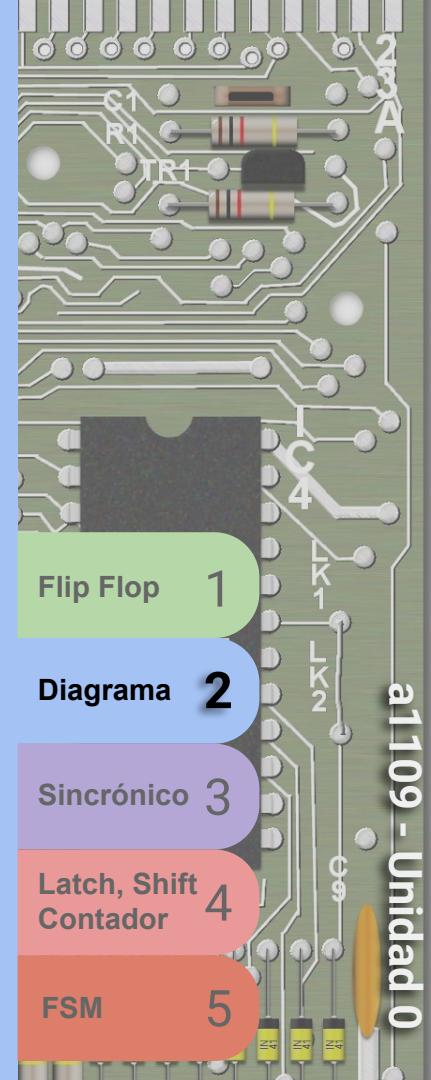
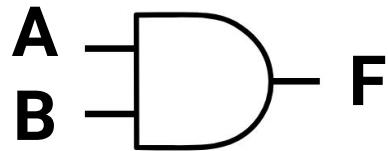
Veamos un diagrama donde representamos en cada renglón una entrada o una salida (con valores 0 o 1) ubicadas en el eje Y. En el eje X tenemos el tiempo. A medida que nos movemos a la derecha el tiempo transcurre.

Veamos una compuerta AND. Vemos la tabla de verdad e identificamos las entradas A y B, y la salida F.

Comenzamos identificando las entradas y salidas.



A	B	F
0	0	0
0	1	0
1	0	0
1	1	1

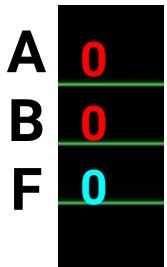


# Ver los gráficos en función del tiempo

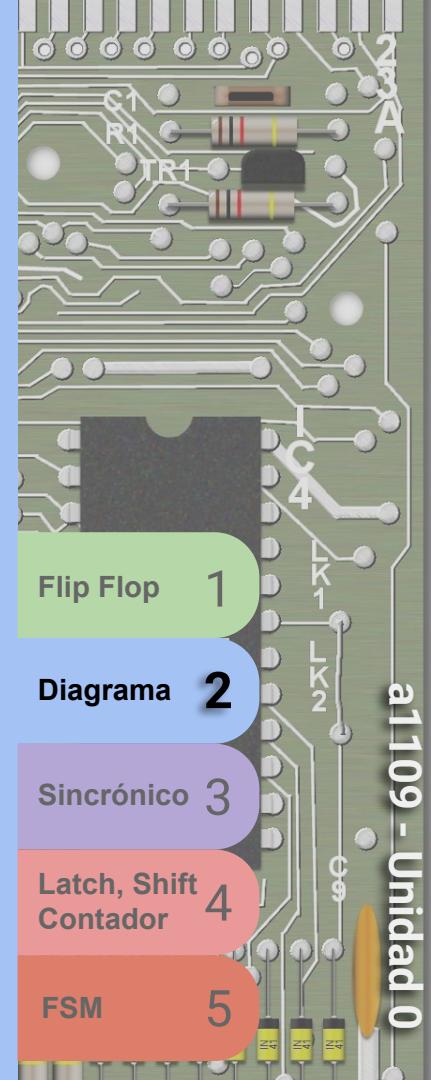
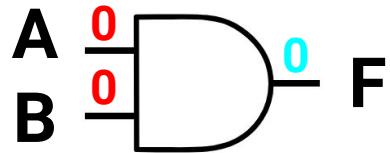
Veamos un diagrama donde representamos en cada renglón una entrada o una salida (con valores 0 o 1) apiladas en el eje Y. En el eje X tenemos el tiempo. A medida que nos movemos a la derecha el tiempo transcurre.

Veamos una compuerta AND. Vemos la tabla de verdad e identificamos las entradas A y B, y la salida F.

Empezamos con las dos líneas de entrada (A,B) en 0. Vemos que F toma 0 en este caso



A	B	F
0	0	0
0	1	0
1	0	0
1	1	1



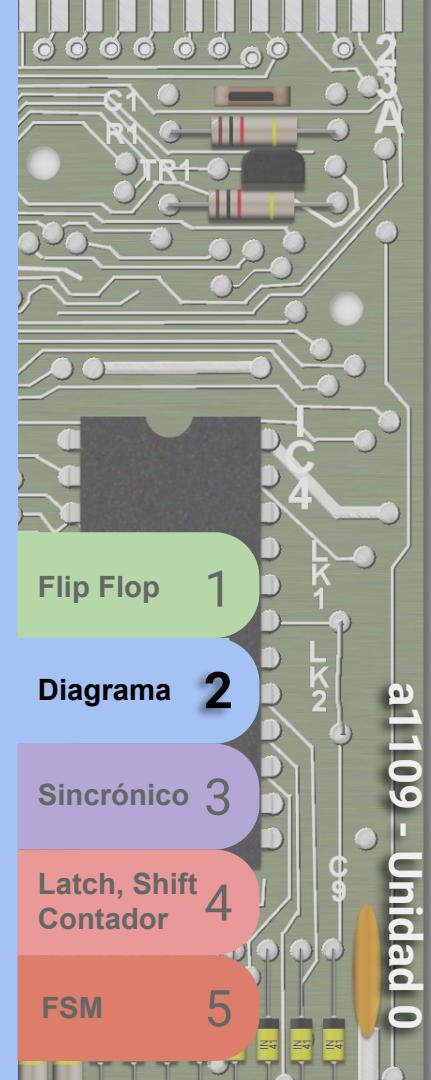
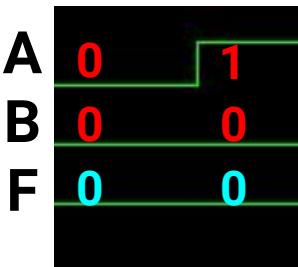
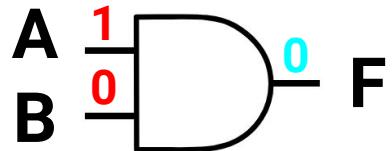
# Ver los gráficos en función del tiempo

Veamos un diagrama donde representamos en cada renglón una entrada o una salida (con valores 0 o 1) apiladas en el eje Y. En el eje X tenemos el tiempo. A medida que nos movemos a la derecha el tiempo transcurre.

Veamos una compuerta AND. Vemos la tabla de verdad e identificamos las entradas A y B, y la salida F.

Cambiamos A=1, vemos que para la función F (AND) el valor continua en 0.

A	B	F
0	0	0
0	1	0
1	0	0
1	1	1



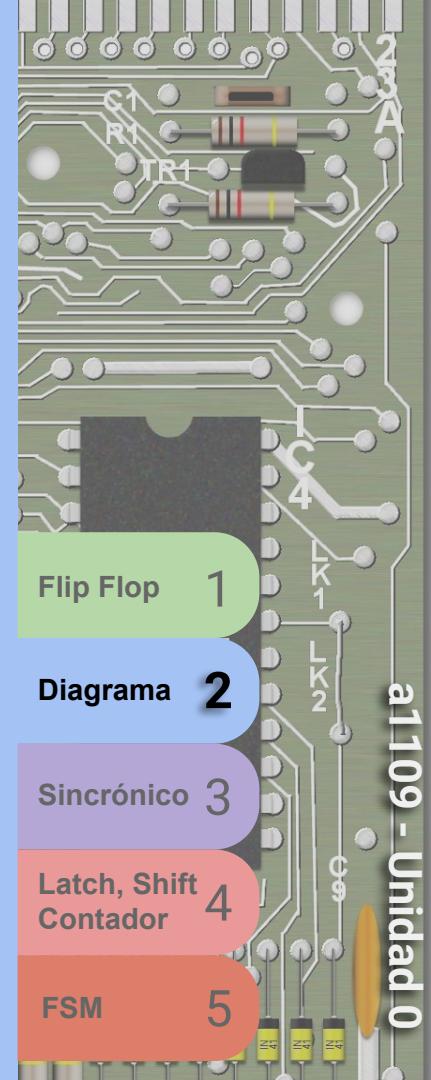
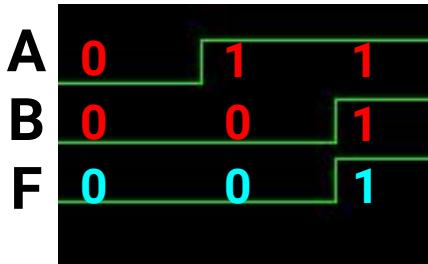
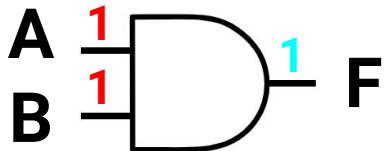
# Ver los gráficos en función del tiempo

Veamos un diagrama donde representamos en cada renglón una entrada o una salida (con valores 0 o 1) apiladas en el eje Y. En el eje X tenemos el tiempo. A medida que nos movemos a la derecha el tiempo transcurre.

Veamos una compuerta AND. Vemos la tabla de verdad e identificamos las entradas A y B, y la salida F.

Ahora cambiamos B=1, por ende A=1, B=1 entonces F=1.

A	B	F
0	0	0
0	1	0
1	0	0
1	1	1



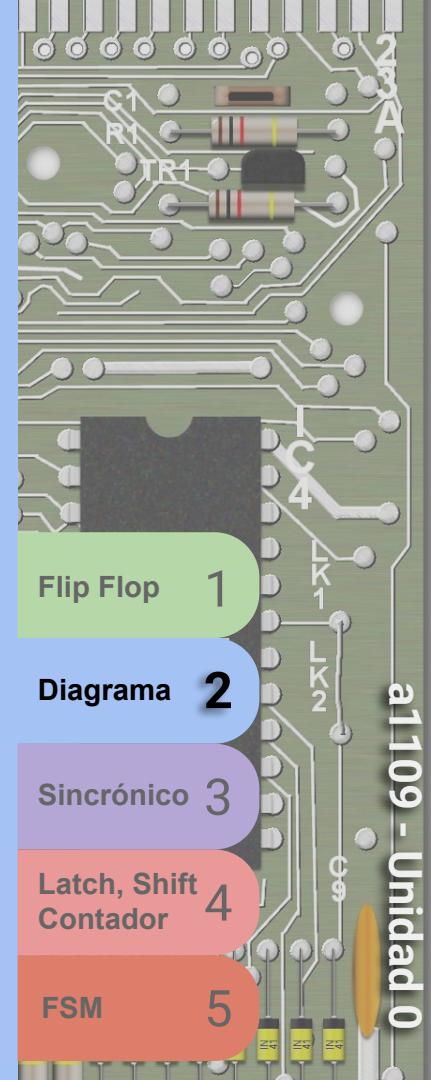
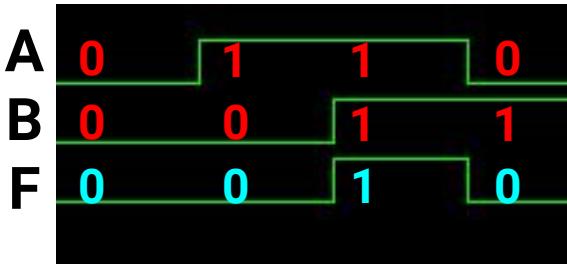
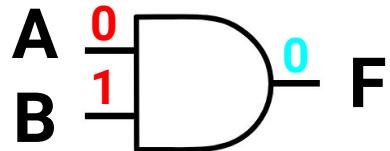
# Ver los gráficos en función del tiempo

Veamos un diagrama donde representamos en cada renglón una entrada o una salida (con valores 0 o 1) apiladas en el eje Y. En el eje X tenemos el tiempo. A medida que nos movemos a la derecha el tiempo transcurre.

Veamos una compuerta AND. Vemos la tabla de verdad e identificamos las entradas A y B, y la salida F.

Luego hacemos A=0, por ende F=0.

A	B	F
0	0	0
0	1	0
1	0	0
1	1	1

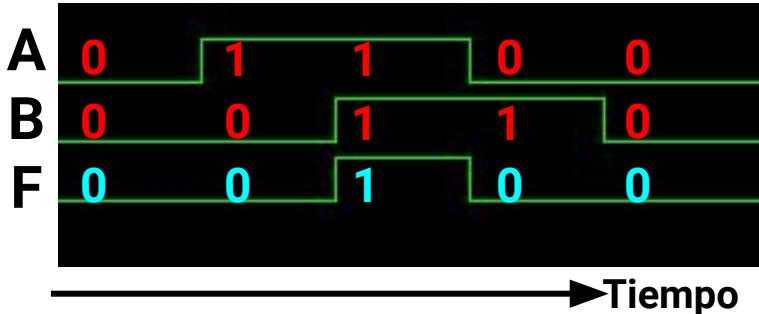
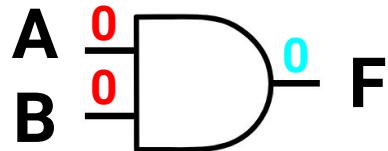


# Ver los gráficos en función del tiempo

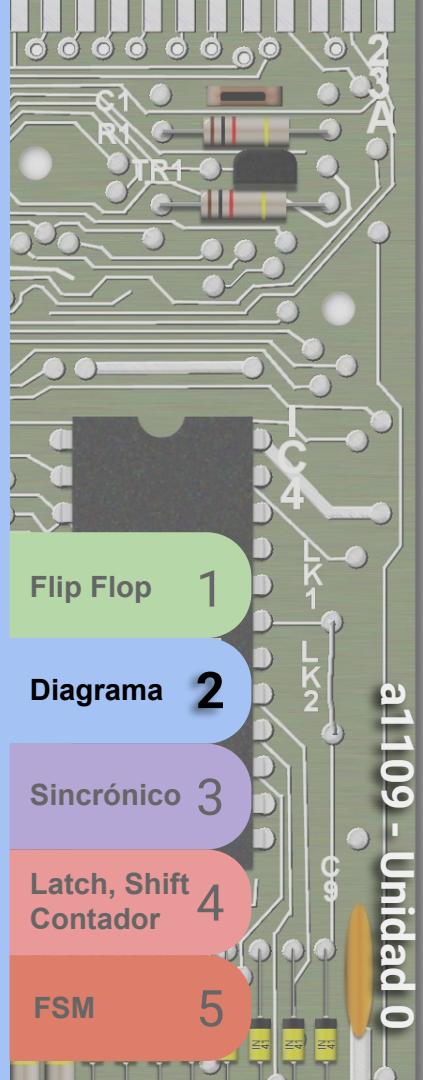
Veamos un diagrama donde representamos en cada renglón una entrada o una salida (con valores 0 o 1) apiladas en el eje Y. En el eje X tenemos el tiempo. A medida que nos movemos a la derecha el tiempo transcurre.

Veamos una compuerta AND. Vemos la tabla de verdad e identificamos las entradas A y B, y la salida F.

A	B	F
0	0	0
0	1	0
1	0	0
1	1	1

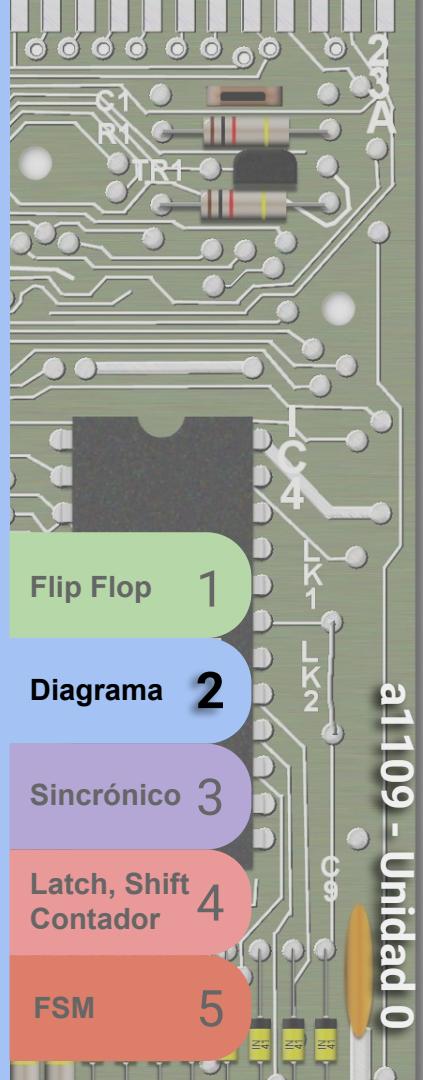
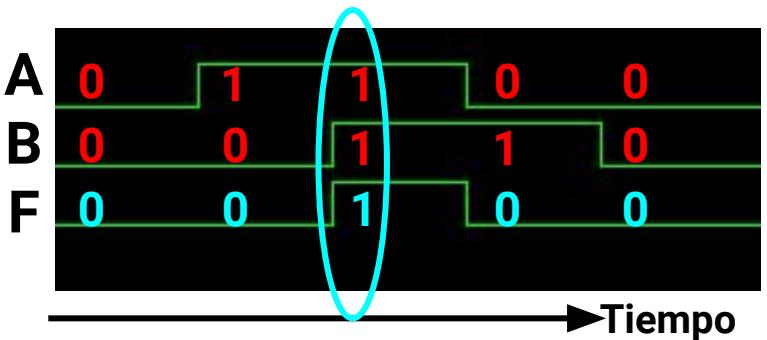


Por último B=0, por ende F continua en 0. Vemos que el diagrama de tiempos representa como fueron cambiando las entradas y qué efecto tuvo esto en la salida... en este caso trivial porque era una simple compuerta AND.

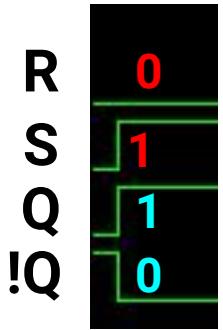


# Ver los gráficos en función del tiempo

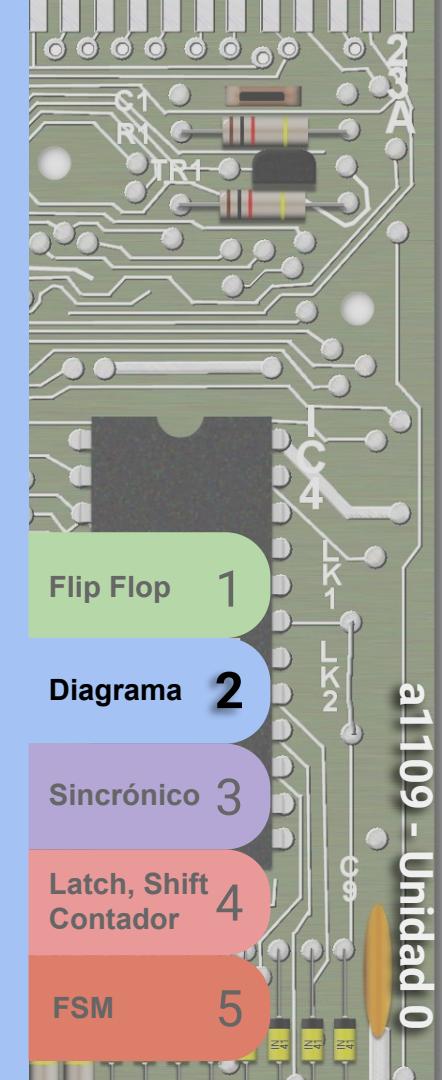
Notemos que en el momento que ambas entradas son 1, el cambio en la salida se produce instantáneamente. Esto evidentemente no ocurre en la vida real. Una magnitud física no puede cambiar en un instante cero. Pero esto lo veremos en detalle en la siguiente unidad de tecnología. A los fines del diagrama de tiempos, podemos elegir representar estos tiempos o no. En este caso no ha sido representado el tiempo de propagación de la compuerta.



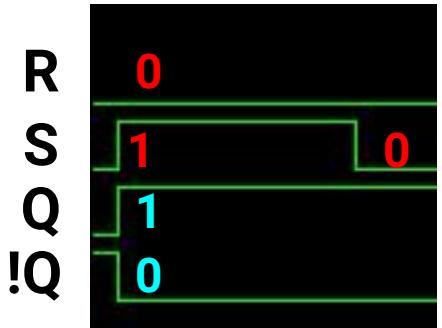
# Ver los gráficos en función del tiempo



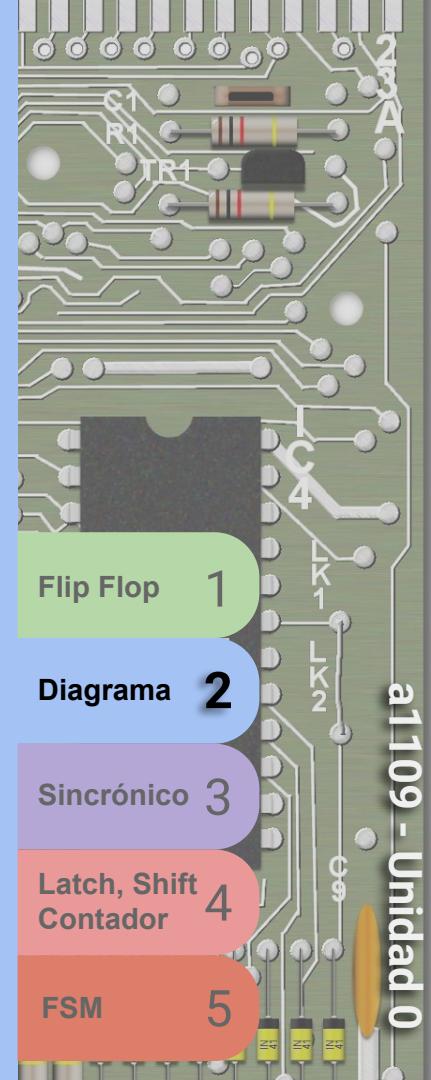
Veamos ahora el caso del Flip Flop RS. Lo establecemos en 1 y vemos que su salida  $Q=1$  y  $\bar{Q}=0$ .



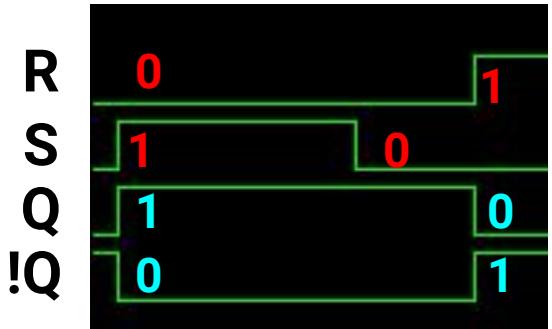
# Ver los gráficos en función del tiempo



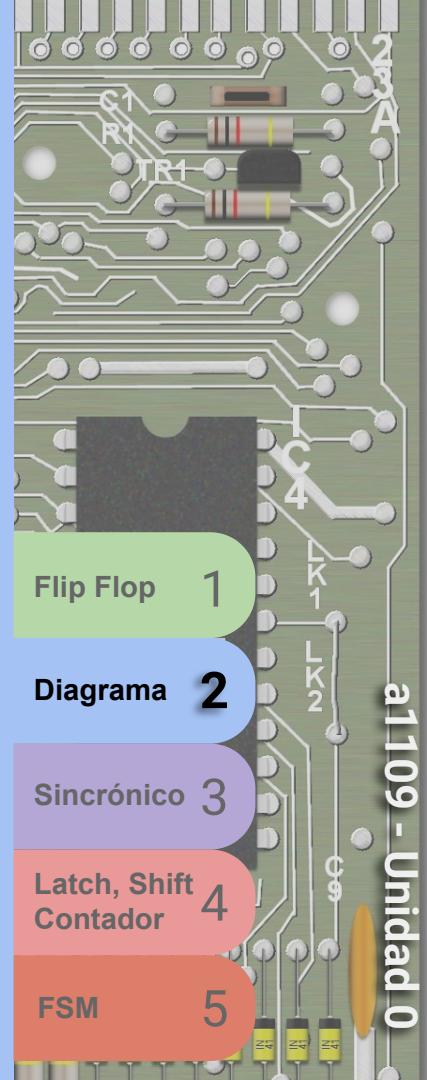
Vemos que cuando S vuelve a 0, se mantienen las salidas Q y !Q.



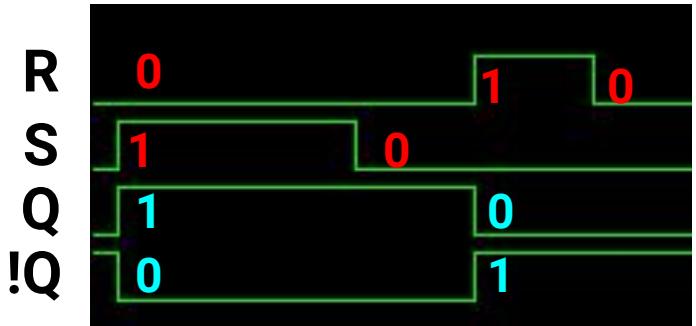
# Ver los gráficos en función del tiempo



Ahora cuando  $R=1$  (siendo  $S=0$ ) vemos que  $Q$  pasa a 0 y  $!Q=1$ . El 1 en  $R$  esta restableciendo el valor cero.

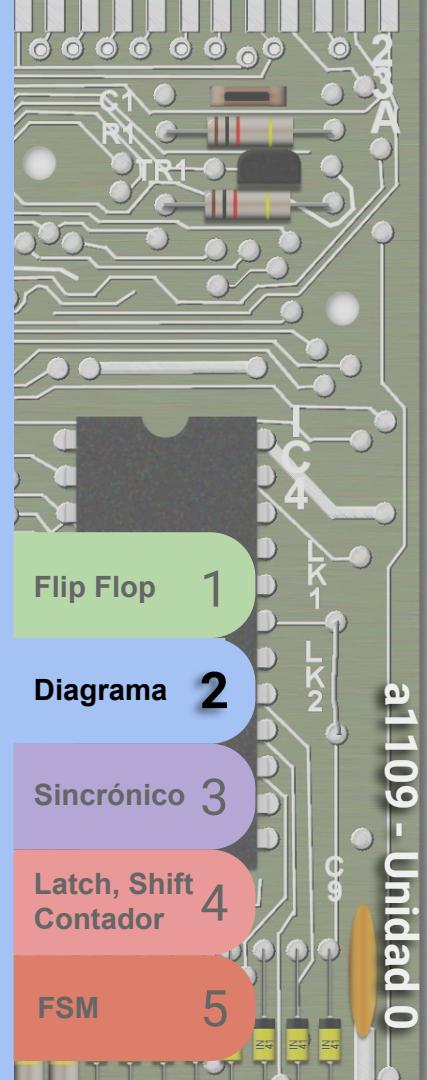


# Ver los gráficos en función del tiempo

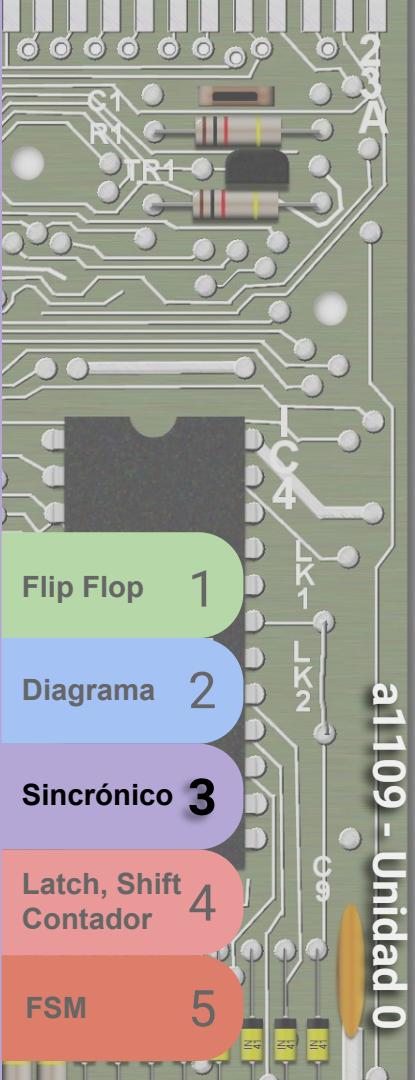


Cuando R vuelve a 0 el FF mantiene su estado en las salidas.

Notamos que no se representa el tiempo de propagación de las compuertas donde el flip flop permanece en estado de transición (inestable). Dejamos como ejercicio realizar el diagrama de tiempos teniendo en cuenta que las transiciones no son inmediatas y representando los momentos donde el flip flop se encuentra inestable.

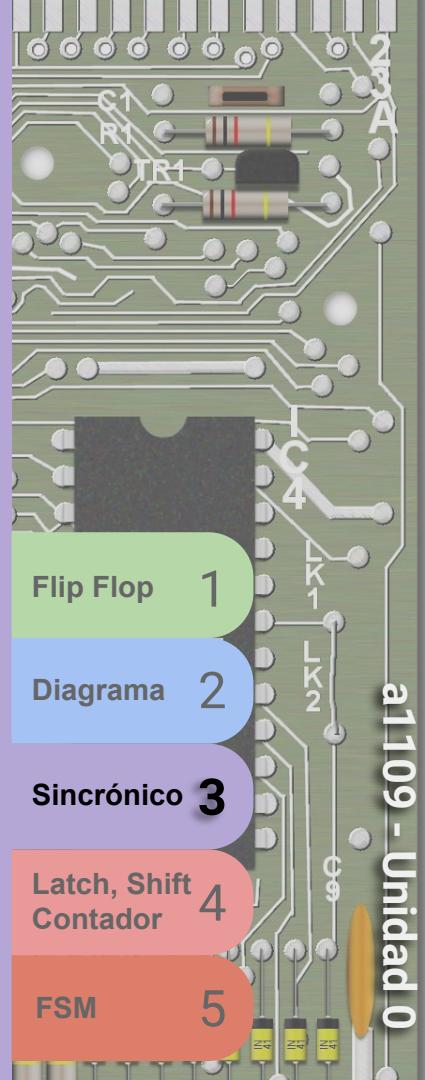
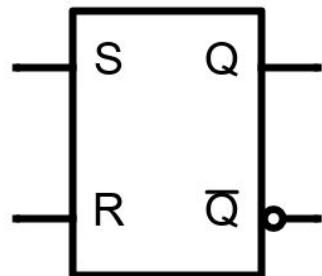
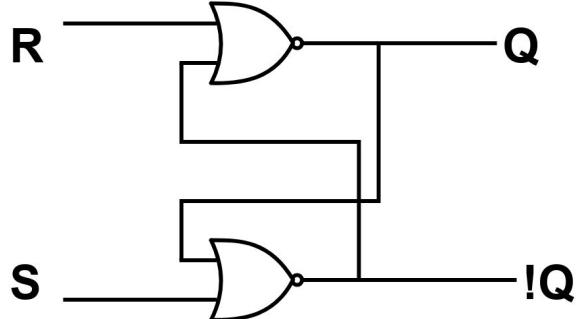


# Circuitos sincrónicos



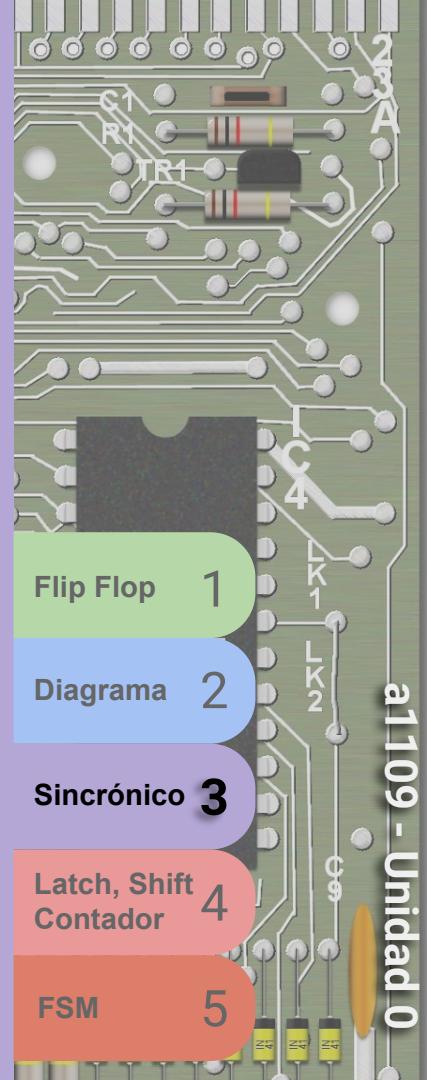
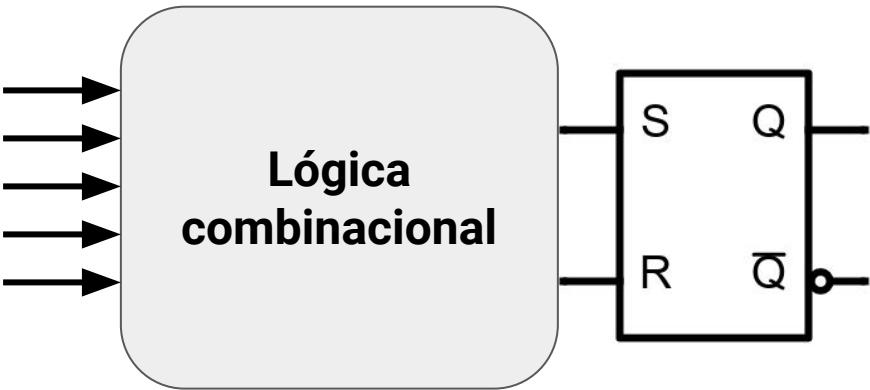
## Diseño modular

Dado que un diseño suele estar compuesto de muchos componentes, conviene representar los componentes que realizan una función específica y definida como un módulo. En este caso vemos que podemos representar el FF RS como una caja con 2 entradas y dos salidas (claramente identificadas) en vez de estar dibujando las compuertas.



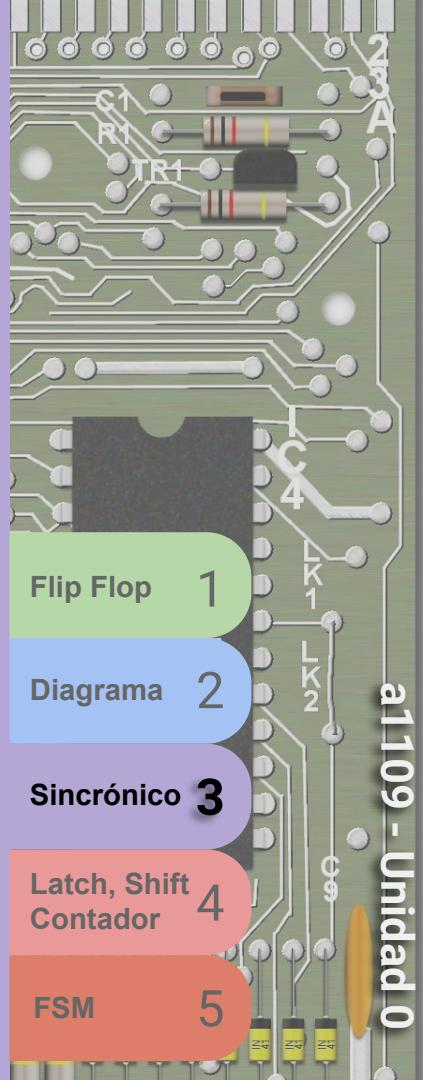
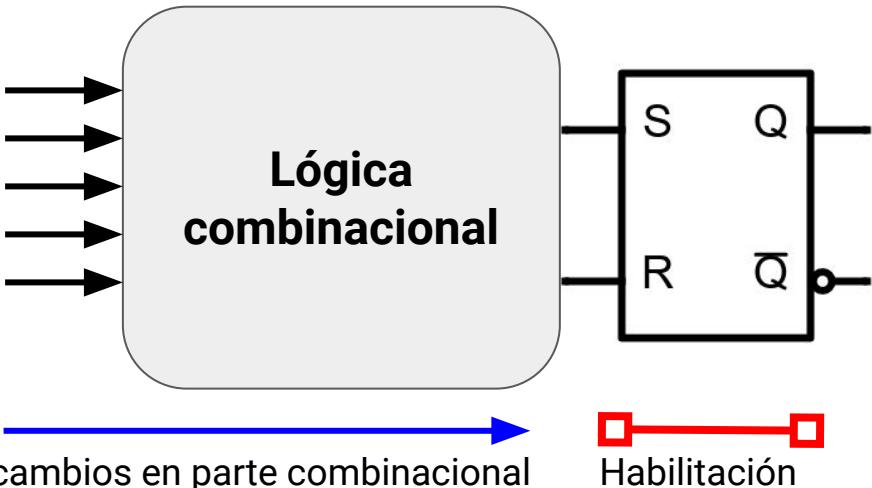
# Sincronismos

Si tenemos un conjunto de circuitos combinacionales que controlan las entradas de un circuito secuencial, es posible que mientras se produzcan los cambios internos en las compuertas las salidas S y R (entradas en el FF) vean cambios hasta que la lógica combinacional termina de comutar. Esto hará que el circuito secuencial vea valores intermedios y actúe en función a ellos.



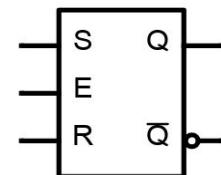
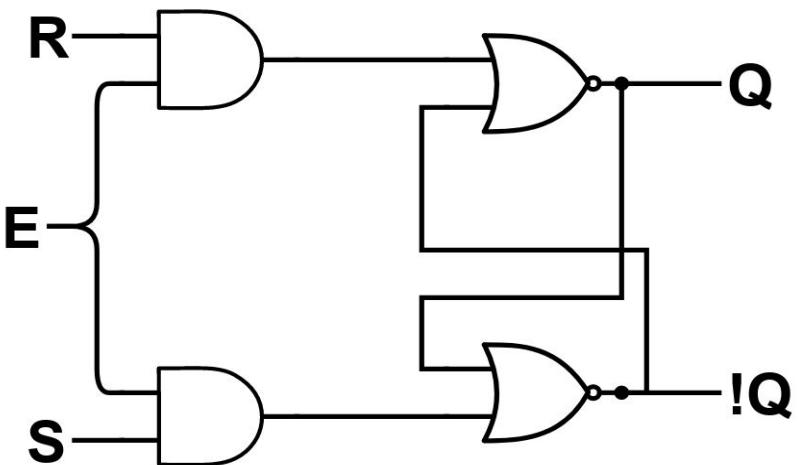
# Sincronismos

Para evitar esto, deberíamos lograr que el circuito secuencial solo le preste atención a sus entradas cuando la lógica combinacional termine de conmutar. Deberíamos tener una forma de habilitar la lógica secuencial únicamente cuando es seguro tomar el valor de las salidas de la lógica combinacional.

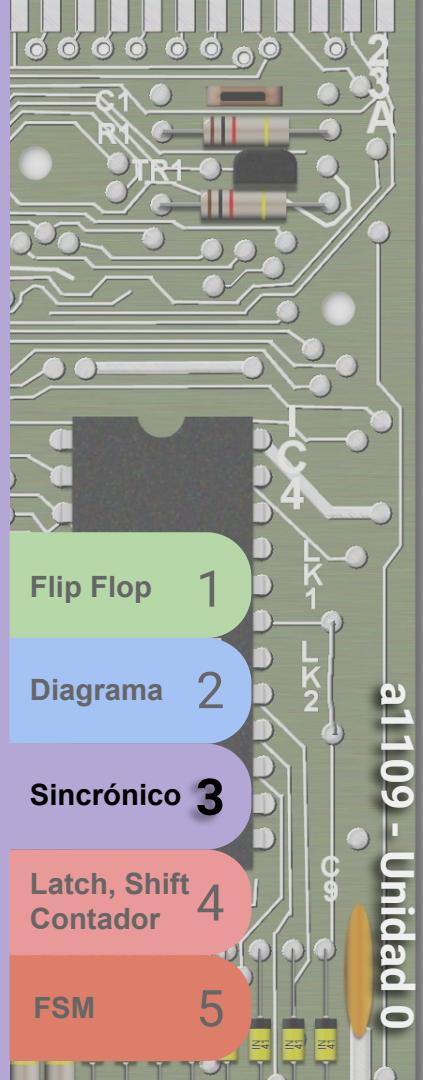


# Flip Flop RS Sincrónico

Con el fin de controlar el momento en el cual el FF RS le “presta atención” a las entradas, agregamos una nueva entrada llamada Enable (Habilitación) y la representamos con la E. Vemos que las entradas R y S ahora pasan (cada una) por una compuerta AND antes de llegar a sus respectivas NOR, y esa AND se comparte en cada caso son la nueva entrada E. A la derecha vemos la representación modular del mismo.

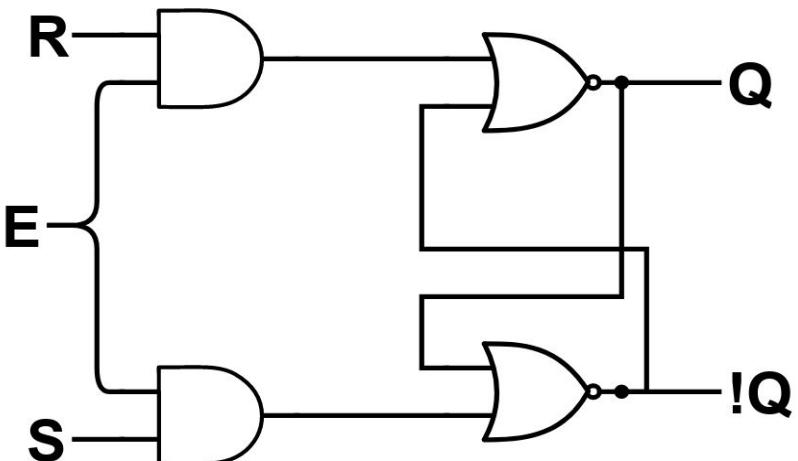


**Flip Flop  
RS  
Sincrónico**

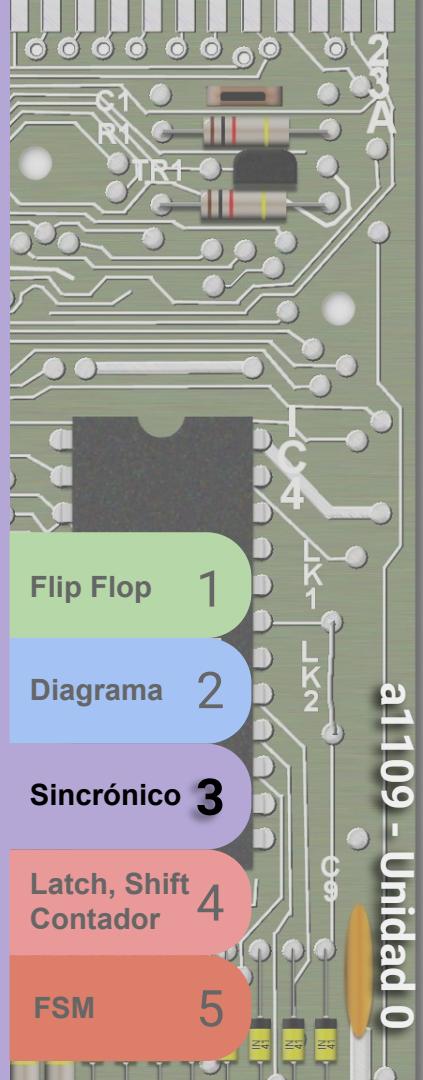


# Flip Flop RS Sincrónico

Vemos que la compuerta AND solo genera un uno cuando ambas entradas son uno. Por ende, mientras  $E=0$ , ambas AND seran 0. Y un FF RS cuyas entradas son ambas 0 mantiene su estado  $Q_{N-1}$ . Cuando  $E=1$  se comporta como un FF RS de los ya vistos.

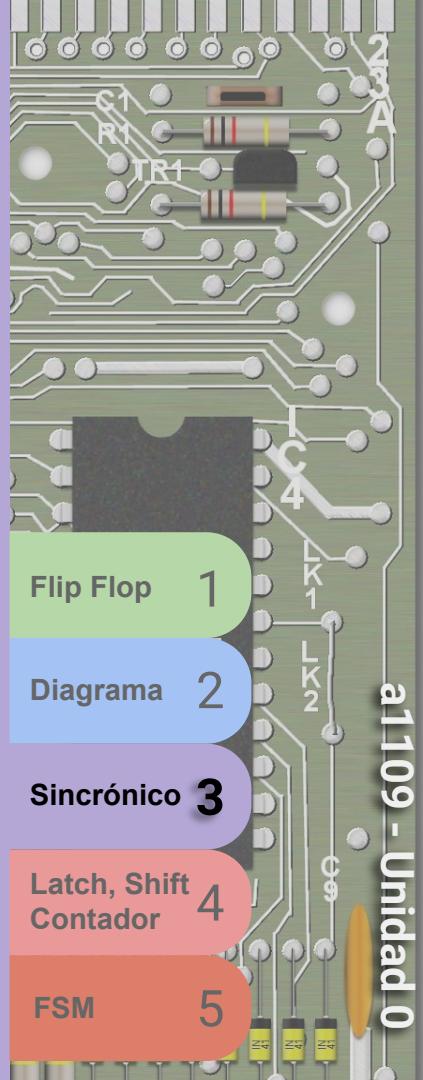
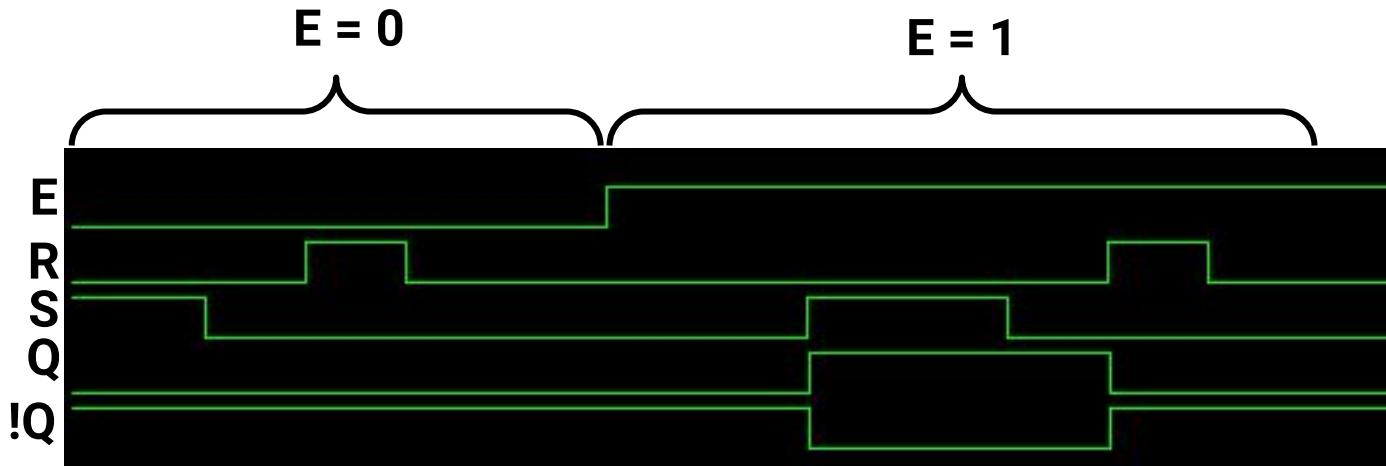


E	R	S	Q
0	0	0	$Q_{N-1}$
0	0	1	$Q_{N-1}$
0	1	0	$Q_{N-1}$
0	1	1	$Q_{N-1}$
1	0	0	$Q_{N-1}$
1	0	1	1
1	1	0	0
1	1	1	?



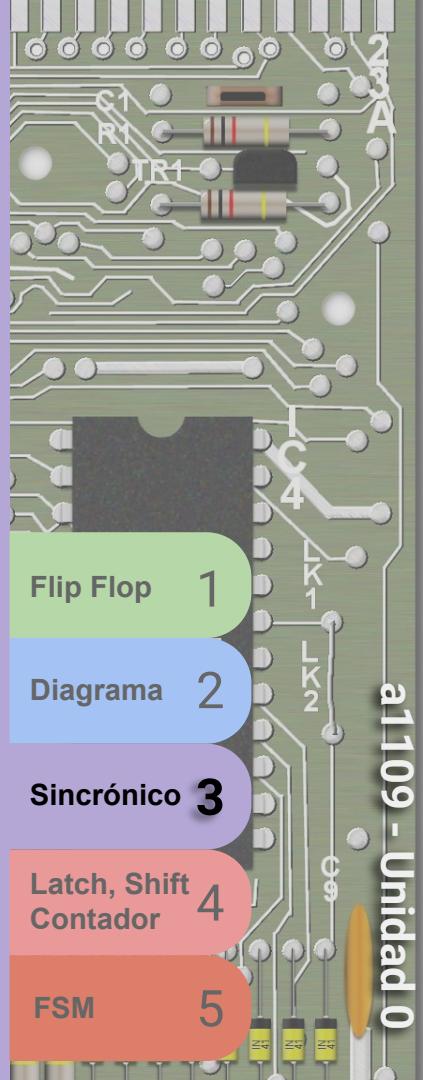
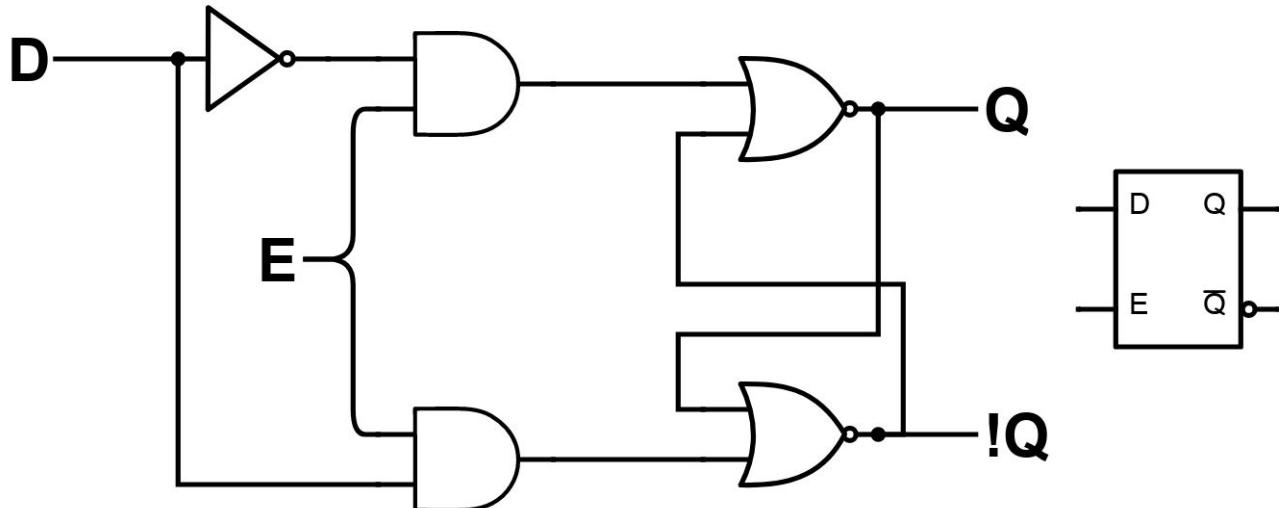
# Flip Flop RS Sincrónico

Vemos que mientras la entrada de Enable se encuentra en bajo, los cambios en las entradas S y R son ignorados. El FF permanece en su estado  $Q=0$  independientemente de lo que pase en R y S. Ahora cuando  $E=1$ , los cambios en R y S son “vistos” por el FF RS.



# Flip Flop D Síncrónico

Hacemos una nueva modificación sobre el Flip Flop RS anterior. Si bien tener dos líneas de entrada (una para memorizar un 0 y una para memorizar un 1) es útil, vemos que con la entrada de sincronismo esto deja de tener sentido. Es más práctico tener una sola entrada de datos (que puede valer 0 o 1) ya que ahora solo se va a memorizar cuando E=1. Agregamos un negador y unimos R y



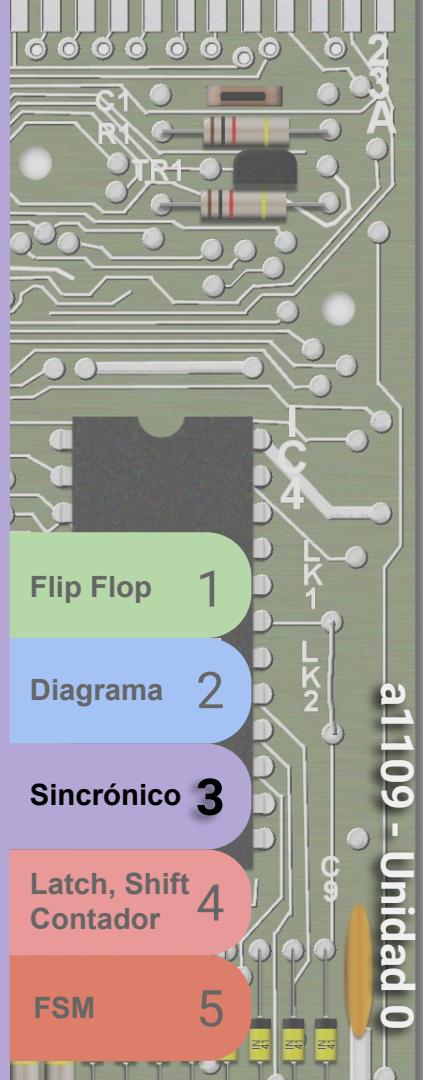
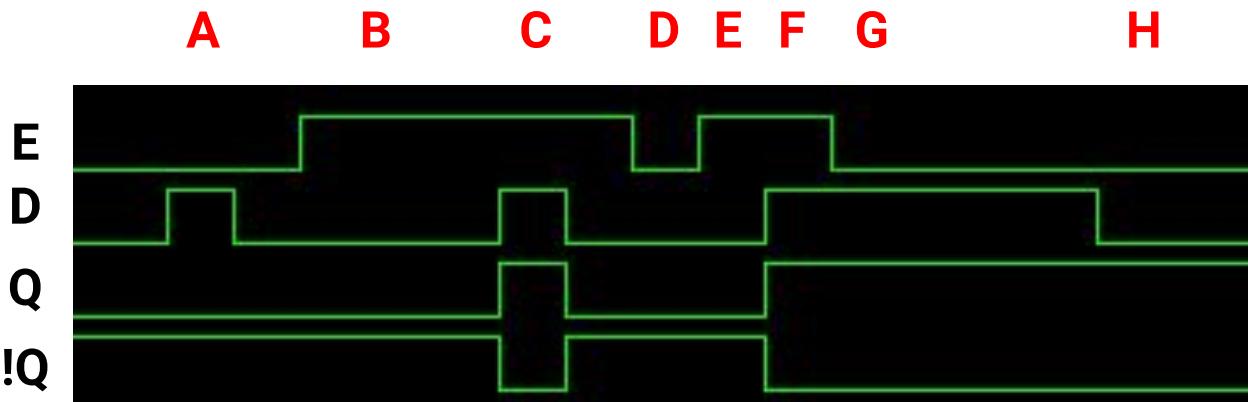
# Flip Flop D Sincrónico

Analicemos en detalle lo que sucede en los siguientes momentos de tiempo.

A- En este punto  $E=0$ , por ende el cambio en  $D$  es ignorado y  $Q$  permanece en su estado anterior  $Q=0$ .

B- Enable pasa a 1. Aquí el valor de  $D$  pasa a  $Q$ . Vemos que como  $D$  está en 0 entonces  $Q=0$ .

C- Ahora con  $E=1$  el valor de  $D=1$ , vemos que durante ese pulso la salida  $Q=D=1$ .



# Flip Flop D Sincrónico

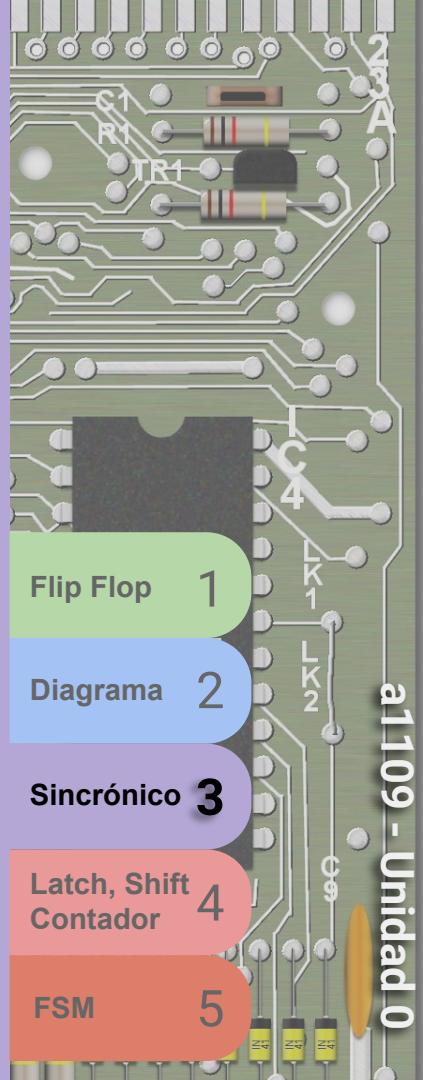
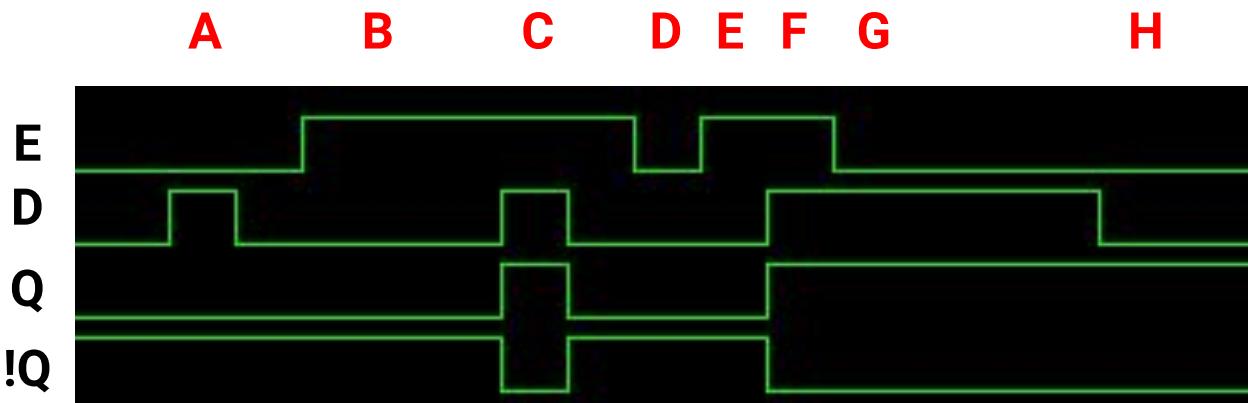
D- Vemos que enable pasa a 0, por ende el valor anterior de Q quedará memorizado.

E- Enable vuelve a 1, pero D se mantiene en 0 por ende Q=0.

F- Ahora D pasa a 1, por ende Q=1.

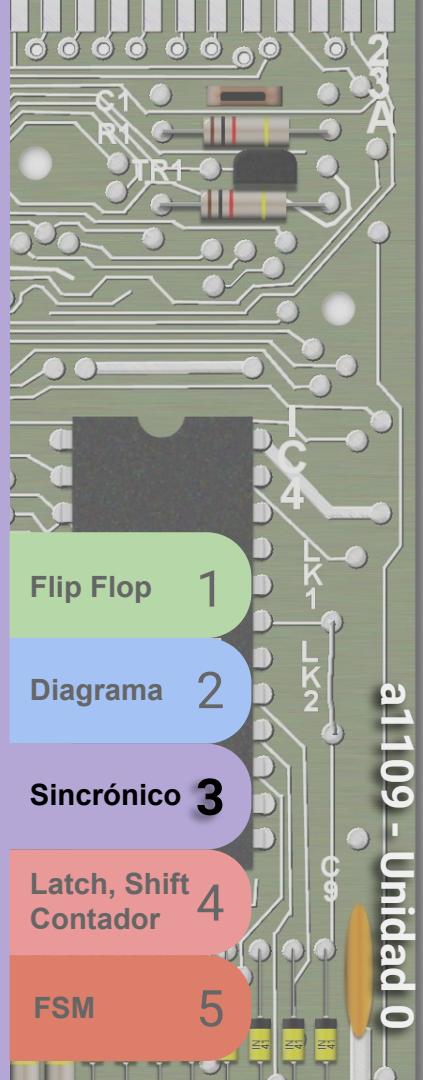
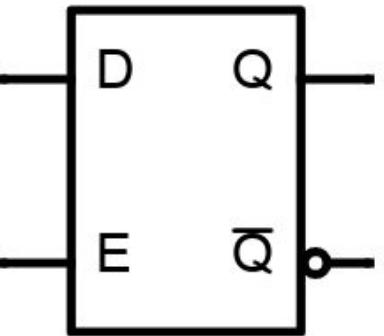
G- Enable ahora es 0, el último valor de Q es 1, por ende quedará con ese valor.

H- Ya al final D=0, pero al ser enable=0 entonces la salida Q no se ve afectada.



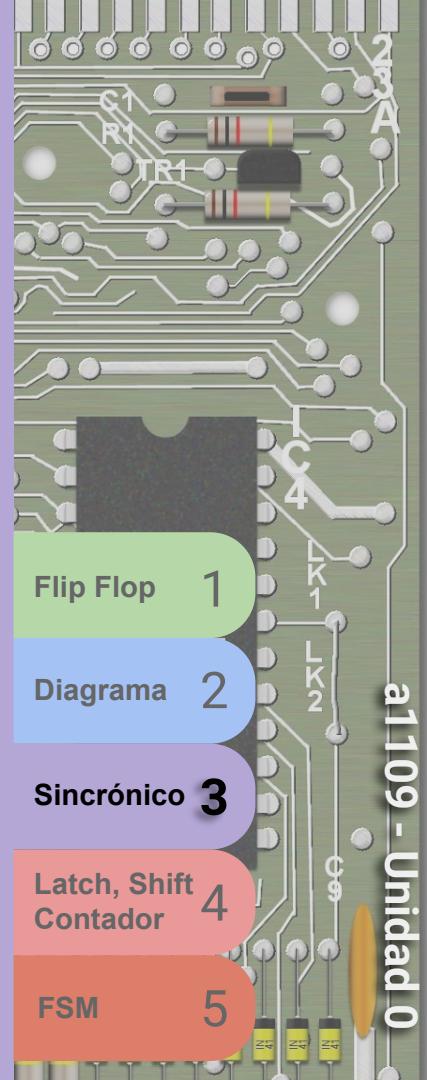
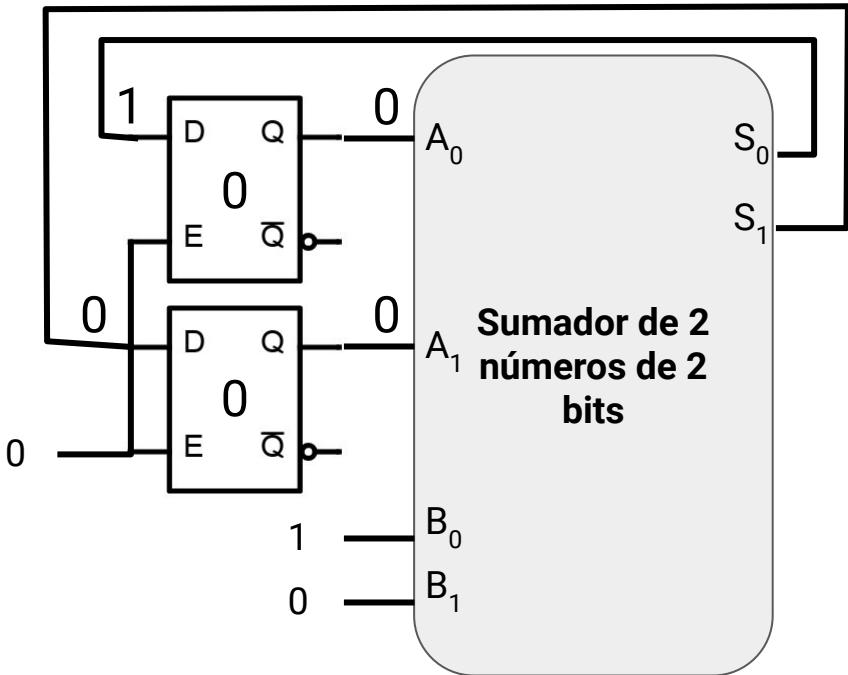
# Flip Flop D Sincrónico por nivel

Este flip flop D sincrónico trabaja “por nivel”, o sea, cuando el nivel de la línea de Enable es alto (existen versiones para bajo, aunque puede simplemente cambiarse con un negador) entonces el flip flop memoriza la entrada D.



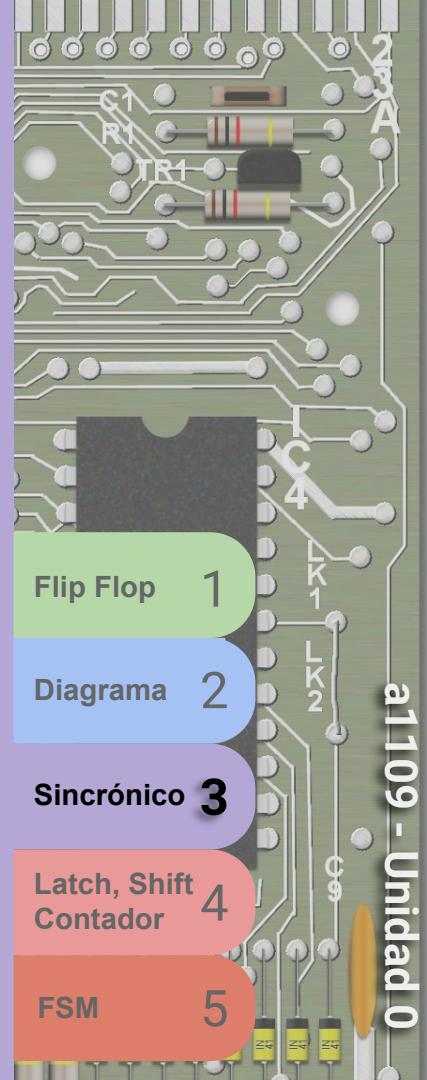
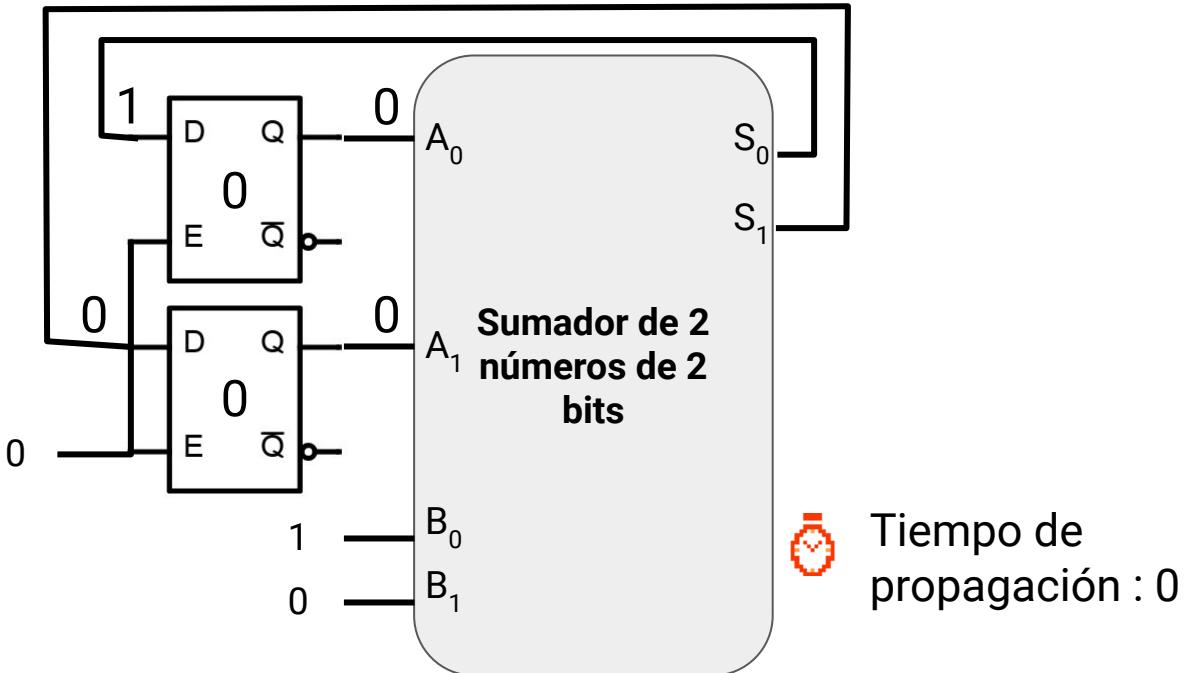
# Condición de carrera

Veamos el siguiente circuito. Es un sumador de 2 números de 2 bits que tiene una entrada constante en 1 ( $01_2$ ), y luego A se obtiene de dos FF y la salida del sumador S vuelve a los FF.



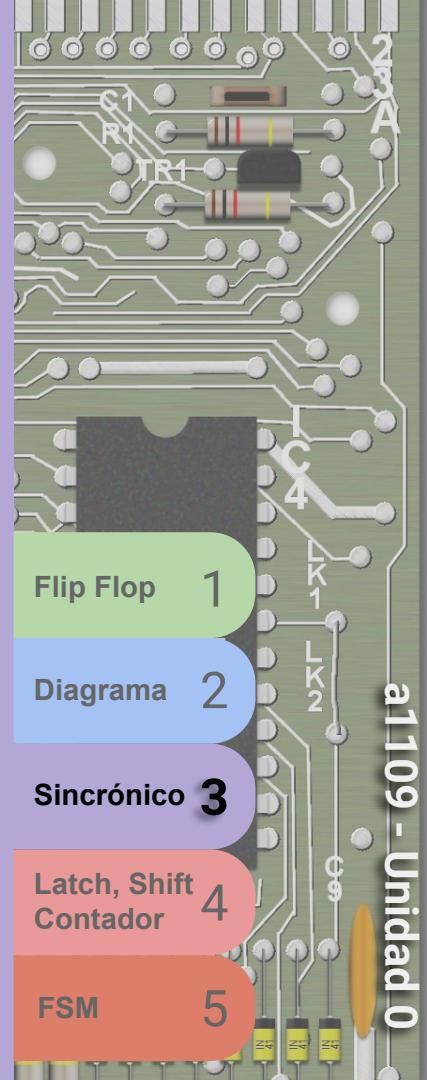
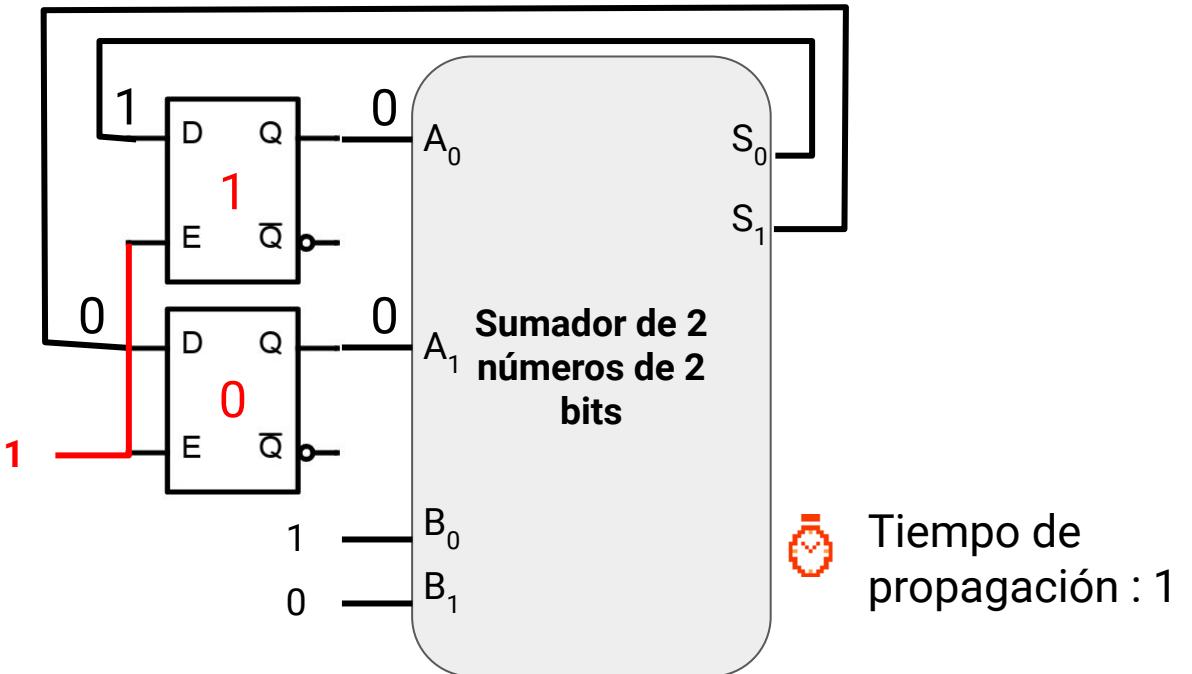
# Condición de carrera

Mientras  $E=0$ , el circuito está quieto. Supongamos que los FF tienen respectivamente 0 y 0 en sus salidas Q, el sumador va a producir un 01 en su salida S. Luego cambiamos  $E=1$ ...



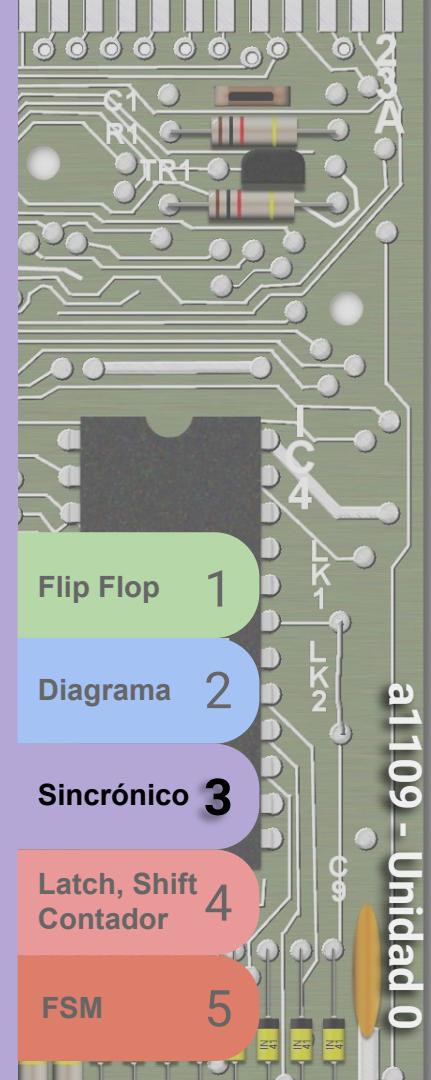
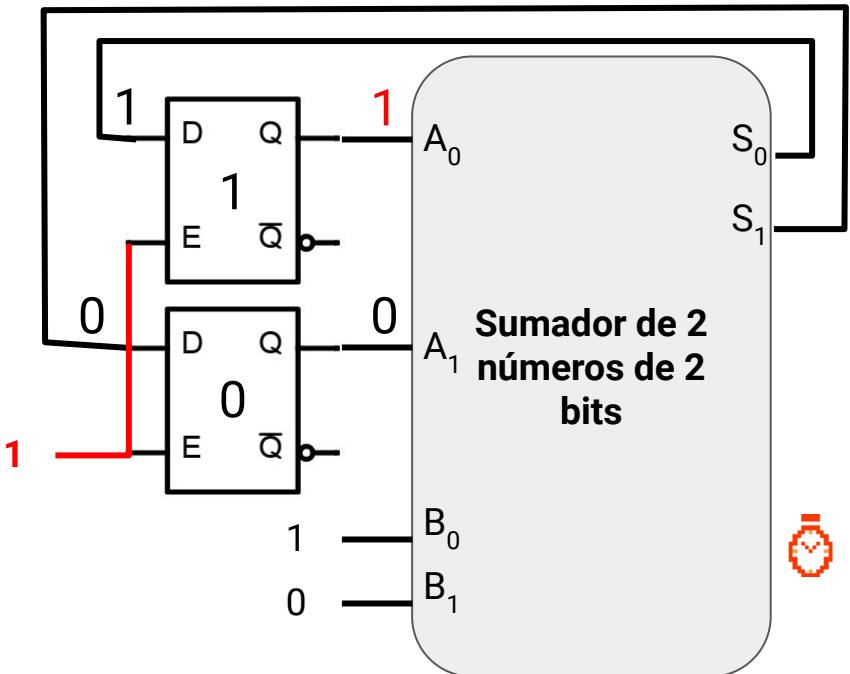
# Condición de carrera

Ahora que  $E=1..$  la salida  $S$  se memoriza en los FF ( $01_2$ ).



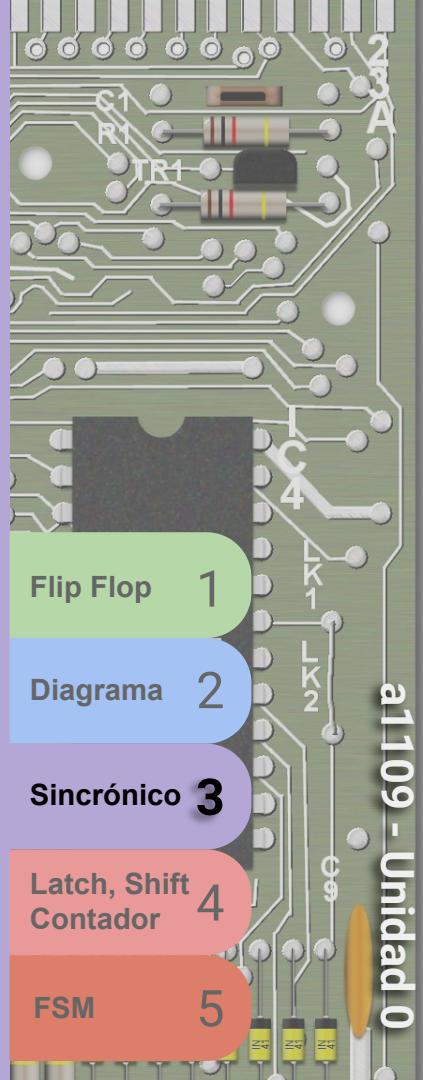
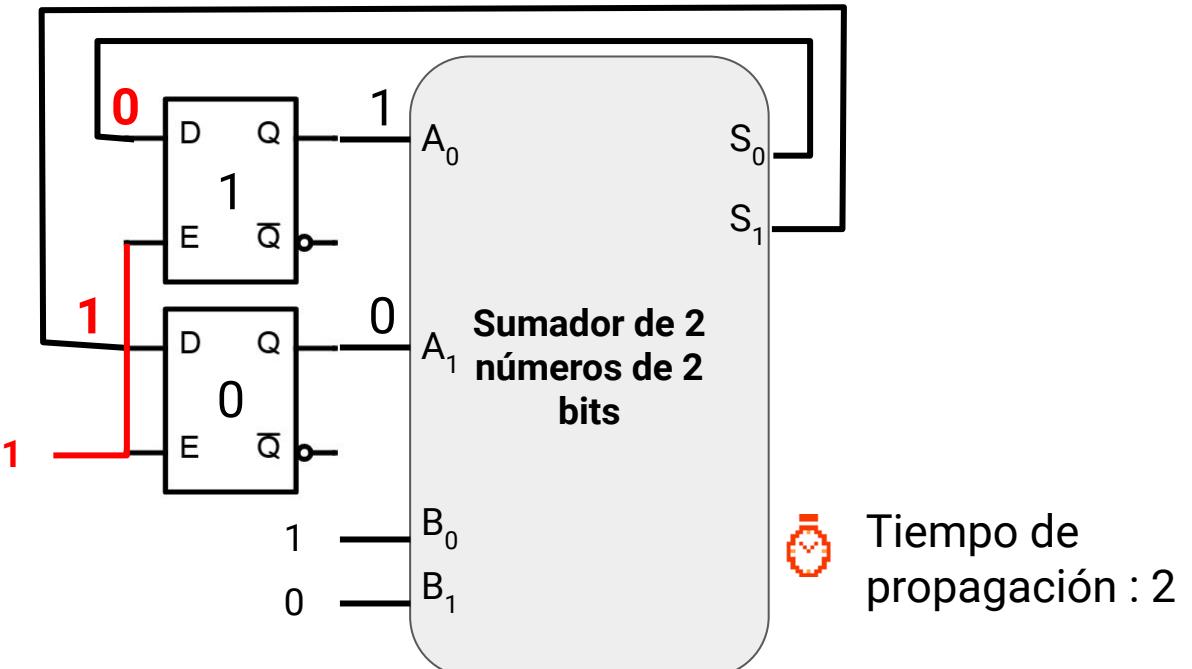
# Condición de carrera

La entrada del sumador ahora es 01.. Por ende el sumador realiza  $01+01$  y su salida pasa a 10...



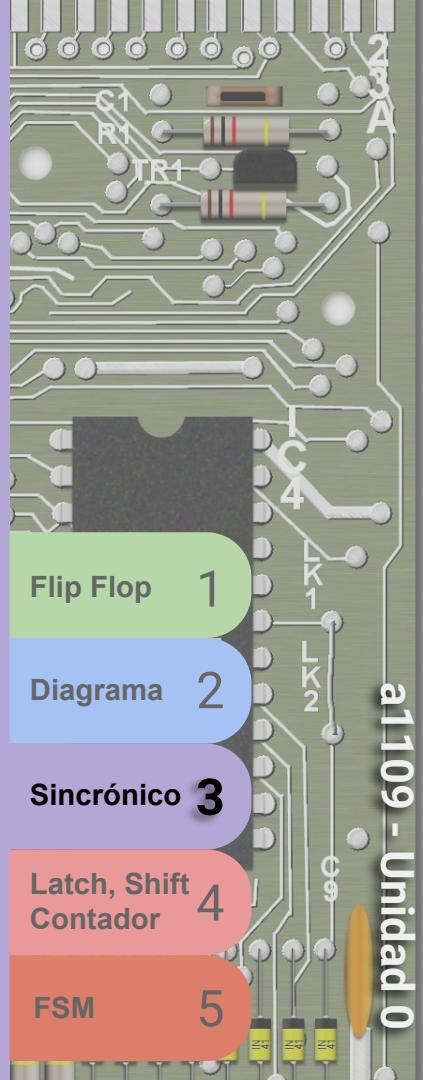
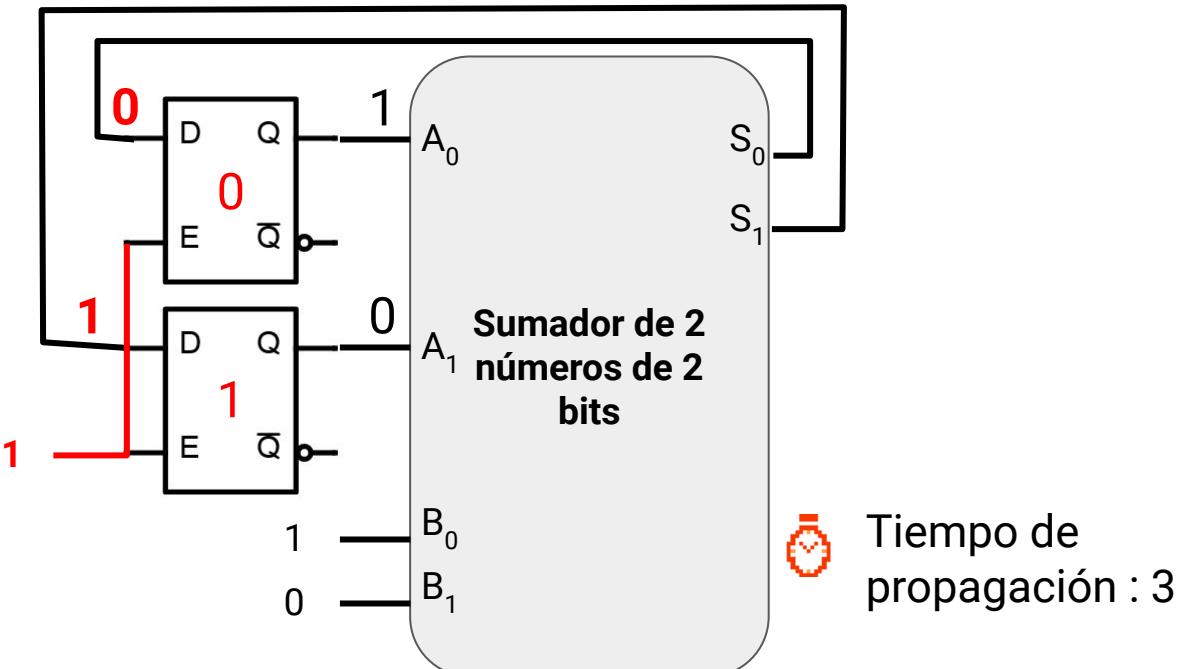
# Condición de carrera

Ahora los FF tienen a su entrada 10, pero vemos que E=1, por ende ese cambio (luego de un tiempo de propagación más) va a impactar en los FF.



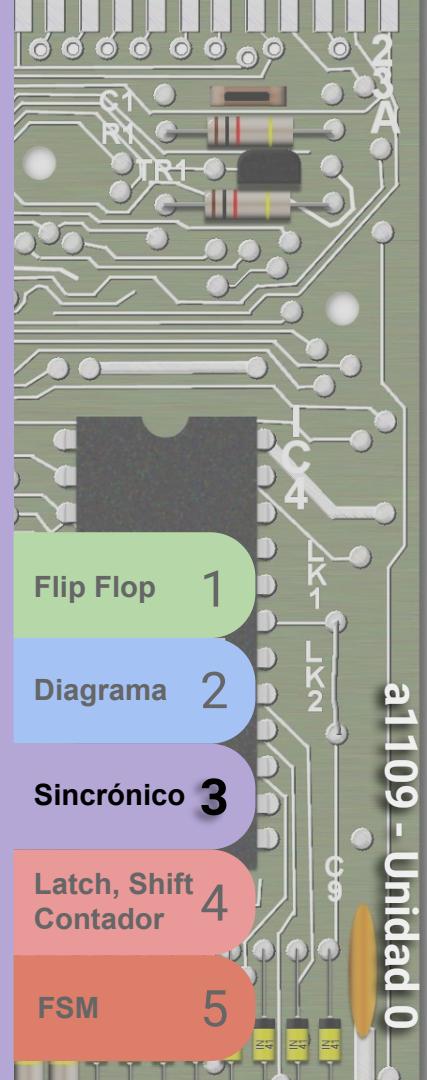
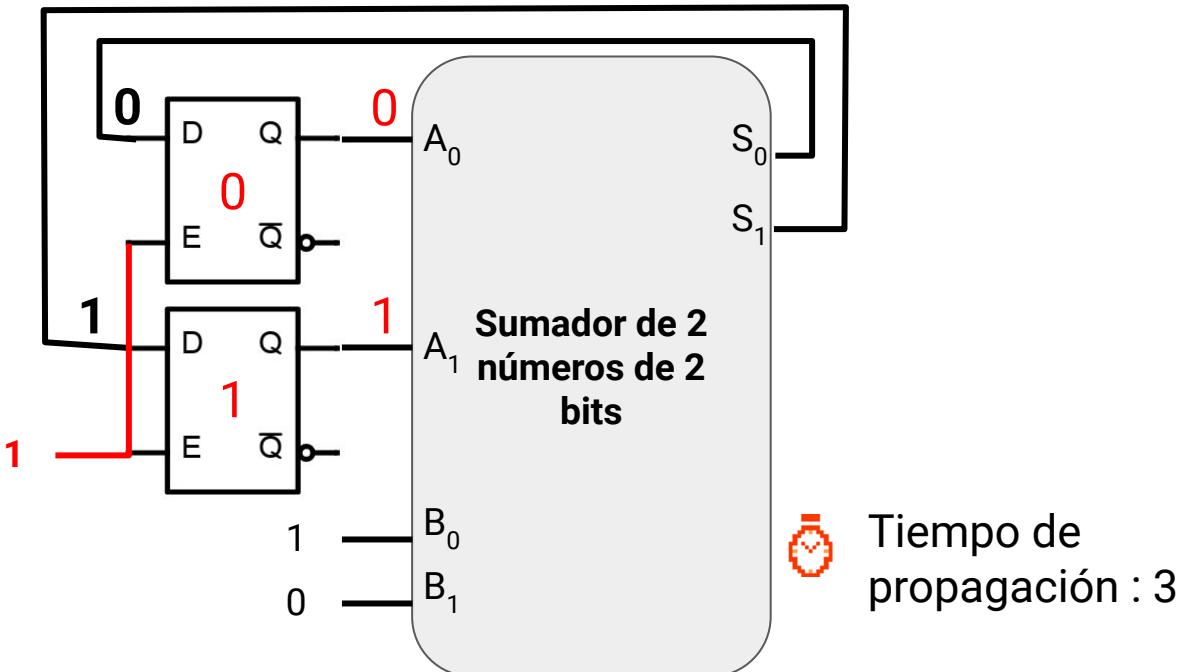
# Condición de carrera

Ahora los FF tienen a su entrada 10, pero vemos que E=1, por ende ese cambio (luego de un tiempo de propagación más) va a impactar en los FF.



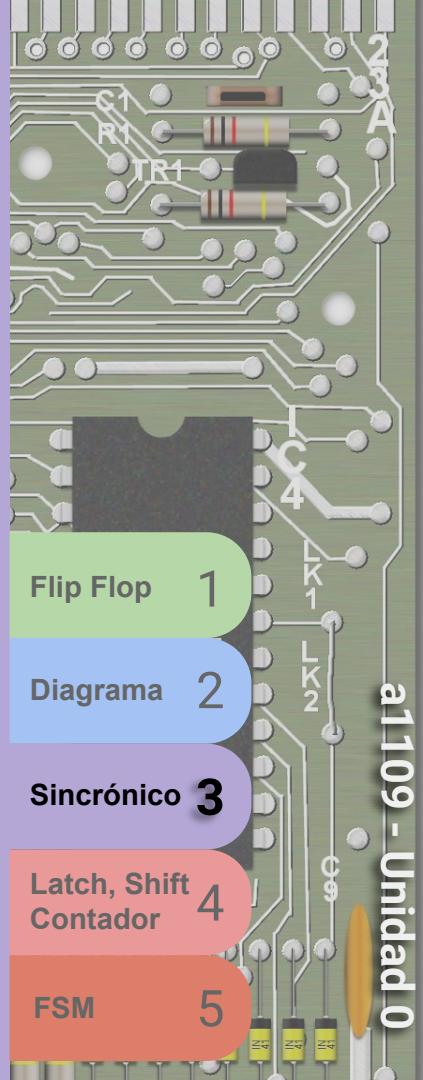
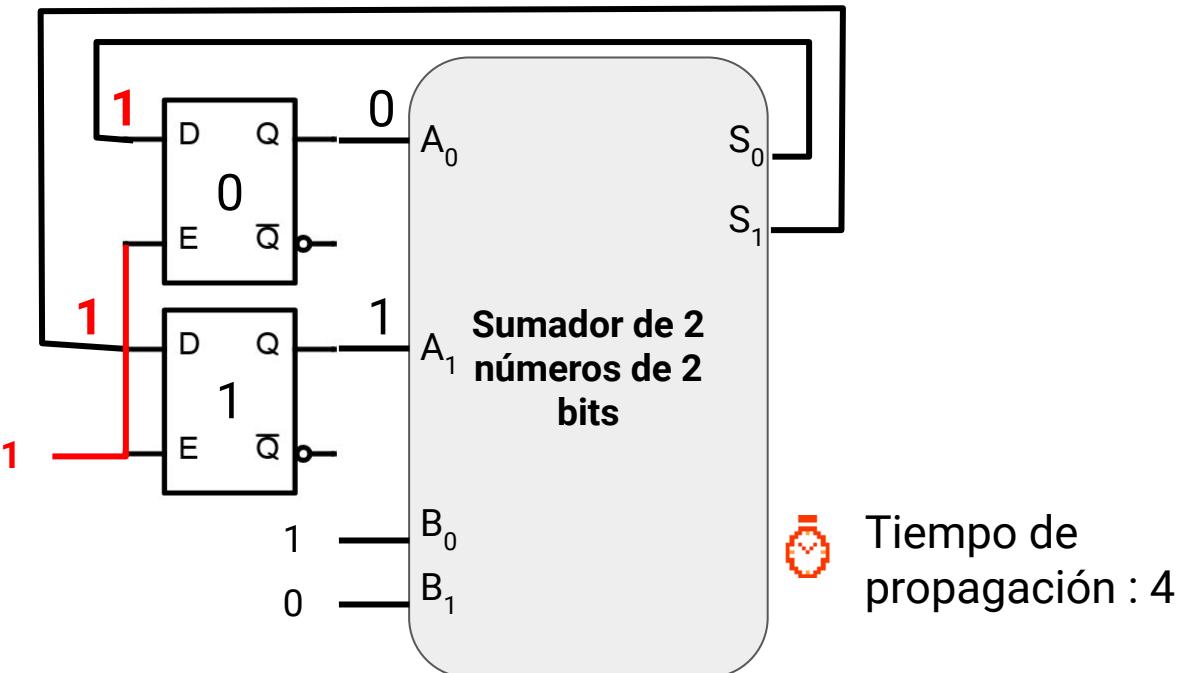
# Condición de carrera

Ahora los FF actualizan sus salidas y el sumador ve 10 en su entrada, por ende suma  $10 + 01$  y eso equivale a 11.



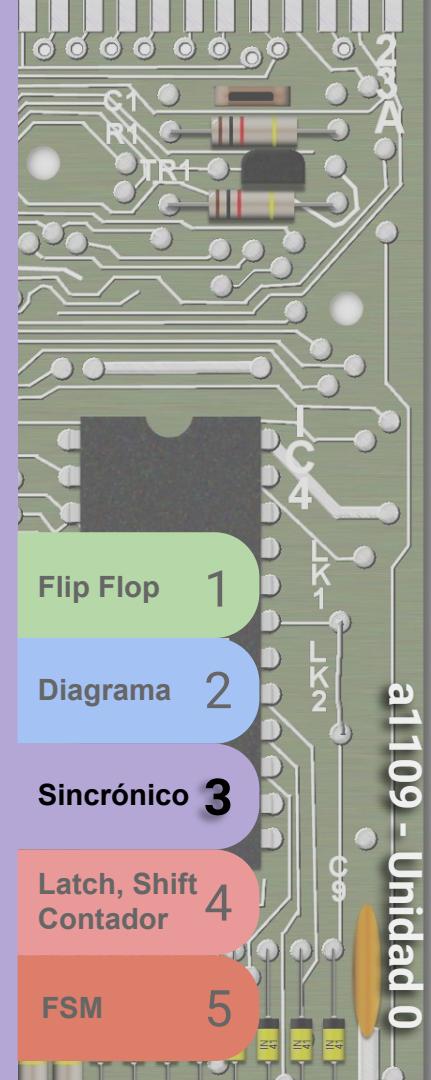
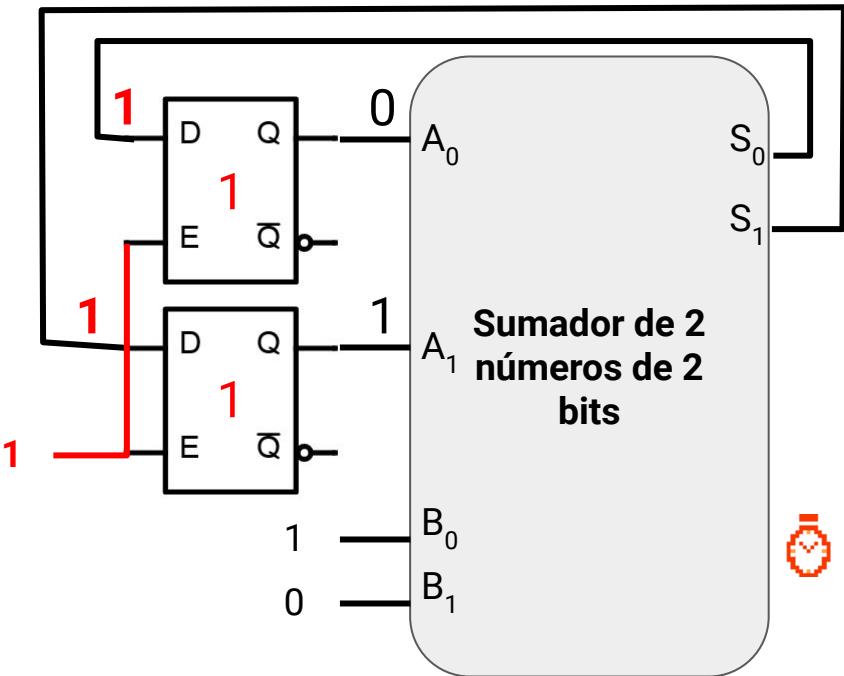
# Condición de carrera

Luego de 4 tiempos de propagación tenemos 11 en la entrada de los FF. Así que como E=1 continua actualizandolos...



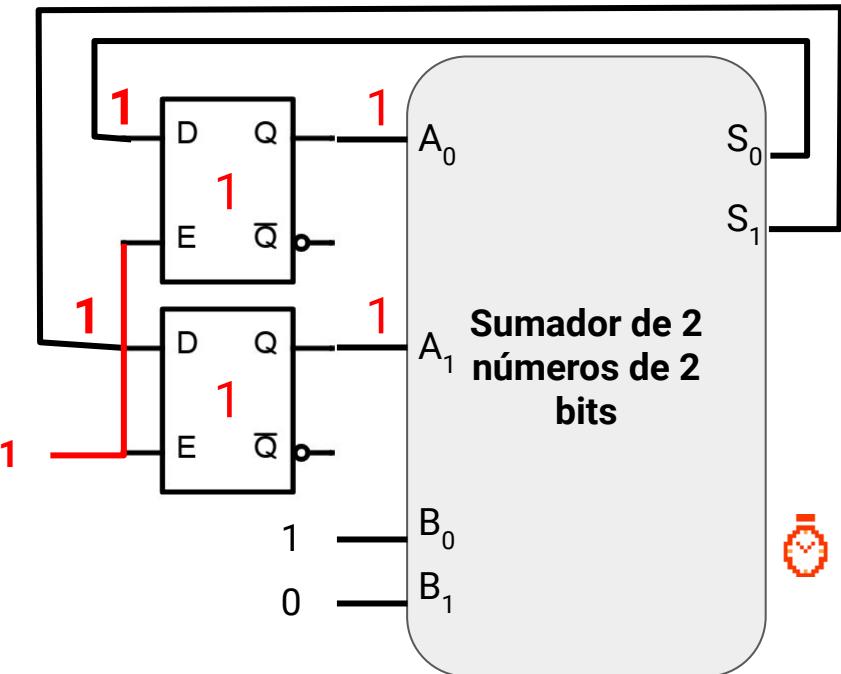
# Condición de carrera

En el quinto tiempo de propagación los FF actualizan su salida a 11.

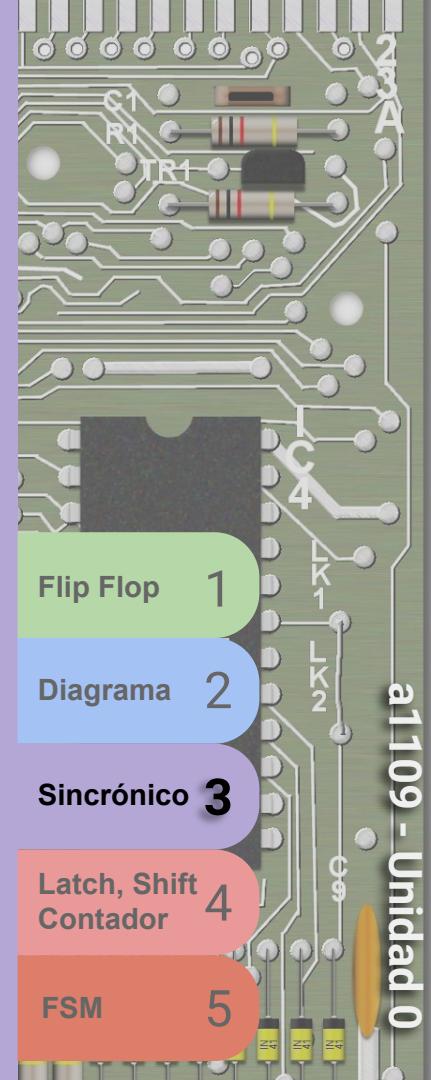


# Condición de carrera

Luego el sumador ve 11 en su entrada y realiza  $11+01$  lo que da 00 de vuelta (no hay carry en este sumador).

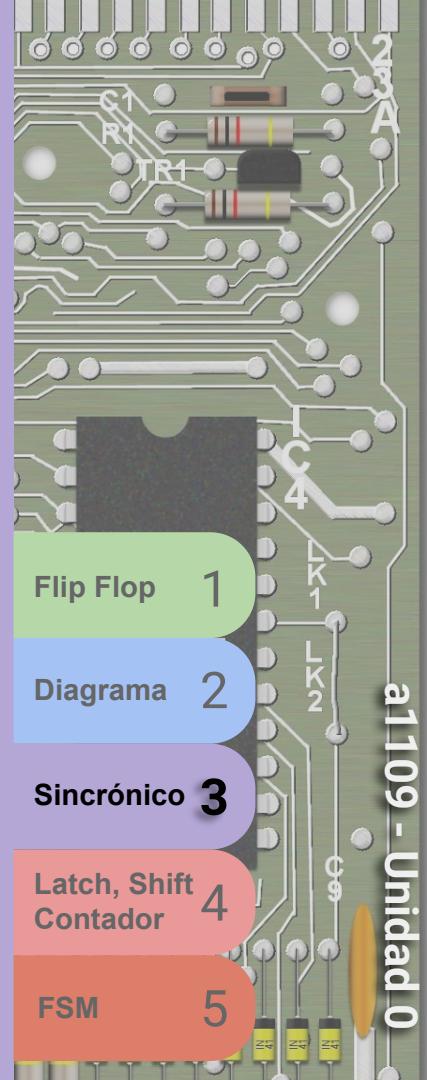
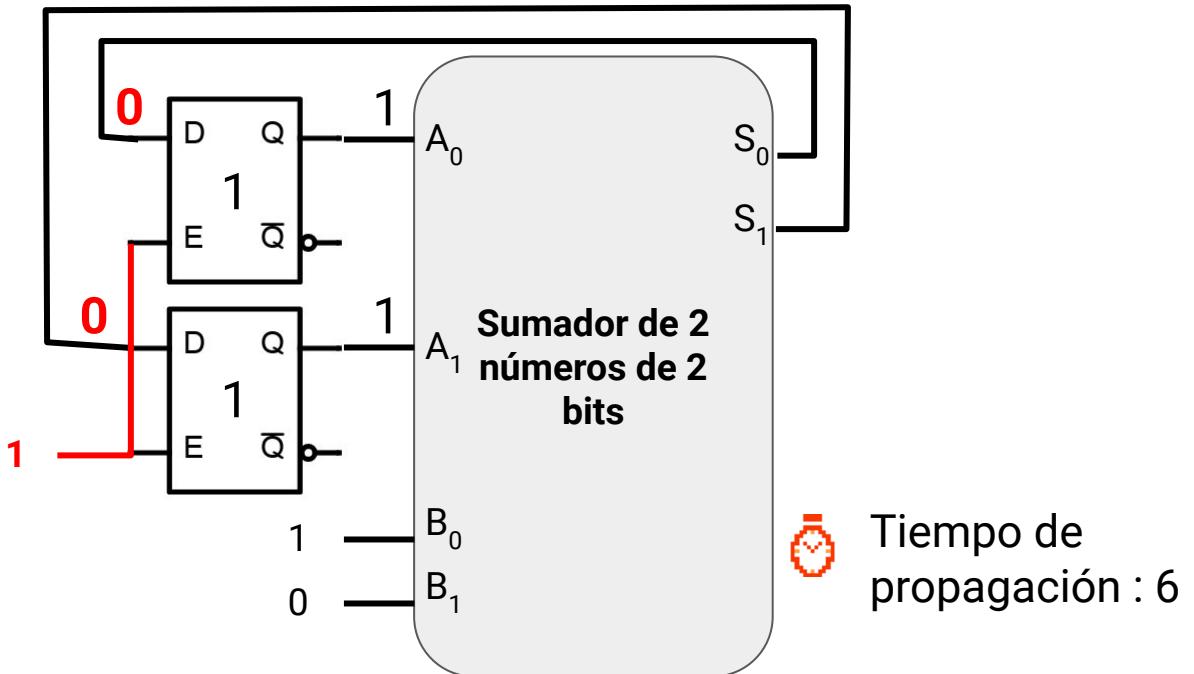


Tiempo de propagación : 5



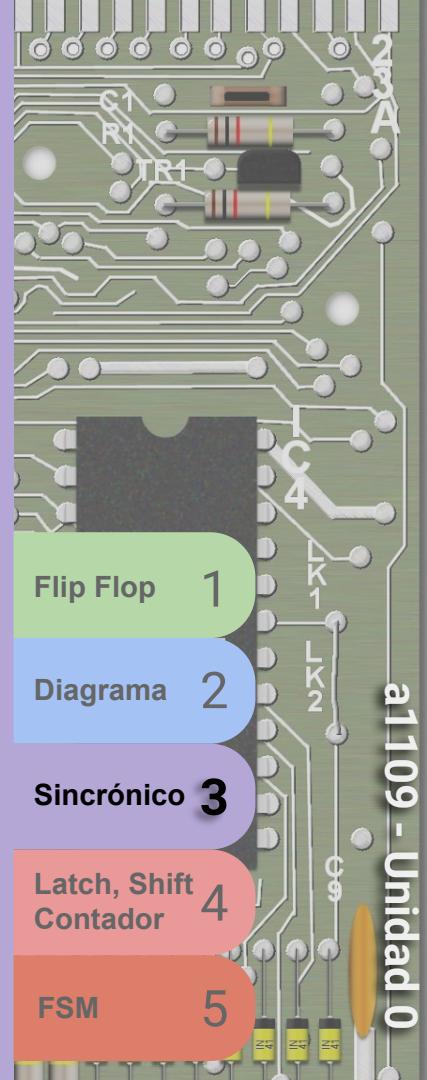
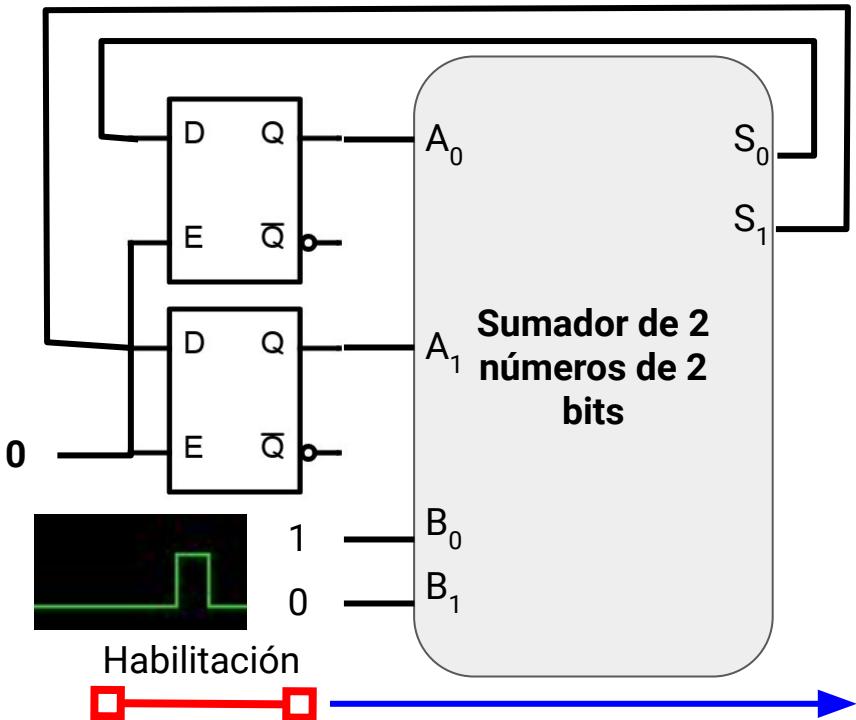
# Condición de carrera

Luego de 6 tiempos de propagación el contador dio toda la vuelta. Esto se debe a que E=1 durante 6 tiempos de propagación.. Y de mantenerse E=1 seguirá sumando sin fin.



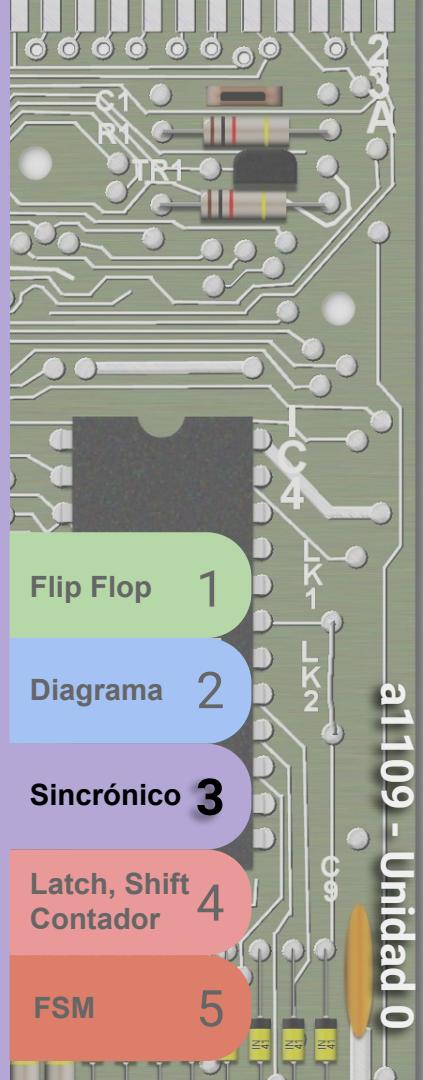
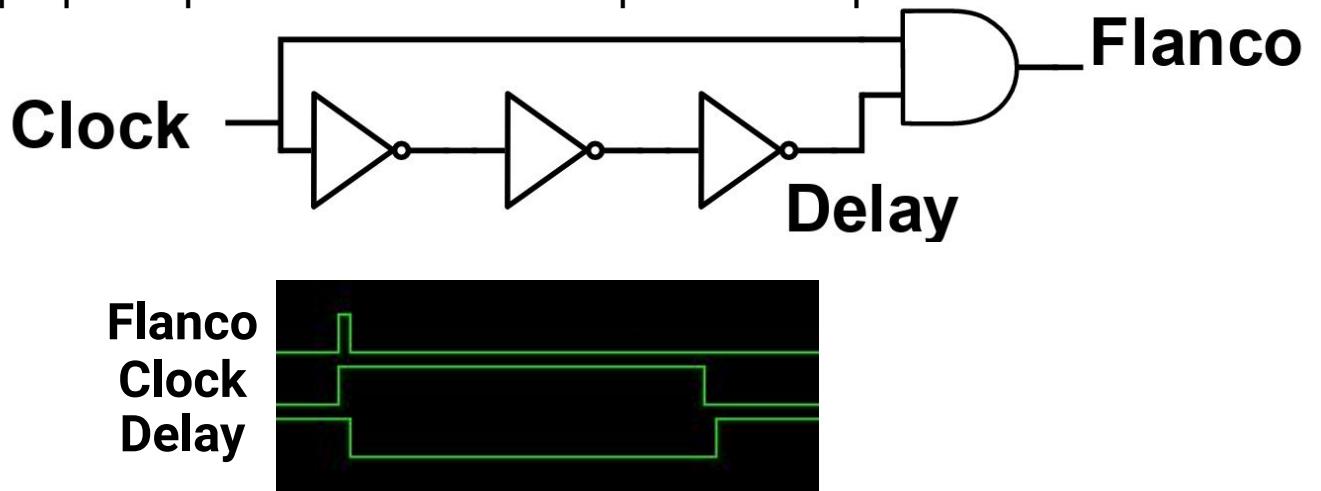
# Condición de carrera

Para evitar esto generamos una señal pulsante en E, o sea una señal con un tiempo en 1 muy pequeño pero suficiente para actualizar los FF.



# Detector de flancos

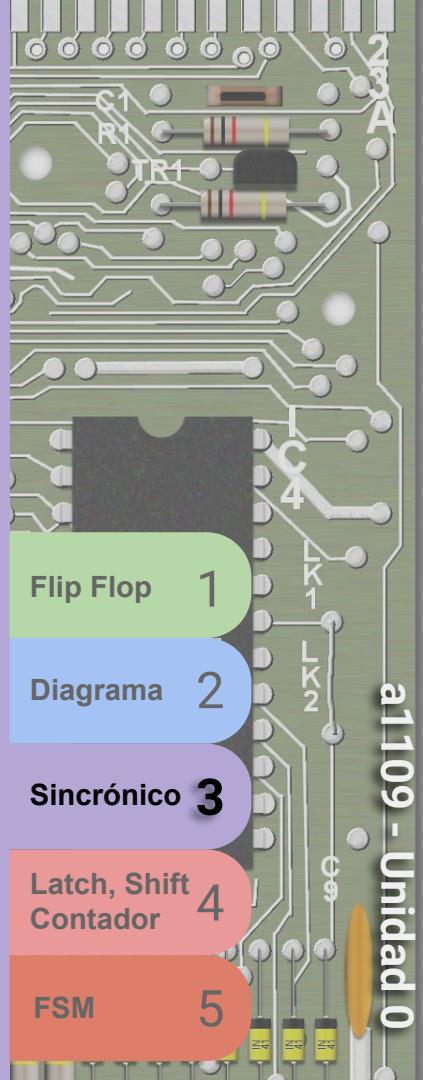
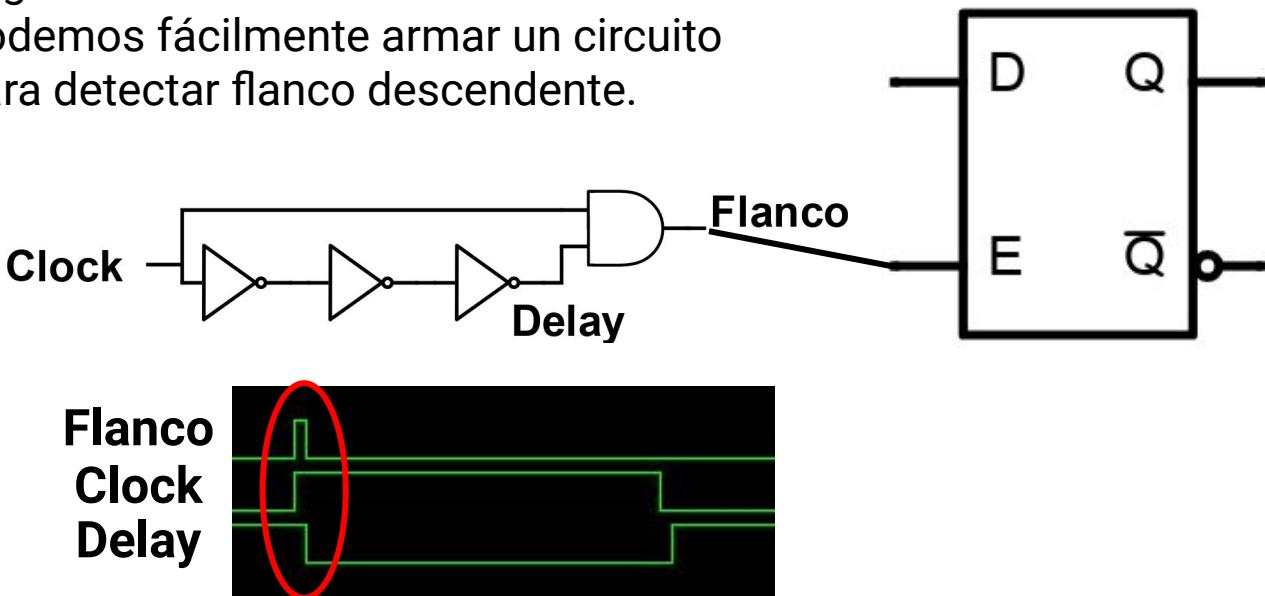
El siguiente circuito es básicamente una compuerta AND, cuyas entradas son Clock y Delay. Vemos que Clock = !Delay, pero la cadena de 3 negadores tarda 3 tiempos de propagación en hacer que Clock = !Delay. Supongamos que Clock=0, vemos que Delay=1 y Flanco = 1 AND 0 = 0. Pero si se produce un cambio de Clock de forma tal que pase de 0 a 1, vemos que por 3 tiempos de propagaciones Flanco = 1 AND 1 = 1. Esto es un tiempo muy pequeño pero suficiente como para ver un pulso.



# Detector de flancos

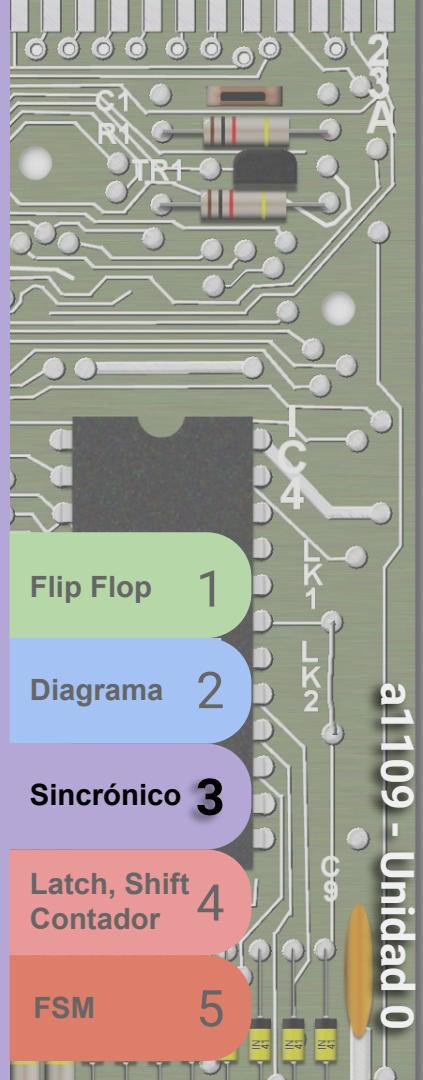
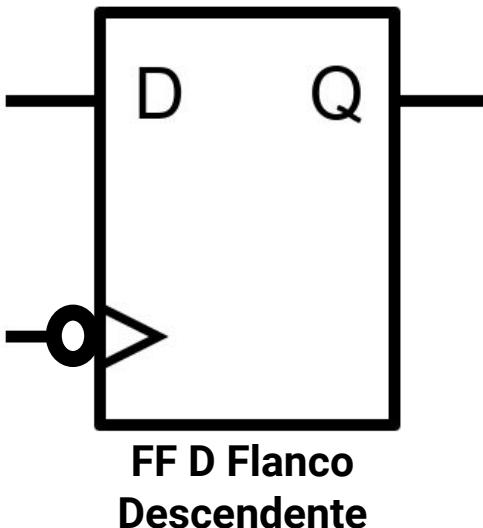
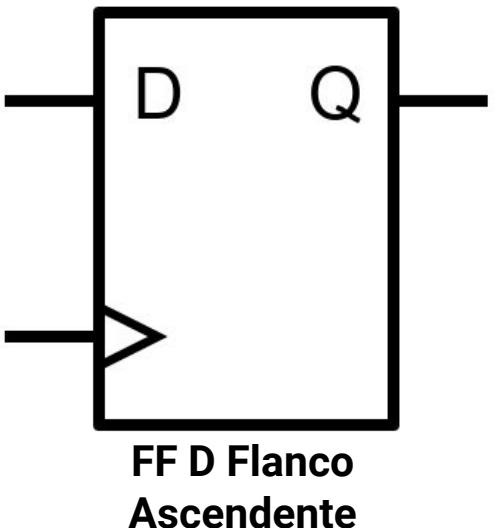
A este detector lo llamamos detector de flanco ascendente (Rising Edge) y ponemos conectarlo en la entrada E de un Flip Flop para lograr que solo funcione cuando la señal de Enable pasa de 0 a 1 (o sea hay un flanco ascendente). El tiempo de duración del pulso de flanco se puede ajustar agregando o quitando negadores de la cadena.

Podemos fácilmente armar un circuito para detectar flanco descendente.



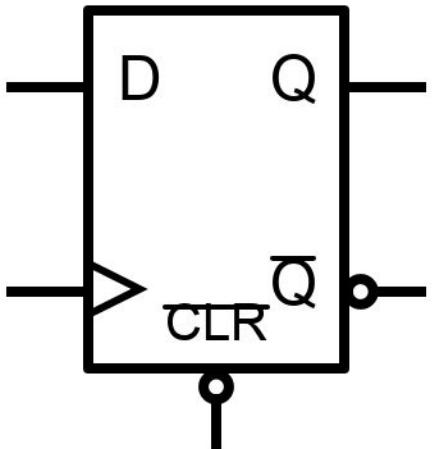
# Flip Flop D Sincrónico por flanco

Combinando el FF D con un detector de flancos (sea por flanco ascendente o flanco descendente) tenemos entonces un FF D sincrónico por flanco. La entrada de habilitación solemos llamarla entrada de clock y el dibujo es un triángulo. Si el mismo está negado indica flanco descendente. Podemos obviar la salida de Q negado si la misma no será utilizada en el circuito.



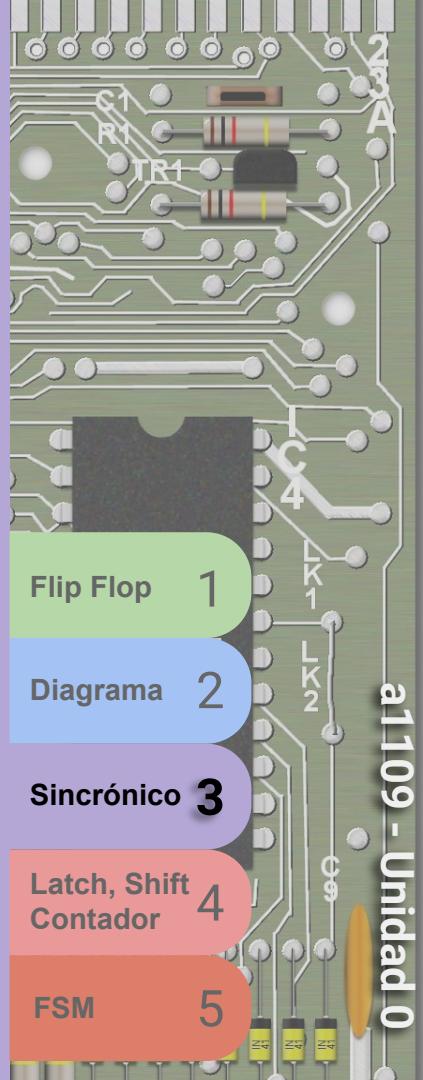
# Entradas forzadas

Cuando trabajamos con FF es conveniente saber con qué valor inician en el momento que se encienden. Veremos que las computadoras utilizar circuitos de reset que generan una señal común para que todo componente con memoria tome un valor predeterminado ante esa señal de reset. Vemos entonces un Flip Flop Síncrono por flanco ascendente con una nueva entrada llamada !CLR. Cuando esta entrada es 0, Q toma valor 0. Si esto ocurre sin importar el valor de clock, es asincrónico.



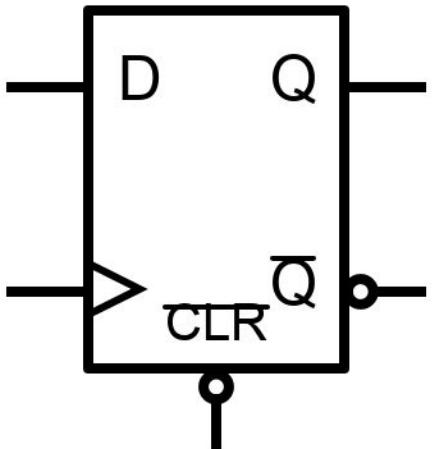
FF D Flanco  
Ascendente con  
clear asincrónico

<b>!CLR</b>	<b>D</b>	<b>Clk</b>	<b>Q<sub>N</sub></b>
1	X	0	Q <sub>N-1</sub>
1	0	↑	0
1	1	↑	1
0	X	X	0



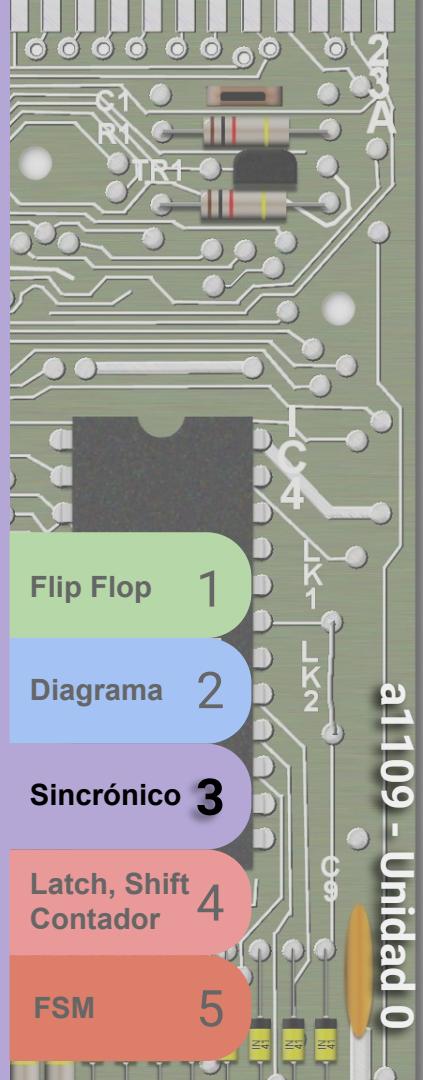
# Entradas forzadas

Si la señal de clear depende del flanco del clock para forzar el valor, entonces es un clear sincrónico. Dependiendo que tan rápido queremos forzar el valor elegimos el tipo que más convenga. Así como existen entradas que fuerzan un 0 también existen entradas que fuerzan un 1. Suelen conocerse como entradas de preset.



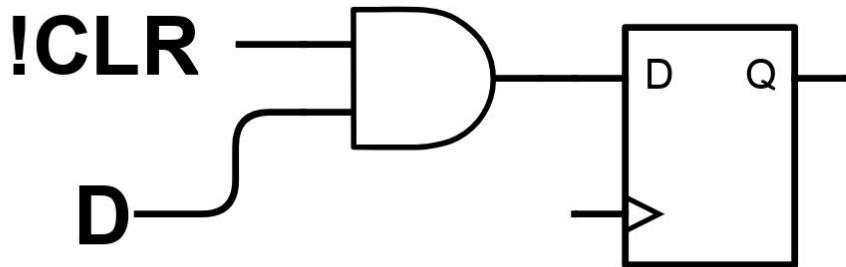
FF D Flanco  
Ascendente con  
clear sincrónico

$\neg \text{CLR}$	D	Clk	$Q_N$
1	X	0	$Q_{N-1}$
1	0	$\uparrow$	0
1	1	$\uparrow$	1
0	X	$\uparrow$	0



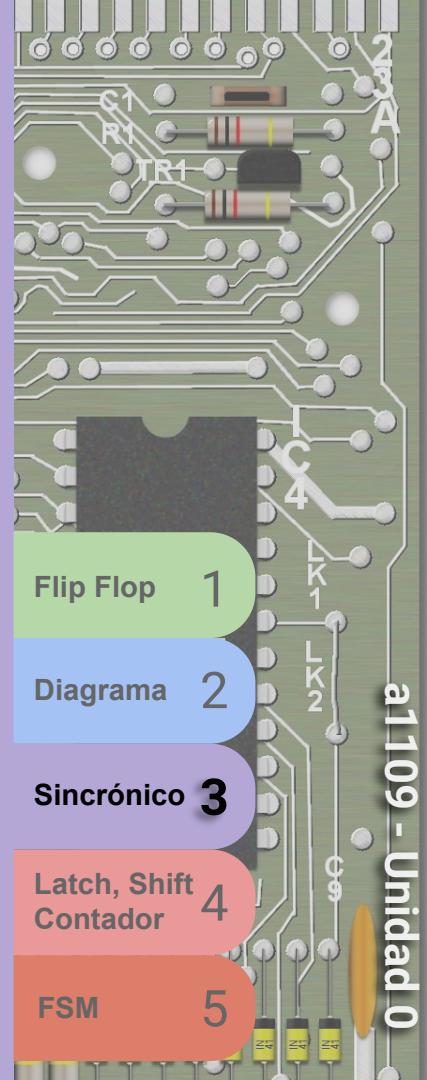
## Entradas forzadas

Vemos que es sumamente sencillo convertir un FF D sin entradas forzadas en un FF D con clear sincrónico. Cuando  $\text{!CLR}=1$ , el valor que toma el FF es la entrada D. Cuando  $\text{!CLR}=0$ , la AND siempre tendrá como salida 0, por ende  $D=0$ ... pero esto ocurre cuando se produce un flanco de clock.



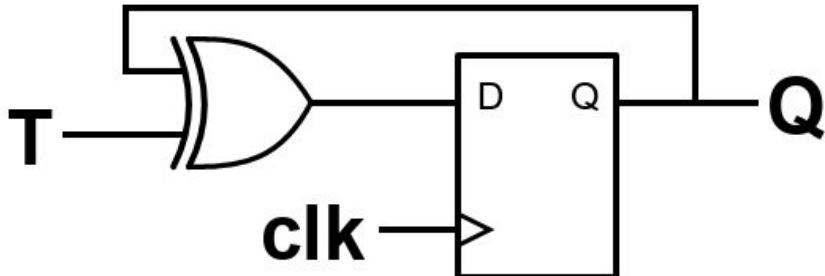
FF D Flanco  
Ascendente con  
clear sincrónico

$\text{!CLR}$	D	Clk	$Q_N$
1	X	0	$Q_{N-1}$
1	0	$\uparrow$	0
1	1	$\uparrow$	1
0	X	$\uparrow$	0

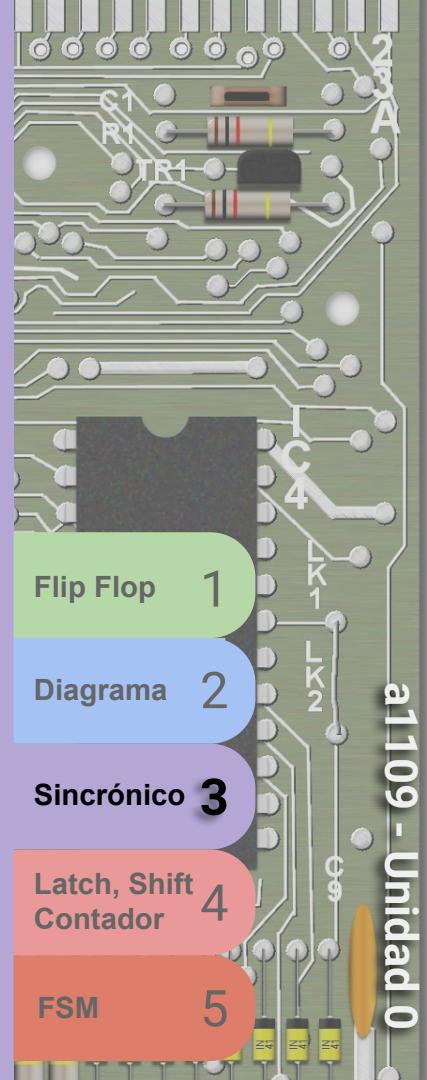


# Flip Flop T

Veamos ahora un nuevo tipo de FF. Si bien vimos que con el RS y un negador podemos construir un FF tipo D, con el FF tipo D y una compuerta XOR podemos construir un FF tipo T. La T viene del inglés toggle (invertir). Este FF posee una entrada T, una entrada de clock (flanco ascendente o descendente según sea el caso) y una salida Q (podemos incluir  $\bar{Q}$  si hace falta). Vemos la tabla de verdad reducida. Cuando  $T=0$ , con cada flanco ascendente la salida Q mantiene su valor. Cuando  $T=1$ , la salida Q toma la inversa del valor anterior. Este FF será útil cuando veamos contadores binarios.

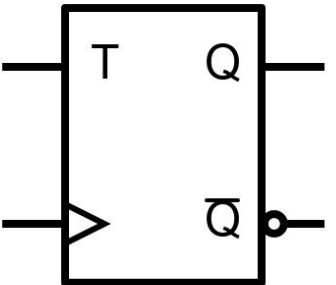
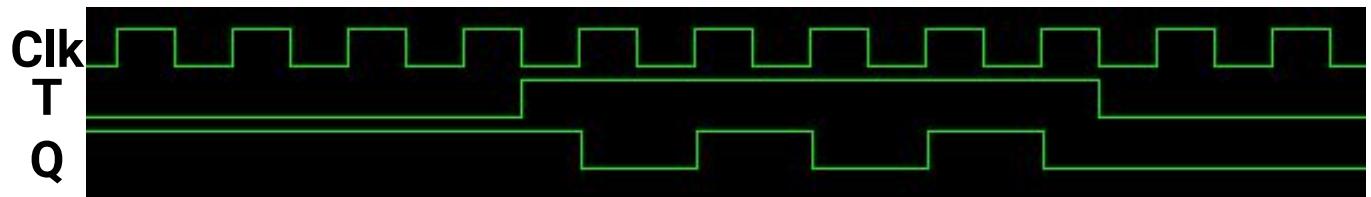


T	Clk	$Q_N$
0	↑	$Q_{N-1}$
1	↑	$\bar{Q}_{N-1}$

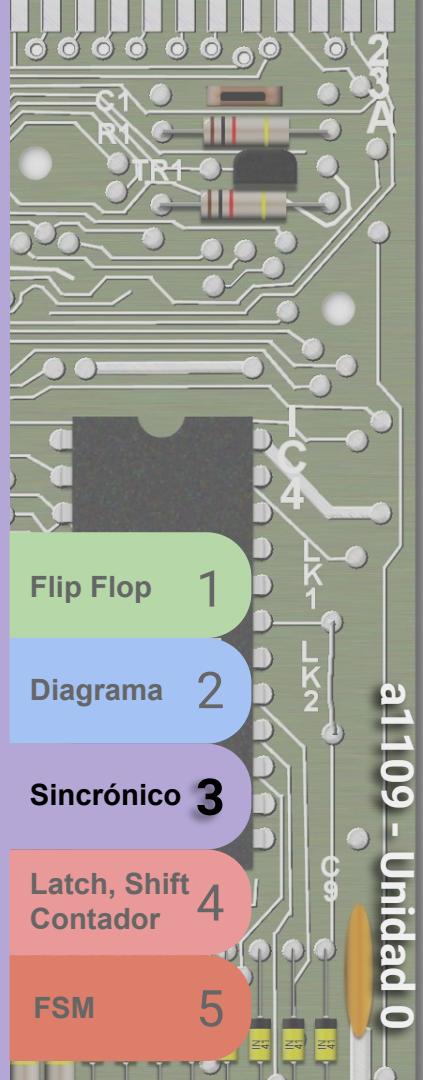


# Flip Flop T

Vemos que  $Q=1$  en este diagrama de tiempo. Comienza con  $T=0$ , por ende cada flanco de clock ascendente mantiene  $Q=1$ . Luego cuando  $T=1$  notemos que con cada flanco ascendente de clock el valor de  $Q$  se invierte. Si prestan atención verán que si dejamos  $T$  en 1 la salida  $Q$  es equivalente a Clock dividido 2, o sea cada dos pulsos de clock tenemos un “pulso” en  $Q$ . Esta característica podrá ser explotada cuando construyamos contadores.



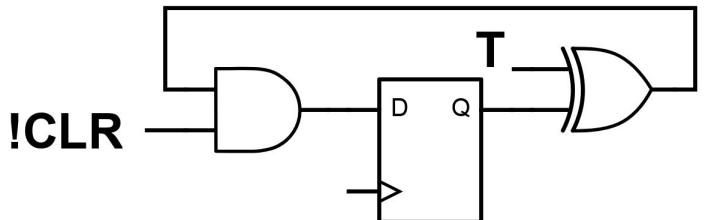
T	Clk	$Q_N$
0	↑	$Q_{N-1}$
1	↑	$\bar{Q}_{N-1}$



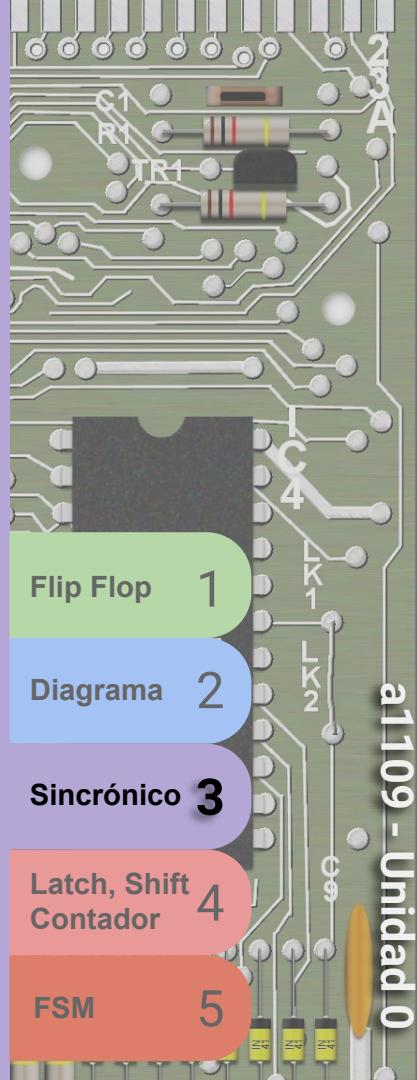
## Flip Flop T

Vemos que podemos usar el mismo método con la compuerta AND para agregar una entrada forzada sincrónica que fuerce el FF T a tomar 0 cuando se produce un flanco de clock.

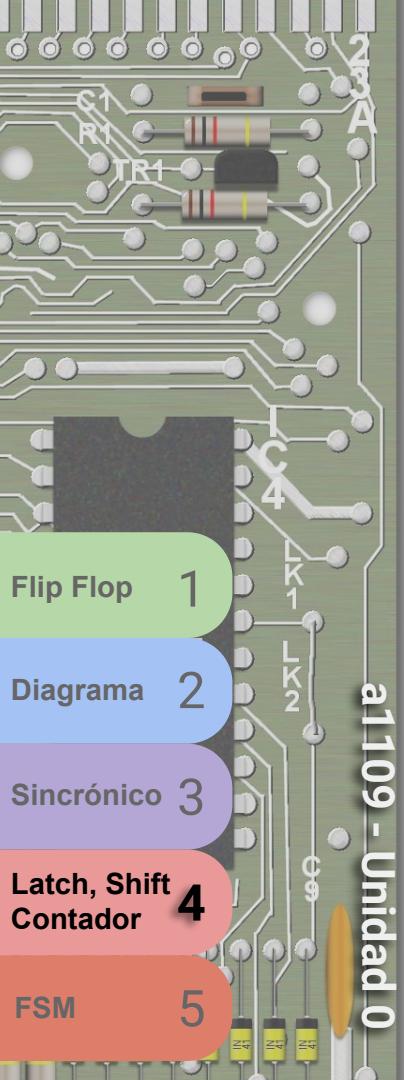
Si bien puede parecer molesto tener que controlar una entrada mas que quizás no utilicemos nunca, es común en electronica definir un valor como predeterminado para una entrada no conectada o usada. Es por esto que utilizamos CLR=0 para forzar el cero. Si dejamos !CLR sin conectar este tomará automáticamente uno. Esto se aclara en la hoja de datos.



!CLR    T    Clk    Q <sub>N</sub>			
1	0	↑	Q <sub>N-1</sub>
1	1	↑	!Q <sub>N-1</sub>
0	X	↑	0

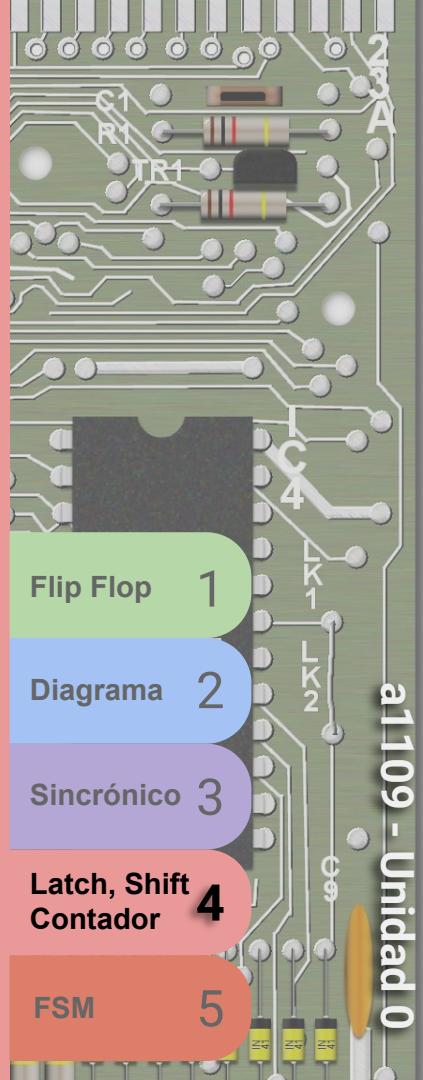
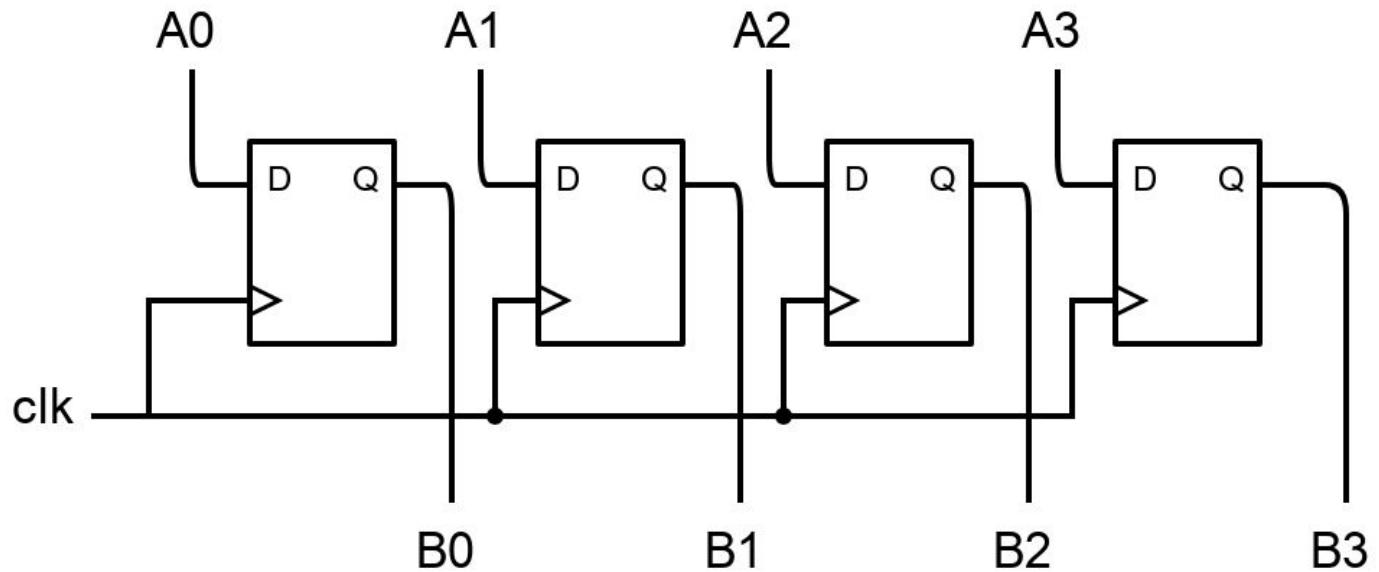


# Registro Latch



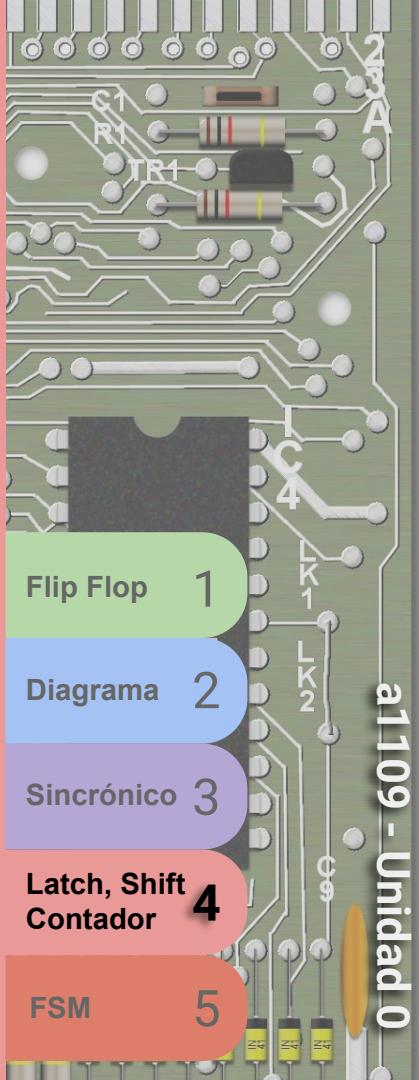
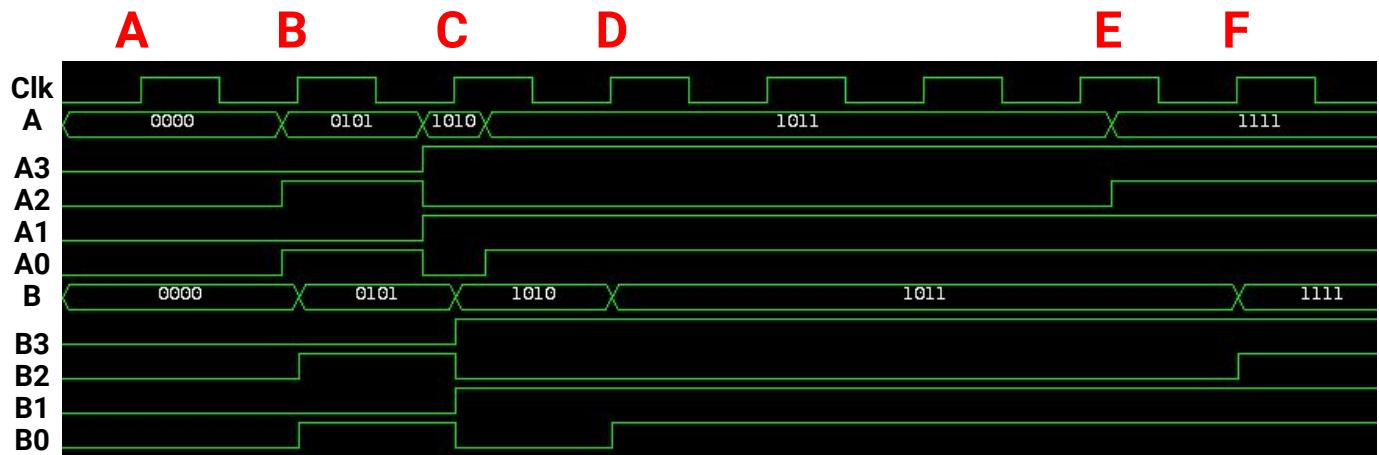
## Latch 4 bits

Este es un registro latch de 4 bits. Como vemos, no es más que 4 FF D conectados a la misma señal de clock. El clock es por flanco ascendente. Podemos asumir que A0A1A2A3 son un número A de 4 bits. Mismo para las salidas B0B1B2B3 son un número B también de 4 bits. Cuando hay un flanco ascendente en clock el valor de B es igual al valor de A. Básicamente almacena 4 bits.



## Latch 4 bits

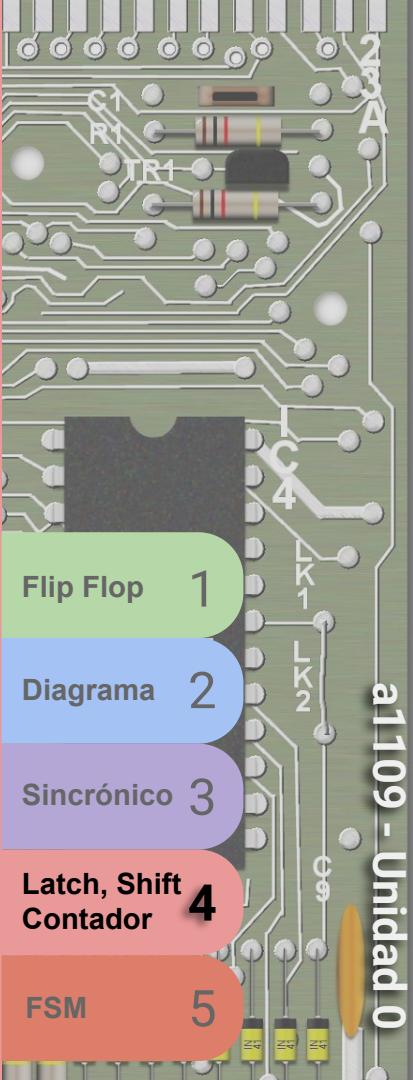
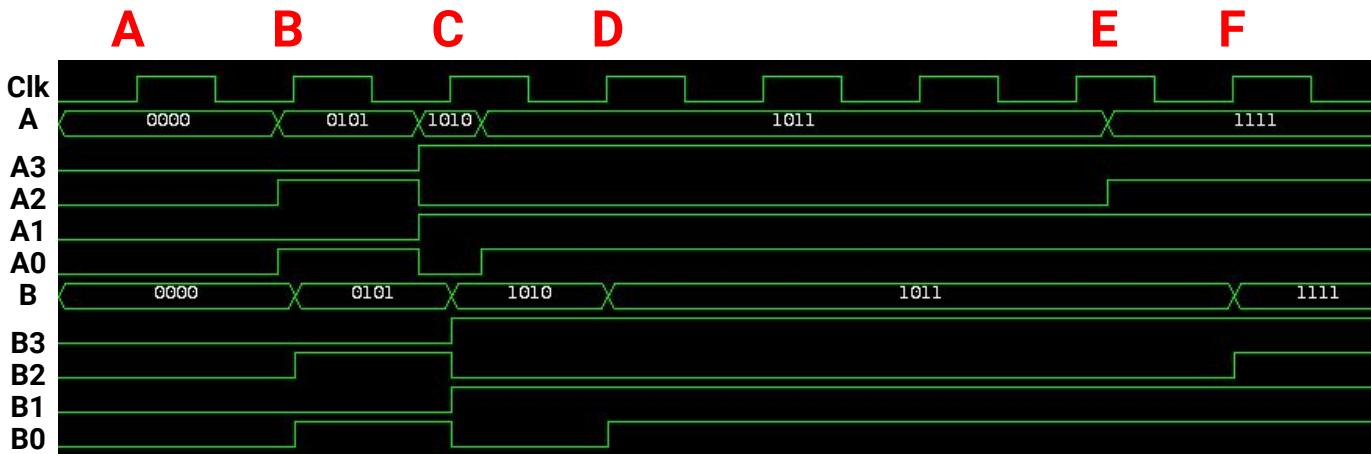
Vemos que este diagrama de tiempo tiene algo nuevo. Vemos que aparte de representar A<sub>3</sub>,A<sub>2</sub>,A<sub>1</sub> y A<sub>0</sub> (como bits que varían entre 0 y 1) vemos también un grupo llamado A. Este grupo es el conjunto de todas las señales A<sub>N</sub> como un número binario. Vemos que es más simple tratarlo todo como un número. Lo mismo con B. Vemos que todos los cambios en B se producen ÚNICAMENTE en los flancos ascendentes de clock.



## Latch 4 bits

A- Vemos que en el flanco ascendente nada parece cambiar, pero esto se debe a que el valor de A=0000 y B=0000 previo al flanco... no hay cambios porque se mantiene el mismo valor.

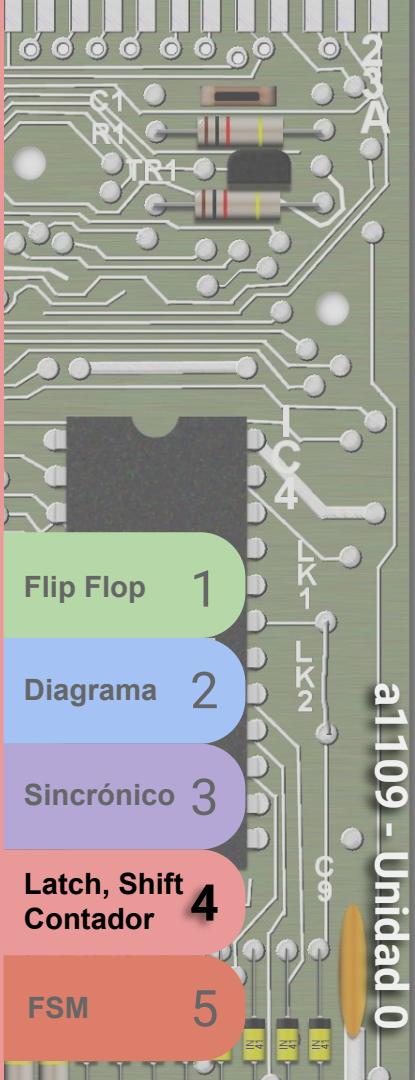
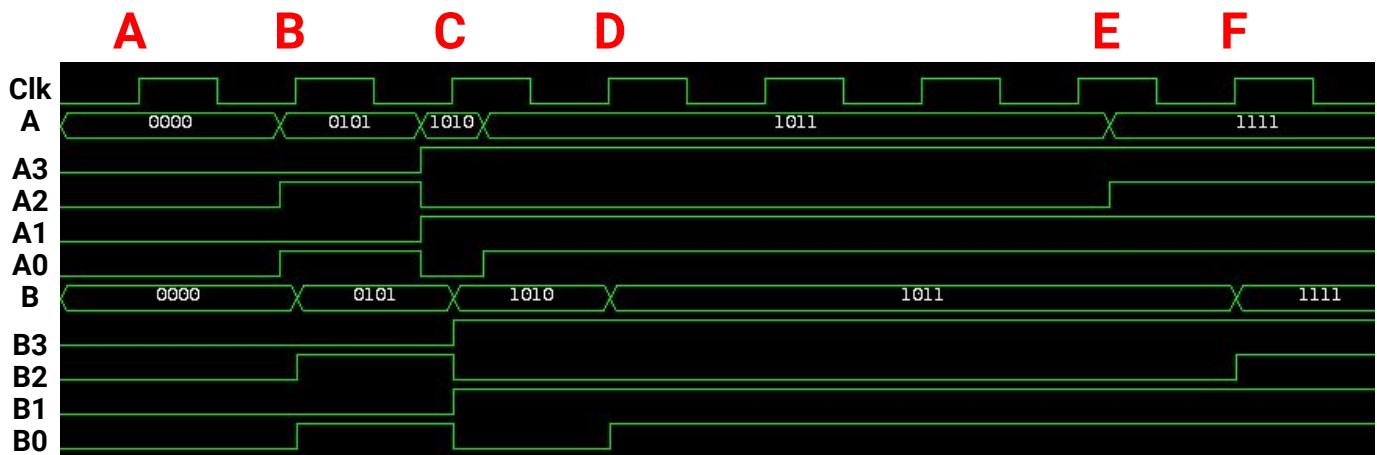
B- Vemos que unos instantes antes del flanco ascendente el valor de A pasa a ser 0101. Notemos que justo en el flanco ascendente el valor de B copia la entrada A y vale 0101. El valor de B solo se actualiza cuando se produce el flanco, no cuando cambia A.



## Latch 4 bits

C- Vemos que en este caso nuevamente A cambia un poco antes del flanco ascendente a 1010 y B cambia a 1010 exactamente con el flanco ascendente de clock. Notemos que antes de que termine el ciclo positivo de clock el valor de A pasa a ser 1011. Sin embargo el valor de B se mantiene en 1010 que era el valor que había en A cuando se produjo el flanco ascendente.

D- Aquí efectivamente B toma 1011.

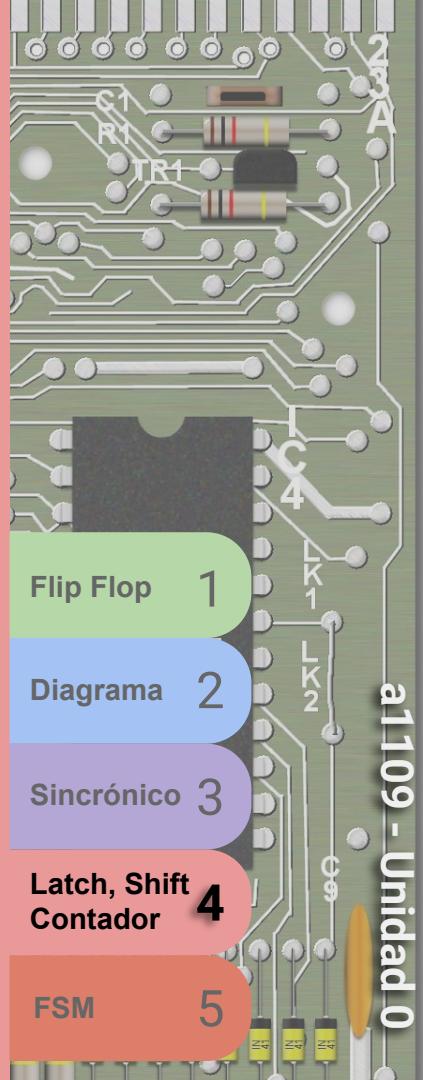
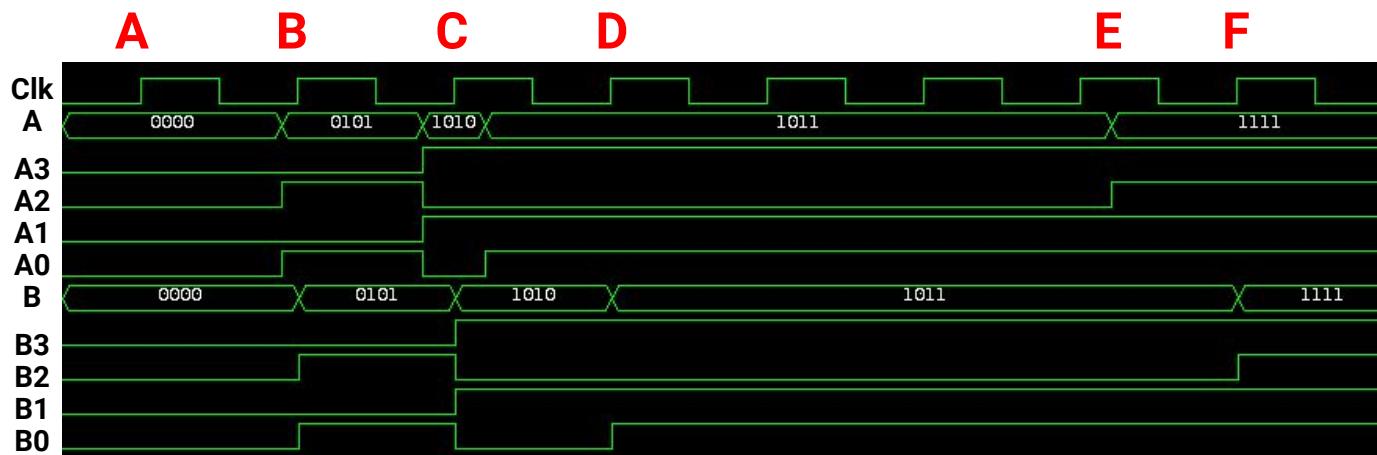


## Latch 4 bits

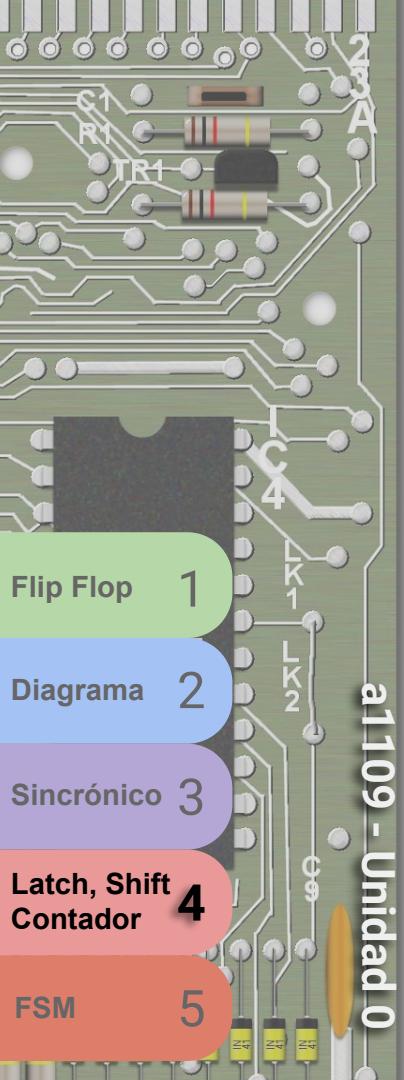
E- Se produce un nuevo cambio en A al valor 1111. Notemos que esto no tiene efecto inmediato en B.

F- Aquí en el flanco ascendente de clock el valor 1111 pasa a B.

Este circuito permite controlar el momento en el que se almacena un valor (que se compone de varios bits). Los registros de las computadoras suelen ser registros tipo Latch (8,16,32 bits o mayores)

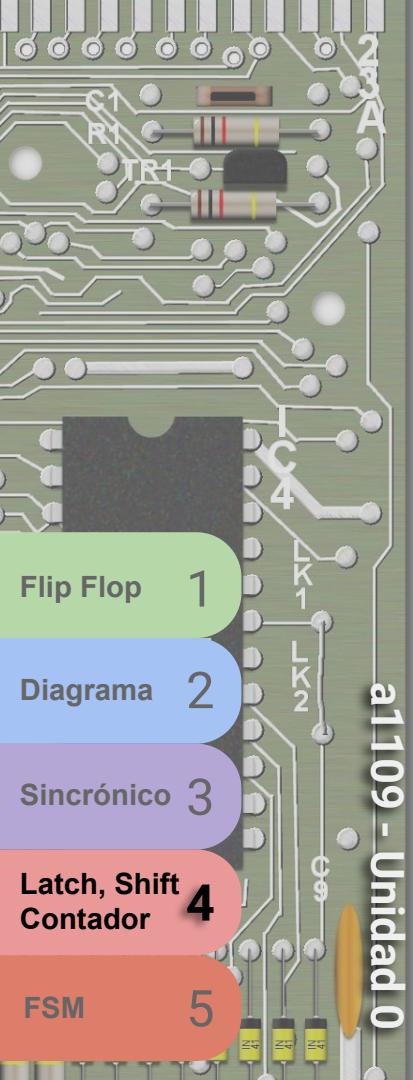
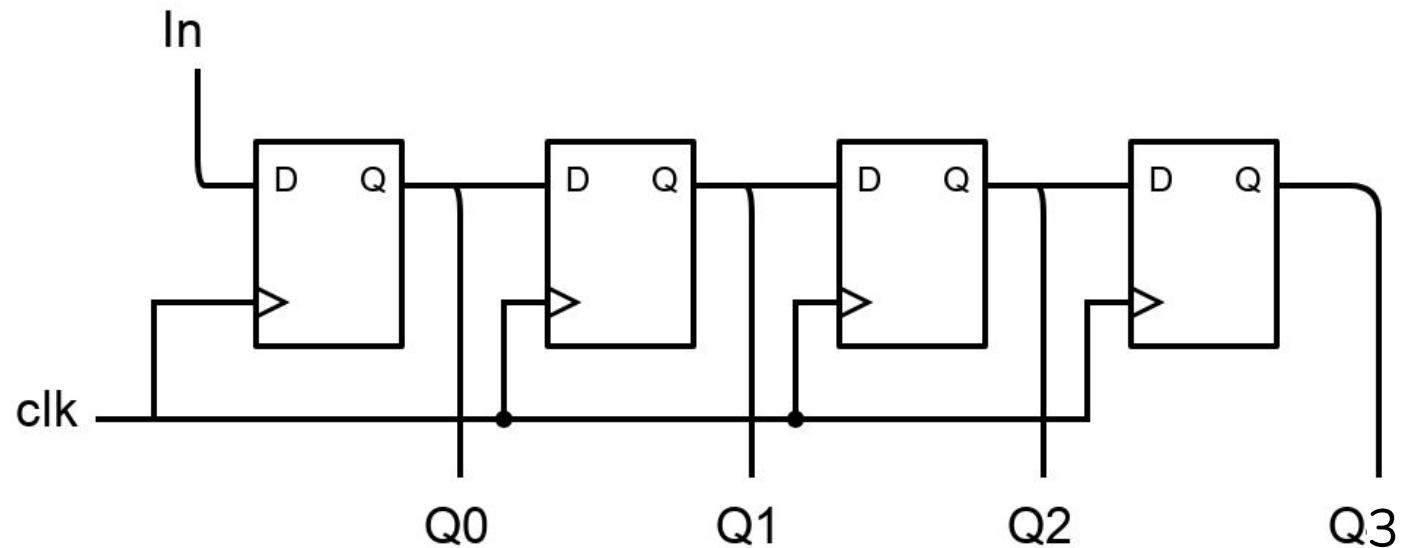


# Registro Shift



# Shift Right 4 bits

En este caso vemos un registro tipo shift (desplaza) con entrada serie (la entrada IN), entrada de clock y posee 4 salidas en paralelo. Vemos que la salida del FF D de la izquierda se encuentra conectada a la entrada D del siguiente FF (y en paralelo a Q0). Este circuito no permite ingresar datos por las salidas Q<sub>N</sub>. Existen versiones más simples que omiten las salidas Q excepto la última, siendo salida serie.



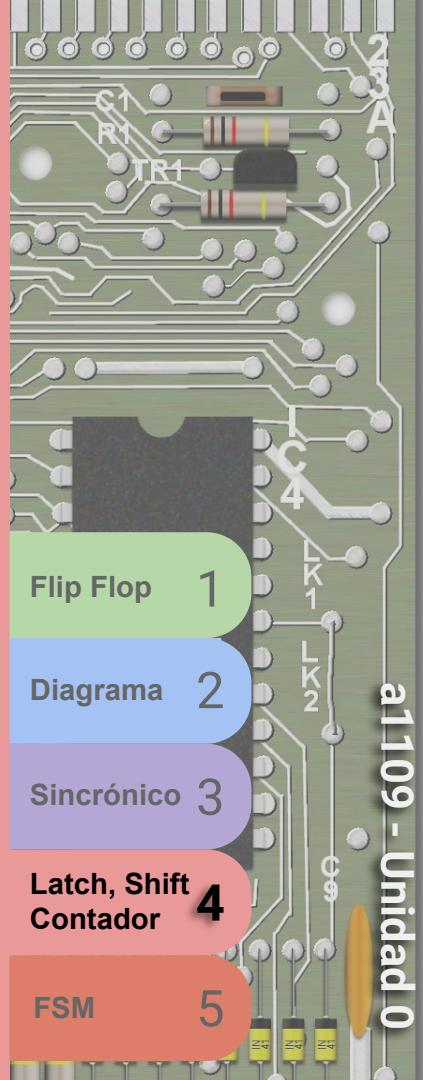
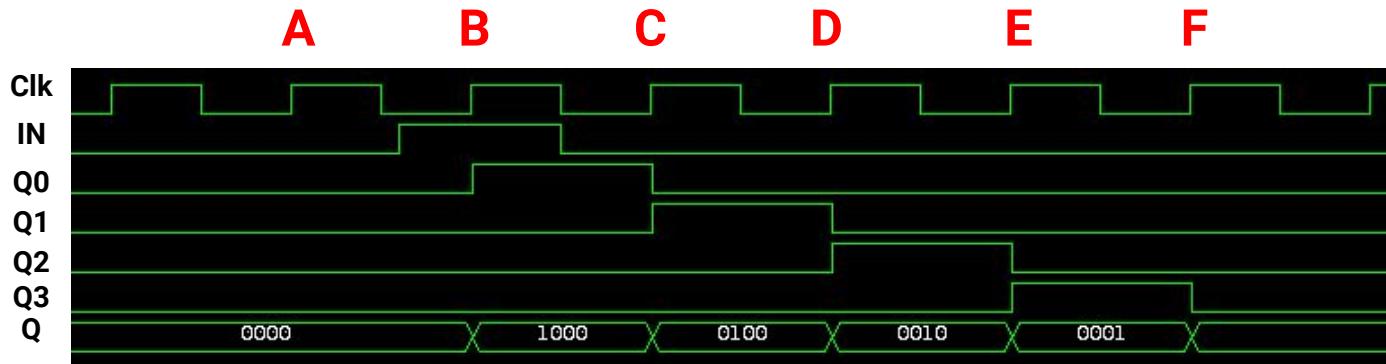
## Shift Right 4 bits

Vemos en el siguiente diagrama de tiempos que existe una señal clock que actualiza todos los flip flops.

A- En este flanco (y el anterior) el valor de IN=0, dado que todos los FF estan en 0, no se producen cambios.

B- Vemos que antes del flanco en B el valor de IN=1, por ende en este flanco el primer FF de la cadena captura este 1. Vemos que poco después el valor de IN=0, sin embargo el primer FF ya tiene capturado el 1.

C- Vemos que el primer FF pasa a 0 (ya que IN=0), pero mientras hace esto su salida Q0=1 por ende Q1 captura este 1. Vemos entonces que el la salida Q=0100.



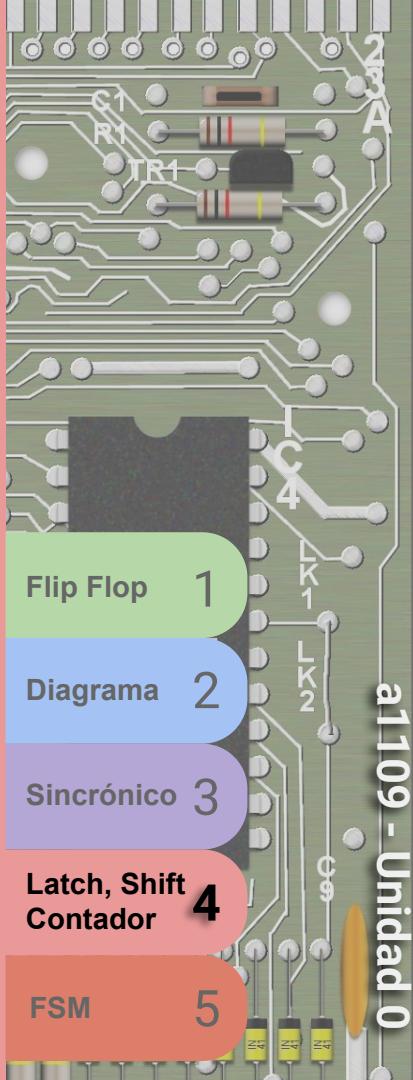
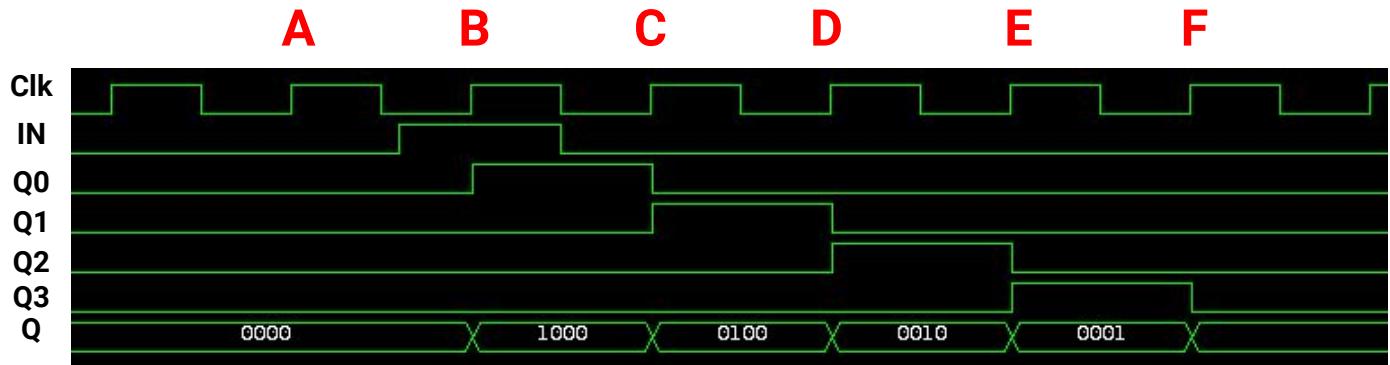
## Shift Right 4 bits

D- Pasa algo similar al punto C. En este caso Q2 todavía ve un 1 proveniente de Q1, el cual pasa a 0. Se ve como este 1 (que se capturó de IN en el instante B) va pasando por la cadena de FF.

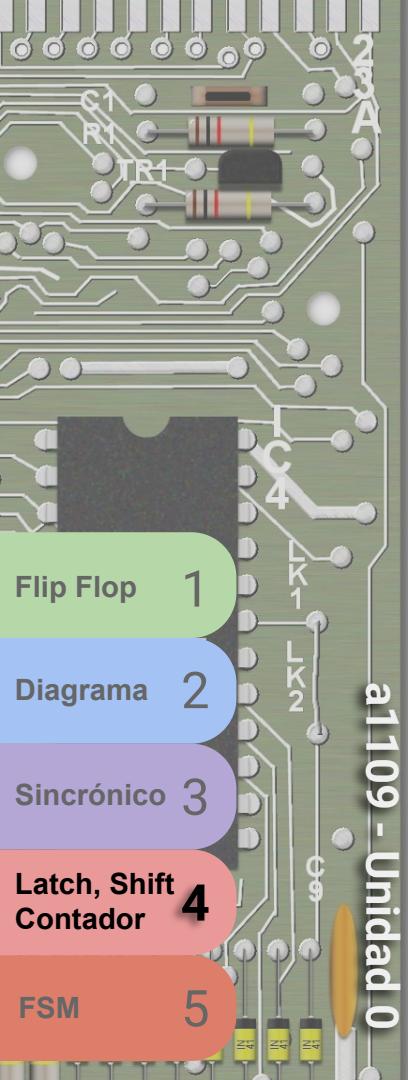
E- Aquí tenemos la última etapa donde el 1 se almacena. Vemos que Q toma 0001.

F- En este flanco todos los FF vuelven a 0.

Vemos que este circuito permite convertir un valor de entrada serializado (bit a bit) y luego de tantos clocks como bits se puedan almacenar vamos a tener el valor en paralelo en la salida B.

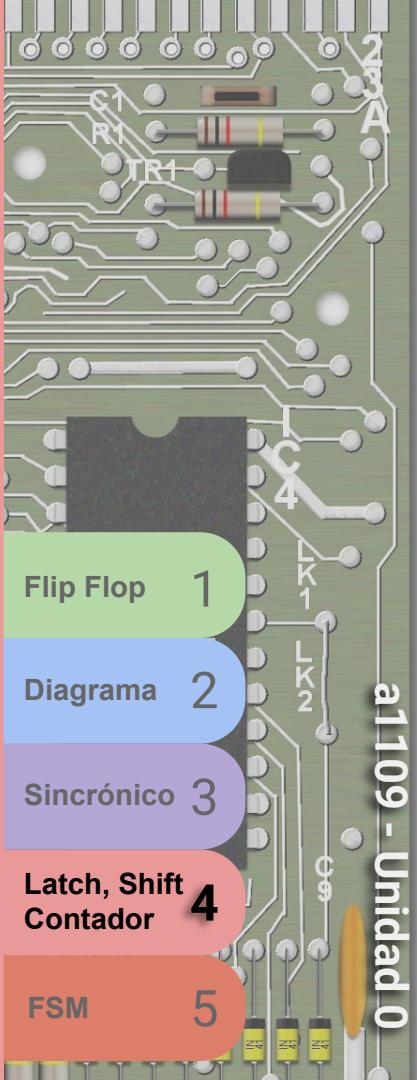
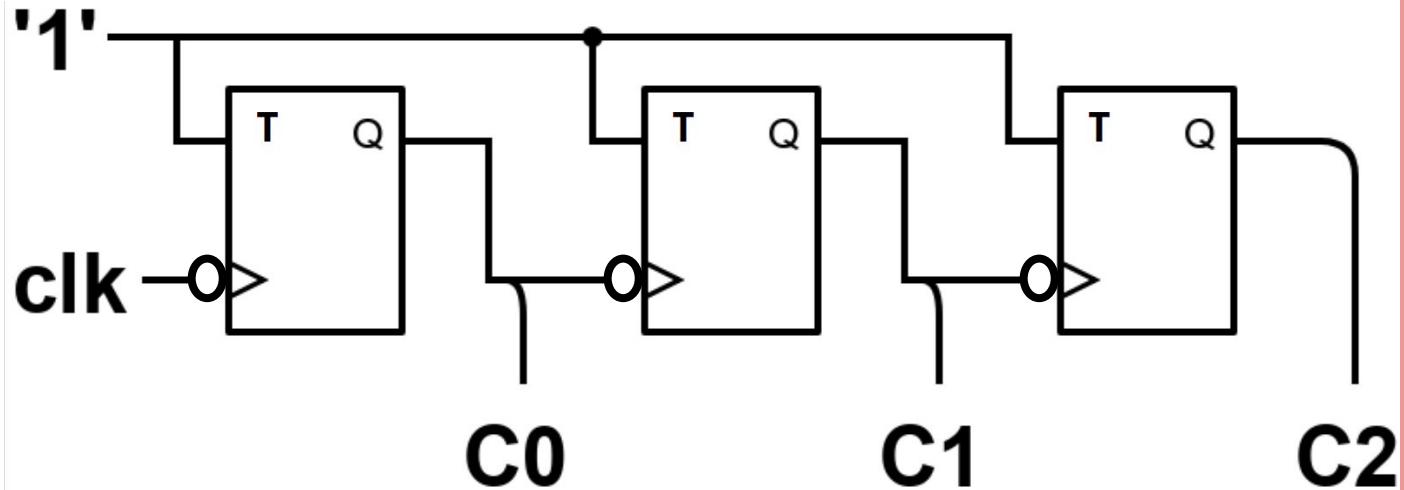


# Contador Asincrónico



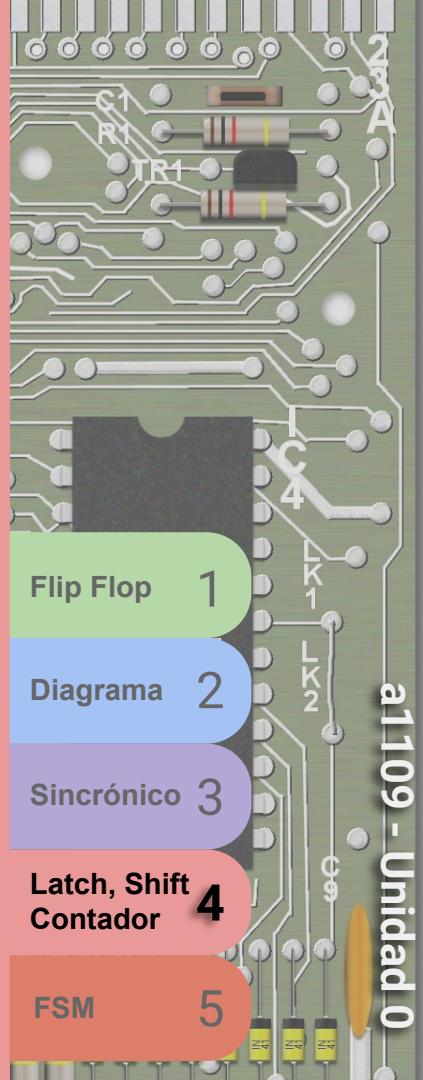
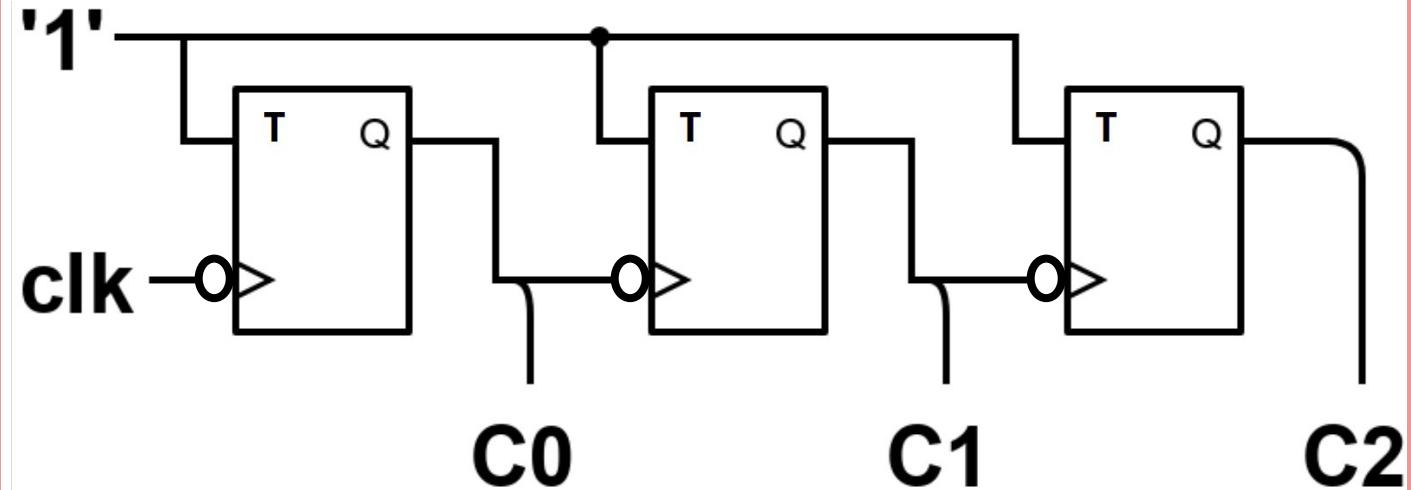
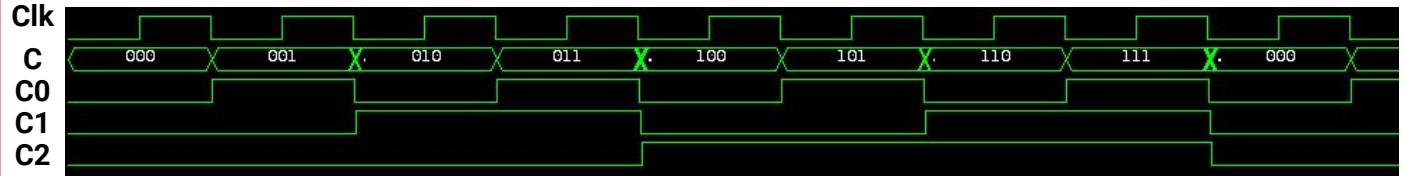
# Contador asincrónico de 3 bits

Vemos ahora un conjunto de FF tipo T (recuerden que este FF invierte la salida Q cuando  $T=1$  y se recibe un flanco de clock, en este caso flanco descendente). Vemos que Clk solo se conecta al primer FF T, luego la salida Q de este FF T pasa a Clk del segundo FF T (y en paralelo a C0). Algo similar pasa con C1 que está conectado a la entrada de clock del tercer FF T. Notemos que todas las entradas T de los FF están conectadas a un 1 fijo.



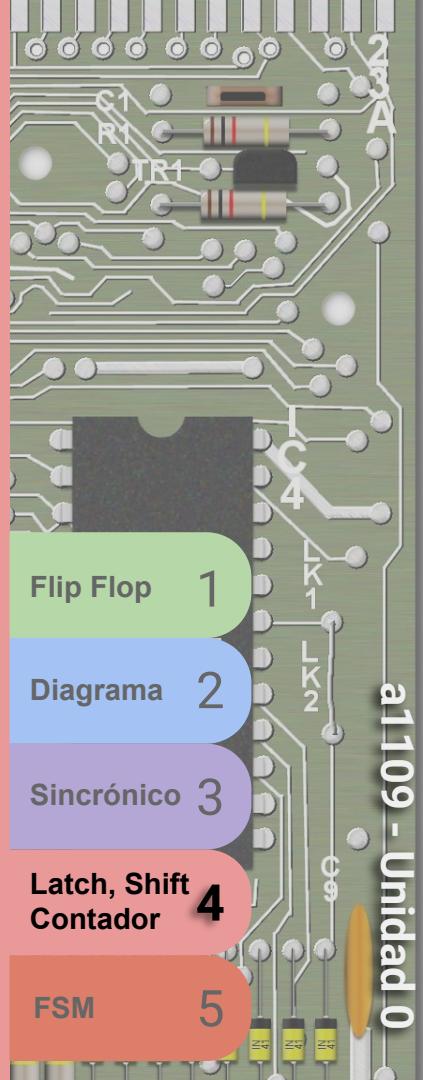
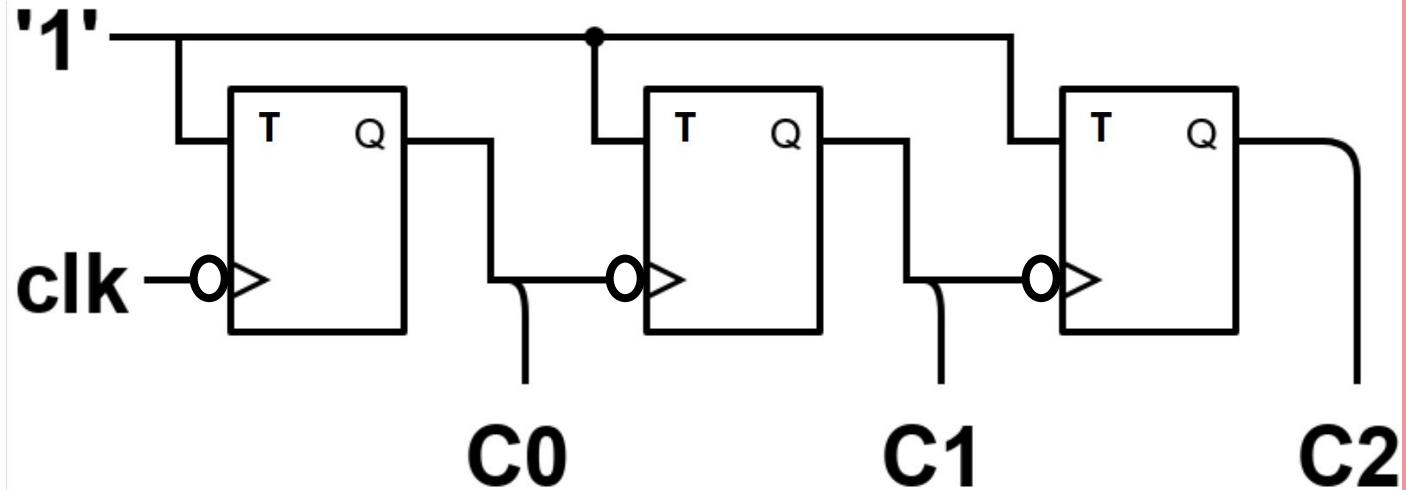
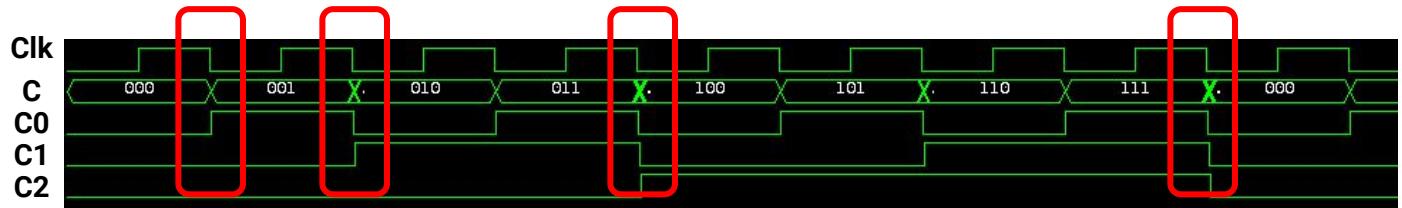
# Contador asincrónico de 3 bits

Vemos que cada flip flop alimenta el clock del FF siguiente, y cada etapa divide por 2 la frecuencia de la etapa anterior. Notemos que se necesita flanco descendente sino cuenta a la inversa.



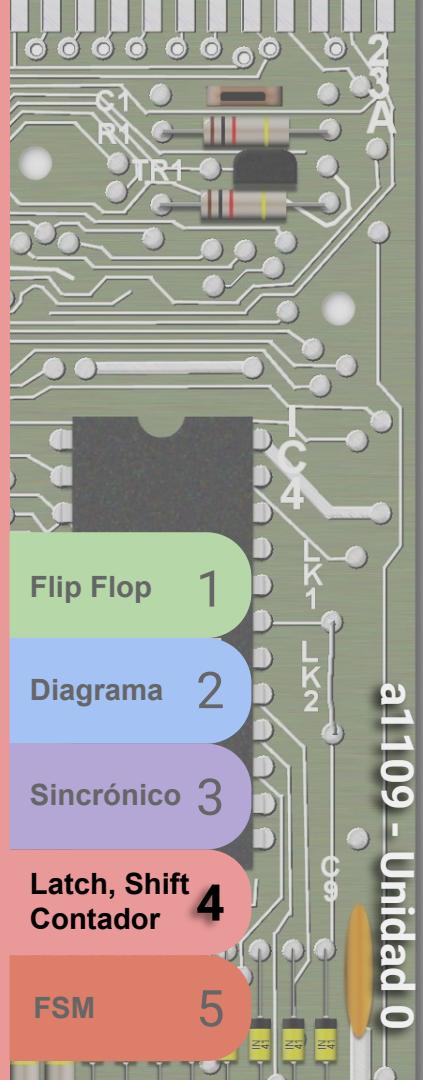
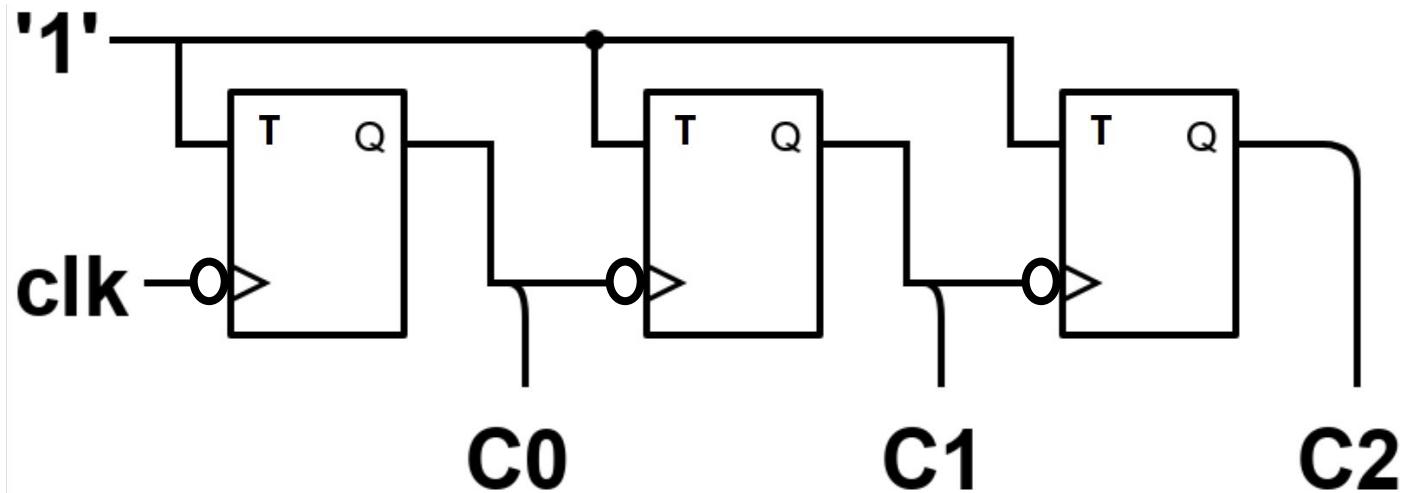
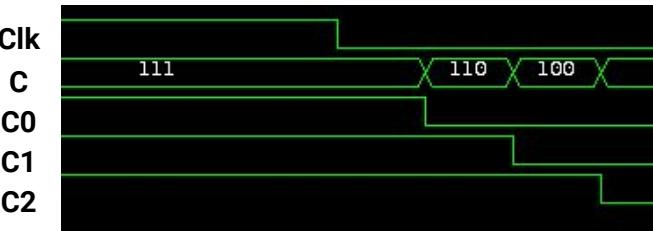
# Contador asincrónico de 3 bits

Vemos que cada flip flop alimenta el clock del FF siguiente, y cada etapa divide por 2 la frecuencia de la etapa anterior. Notemos que se necesita flanco descendente sino cuenta a la inversa.

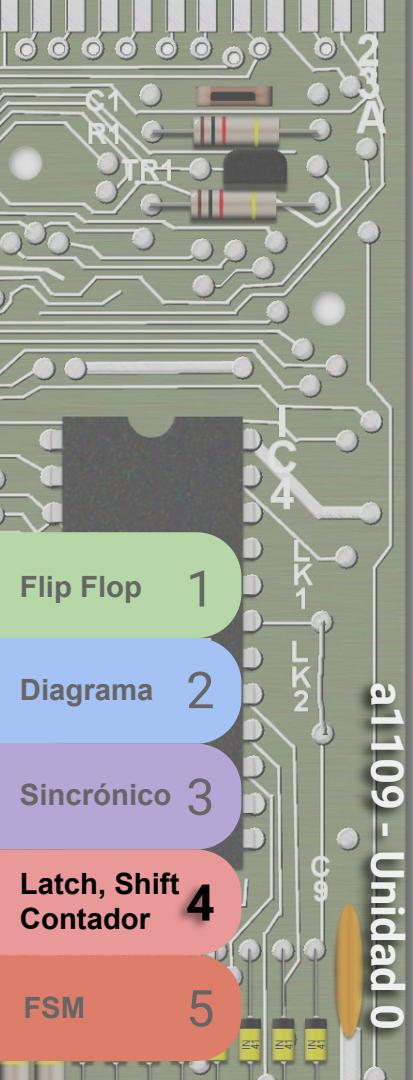


# Contador asincrónico de 3 bits

Si hacemos zoom en algunos momentos, como cuando pasa de 111 a 000, vemos que hay unos estados intermedios. Esto ocurre porque el contador es asincrónico. Esto impone un límite a la frecuencia máxima de operación y genera estados intermedios no deseados. Vemos que en este caso C0, C1 y C2 pasan a 0 pero a distintos tiempos generando valores intermedios inválidos.



# Contador Sincrónico $2^N$



Flip Flop 1

Diagrama 2

Sincrónico 3

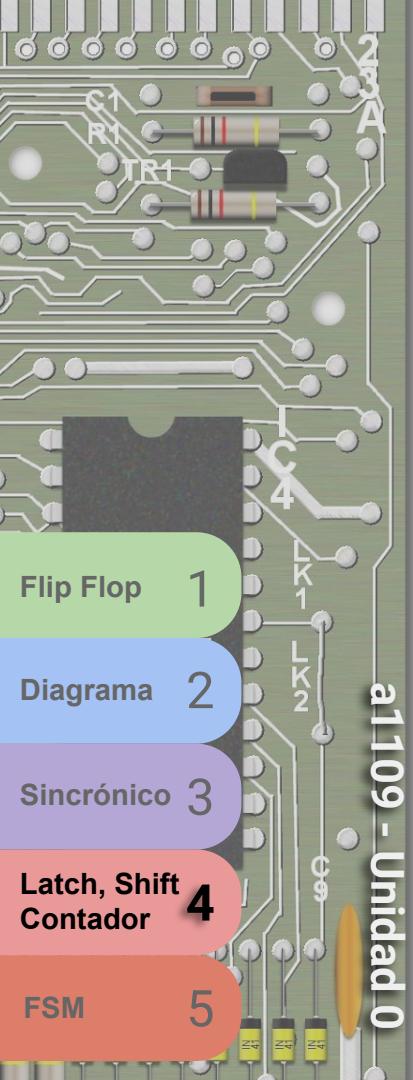
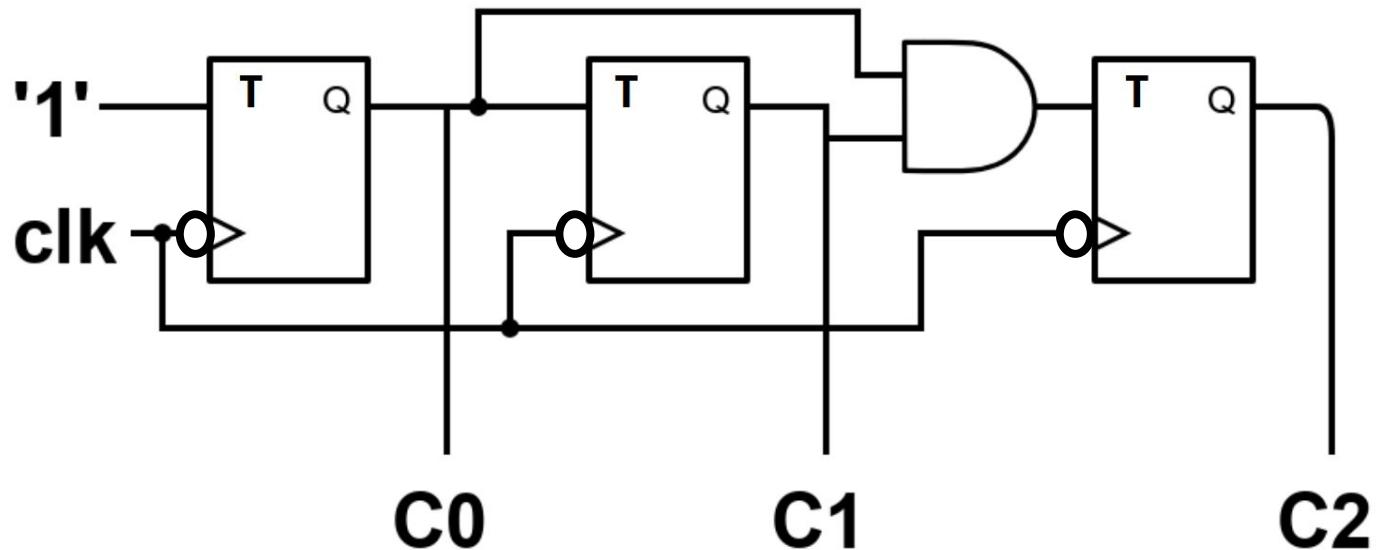
Latch, Shift  
Contador 4

FSM 5

a1109 - Unidad 0

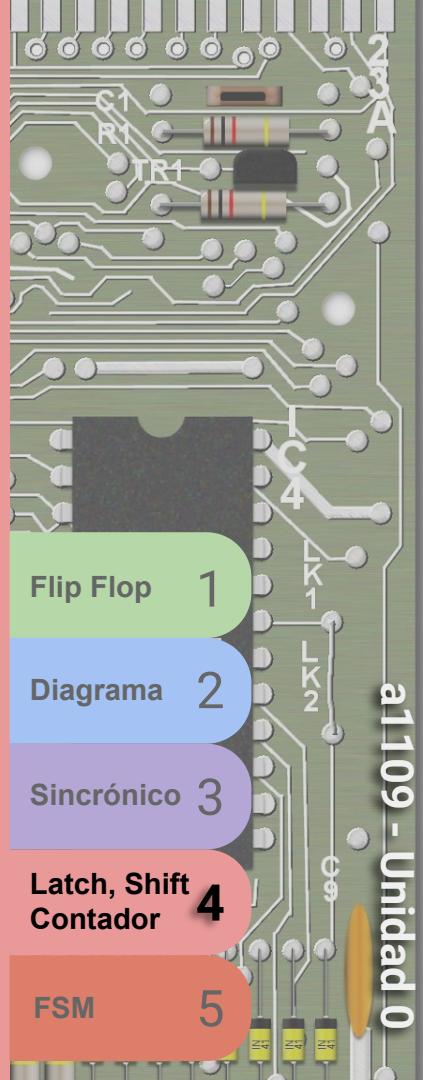
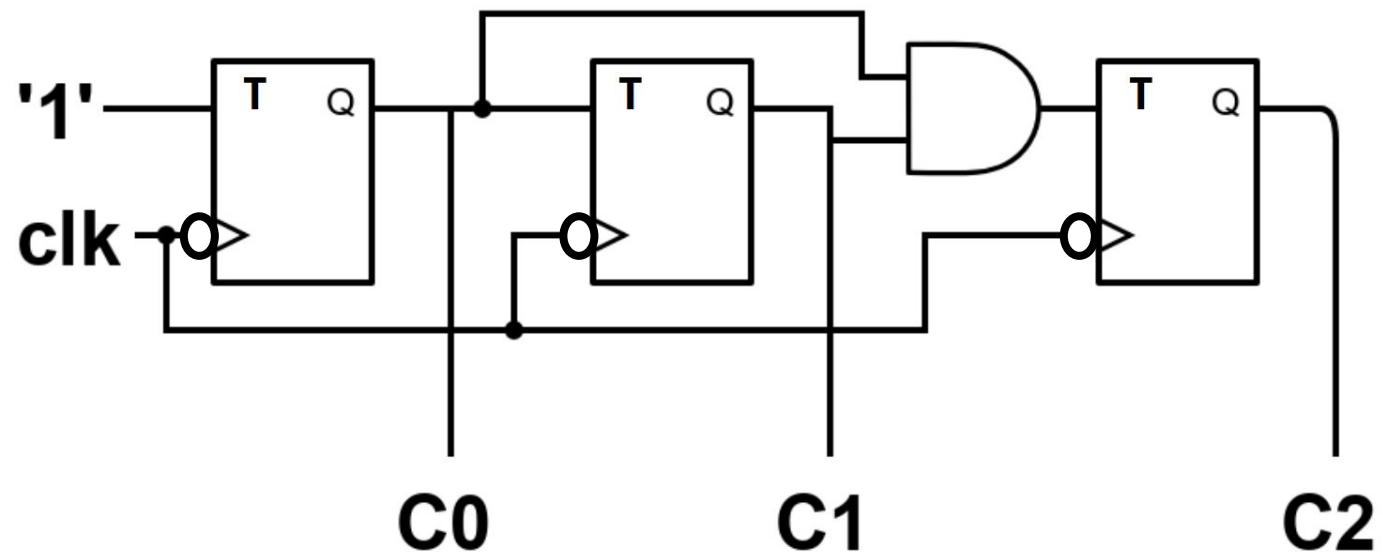
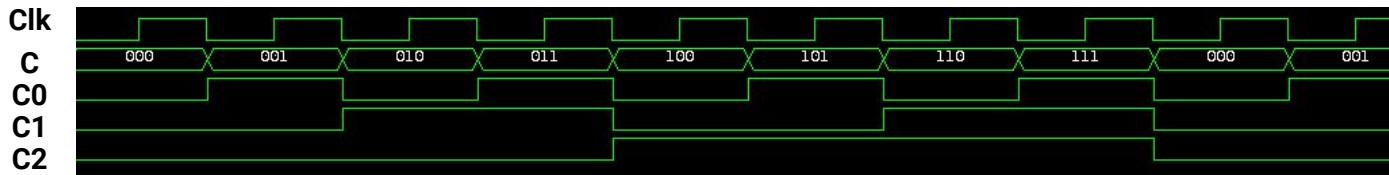
# Contador sincrónico de 3 bits

Con el objetivo de eliminar estados intermedios inválidos, implementamos un contador sincrónico. Vemos que Clk se conecta a todos los FF T al mismo tiempo. Pero ahora la señal T de los flip flops ya no es 1 en todos los casos. Vemos que solo el primer FF T tiene un 1 constante por ende generará la mitad de la frecuencia de clock en cada flanco ascendente.



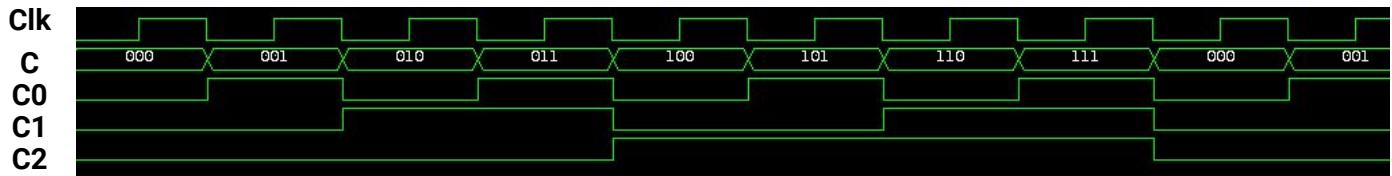
# Contador sincrónico de 3 bits

Vemos que en cada flanco descendente el contador incrementa en 1 el valor de C.

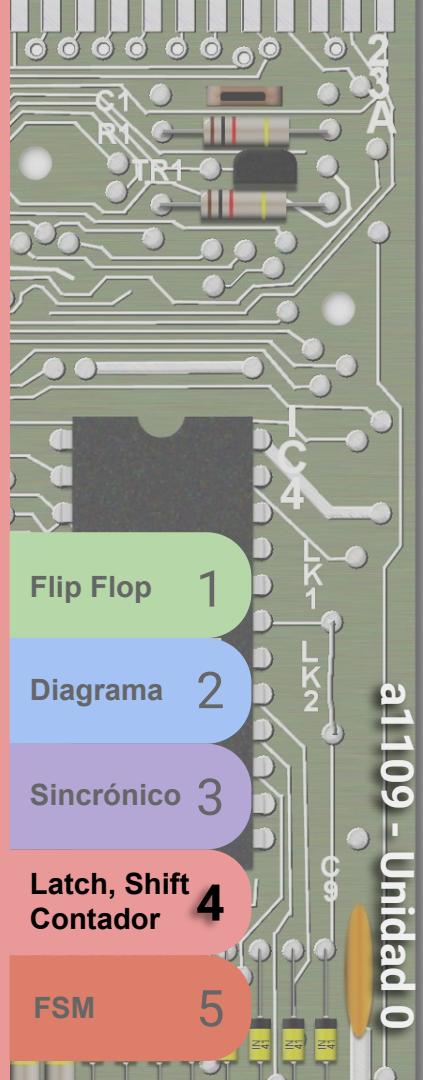
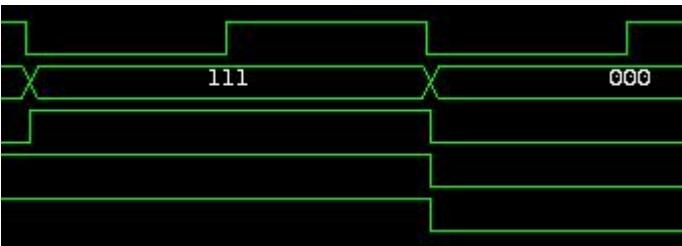


# Contador sincrónico de 3 bits

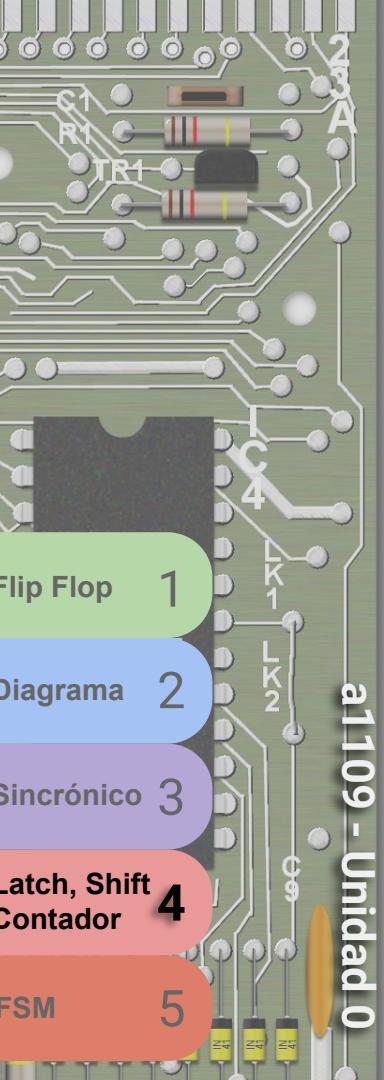
Vemos que en cada flanco descendente el contador incrementa en 1 el valor de C.



Ahora no existen los estados intermedios que aparecían en el contador asincrónico, todos los FF T en este caso cambian con el flanco descendente de la señal “sincrónica” de clock. Es por este motivo que este contador , si bien es más complejo de implementar, funciona sin estados intermedios indeseables.

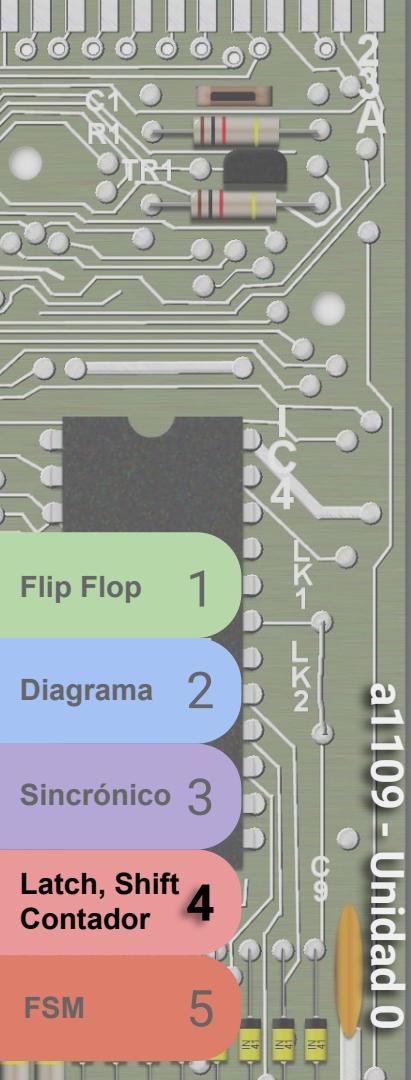
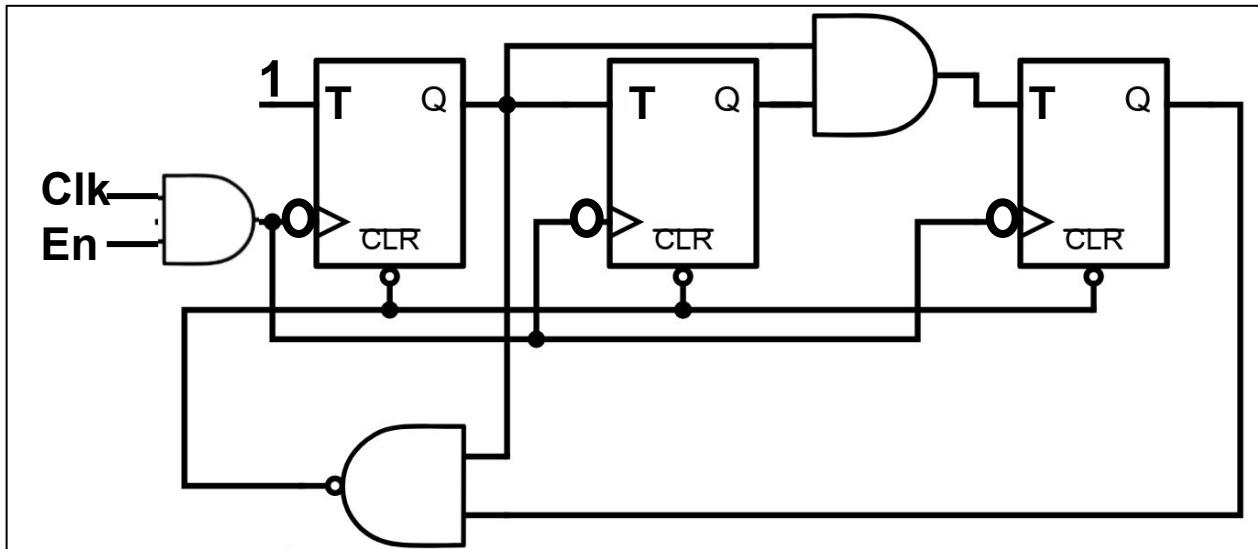


# Contador Sincrónico módulo N



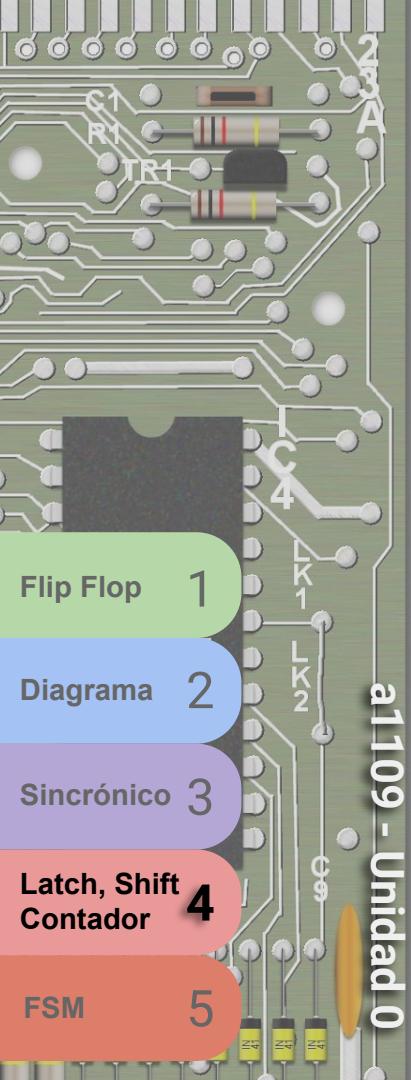
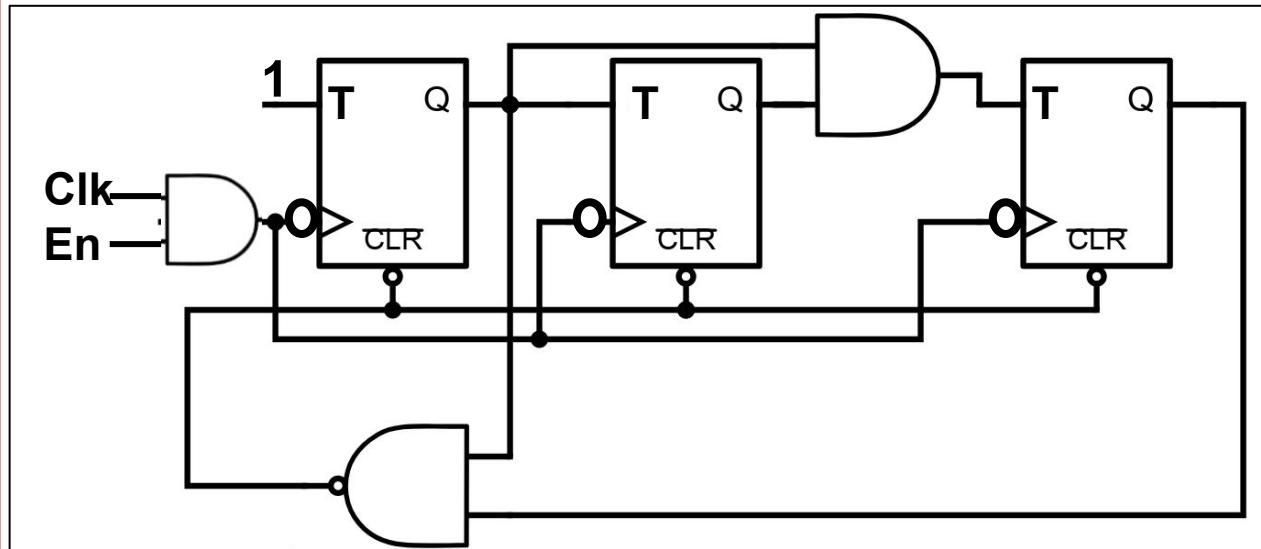
# Contador sincrónico módulo 5

El siguiente circuito posee unas modificaciones. Las salidas C fueron quitadas para simplificar el gráfico. La entrada En (enable) puede ser usada para “frenar” el contador. Si En=0, el contador no cuenta. Si En=1 entonces con cada pulso de clock el contador incrementa su cuenta. Esto resultará muy útil si deseamos detener la cuenta.



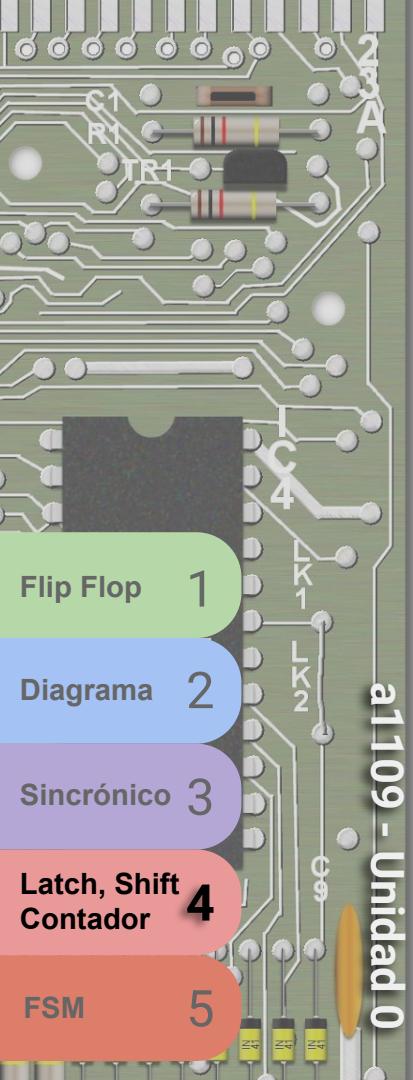
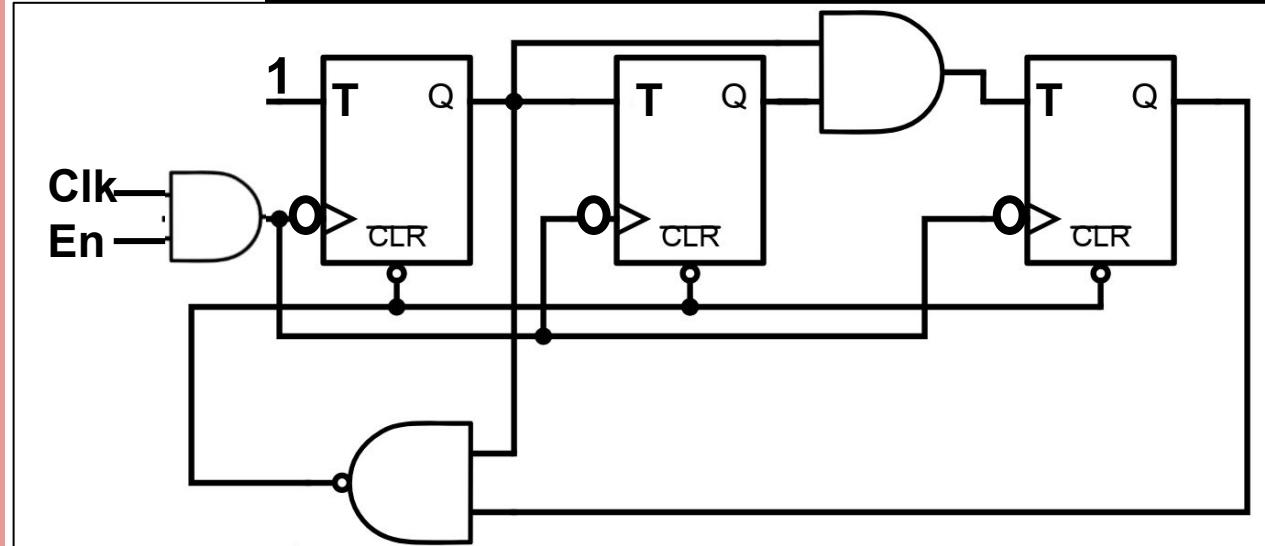
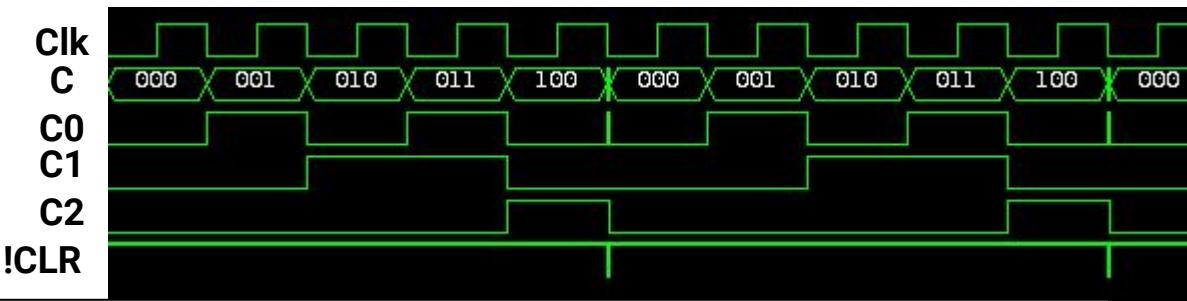
# Contador sincrónico módulo 5

La otra modificación más notoria es el uso de las entradas de reset ( $\overline{CLR}$ ) en cada FF T. Cuando  $\overline{CLR}=0$  el FF T tiene como salida  $Q=0$ . Cuando  $\overline{CLR}=1$  opera como un FF T normal. Vemos que existe una NAND entre la salida del primer FF y la salida del último. Esto quiere decir que cuando la cuenta sea: 1x1 (notemos que no verifica el bit del medio) la compuerta NAND genera un 0 a su salida, haciendo que todos los FF T pasen a 0.



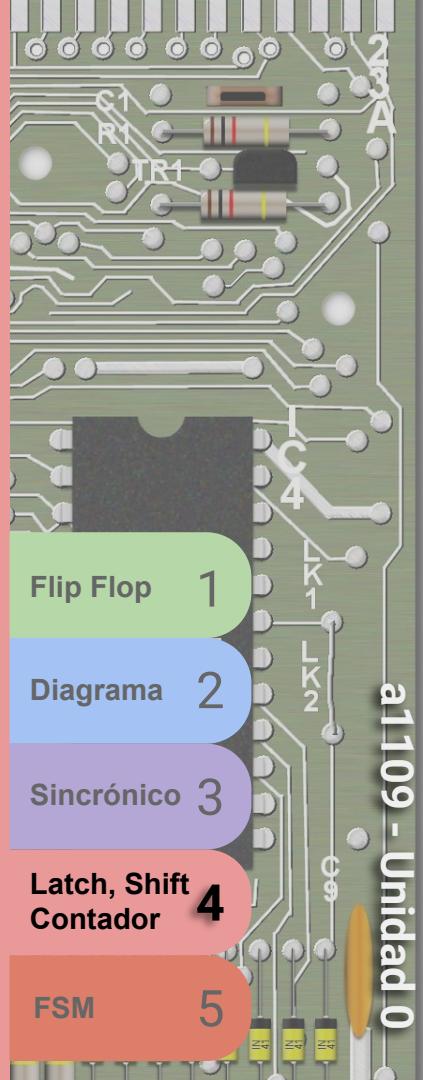
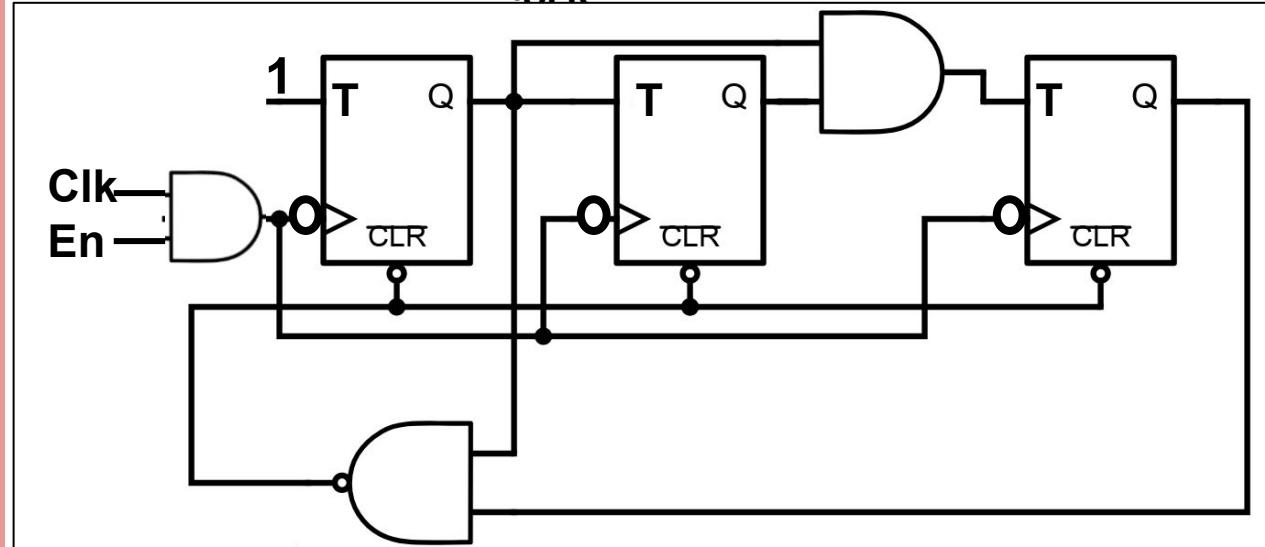
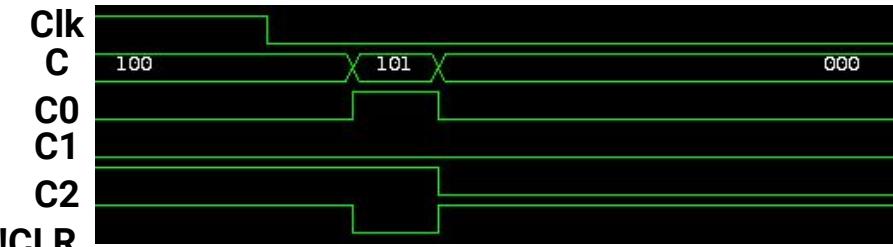
# Contador sincrónico módulo 5

Vemos en el diagrama de tiempos que el contador va de 000 a 100 y luego vuelve a 000. Hagamos zoom en este cambio...



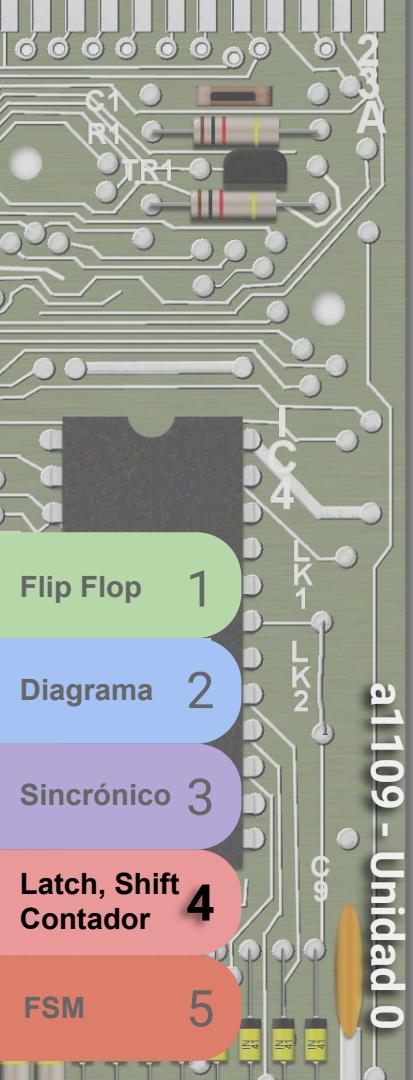
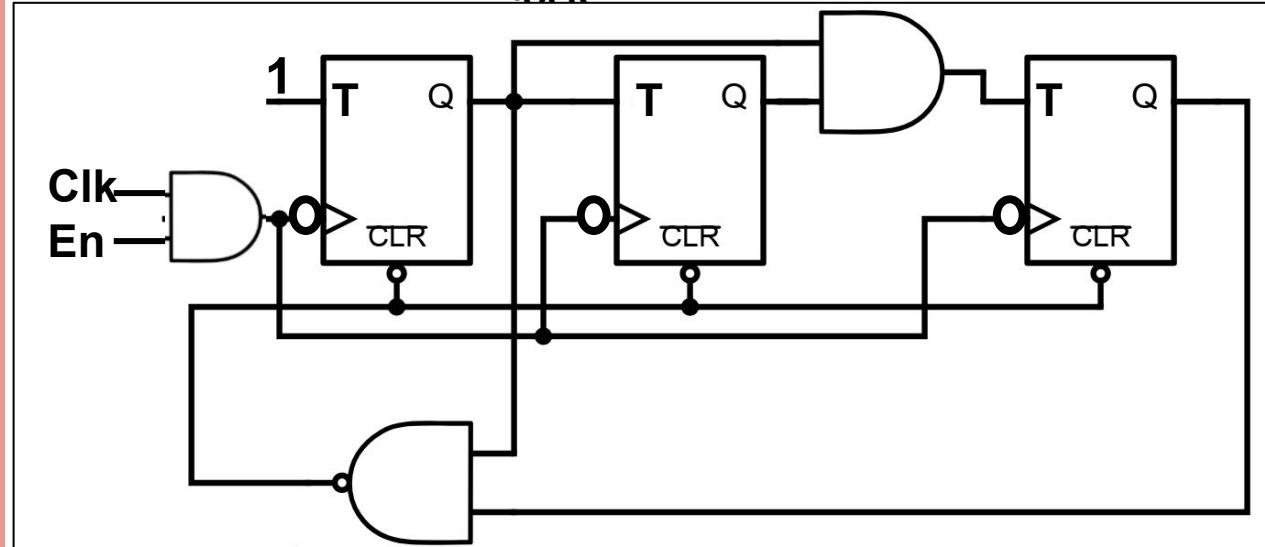
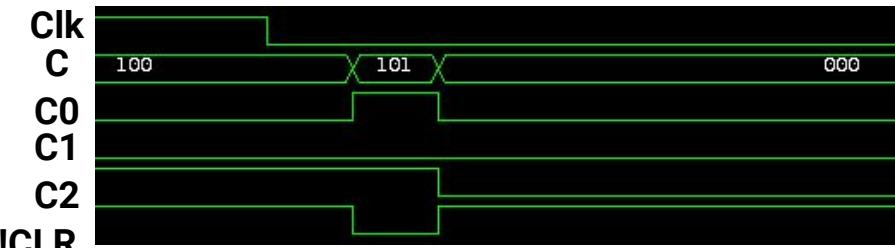
# Contador sincrónico módulo 5

Vemos que por un instante de tiempo muy corto el contador toma efectivamente 101.. Esto es capturado por la NAND que genera el pulso en !CLR y los FF T vuelven todos a 000.



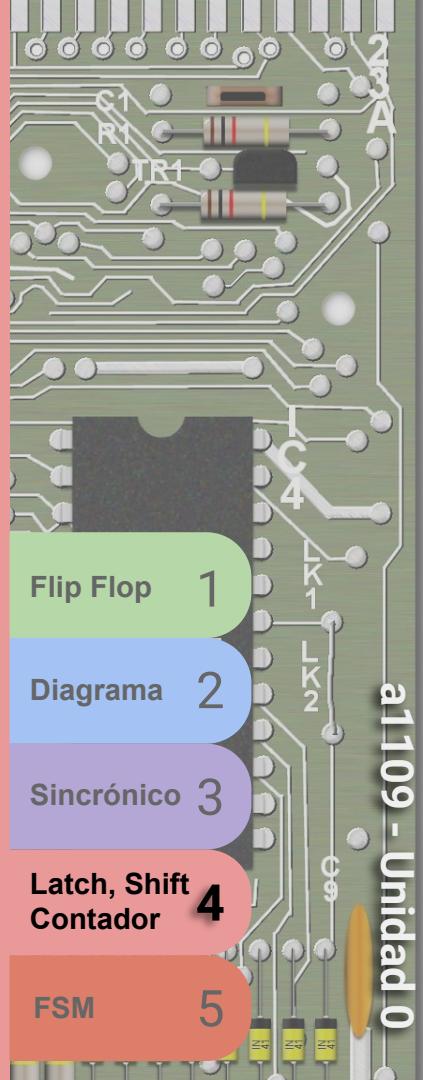
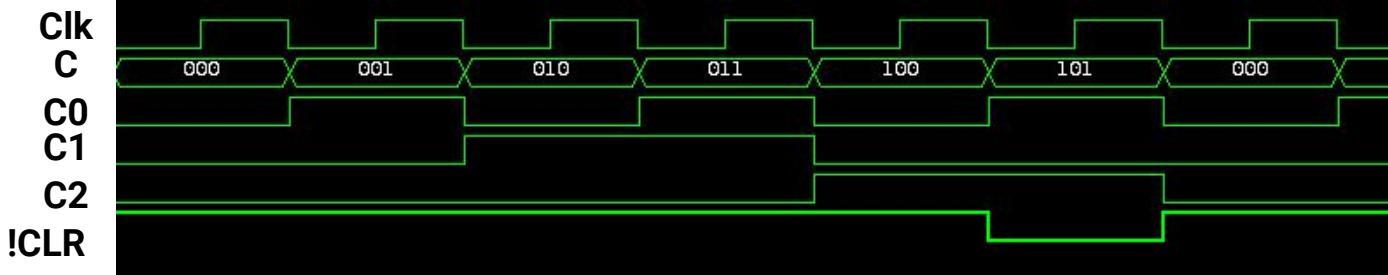
# Contador sincrónico módulo 5

Esto se debe a que la señal de CLR es asincrónica. O sea, cambia los FF T a 0 inmediatamente. Existen FF T con señales de CLR sincrónicas las cuales fuerzan el 0 cuando hay un flanco en Clk.

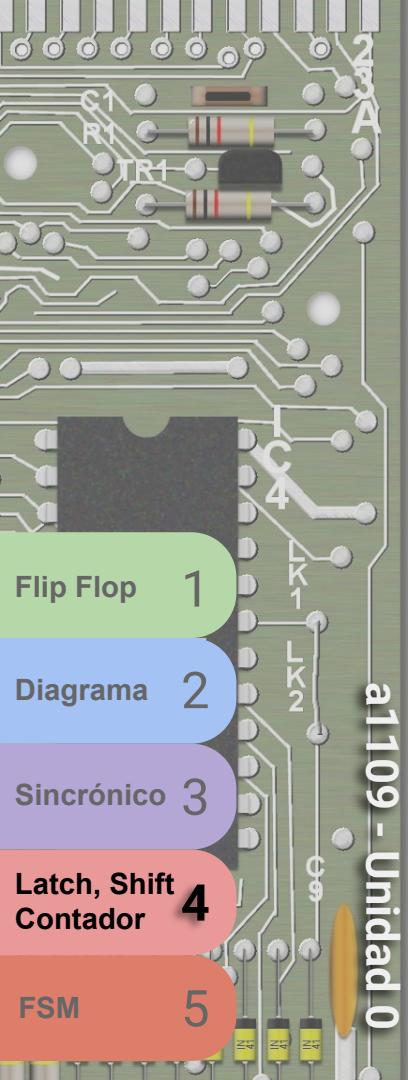


# Contador sincrónico módulo 6

Si cambiamos la tecnología de los FF T para que utilicen reset sincrónicos, vemos que el contador cuenta un valor extra!!!. Ahora el 101 es parte de la cuenta. Esto se debe a que la compuerta NAND está pensada para detectar el valor 1x1, y vemos que cuando eso ocurre generar un 0 para !CLR. Sin embargo como ahora el FF T solo presta atención a !CLR en el flanco descendente, hay que esperar al siguiente pulso de clock para que los FF T pasen a 0. Vemos que esto agrega un valor más, que puede ser compensado fácilmente cambiando la NAND para detectar la combinación 100 en vez de 101. Vemos que en este caso el cambio de 101 a 000 se produce de forma limpia... sin estados indeseables de por medio.

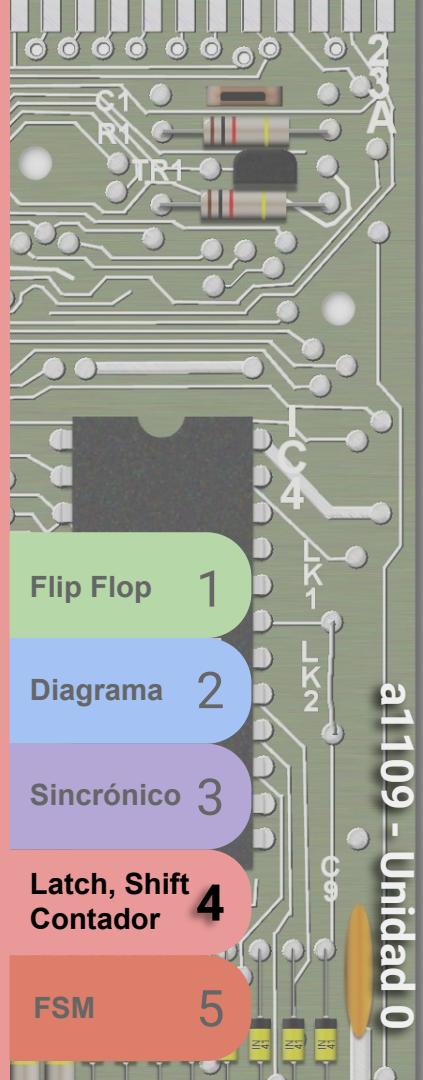
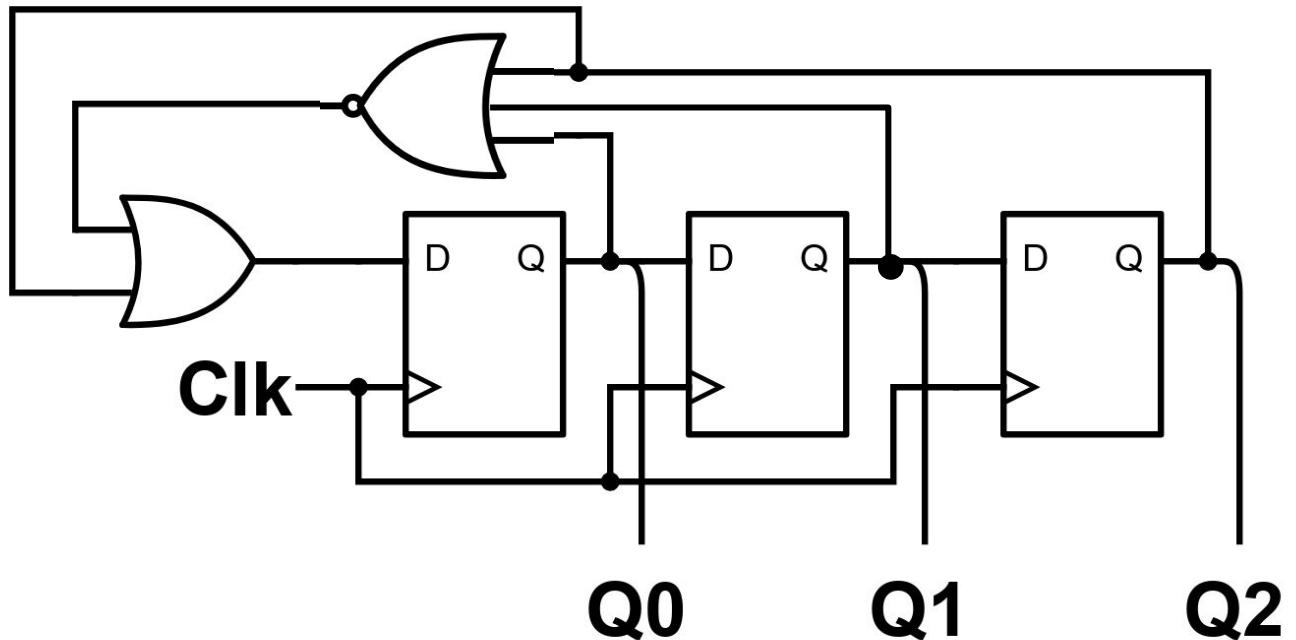


# Contador Sincrónico módulo N anillo



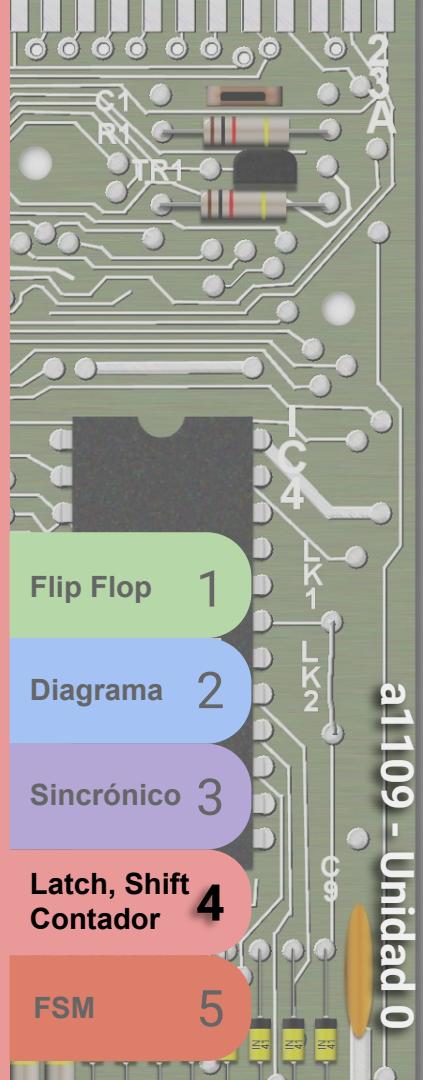
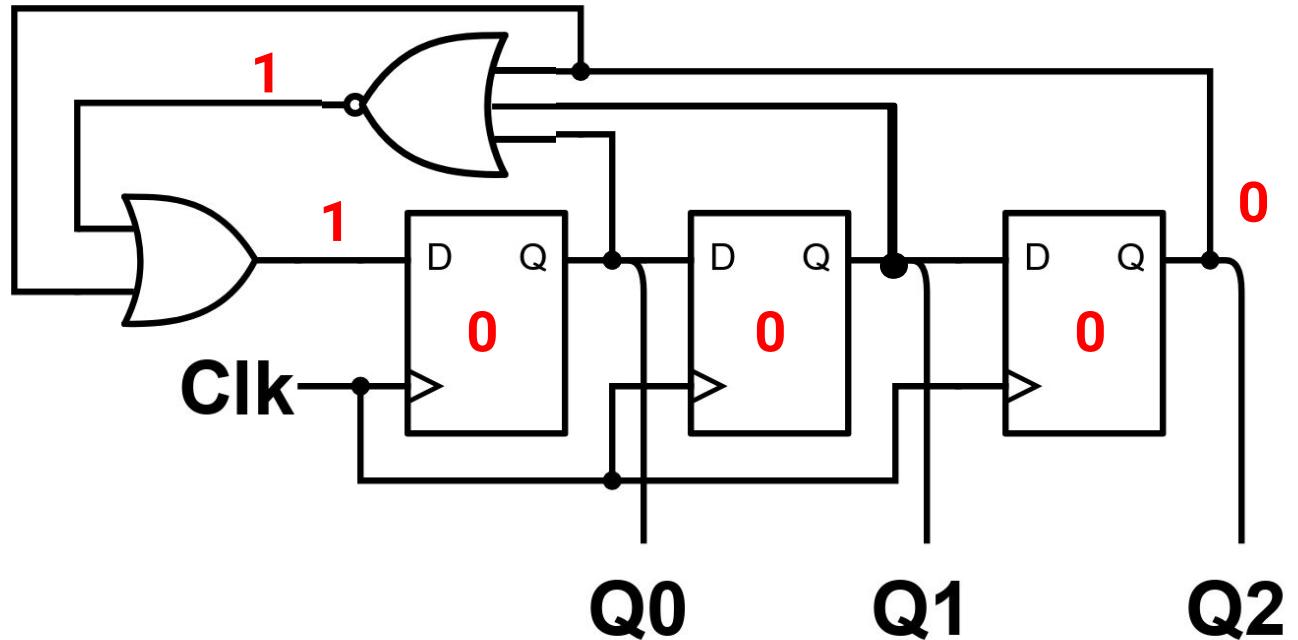
# Contador sincrónico módulo N anillo

Este contador posee 3 FF tipo D, encadenados como un shift register, pero notemos la compuerta NOR sobre las 3 salidas. Esta NOR solo dará un 1 cuando el contador sea 000. Esto ocurre cuando el mismo se enciende. Ese 1 de la NOR se encuentra como entrada en el primer FF D mediante la OR.



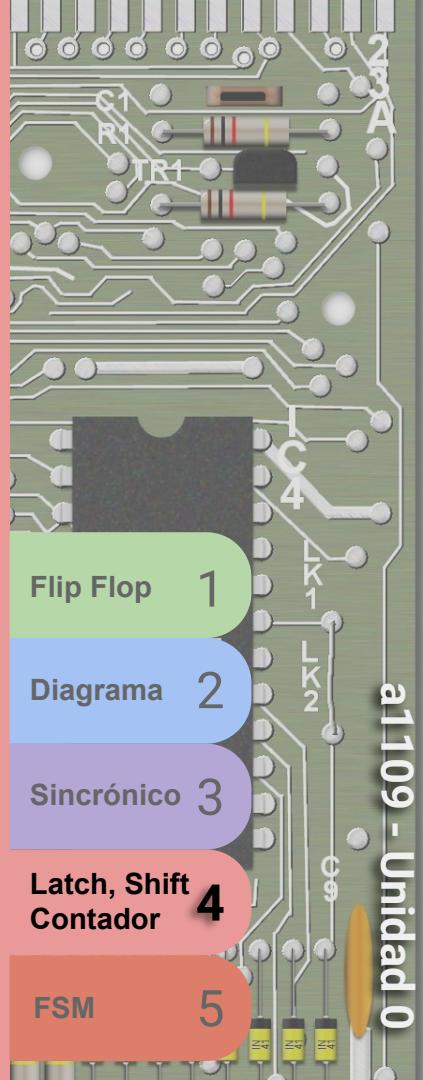
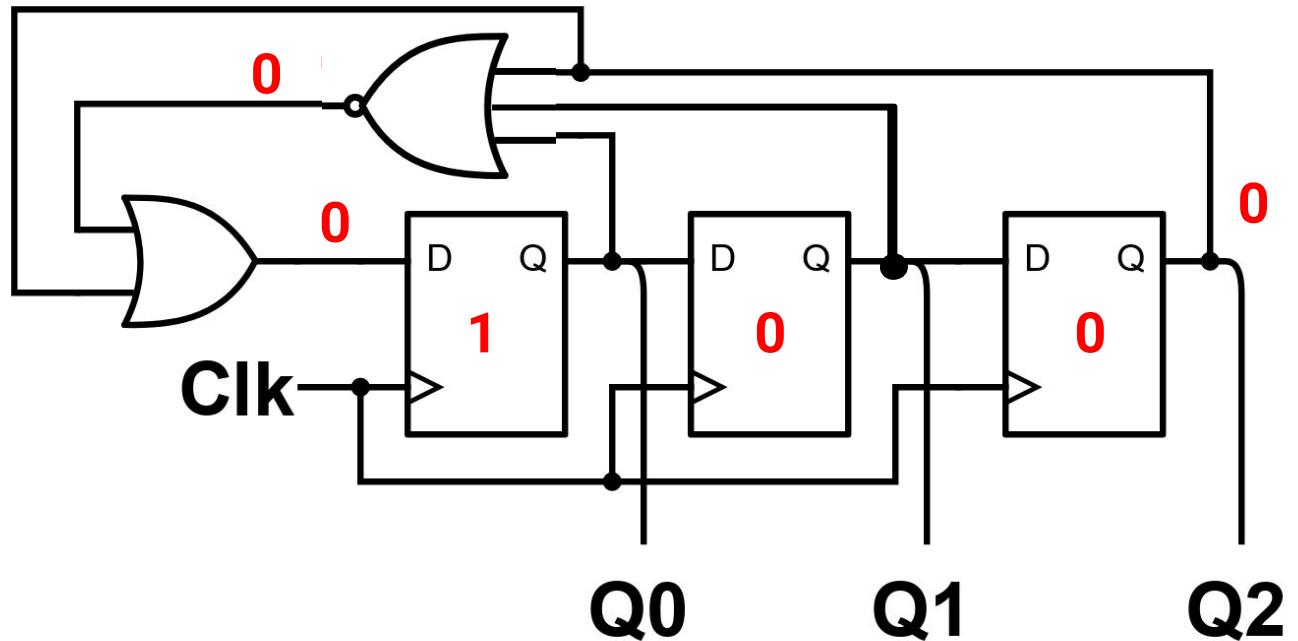
# Contador sincrónico módulo N anillo

Vemos que la OR tiene como entradas la salida de la NOR (que detecta 000 y genera un 1 en ese caso) y la salida del último FF. Cuando el circuito se enciende, y todavía NO llega el primer clock, el mismo tiene un 1 esperando entrar en el primer FF. En el primer clock vemos que ese 1 se captura y la NOR vuelve a 0.



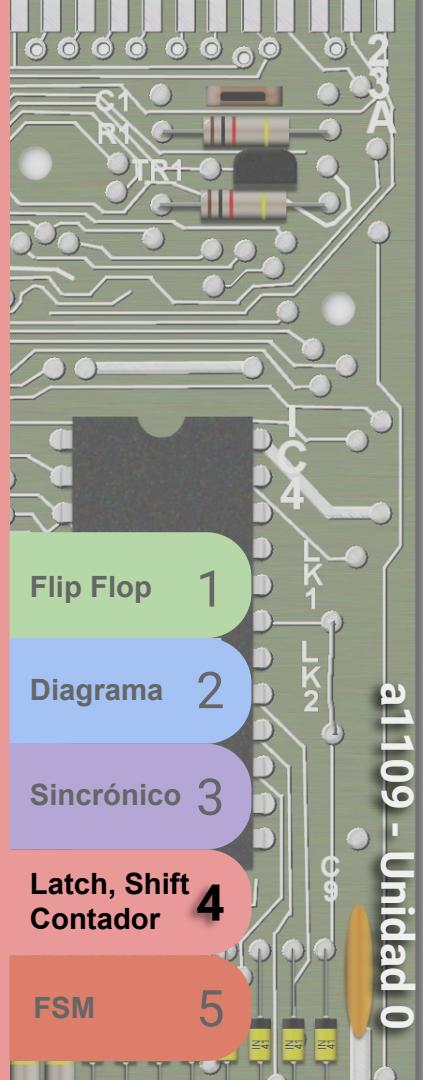
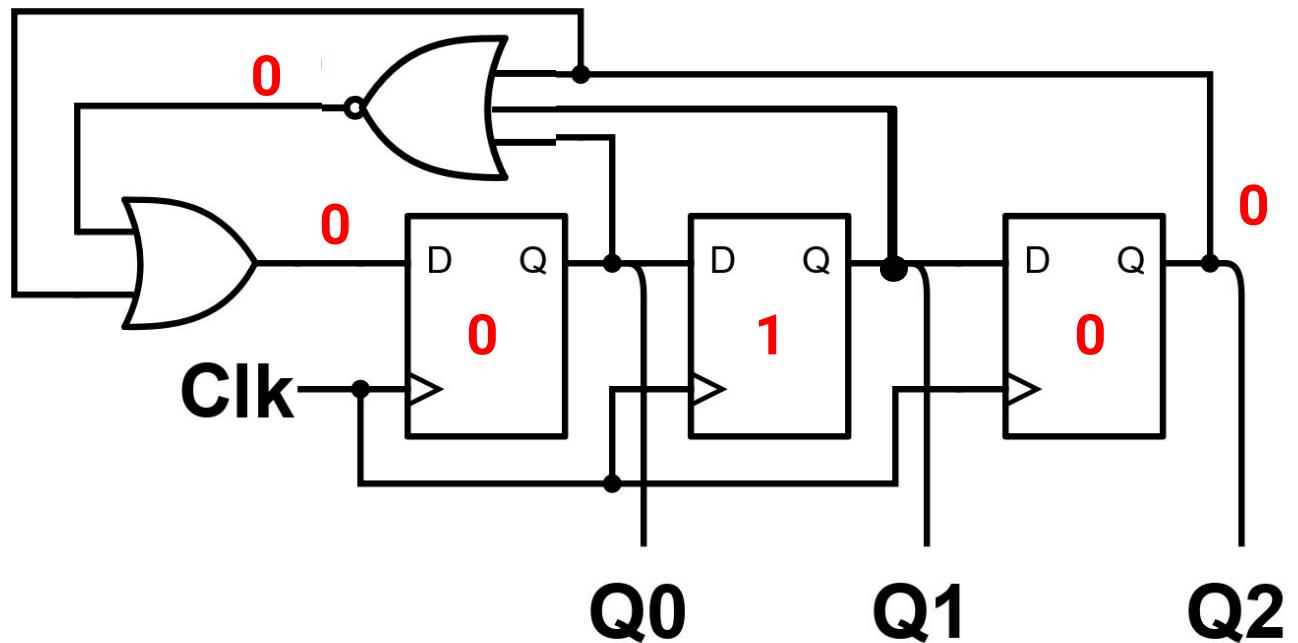
# Contador sincrónico módulo N anillo

Cuando llega el primer pulso de clock, vemos que el 1 ahora se capturó en el primer FF. Vemos que NOR pasa a 0 (y veremos que nunca más será 1) y la entrada de D del primer FF ahora tiene un 0.



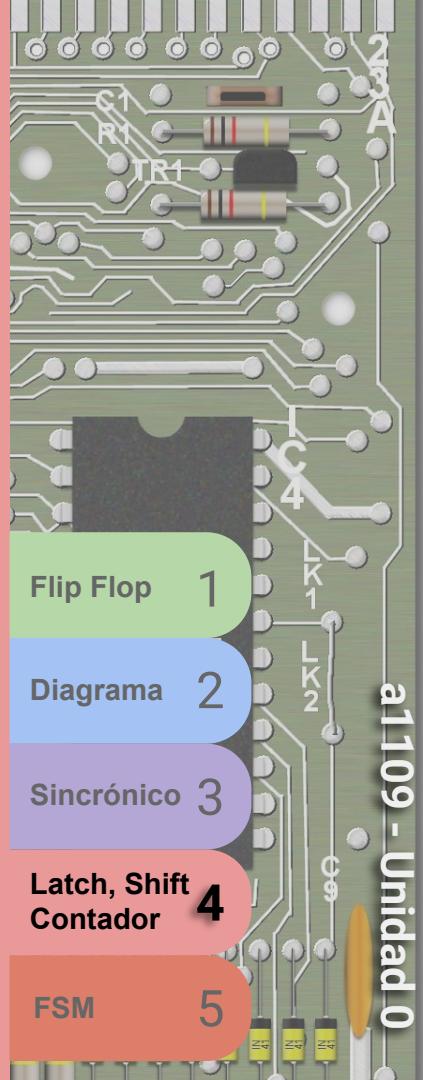
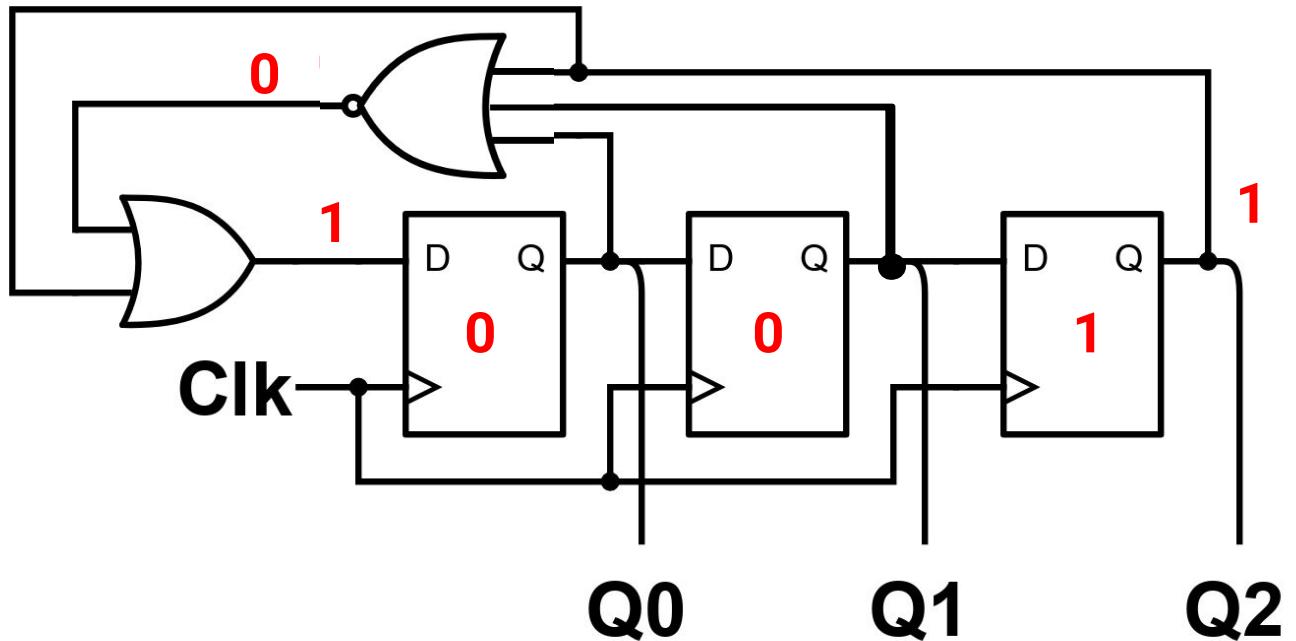
# Contador sincrónico módulo N anillo

Con el segundo pulso de clock vemos que ahora el segundo FF tiene el 1, el resto todo 0.



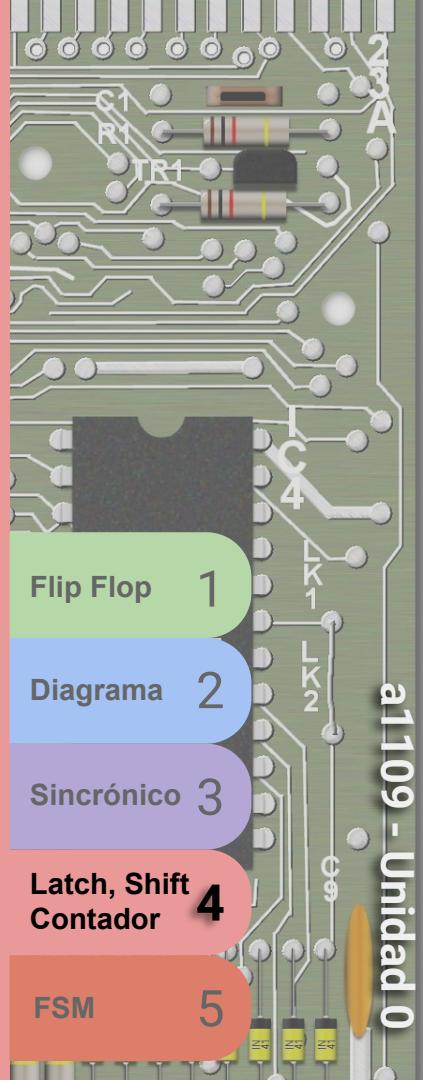
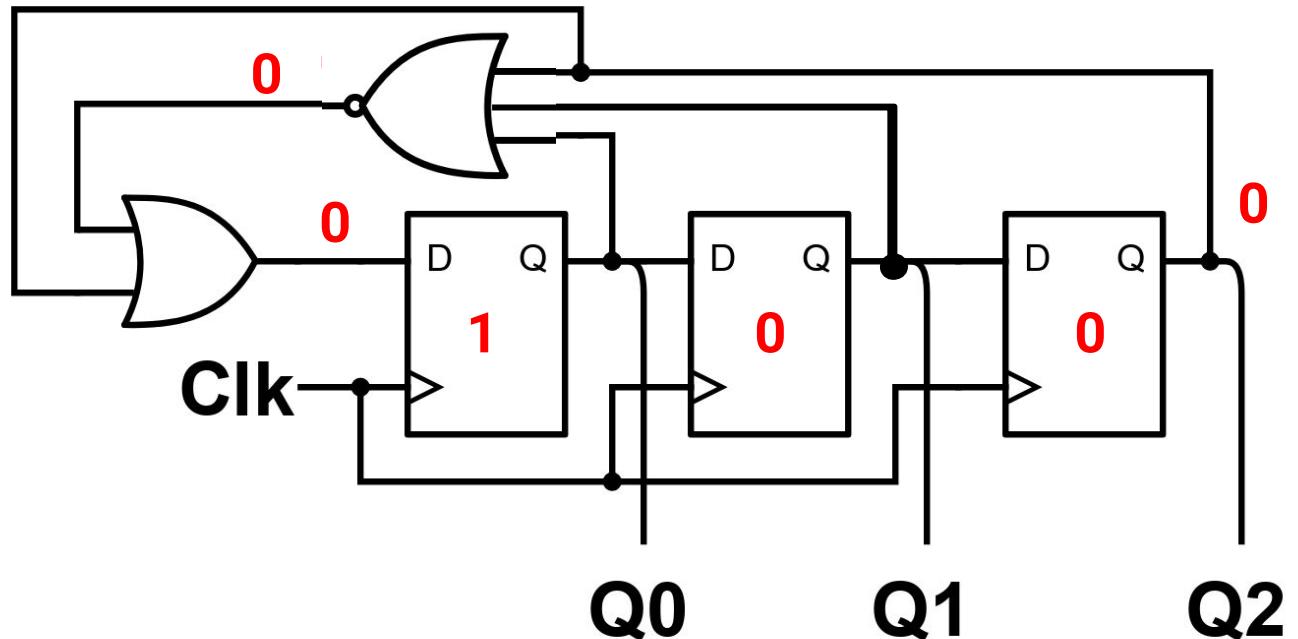
# Contador sincrónico módulo N anillo

En el tercer pulso de clock ahora el último FF tiene el 1, Vemos que como esta salida se realimenta por la OR, ahora el 1 queda disponible para el primer FF...

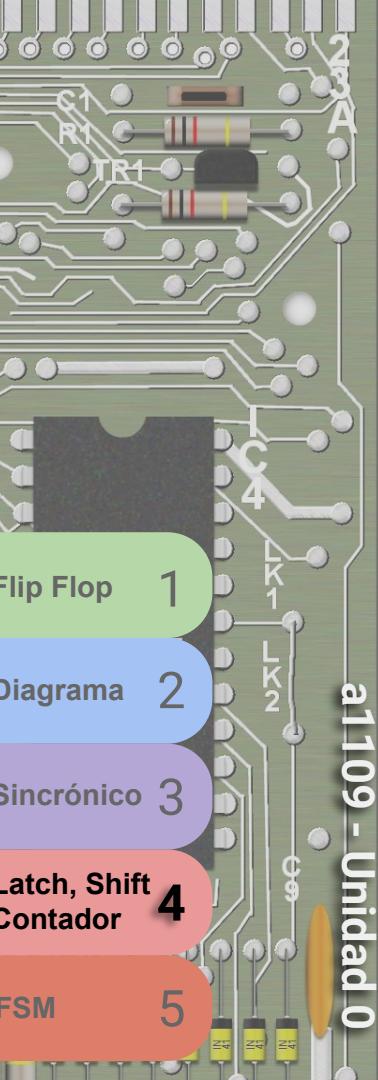


# Contador sincrónico módulo N anillo

Con el cuarto pulso de clock volvemos al estado donde solo el primer FF tiene un 1. Vemos que la cuenta pasó de 000 (antes del primer clock por única vez) a la secuencia 100, 010, 001 y luego se repite 100, 010, 001. Así que con 3 FF tenemos un contador módulo 3, o sea cuenta 3 elementos distintos.

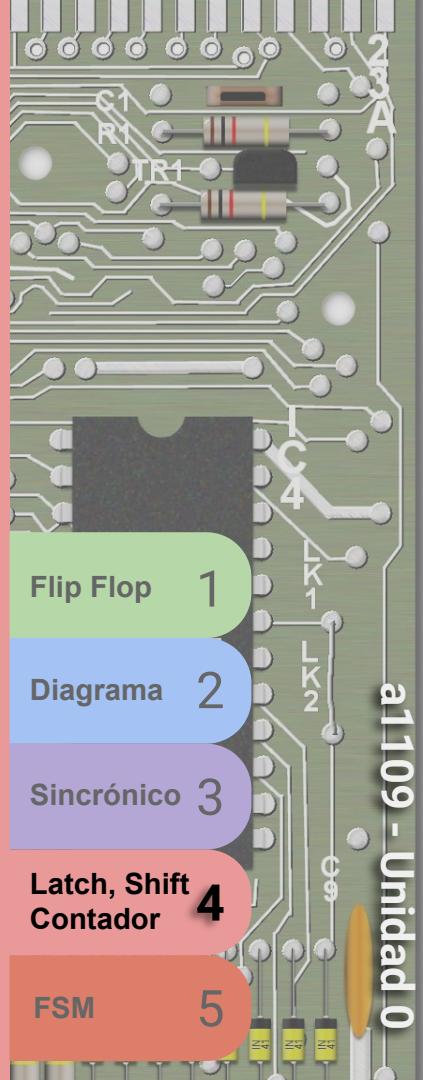
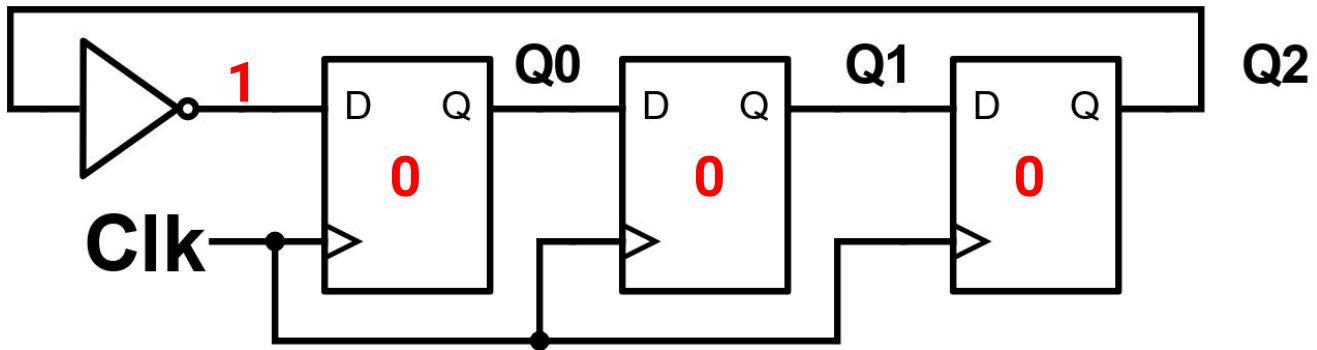


# Contador Sincrónico módulo $2^*N$



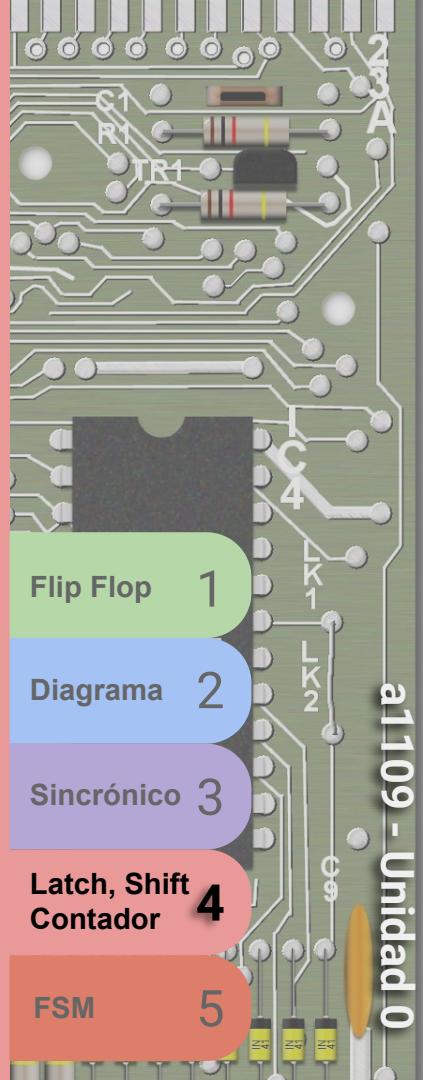
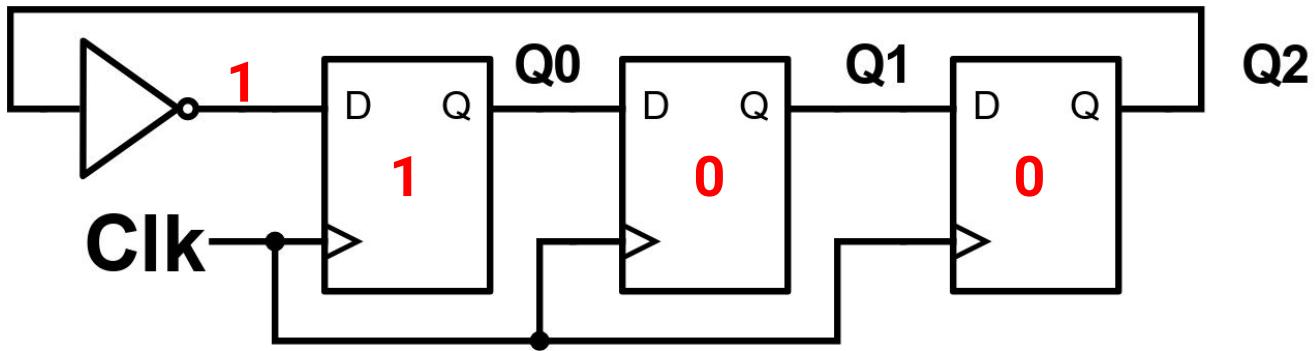
# Contador sincrónico módulo $2^*N$

El siguiente contador está formado por 3 FF tipo D. Vemos que también tiene una estructura tipo shift register. La entrada es equivalente a la salida del último FF pero invertida. Cuando el mismo se enciende, se encuentra todo en 0.



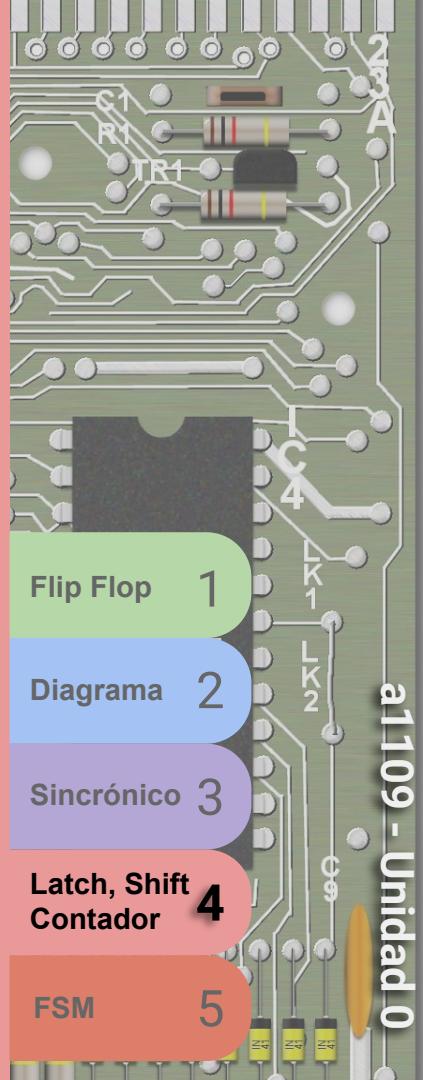
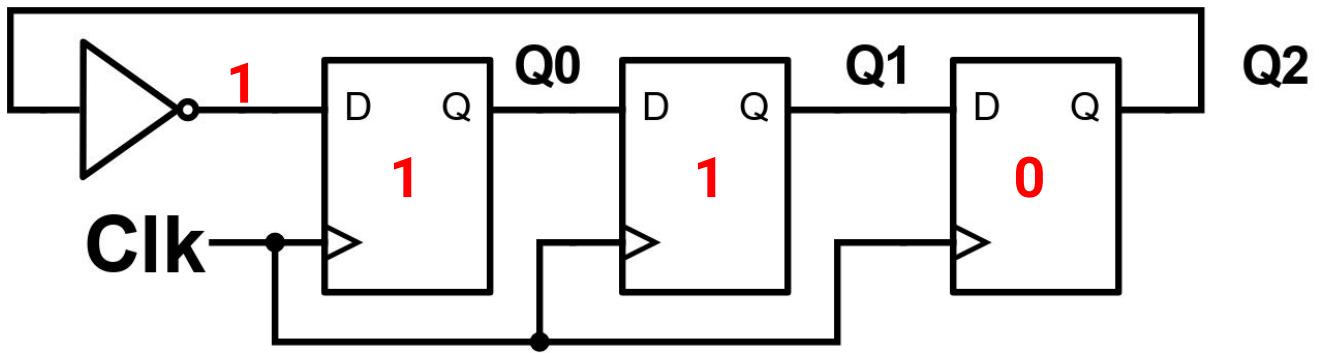
# Contador sincrónico módulo $2^*N$

En el primer pulso de clock tenemos un 1 en el primer FF, pero el resto todo en 0. Notemos que la entrada de la cadena sigue siendo 1.



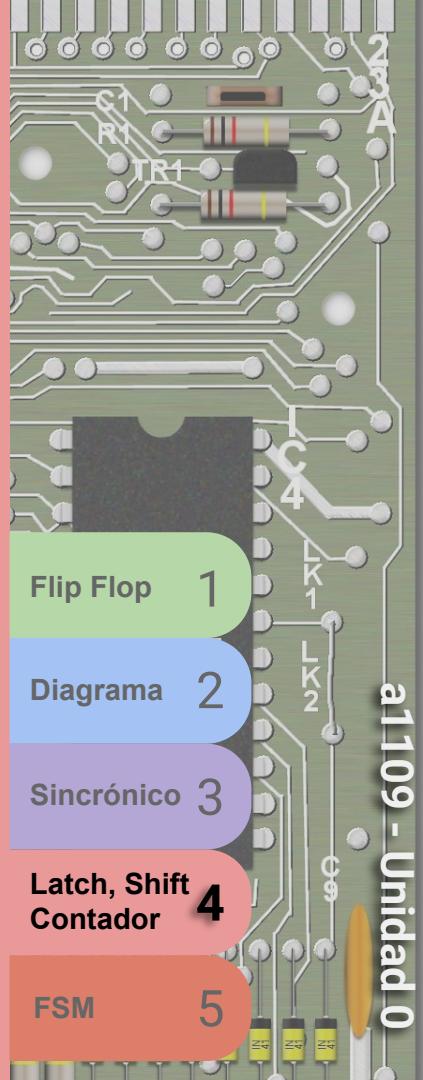
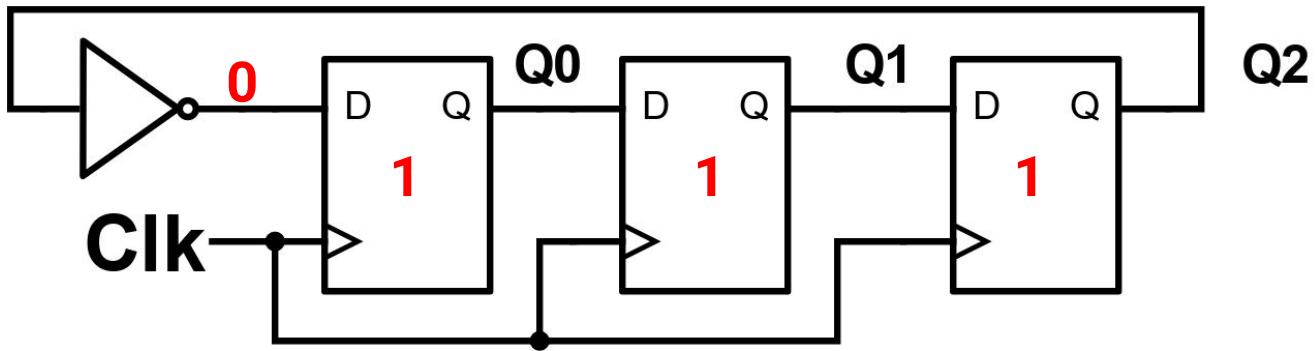
# Contador sincrónico módulo $2^*N$

En el segundo pulso de clock tenemos un 1 en el segundo FF.



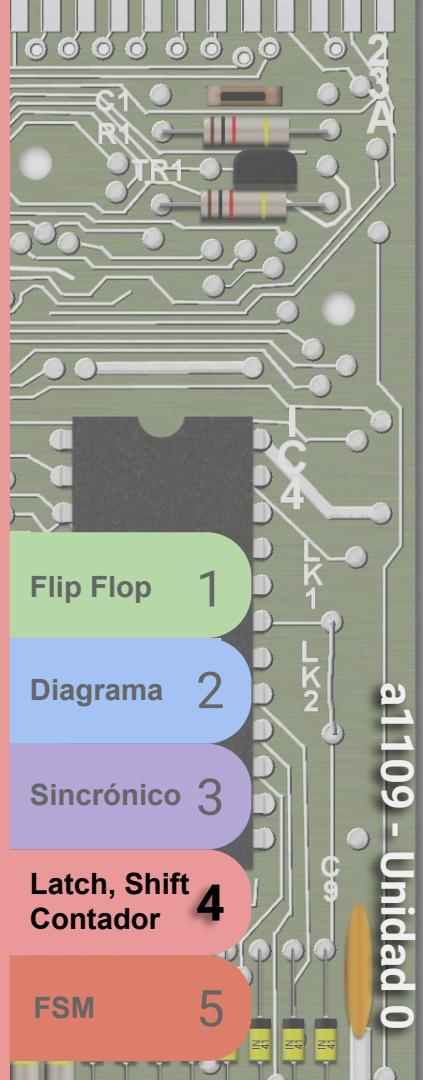
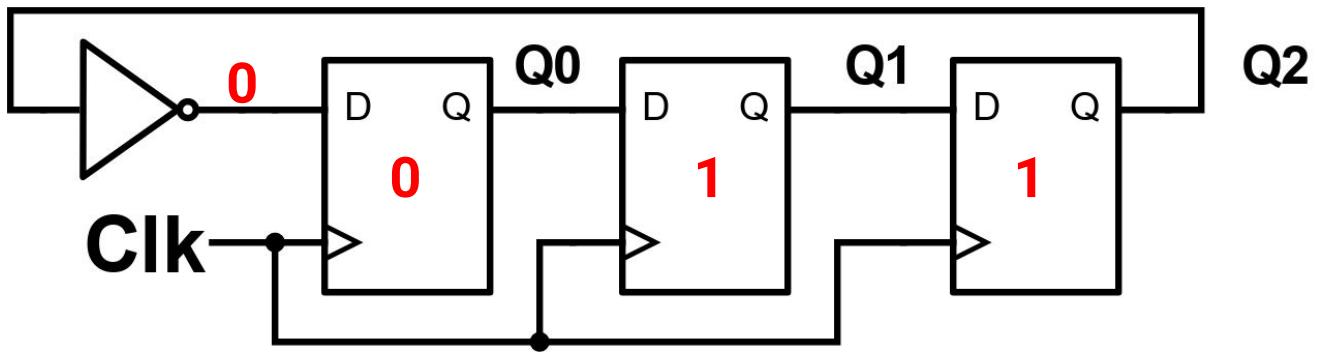
# Contador sincrónico módulo $2^*N$

En el tercer pulso de clock tenemos ahora todos los FF con un 1 pero ahora la entrada cambia a 0.



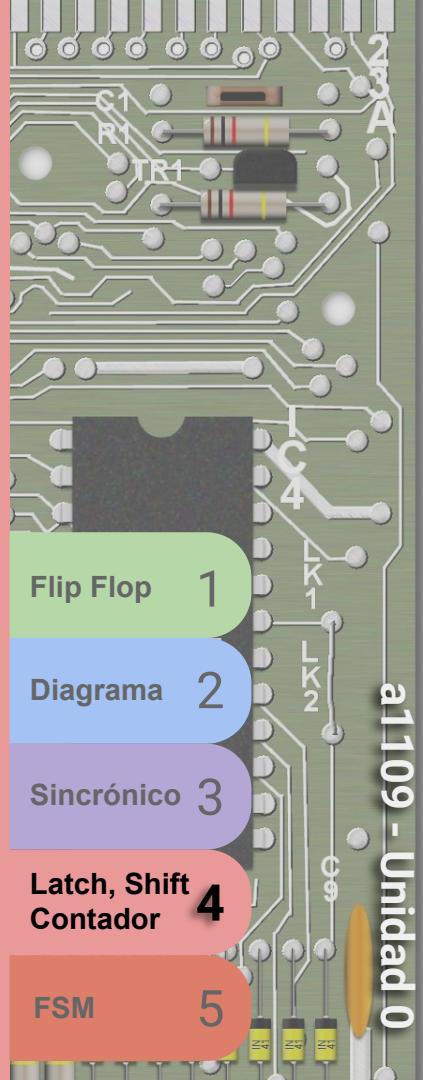
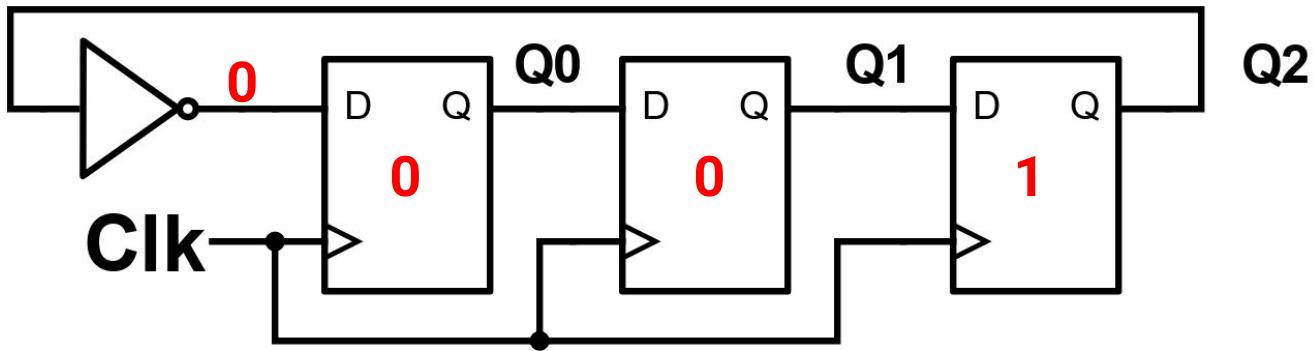
# Contador sincrónico módulo $2^*N$

En el cuarto pulso de clock tenemos un 0 en el primer FF, el resto se mantiene en 1.



# Contador sincrónico módulo $2^*N$

En el quinto pulso de clock tenemos la cuenta 001, con la entrada todavía en 0.

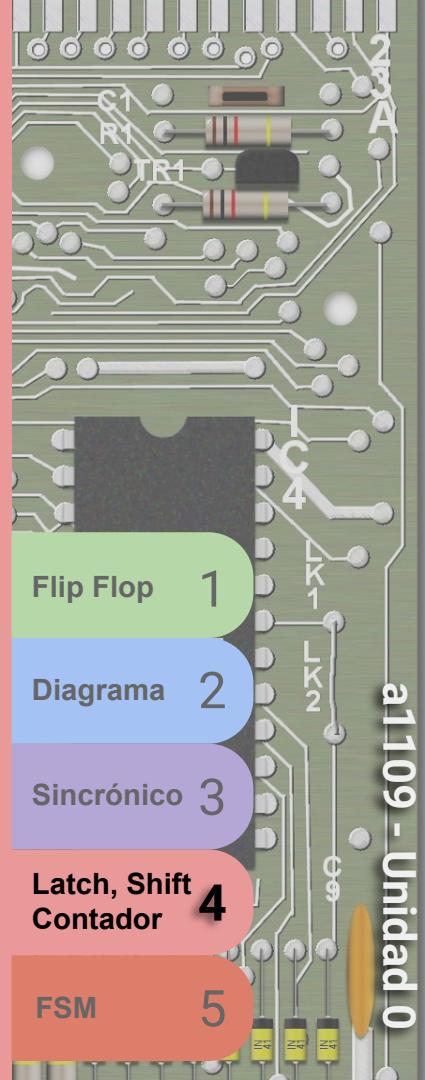
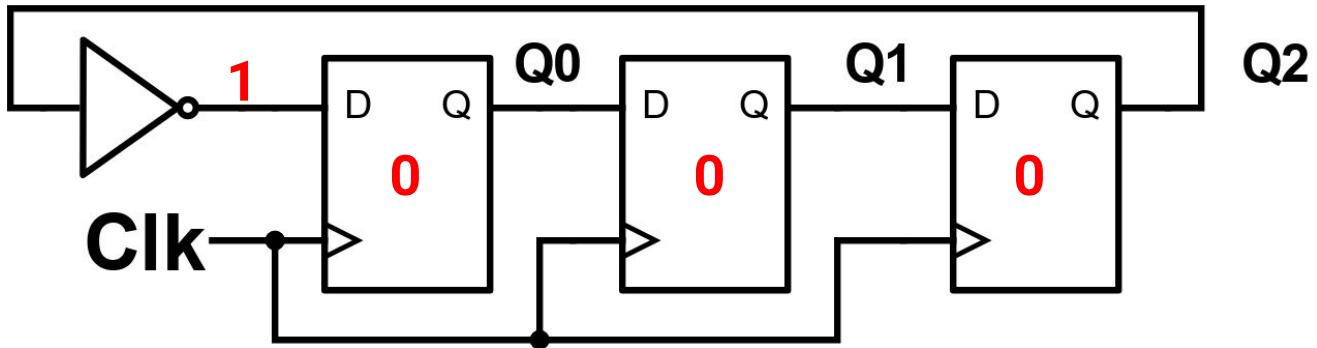


# Contador sincrónico módulo $2^N$

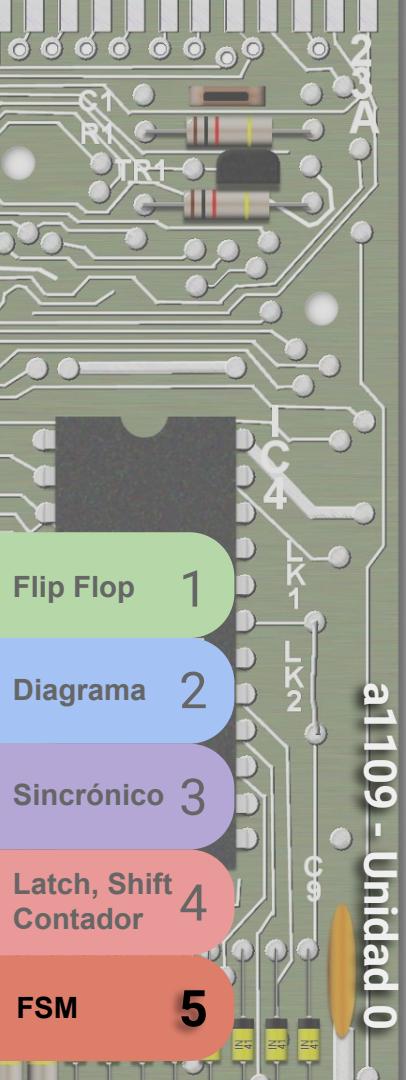
En el sexto pulso de clock volvemos al estado original con todos los FF D en cero y un uno en la entrada.

Este contador cuenta la secuencia: 000, 100, 110, 111, 011, 001 y luego se repite. Vemos que el módulo de cuenta es  $2^N$  siendo N la cantidad de FF.

Vemos que su implementación es sencilla en desmedro de la cantidad de elementos que cuenta. Con la misma cantidad de FF podemos contar  $2^N$  elementos utilizando un contador sincrónico.



# Máquinas de estado finito (FSM)



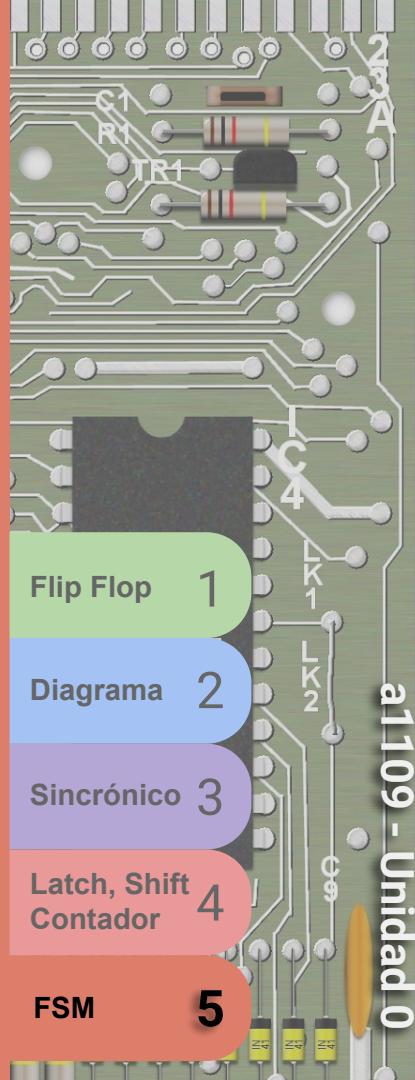
Ahora que somos especialistas en circuitos combinacionales y secuenciales, podemos analizar las particularidades de cada tipo.

## Combinacionales

- Se describen enteramente por su tabla de verdad siempre (sin memoria).
- El tiempo es solo importante para calcular el retardo máximo desde que una entrada afecta la salida.
- Dado un circuito combinacional desconocido, podemos reconstruir su tabla de verdad aplicando todas las entradas posibles.

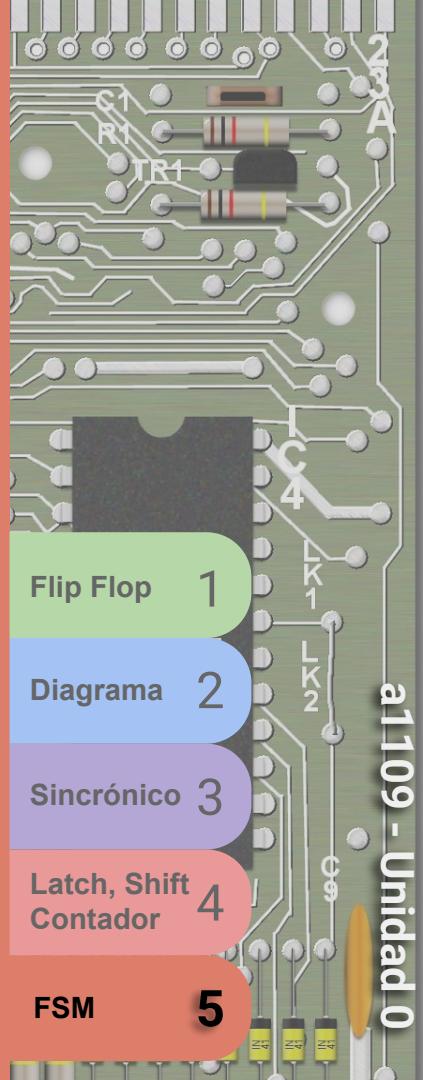
## Secuenciales

- Su tabla de verdad tiene valores que dependen del estado anterior (memoria).
- El tiempo es un factor fundamental para el análisis de estos circuitos. La salida puede cambiar en función del tiempo.
- Dado un circuito secuencial desconocido, no siempre podemos reconstruir su función en base a las entradas.



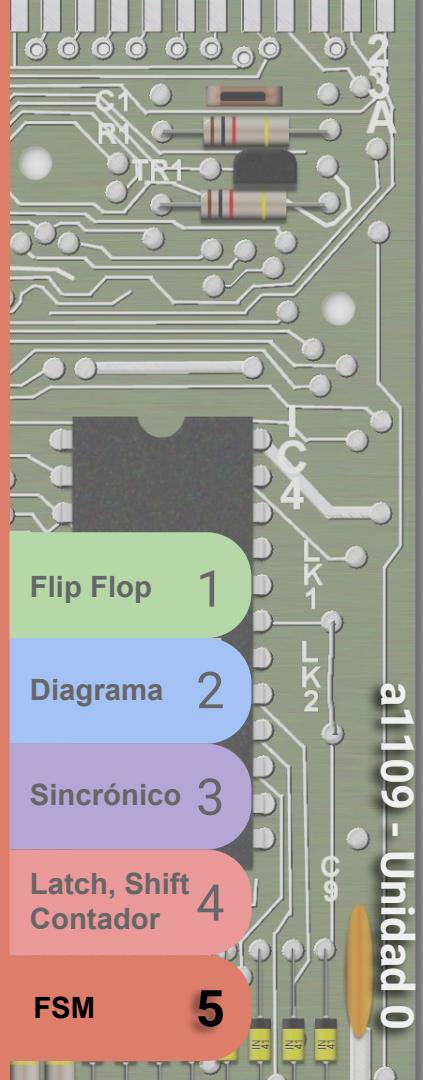
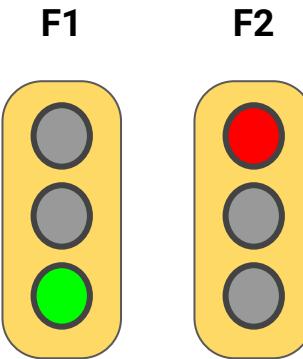
# Una esquina con semáforo

Esta es la intersección de la esquina de la universidad. Vemos que hay un semáforo que controla el tránsito en dos sentidos, el de la avenida ex Kennedy y el de la calle Florencio Varela. A estos sentidos los llamamos fases. Tenemos la fase 1 (F1) que controla la avenida (doble mano) y la fase 2 (F2) que controla la calle.



# **Una esquina con semáforo**

Vemos que en algunos momentos F1 está en verde y F2 esta en rojo. Podemos asumir por ejemplo tiempo de verde en F1 de 14 segundos (utilizamos números pequeños para simplificar la lógica, pero el diseño es escalable a cualquier número).

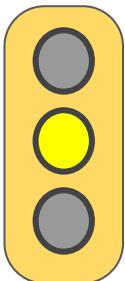


# Una esquina con semáforo

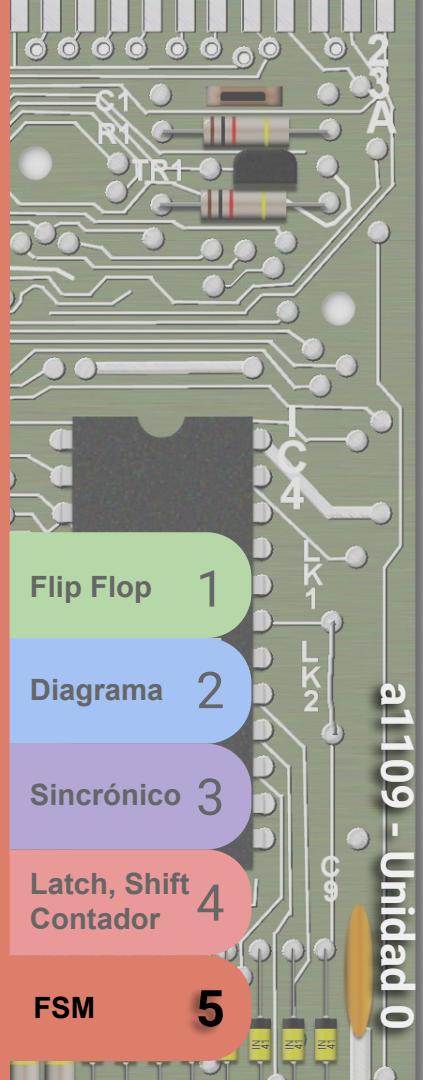
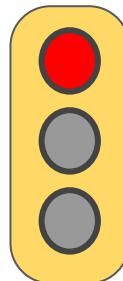
Luego F1 pasa a amarillo (mientras F2 sigue en rojo) durante un tiempo (ej: 5 segundos).



F1



F2

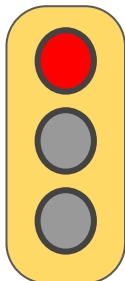


# Una esquina con semáforo

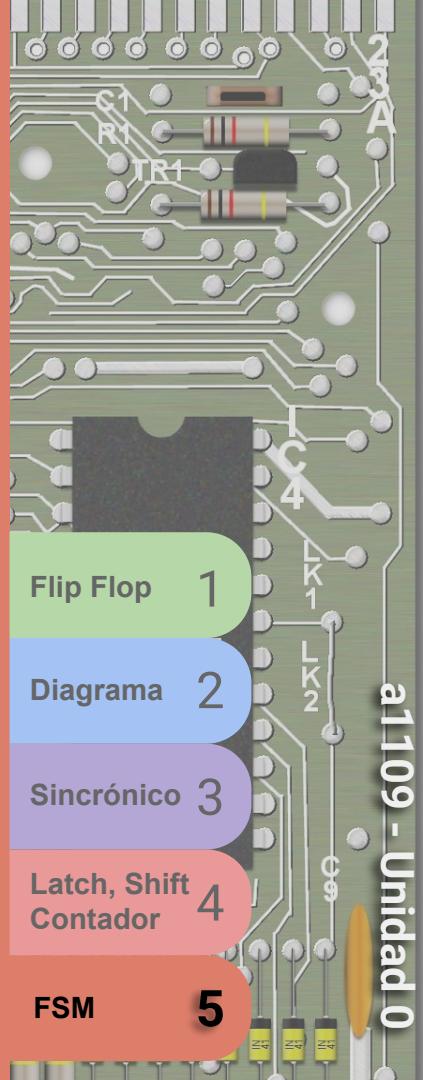
Luego F1 pasa a rojo y podemos poner F2 en verde. Notemos que no hace falta pasar por un amarillo intermedio en F2. Este tiempo de verde puede ser menor que el de F1 para priorizar la avenida, digamos que es 10 segundos.



F1



F2

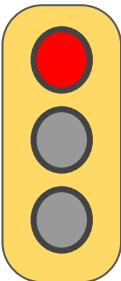


# Una esquina con semáforo

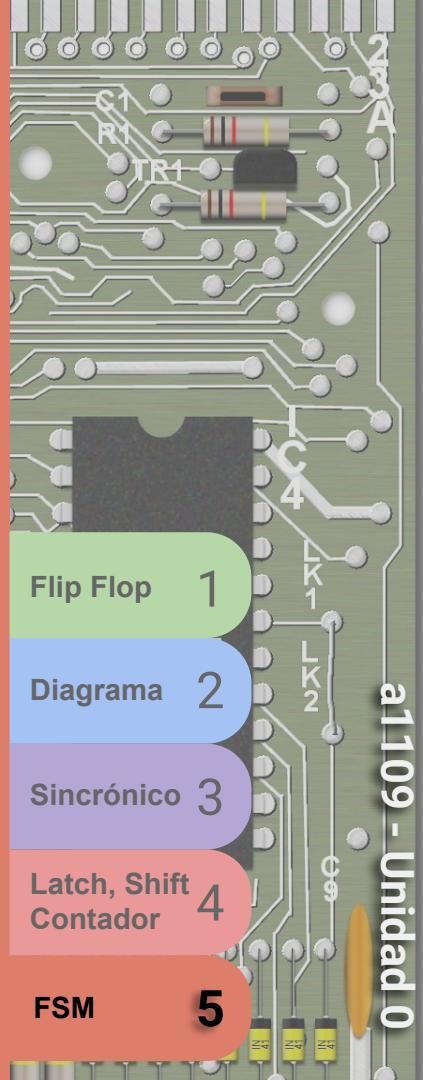
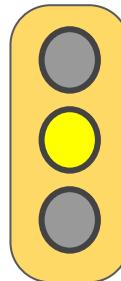
Pasado el tiempo de verde de F2 ahora tenemos amarillo en F2 y esperamos por ejemplo 3 segundos. El siguiente paso es repetir la secuencia volviendo a F1 en verde y F2 en rojo.



F1



F2

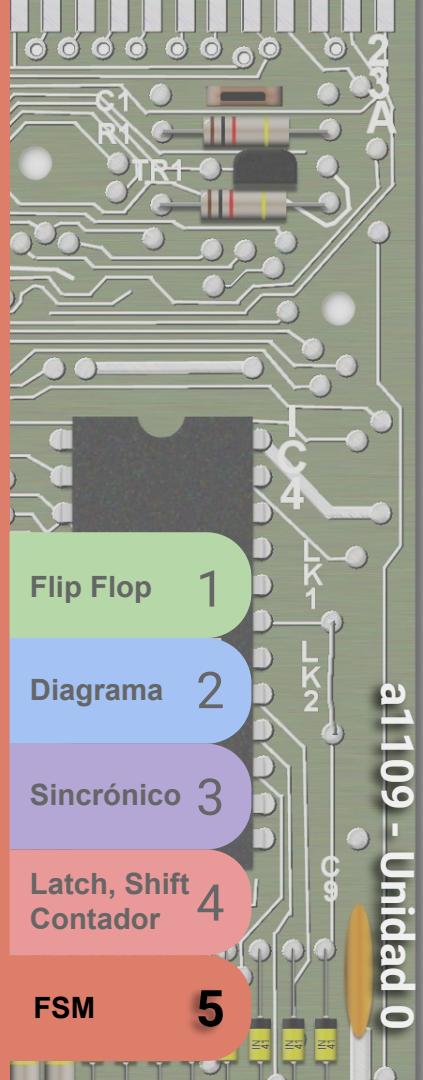


# **Una esquina con semáforo**

Esta secuencia la podemos representar con lo que se conoce un autómata finito, también conocido como máquina de estados finitos. La máquina de estados está compuesta de 3 elementos:

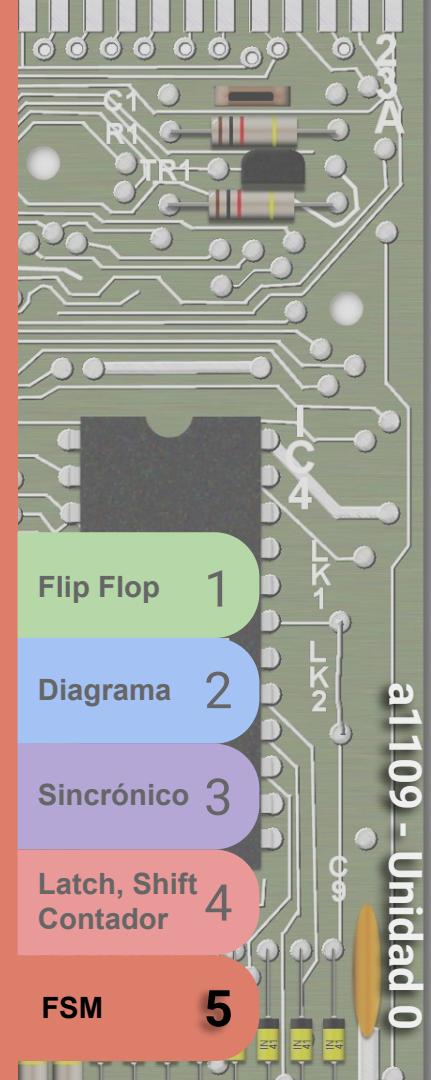
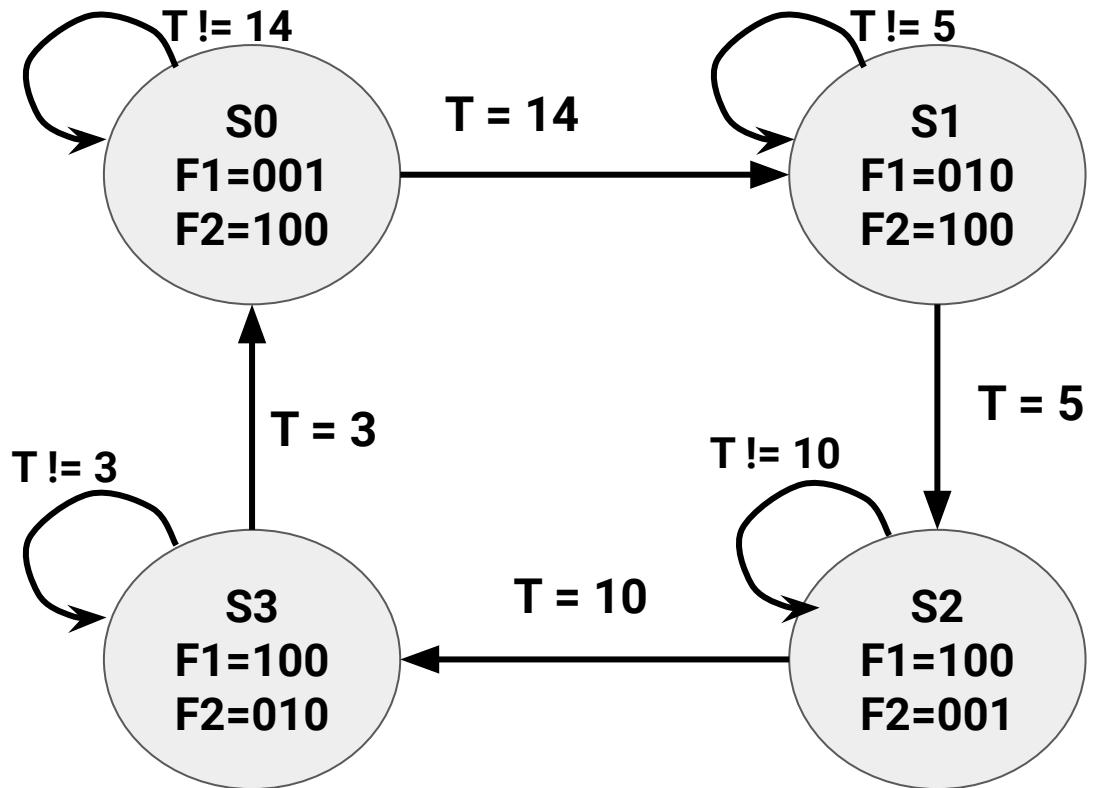
- Entradas
  - Estados
  - Salidas

Luego la máquina de estado representa las transiciones entre estados. Como entrada en el caso del semáforo utilizaremos un contador de tiempos. Por cuestiones de espacio nuestro contador será de 4 bits (pudiendo contar desde 0000 hasta 1111). Las salidas serán las 6 lámparas de F1 y F2 , o sea Rojo1, Amarillo1, Verde1, Rojo2, Amarillo2 y Verde2 (R1,A1,V1,R2,A2,V2). Hemos visto que hay 4 estados posibles, por ende los llamaremos S0, S1, S2 y S3.



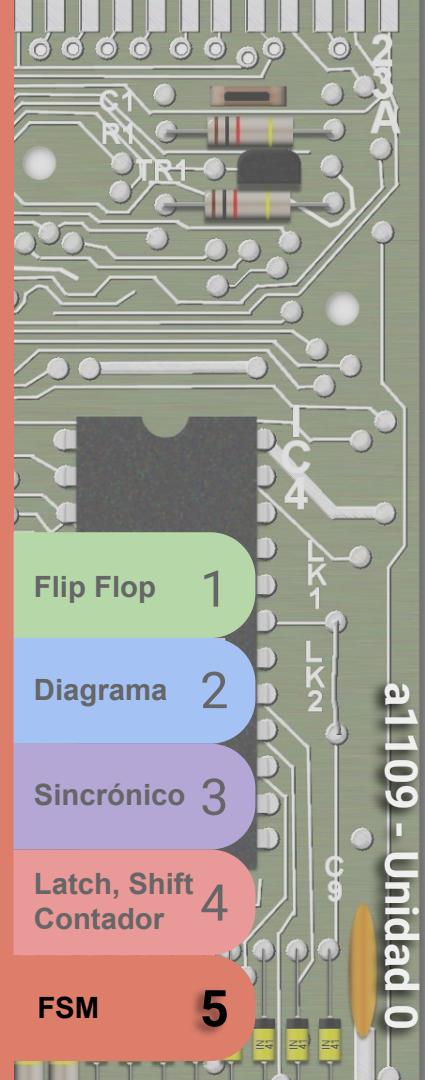
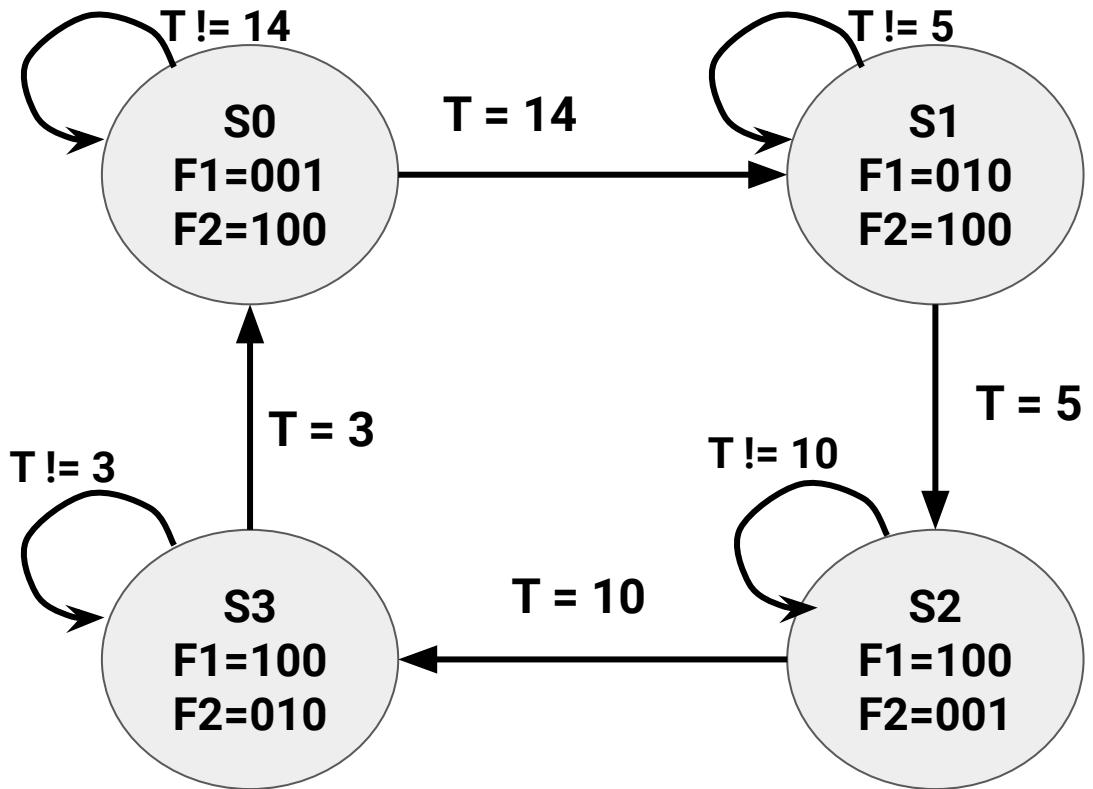
# Máquina de estados finitos

El diagrama representa las transiciones (cuando el tiempo de cada etapa llega al esperado) y el valor de las fases ( $F_n=RAV$ ).



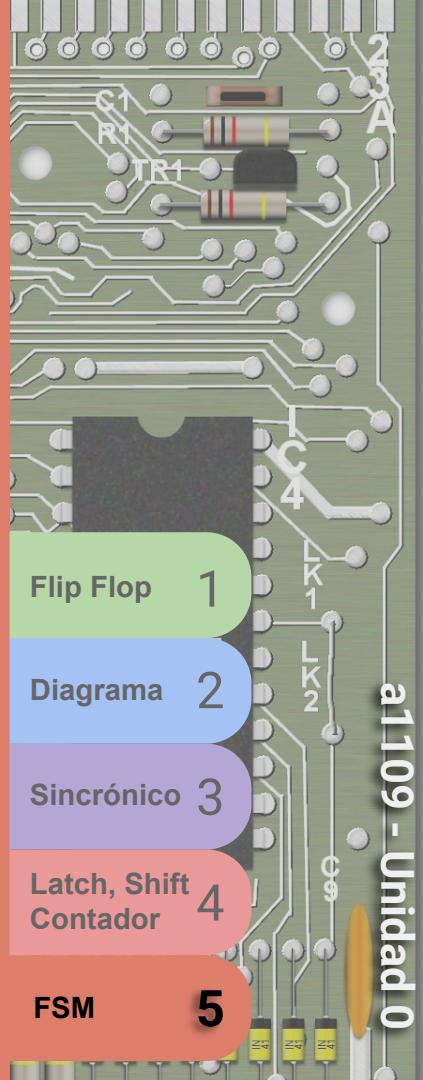
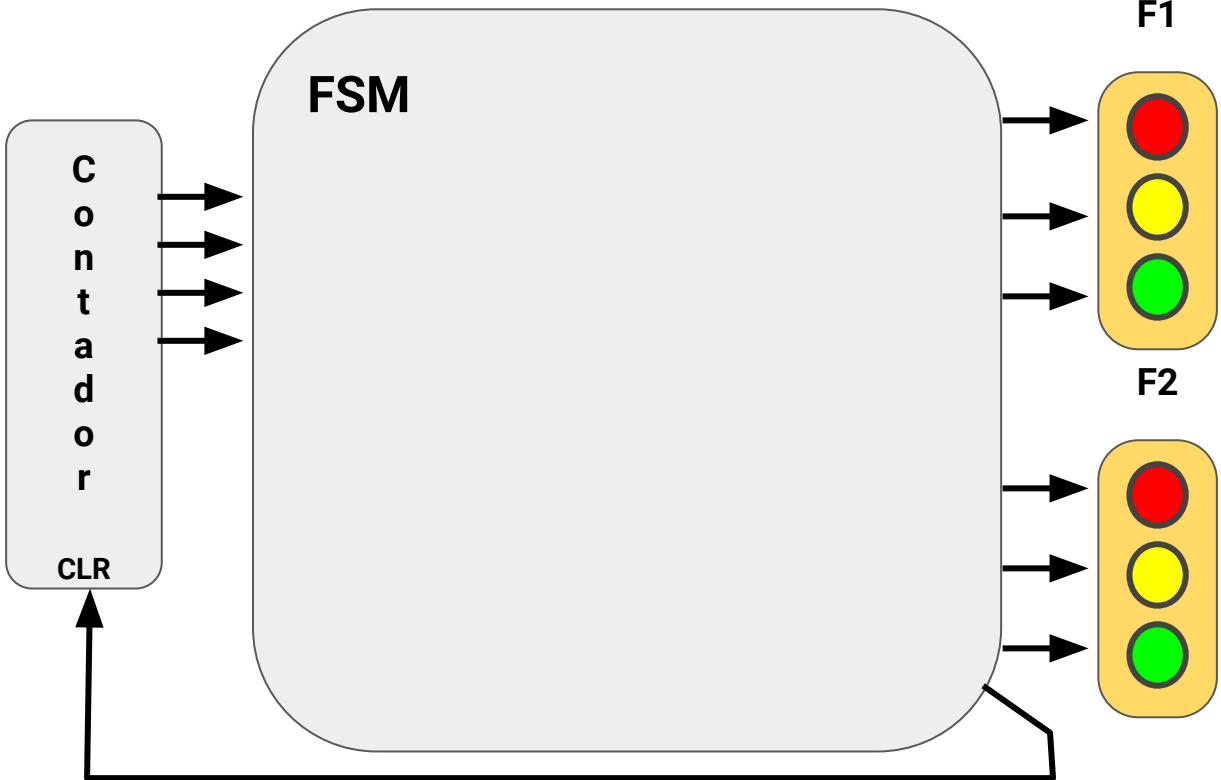
# Máquina de estados finitos

Para memorizar 4 estados usamos FF tipo D (00,01,10,11).



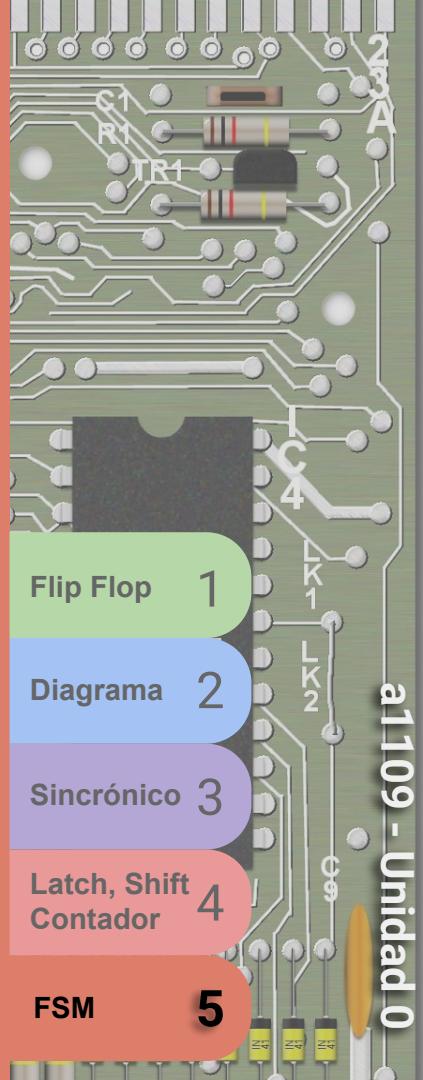
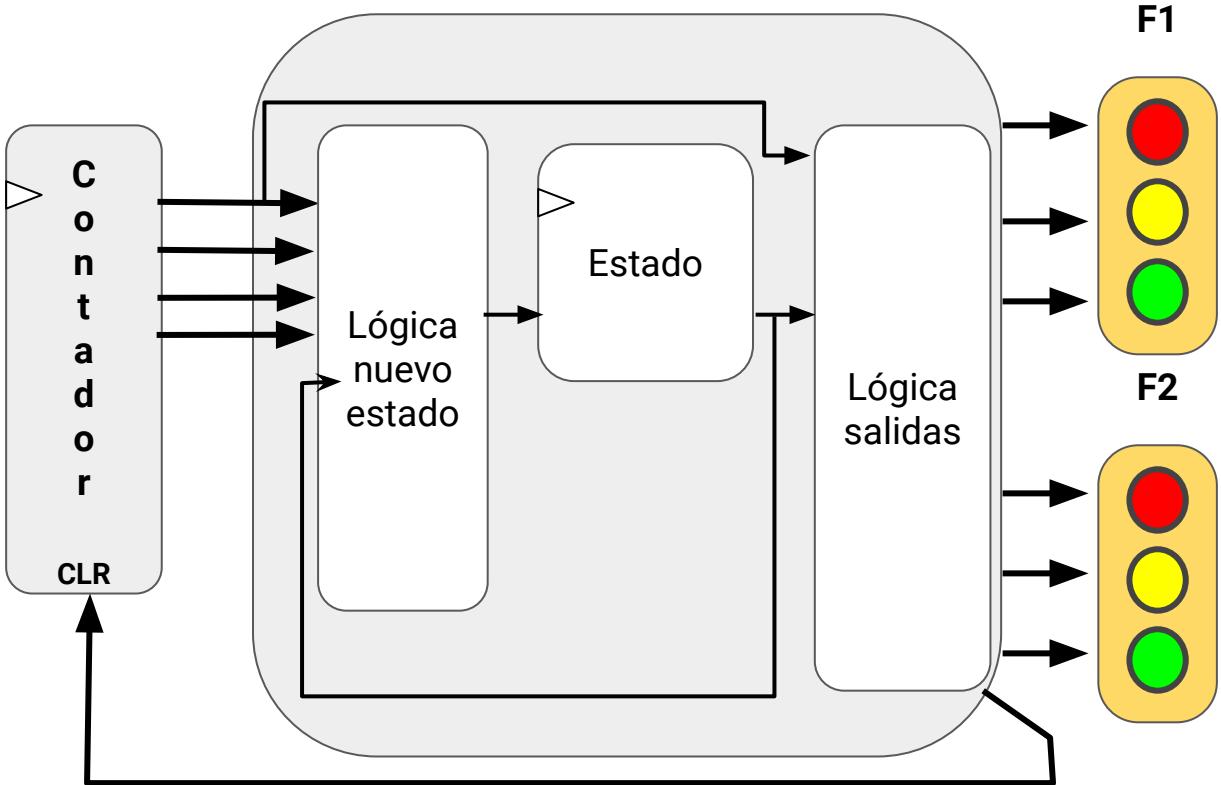
# Máquina de estados finitos (FSM)

Vemos que la FSM tiene como entrada los 4 bits del contador y como salidas las 6 lámparas y una señal de reset para el contador. Con esta última limitamos el módulo del contador.



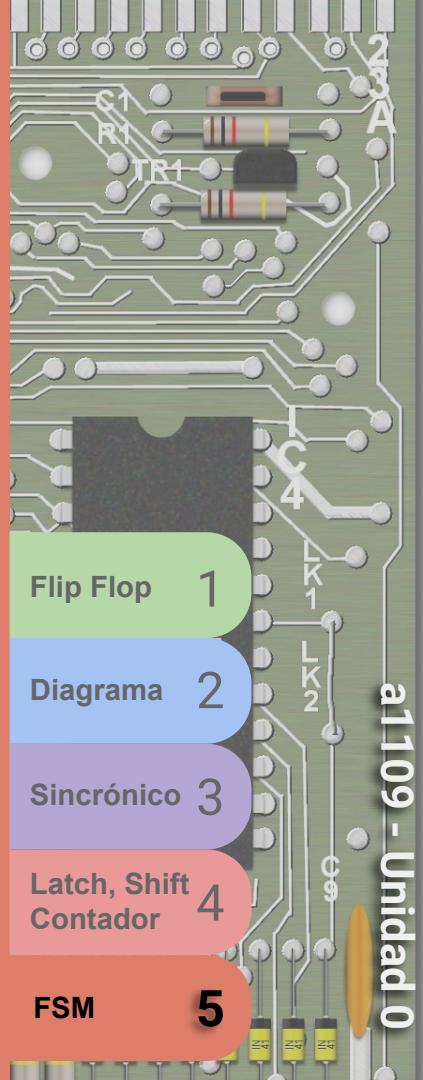
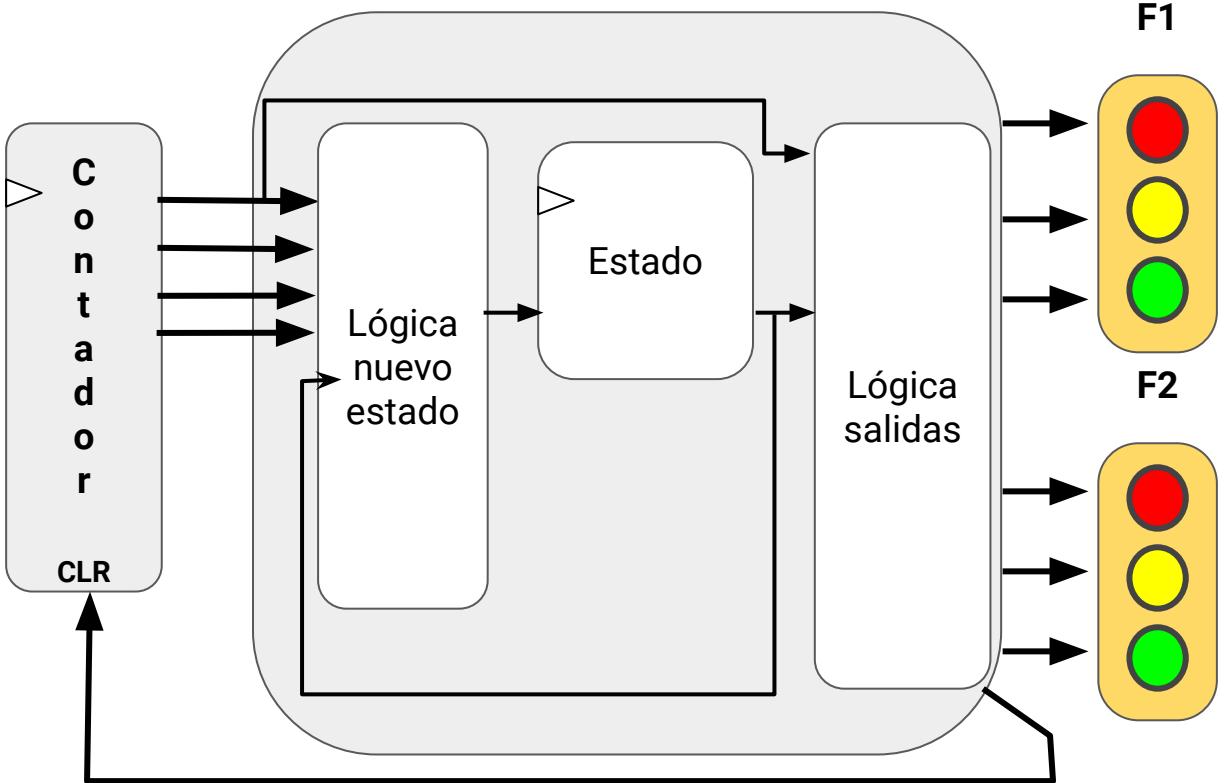
# Máquina de estados finitos (FSM)

Utilizaremos un modelo de FSM llamado de Mealy. En este existe una lógica combinacional que calcula el nuevo estado utilizando las entradas y el estado anterior.



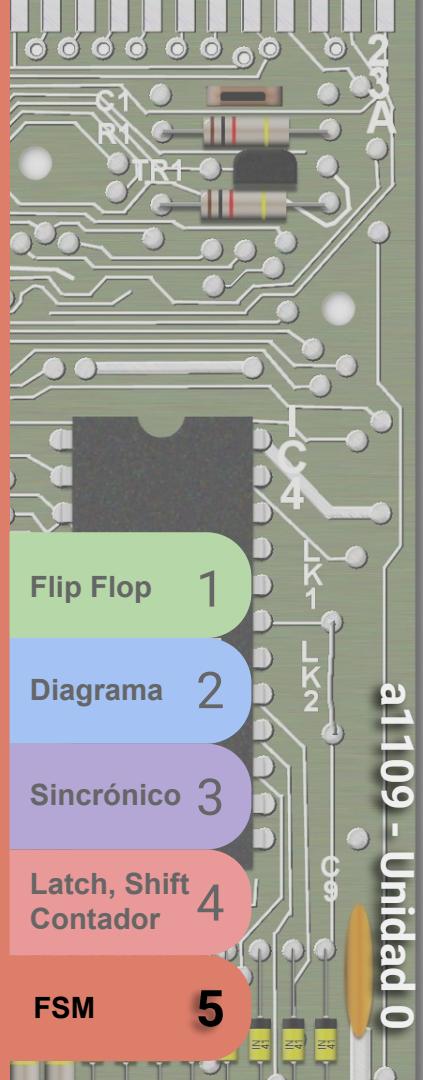
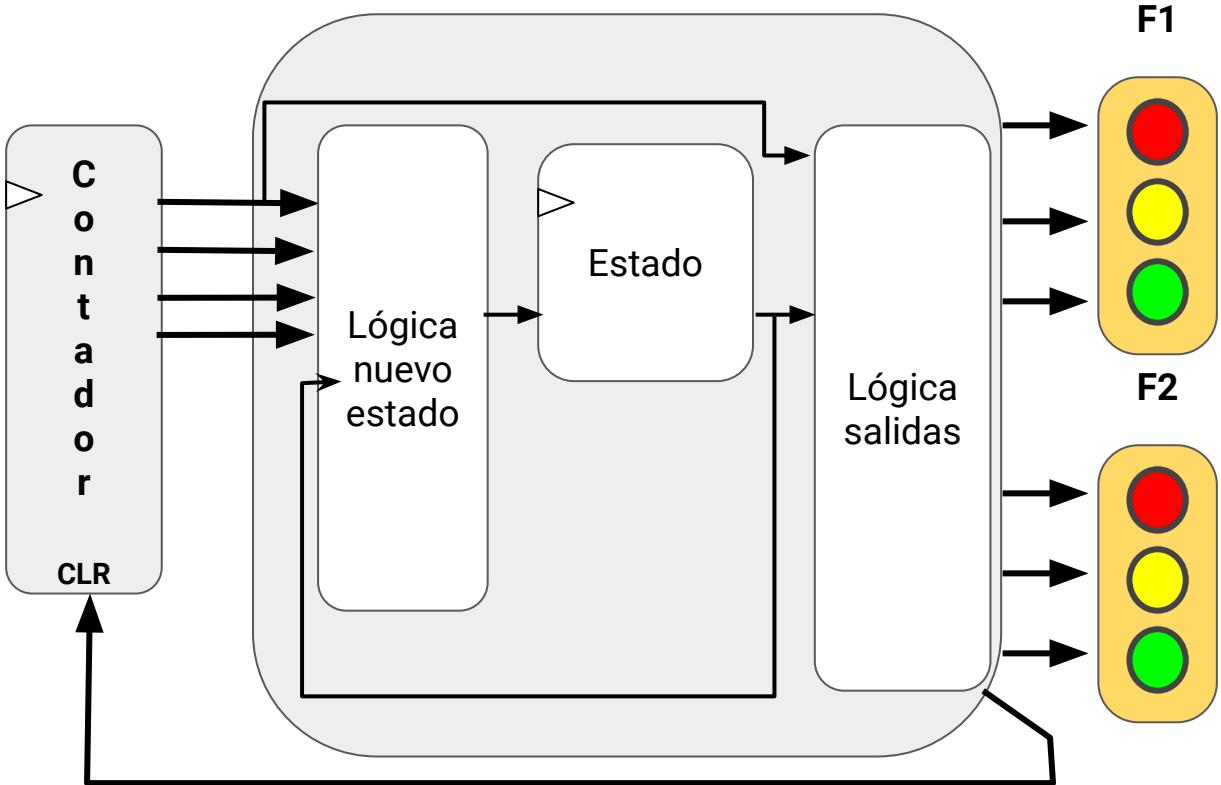
# Máquina de estados finitos (FSM)

La lógica de salida toma el estado actual y las entradas para definir qué valor deben tomar las salidas.



# Máquina de estados finitos (FSM)

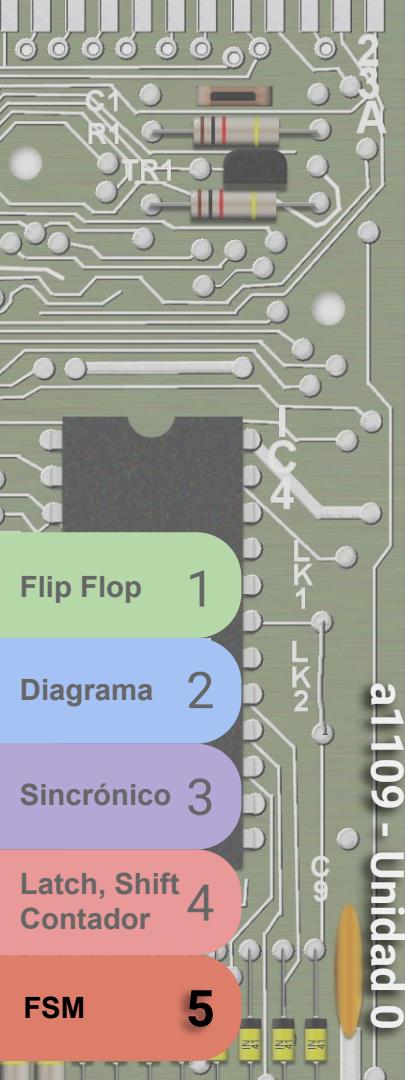
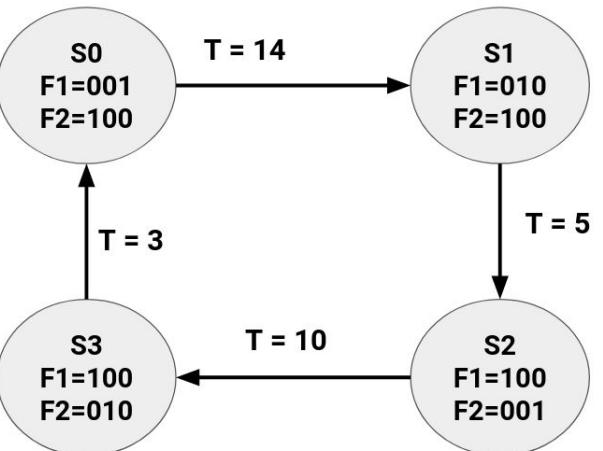
El estado no es otra cosa más que un conjunto de Flip flops, que son el único elemento secuencial de la FSM. Notemos que tiene una entrada de Clock (igual que el contador).



# Máquina de estados finitos (FSM)

Planteamos entonces la lógica de nuevo estado. Vemos que según el estado actual y el valor del contador (recuerden que el contador con reset síncrono toma un valor extra en la cuenta, por eso contamos los tiempos menos uno) se calcula el nuevo estado. Esto podemos implementarlo con una compuerta AND por cada uno de nuevo estado.

<b>Entrada</b>	<b>Estado</b>	<b>Nuevo estado</b>
1101(13)	00	01
0100(4)	01	10
1001(9)	10	11
0010(2)	11	00

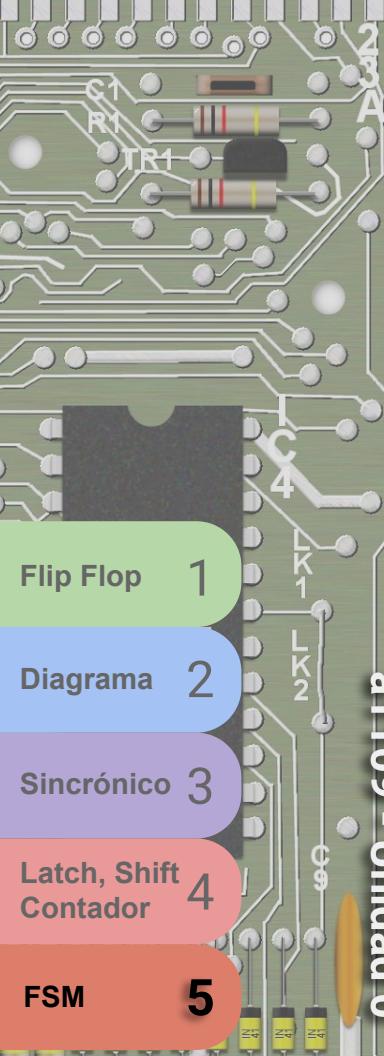
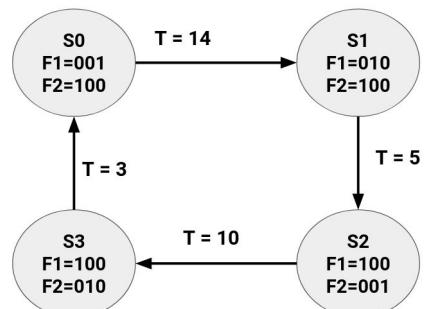
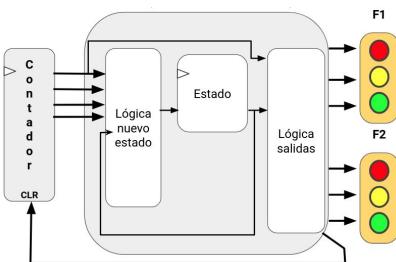


# Máquina de estados finitos (FSM)

Hacemos lo mismo con la lógica de salida. La separamos en dos tablas ya que las salidas F1 y F2 no dependen realmente de la entrada sino que solo del estado. Sin embargo la señal de CLR que resetea el contador depende tanto del estado actual como del valor actual del contador. Esta lógica se encarga de cambiar el módulo del contador de acuerdo al estado actual. Vemos que siempre está en 1 excepto en los cambios.

Entrada Estado CLR		
1101	00	0
0100	01	0
1001	10	0
0010	11	0
XXXX	XX	1

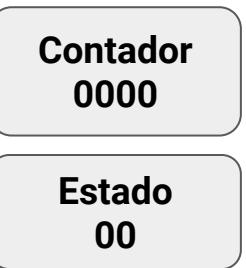
Estado	F1	F2
00	001	100
01	010	100
10	100	001
11	100	010



# Máquina de estados finitos (FSM)

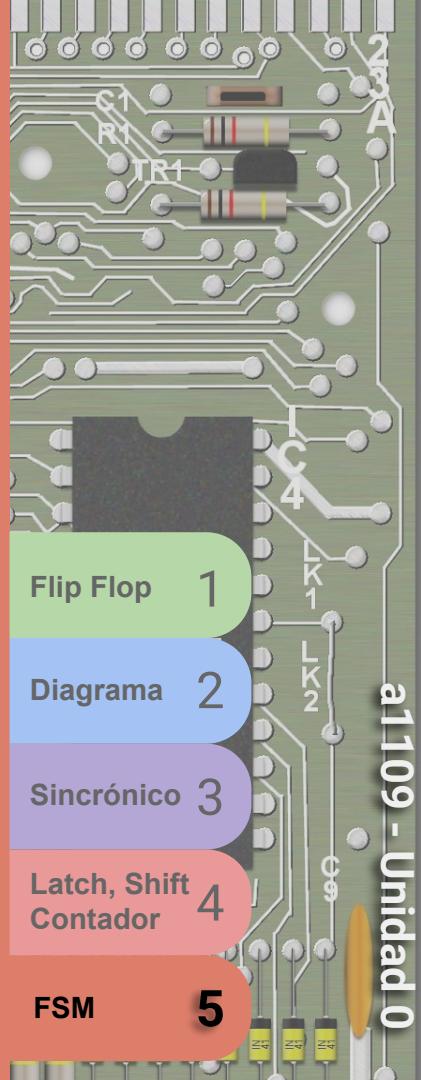
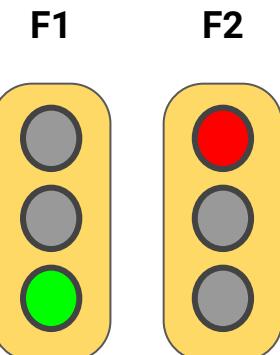
Podemos simular las entradas.. El contador comienza en 0000 y el estado en 00. Vemos que CLR=1 en este caso y que F1 y F2 están en verde y rojo respectivamente. Vemos que no existe una combinación que cambie a un nuevo estado por ende queda en 00.

Entrada	Estado	Nuevo estado
1101	00	01
0100	01	10
1001	10	11
0010	11	00



Entrada	Estado	CLR
1101	00	0
0100	01	0
1001	10	0
0010	11	0
XXXX	XX	1

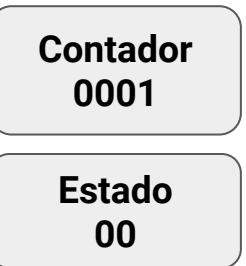
Estado	F1	F2
00	001	100
01	010	100
10	100	001
11	100	010



# Máquina de estados finitos (FSM)

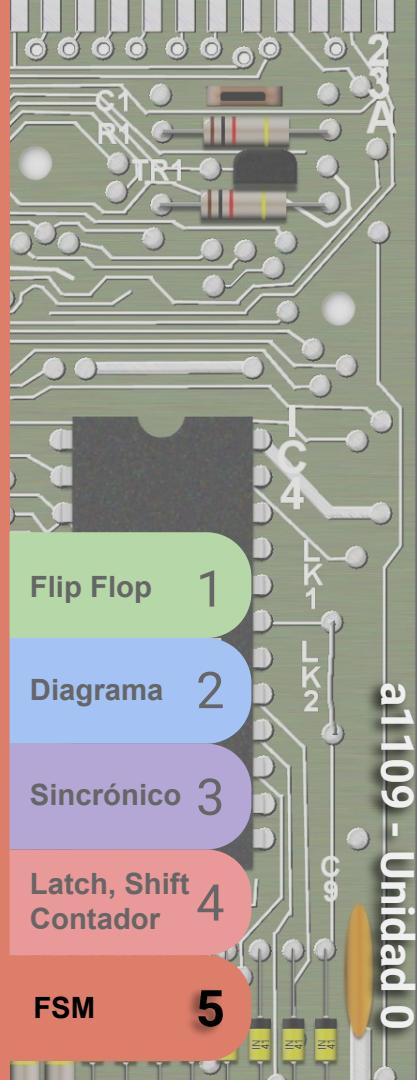
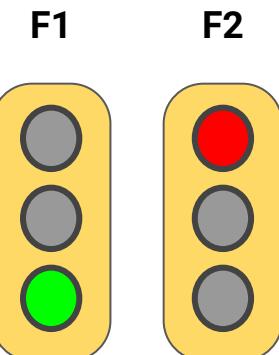
Contador se incrementa en 1, pero vemos que seguimos en el mismo estado 00 y la señal de CLR sigue en 1. Por ahora no hay cambio de estado... seguimos avanzando el contador dejando pasar el tiempo....

Entrada	Estado	Nuevo estado
1101	00	01
0100	01	10
1001	10	11
0010	11	00



Entrada	Estado	CLR
1101	00	0
0100	01	0
1001	10	0
0010	11	0
XXXX	XX	1

Estado	F1	F2
00	001	100
01	010	100
10	100	001
11	100	010



# Máquina de estados finitos (FSM)

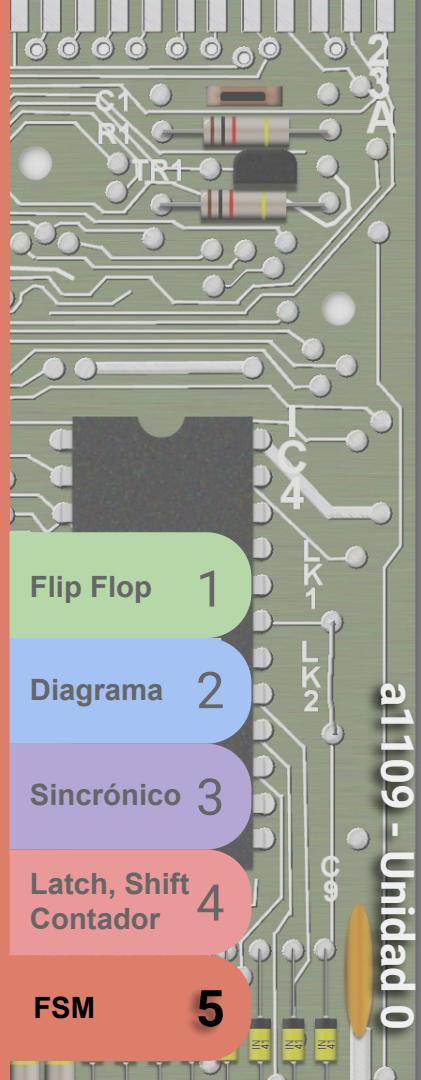
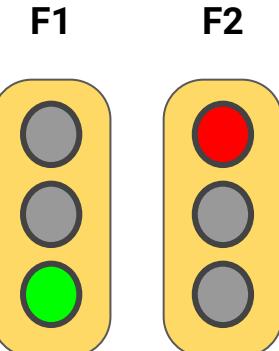
Ahora el contador llega a 1101, y vemos que ahora identificamos este valor en las tablas... Cuando entrada=1101 y Estado=00, entonces nuevo estado es 01 y la señal CLR=0.

Entrada	Estado	Nuevo estado
1101	00	01
0100	01	10
1001	10	11
0010	11	00



Entrada	Estado	CLR
1101	00	0
0100	01	0
1001	10	0
0010	11	0
XXXX	XX	1

Estado	F1	F2
00	001	100
01	010	100
10	100	001
11	100	010



# Máquina de estados finitos (FSM)

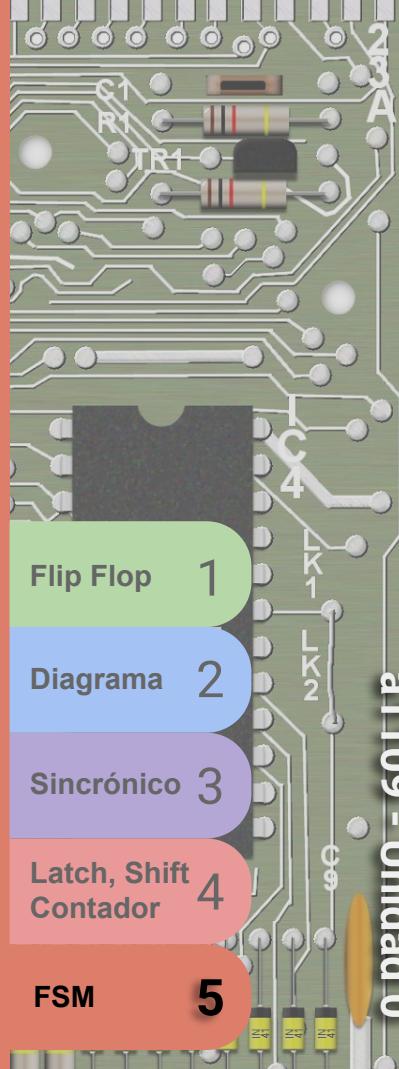
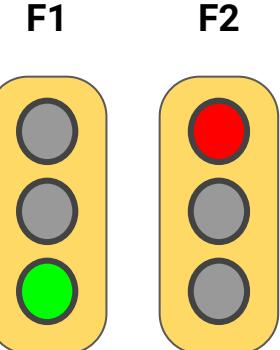
La lógica combinacional cambia el valor de estado y el CLR en 0 va a resetear el contador para que en el próximo ciclo comience en 0000.

Entrada	Estado	Nuevo estado
1101	00	01
0100	01	10
1001	10	11
0010	11	00



Entrada	Estado	CLR
1101	00	0
0100	01	0
1001	10	0
0010	11	0
XXXX	XX	1

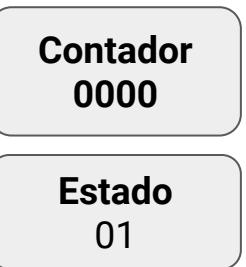
Estado	F1	F2
00	001	100
01	010	100
10	100	001
11	100	010



# Máquina de estados finitos (FSM)

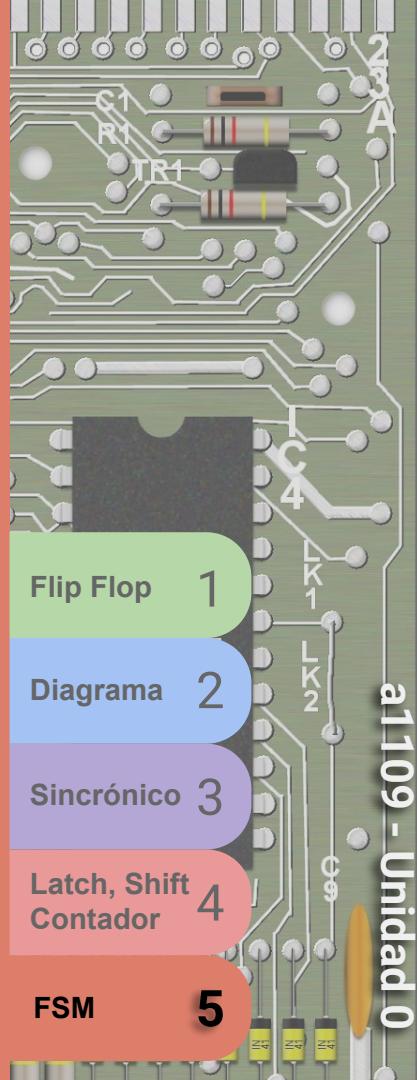
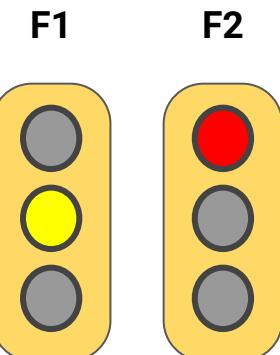
Ahora que el contador vuelve a 0000, el valor de CLR pasa de nuevo a 1 (dejando el contador libre para correr) y la lógica de salida decodifica estado 01, por ende F1=Amarillo y F2=Rojo.

Entrada	Estado	Nuevo estado
1101	00	01
0100	01	10
1001	10	11
0010	11	00



Entrada	Estado	CLR
1101	00	0
0100	01	0
1001	10	0
0010	11	0
XXXX	XX	1

Estado	F1	F2
00	001	100
01	010	100
10	100	001
11	100	010



# Máquina de estados finitos (FSM)

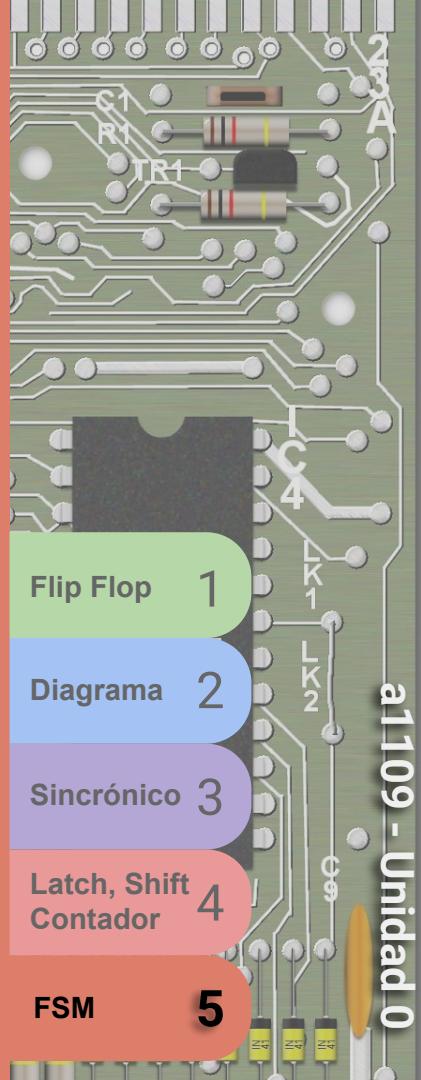
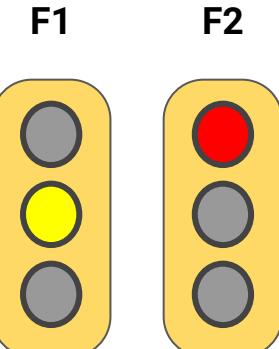
Sigue pasando el tiempo y el contador llega a 0100. Vemos que para entrada=0100 y estado=01 tenemos nuevo estado=10. También esta combinación hace CLR=0

Entrada	Estado	Nuevo estado
1101	00	01
0100	01	10
1001	10	11
0010	11	00



Entrada	Estado	CLR
1101	00	0
0100	01	0
1001	10	0
0010	11	0
XXXX	XX	1

Estado	F1	F2
00	001	100
01	010	100
10	100	001
11	100	010



# Máquina de estados finitos (FSM)

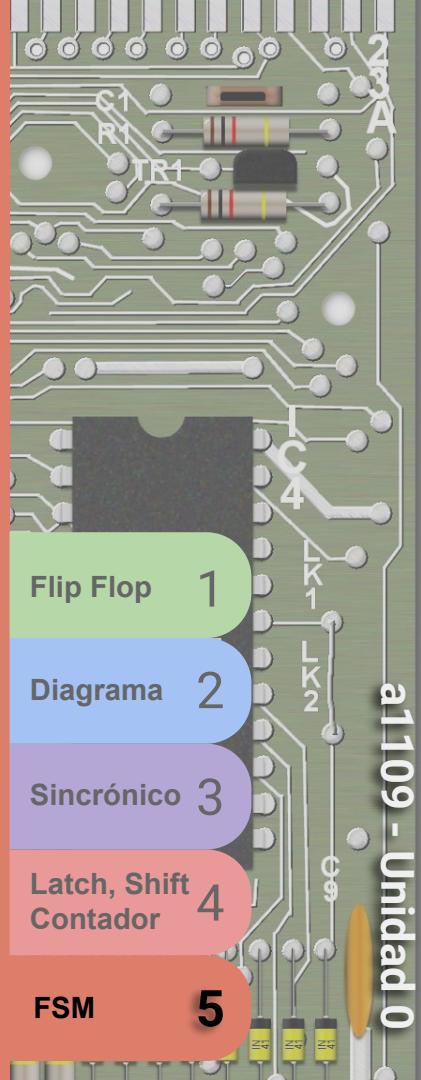
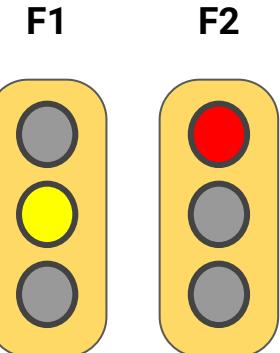
Por ende actualizamos el registro de estado y la salida CLR.

Entrada	Estado	Nuevo estado
1101	00	01
0100	01	10
1001	10	11
0010	11	00



Entrada	Estado	CLR
1101	00	0
0100	01	0
1001	10	0
0010	11	0
XXXX	XX	1

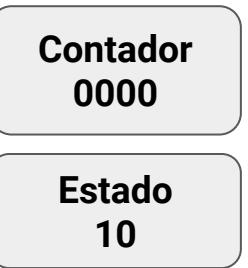
Estado	F1	F2
00	001	100
01	010	100
10	100	001
11	100	010



# Máquina de estados finitos (FSM)

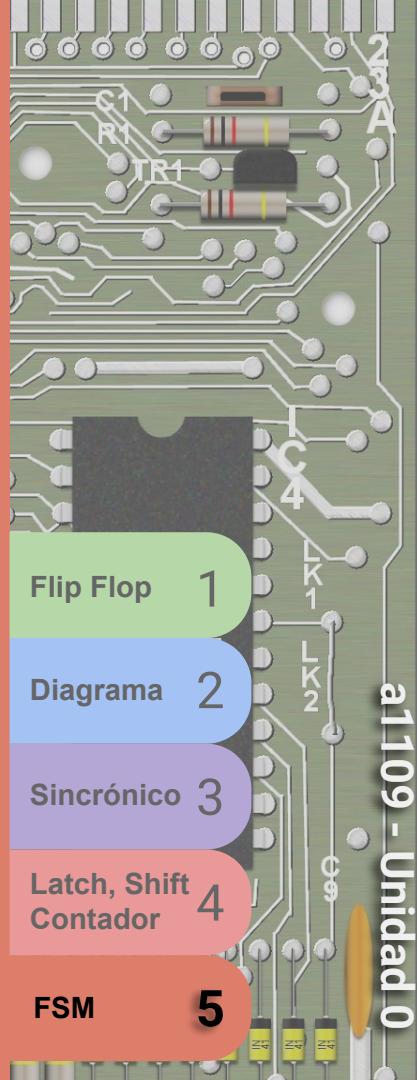
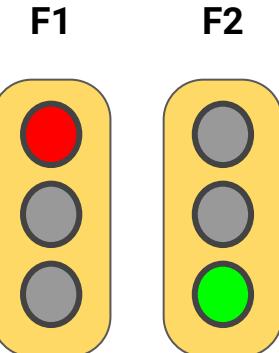
Vemos que ahora las salidas F1 y F2 representan el nuevo estado y el contador vuelve a 0000, lo cual cambia la señal CLR que ahora vuelve a estar en 1 dejando libre al contador para que corra.

Entrada	Estado	Nuevo estado
1101	00	01
0100	01	10
1001	10	11
0010	11	00



Entrada	Estado	CLR
1101	00	0
0100	01	0
1001	10	0
0010	11	0
XXXX	XX	1

Estado	F1	F2
00	001	100
01	010	100
10	100	001
11	100	010



# Máquina de estados finitos (FSM)

Pasa de nuevo el tiempo y llegamos al valor 1001 del contador.

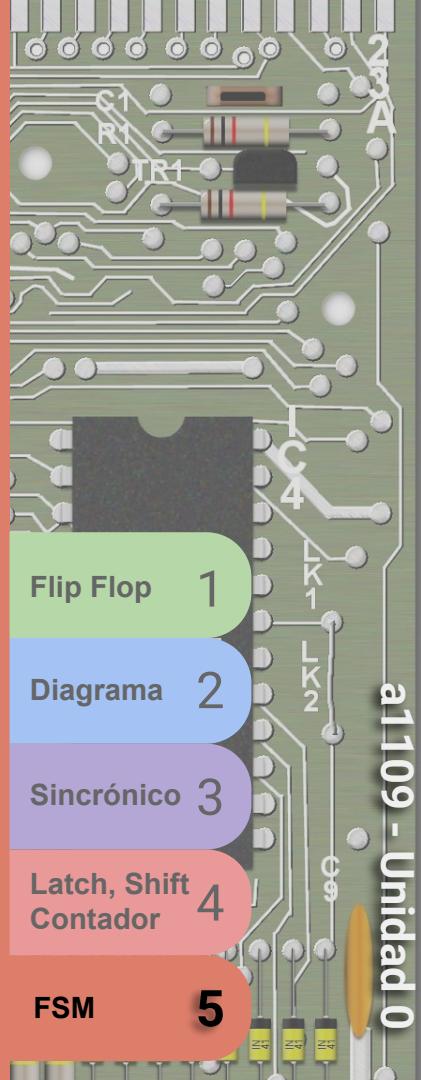
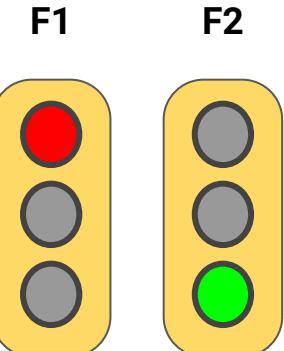
Para entrada=1001 y estado=10 tenemos que nuevo estado es igual a 11, CLR=0 y van a cambiar F1 y F2.

Entrada	Estado	Nuevo estado
1101	00	01
0100	01	10
1001	10	11
0010	11	00



Entrada	Estado	CLR
1101	00	0
0100	01	0
1001	10	0
0010	11	0
XXXX	XX	1

Estado	F1	F2
00	001	100
01	010	100
10	100	001
11	100	010



# Máquina de estados finitos (FSM)

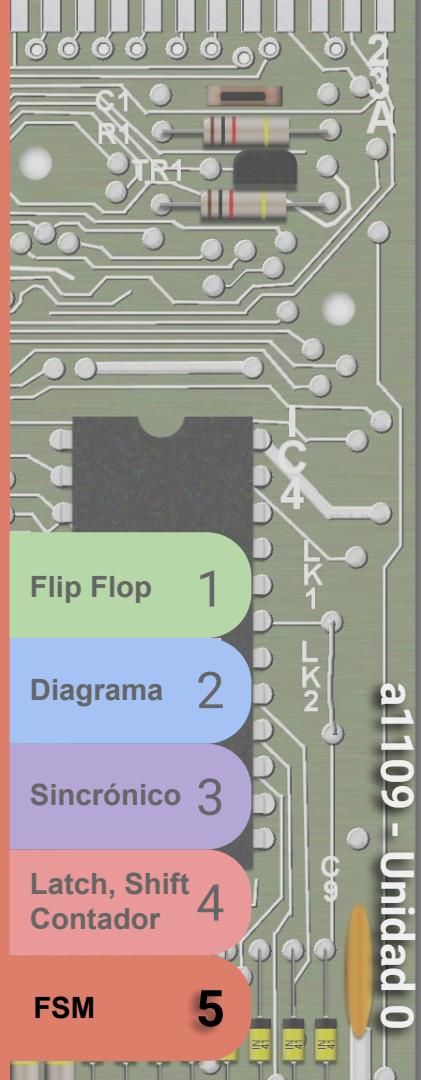
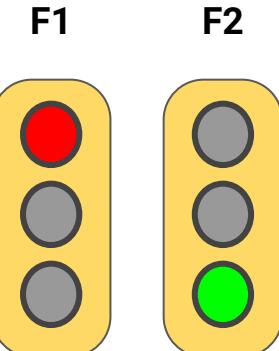
La señal CLR resetea el contador que va a volver a 0000.

Entrada	Estado	Nuevo estado
1101	00	01
0100	01	10
1001	10	11
0010	11	00



Entrada	Estado	CLR
1101	00	0
0100	01	0
1001	10	0
0010	11	0
XXXX	XX	1

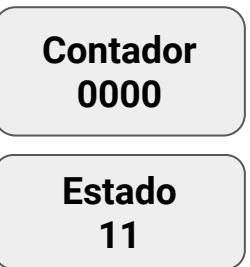
Estado	F1	F2
00	001	100
01	010	100
10	100	001
11	100	010



# Máquina de estados finitos (FSM)

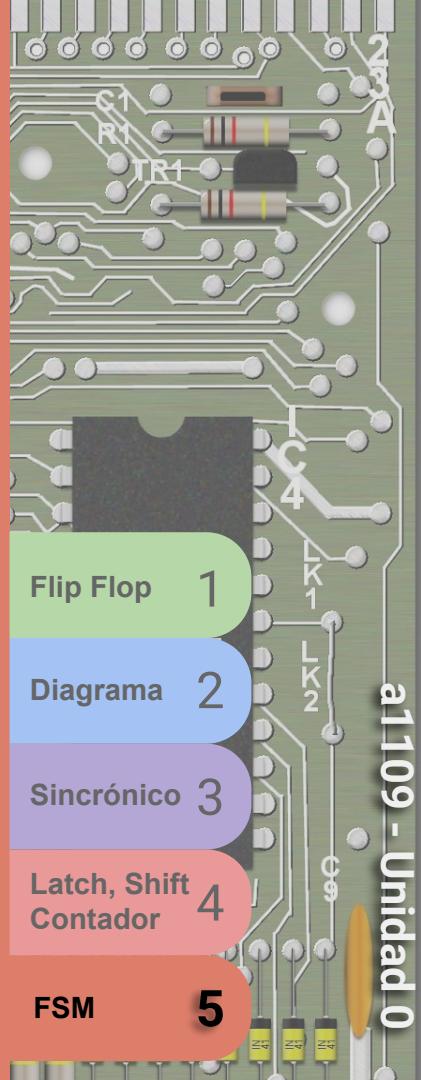
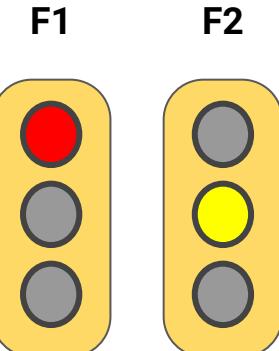
Ahora que F1 y F2 representan este estado, CLR vuelve a 1, y contador comienza la cuenta nuevamente....

Entrada	Estado	Nuevo estado
1101	00	01
0100	01	10
1001	10	11
0010	11	00



Entrada	Estado	CLR
1101	00	0
0100	01	0
1001	10	0
0010	11	0
XXXX	XX	1

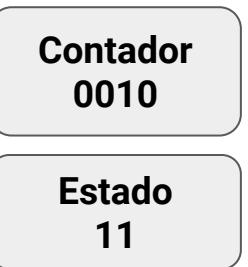
Estado	F1	F2
00	001	100
01	010	100
10	100	001
11	100	010



# Máquina de estados finitos (FSM)

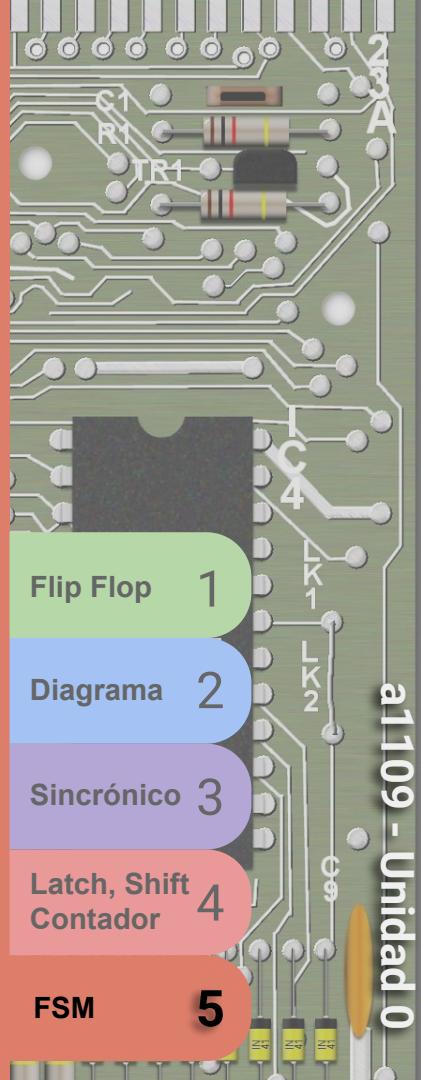
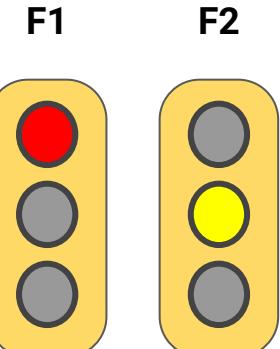
Ahora entrada=0010 y estado=11 generan el nuevo estado 00.  
CLR pasa de nuevo a 0 y se actualizan F1 y F2.

Entrada	Estado	Nuevo estado
1101	00	01
0100	01	10
1001	10	11
0010	11	00



Entrada	Estado	CLR
1101	00	0
0100	01	0
1001	10	0
0010	11	0
XXXX	XX	1

Estado	F1	F2
00	001	100
01	010	100
10	100	001
11	100	010



# Máquina de estados finitos (FSM)

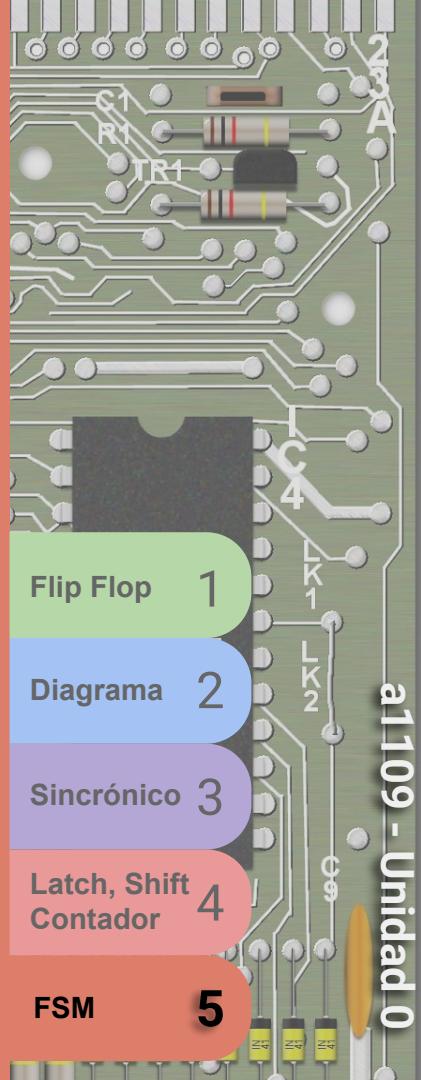
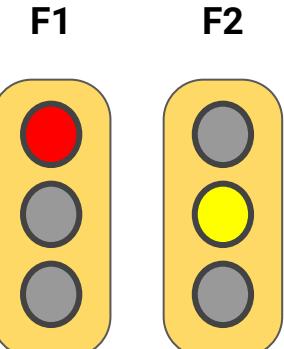
Cambia el estado y se resetea el contador..

Entrada	Estado	Nuevo estado
1101	00	01
0100	01	10
1001	10	11
0010	11	00



Entrada	Estado	CLR
1101	00	0
0100	01	0
1001	10	0
0010	11	0
XXXX	XX	1

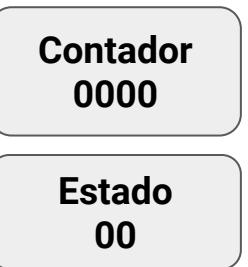
Estado	F1	F2
00	001	100
01	010	100
10	100	001
11	100	010



# Máquina de estados finitos (FSM)

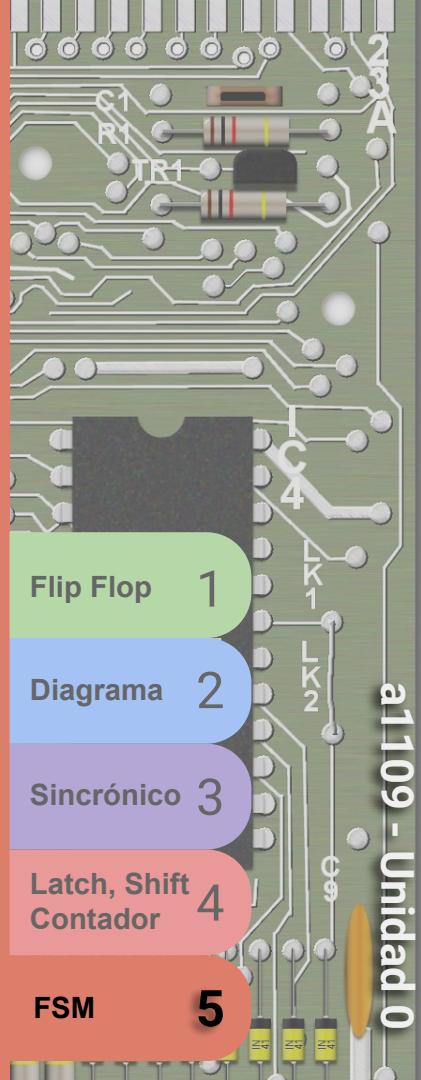
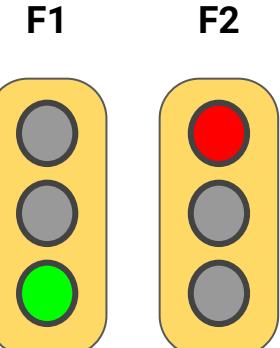
Volvemos al estado 00 original, y la secuencia vuelve a comenzar.  
Hemos simulado todos los estados con sus correspondientes entradas y salidas.

Entrada	Estado	Nuevo estado
1101	00	01
0100	01	10
1001	10	11
0010	11	00



Entrada	Estado	CLR
1101	00	0
0100	01	0
1001	10	0
0010	11	0
XXXX	XX	1

Estado	F1	F2
00	001	100
01	010	100
10	100	001
11	100	010



## Parte combinacional

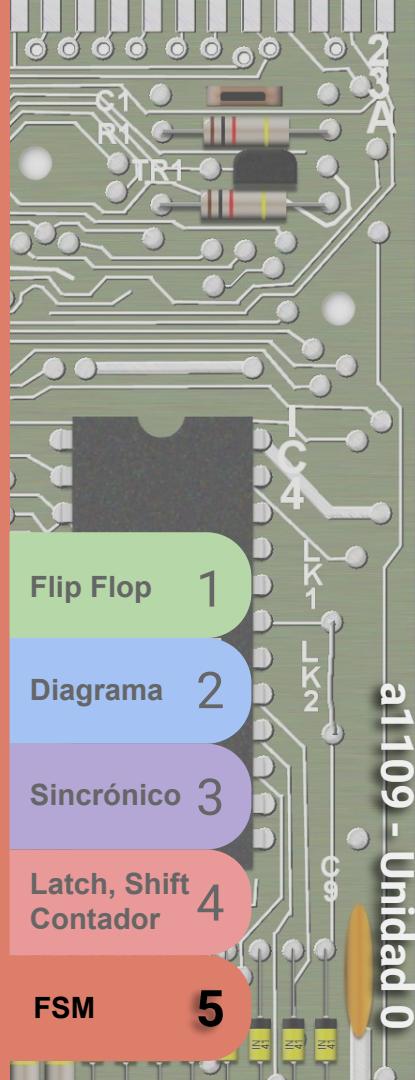
La primer tabla representa la lógica combinacional de salida (basado en el estado). Vemos que son 6 salidas y su implementación es trivial (por ejemplo  $R_1=S_1$ ,  $R_2=\neg S_1$ ,  $A_1=\neg S_1 \cdot S_0$  ...etc).

Estado	F1	F2
$S_1 S_0$	$R_1 A_1 V_1$	$R_2 A_2 V_2$
00	001	100
01	010	100
10	100	001
11	100	010

En el caso de CLR, cada 0 es una compuerta AND de 6 entradas identificando las entradas (por ejemplo  $A \cdot B \cdot \neg C \cdot D \cdot \neg S_1 \cdot \neg S_0$  sería una entrada a identificar) y luego todas las AND se unen con una NOR (ya que debe generar 0 cuando se encuentran los casos).

Entrada	Estado	CLR
ABCD	$S_1 S_0$	
1101	00	0
0100	01	0
1001	10	0
0010	11	0
XXXX	XX	1

Esto no reviste mayor complejidad, son circuitos simples de implementar.



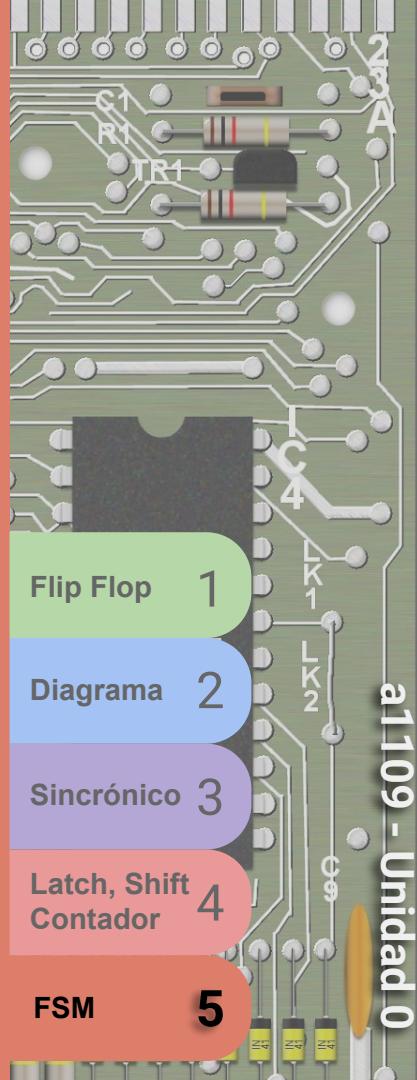
## Parte Secuencial

La tabla de transición de estados que usamos es muy útil para identificar los cambios, pero es una tabla reducida.

Entrada	Estado	Nuevo estado
1101	00	01
0100	01	10
1001	10	11
0010	11	00

Notemos que no se representa que ocurre cuando las entradas toman otros valores, como 0000, 0001, etc. Una forma más correcta de expresarla sería la siguiente.

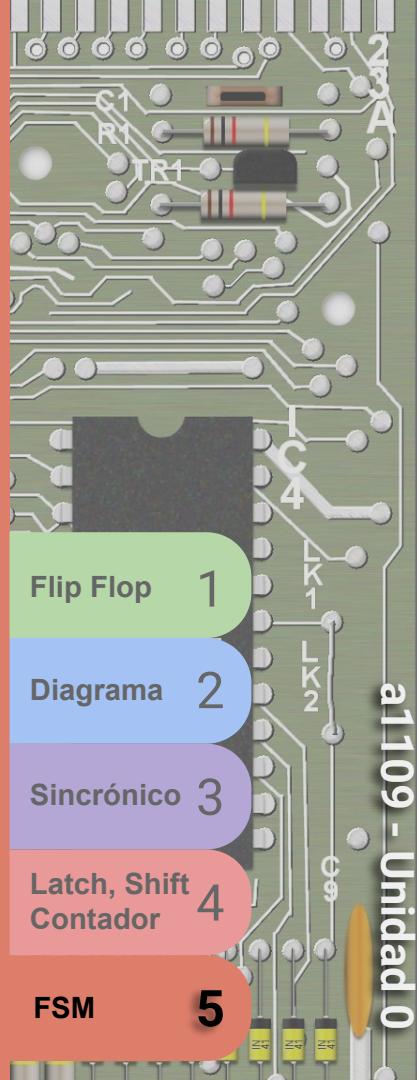
Entrada	Estado	Nuevo estado
1101	00	01
0100	01	10
1001	10	11
0010	11	00
XXXX	XX	Estado <sub>N-1</sub>



## Parte Secuencial

Vemos ahora que para cualquier estado no especificado el valor de nuevo estado es igual a estado. O sea, mantiene el valor. Dado que las entradas provienen de un contador las mismas tomarán valores más allá de los representados en la tabla. Si lo pensamos, hay 4 bits de entrada y 2 de estado.. Por ende tenemos 6 bits efectivos lo cual daría 64 combinaciones en la tabla.

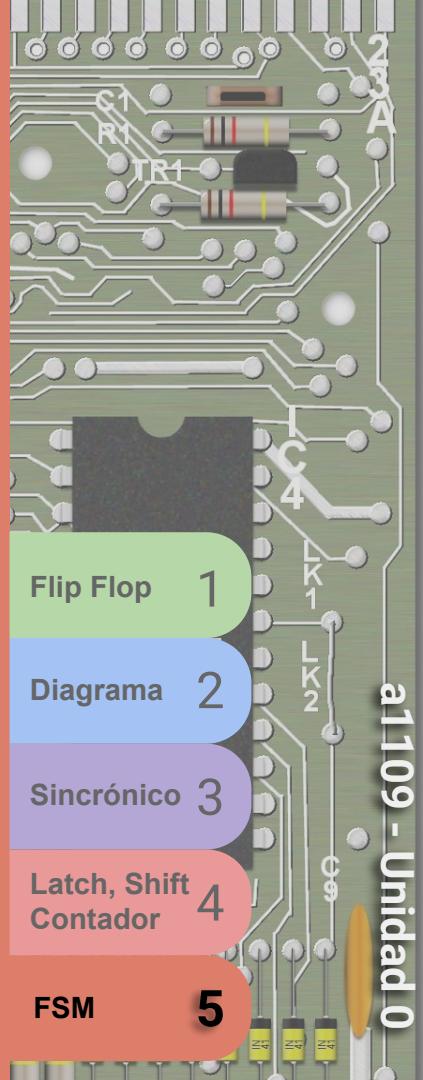
Entrada	Estado	Nuevo estado
1101	00	01
0100	01	10
1001	10	11
0010	11	00
0000	00	Estado <sub>N-1</sub>
.		
.		
.		
1111	11	Estado <sub>N-1</sub>



## Parte Secuencial

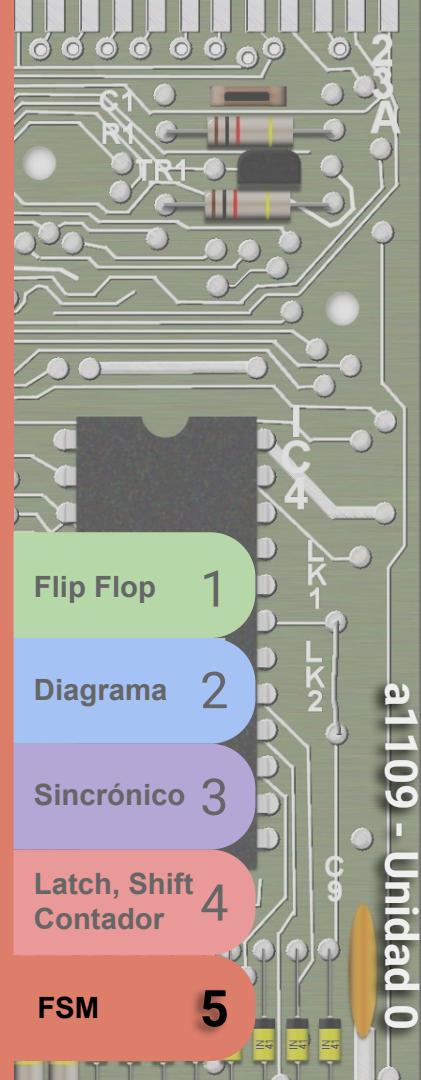
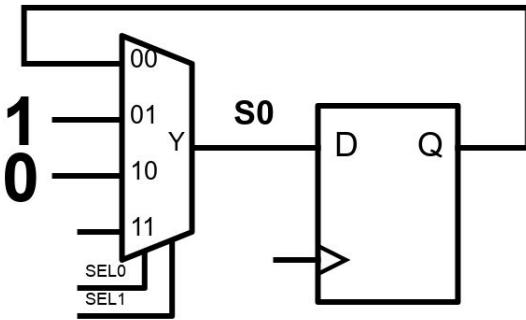
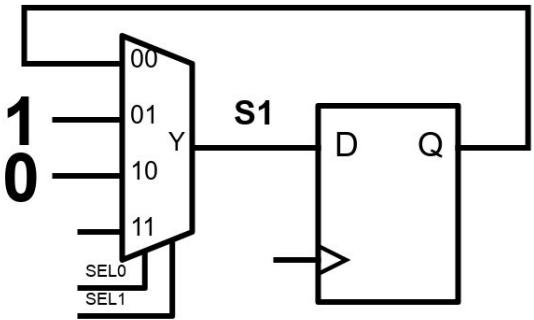
Vemos que resolver esta tabla de verdad para Nuevo estado ya no es trivial, porque debemos completar el caso de Estado<sub>N-1</sub> para cada una de las 64 combinaciones donde se mantenga el estado.

Entrada	Estado	Nuevo estado
1101	00	01
0100	01	10
1001	10	11
0010	11	00
0000	00	Estado <sub>N-1</sub>
.	.	
.	.	
1111	11	Estado <sub>N-1</sub>



## Parte Secuencial

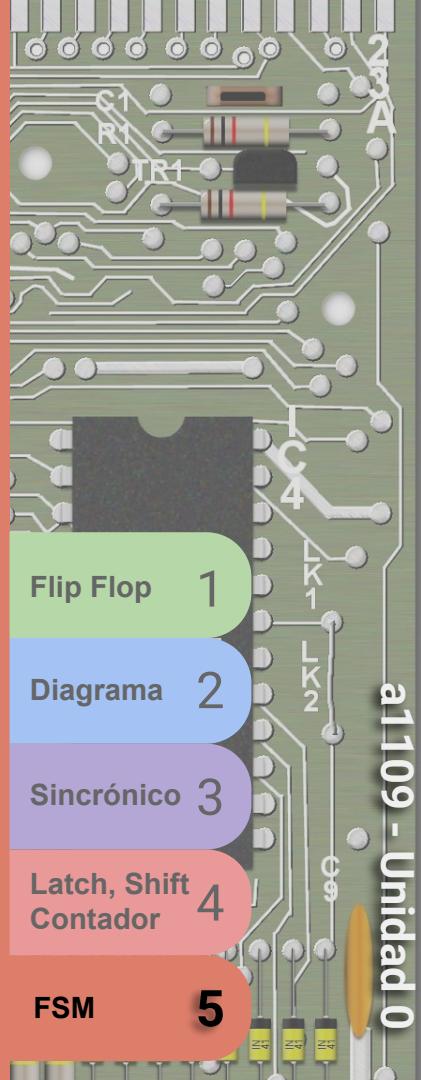
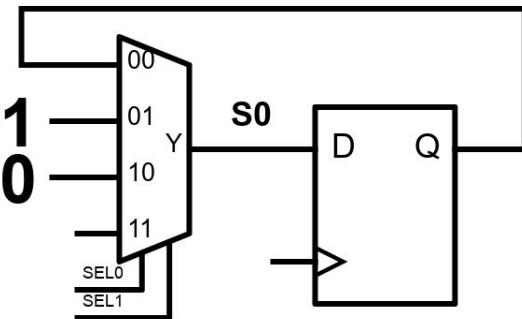
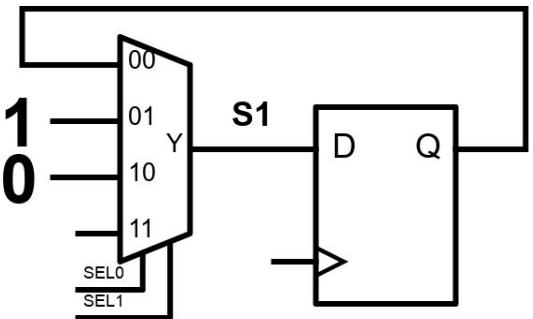
Dado que en la gran mayoría de las combinaciones de entrada y estado tenemos que repetir el valor de estado, simplemente planteamos un circuito como el que se ve abajo. En este circuito cada FF se realimenta y su entrada se selecciona con un MUX. Si selección del mux es 00, el FF mantiene estado. Si selección es igual a 01, el FF toma un valor 1 forzado, y el selección es igual a 10 el FF toma un valor 0 forzado. Vemos que ahora para controlar el nuevo estado necesitamos dos bits por cada FF tipo D. Sin embargo esto simplifica mucho la tabla de verdad.



## Parte Secuencial

Vemos que si bien ahora controlamos Sel S1 y Sel S0 y donde antes teníamos 0 ahora tenemos 10 y donde teníamos 1 ahora tenemos 01, todos los estados donde mantenemos valor toman 00. Esto simplifica mucho la implementación ya que solo nos tenemos que encargar de los unos en Sel S0 y Sel S1

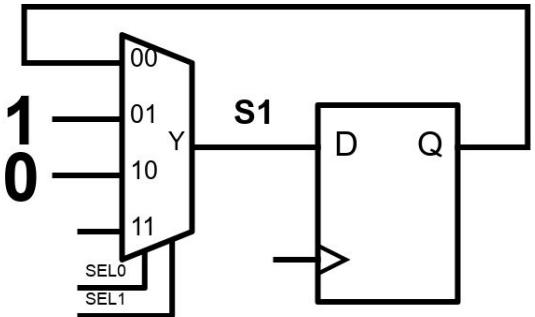
Entrada	Estado	Sel S1	Sel S0
1101	00	10 (0)	01 (1)
0100	01	01(1)	10(0)
1001	10	01(1)	01(1)
0010	11	10(0)	10(0)
0000	00	00	00
.	.		
.	.		
.	.		
1111	11	00	00



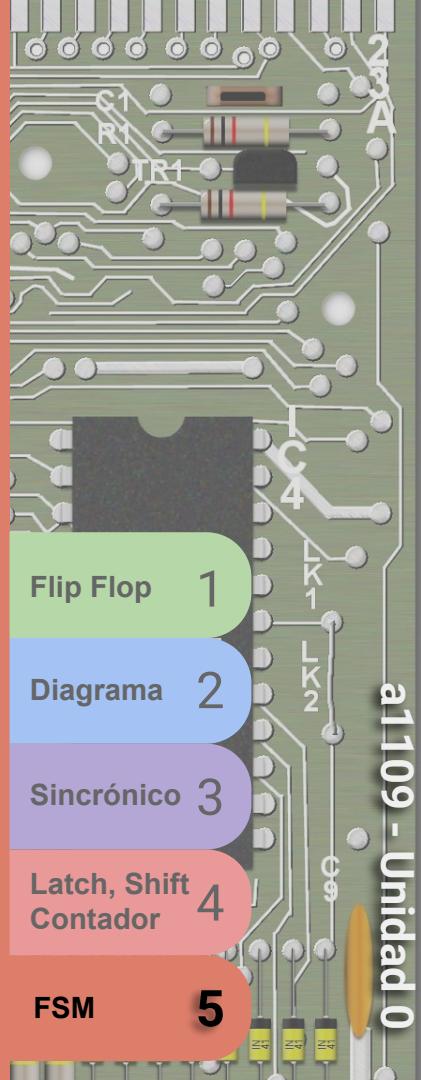
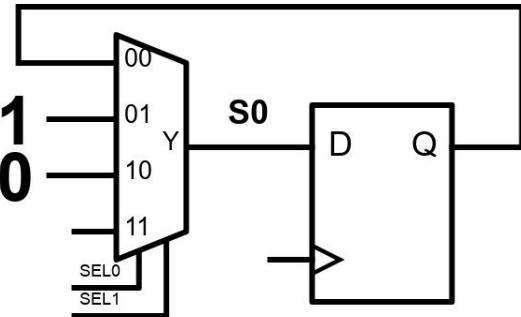
## Parte Secuencial

Esta tabla de transición requiere solamente 8 compuertas AND de 6 entradas y cuatro compuertas OR para unir de a 2 compuertas AND. Por ejemplo para Sel S1 H tenemos  $A \cdot B \cdot \neg C \cdot D \cdot \neg S_0 \cdot S_1 \text{ OR } \neg A \cdot \neg B \cdot C \cdot \neg D \cdot S_1 \cdot S_0$

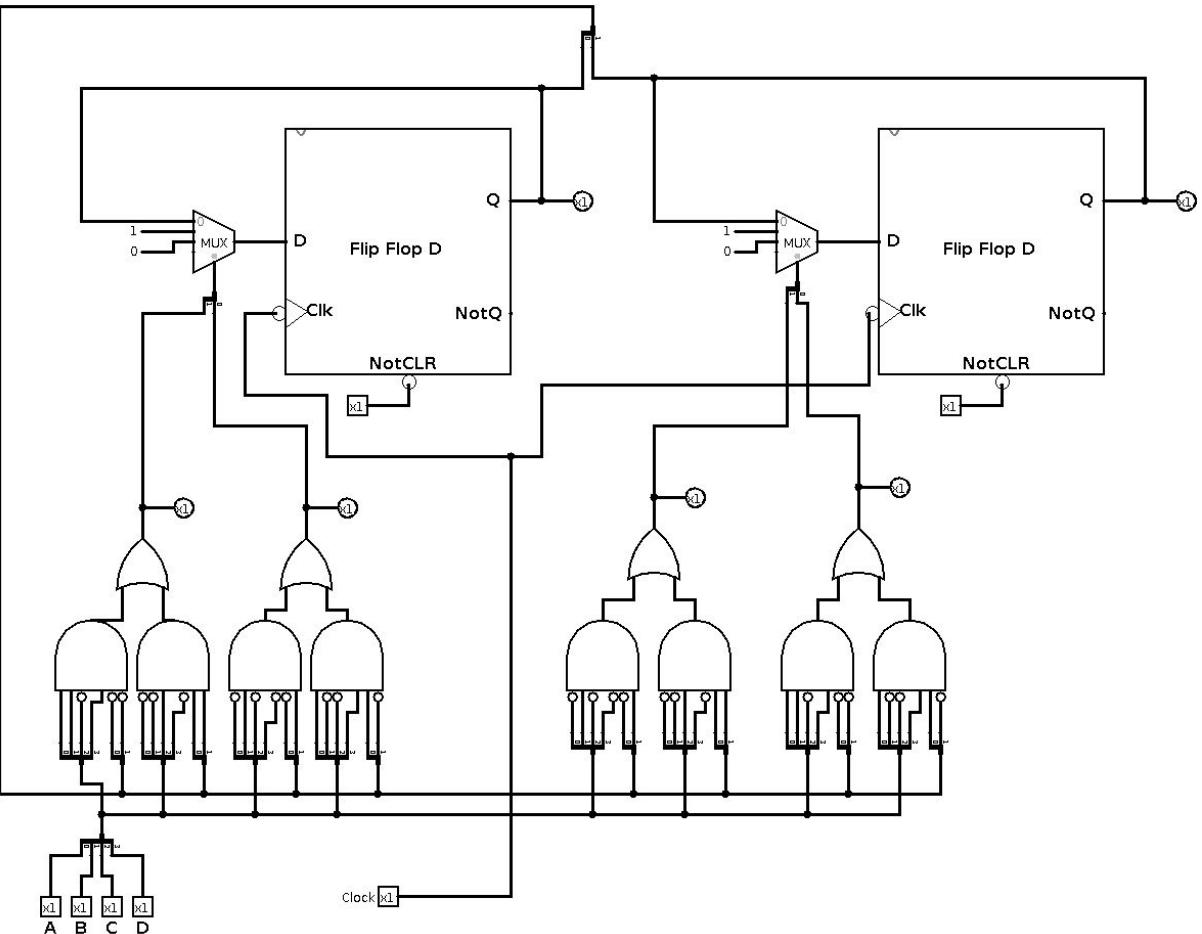
Vemos que es trivial implementarlo de esta forma sin tener que escribir las 64 combinaciones.



Entrada ABCD	Estado $S_1 S_0$	Sel S1 HL	Sel S0 HL
1101	00	10	01
0100	01	01	10
1001	10	01	01
0010	11	10	10
0000	00	00	00
.	.		
.	.		
.	.		
1111	11	00	00



# Parte Secuencial



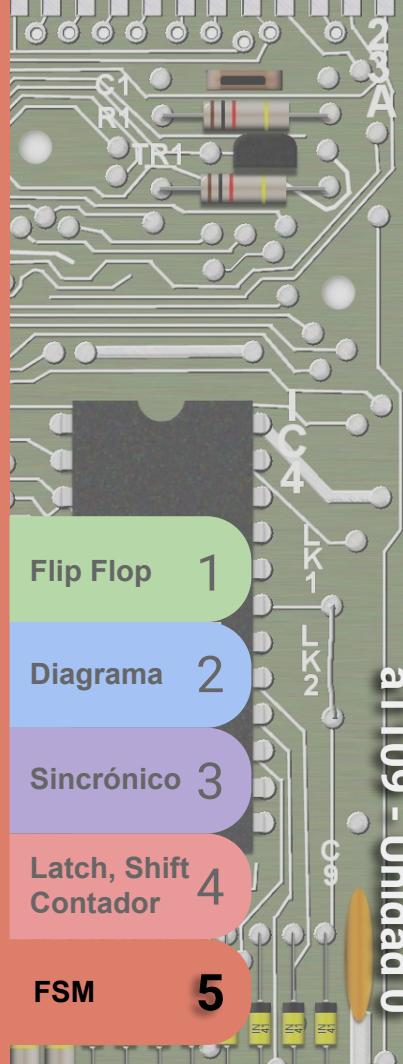
Flip Flop 1

Diagrama 2

Sincrónico 3

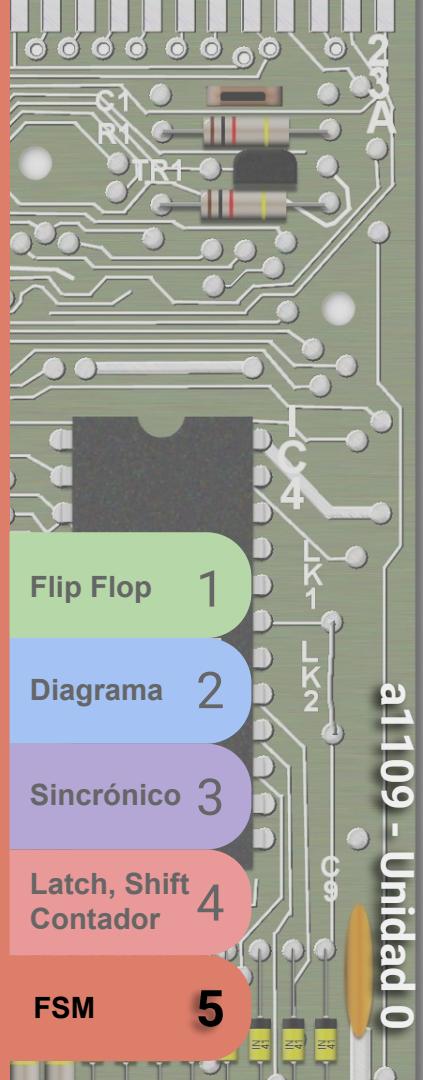
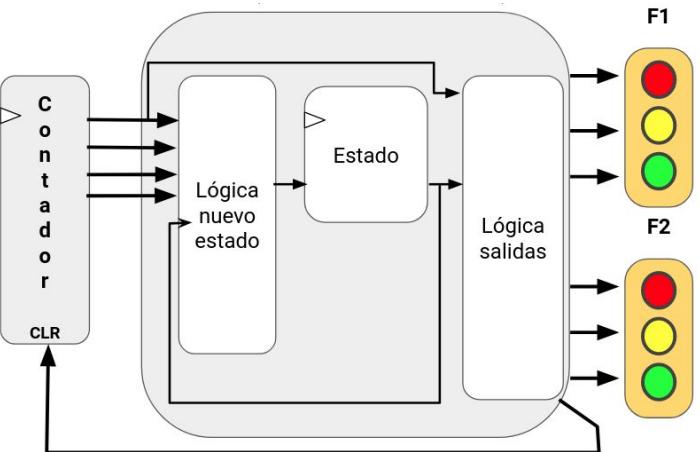
Latch, Shift  
Contador 4

FSM 5

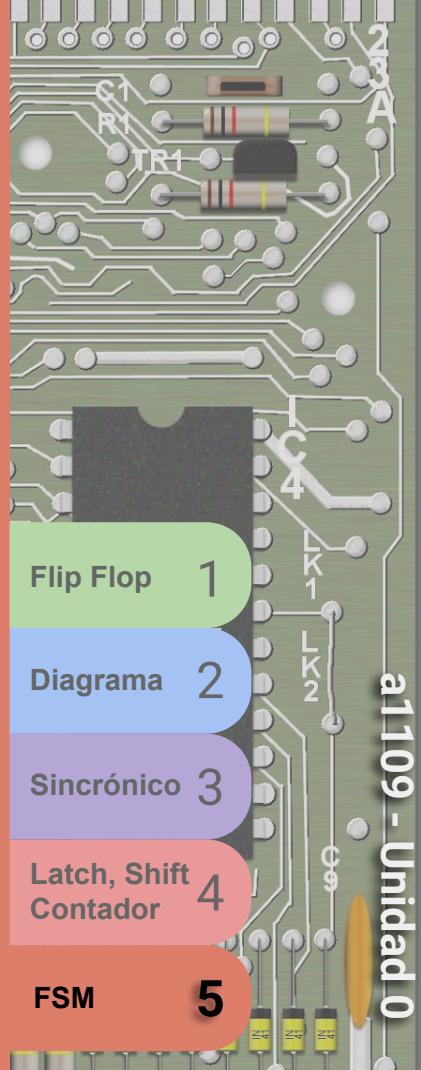
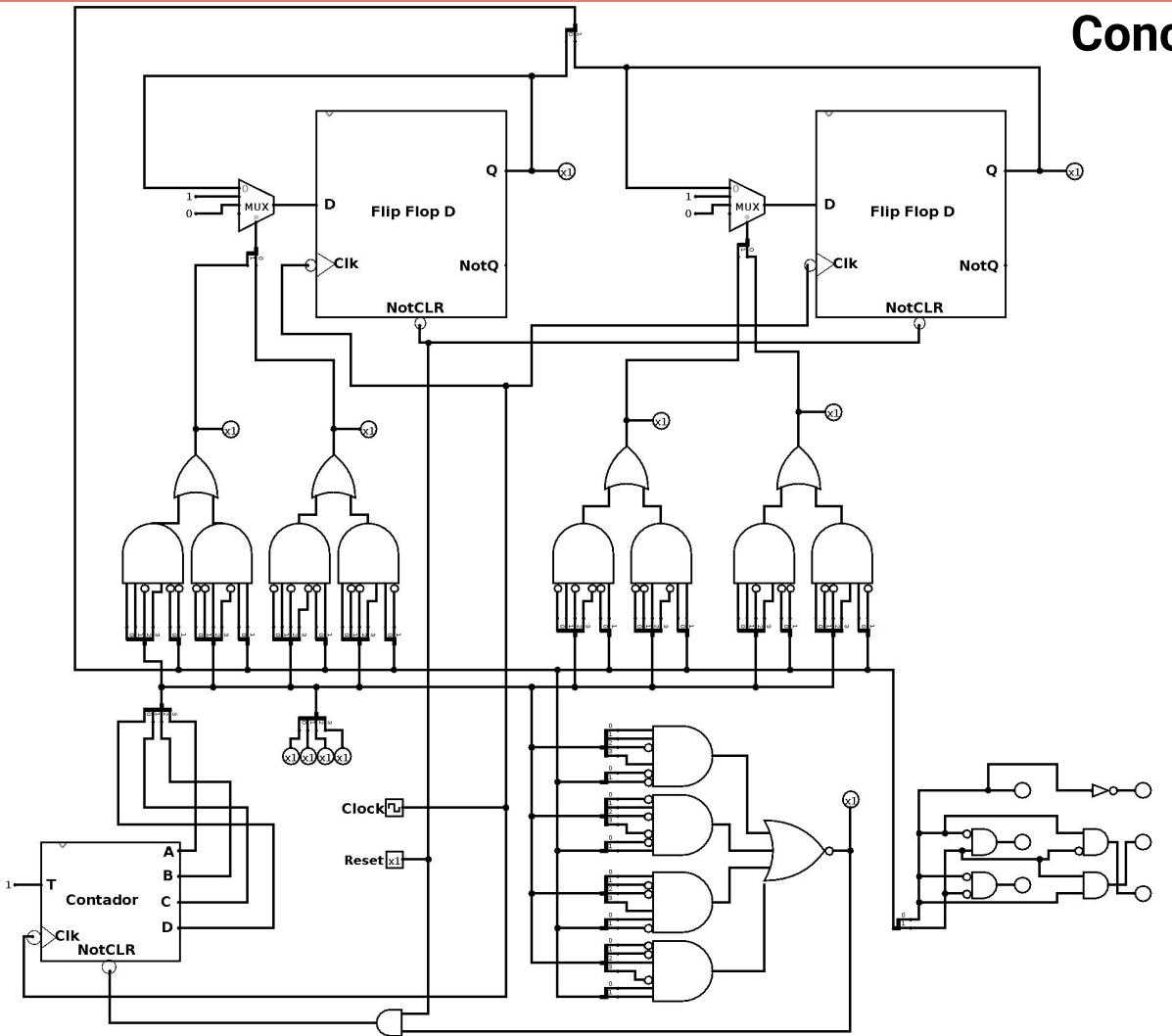


# Conclusiones

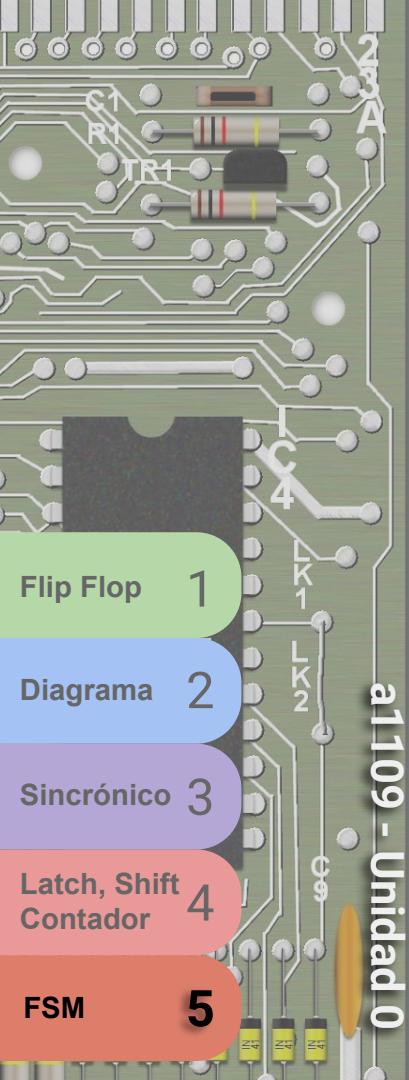
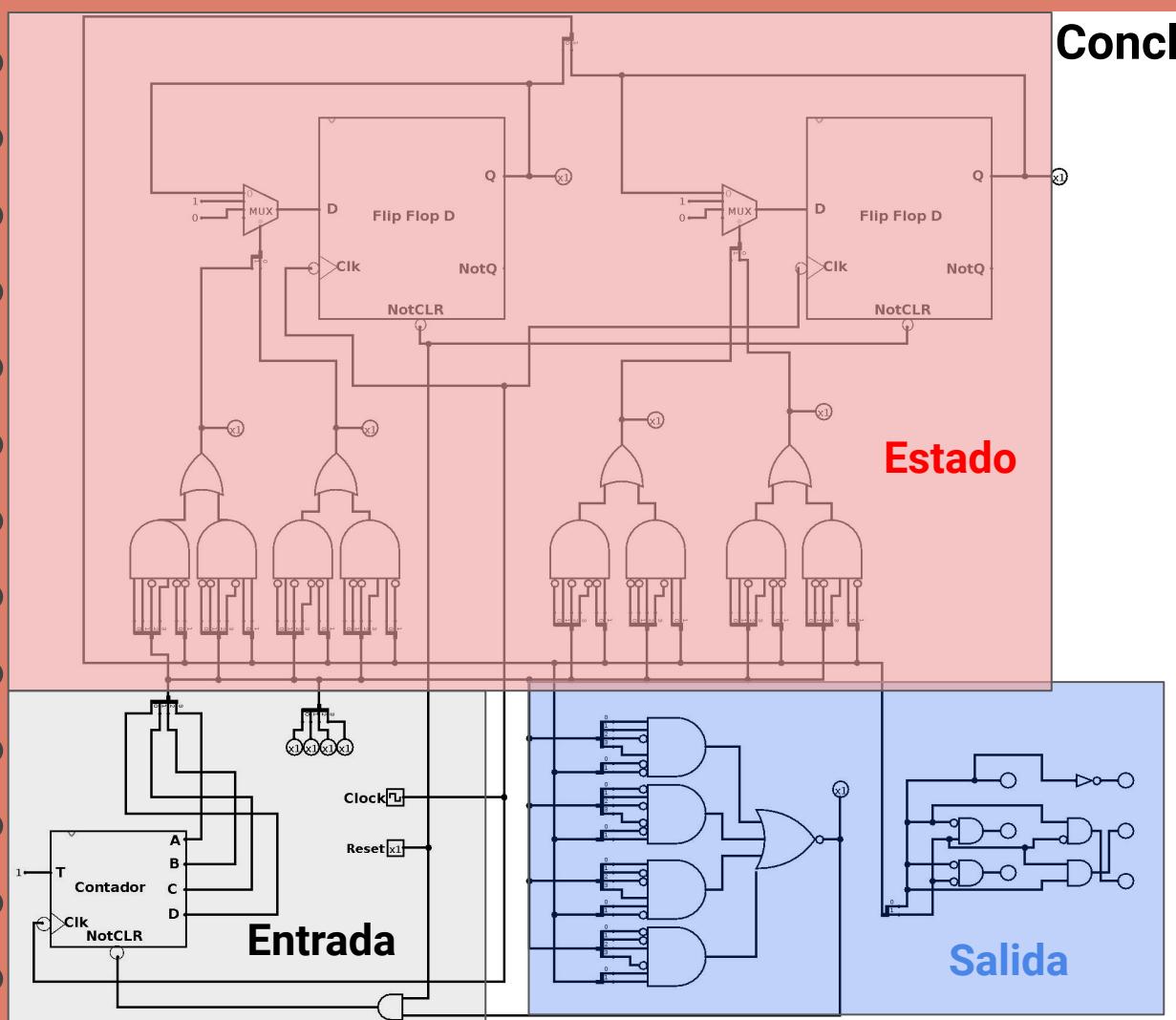
Con todo lo visto de lógica combinatoria y secuencial podemos implementar la FSM del semáforo. Lo más importante es tener en claro cuales son las entradas, cuáles son las salidas y que estados intermedios hacen falta. Vemos que en el caso del semáforo la entrada es el contador, y las salidas son las lámparas y el control del contador. Toda la lógica de salida es combinacional, y la lógica para calcular el nuevo estado también es combinacional teniendo en cuenta los estados realimentados. Lo que es secuencial son los FF en sí que almacenan el estado.



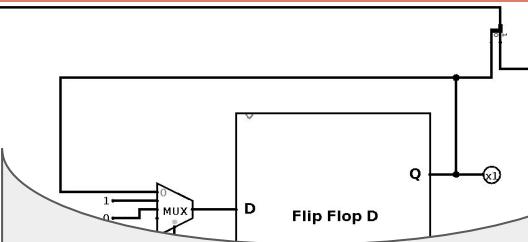
# Conclusiones



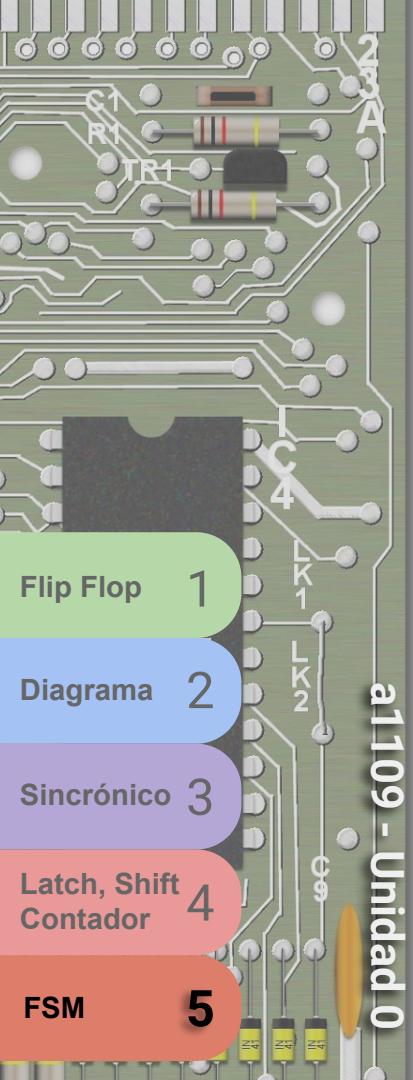
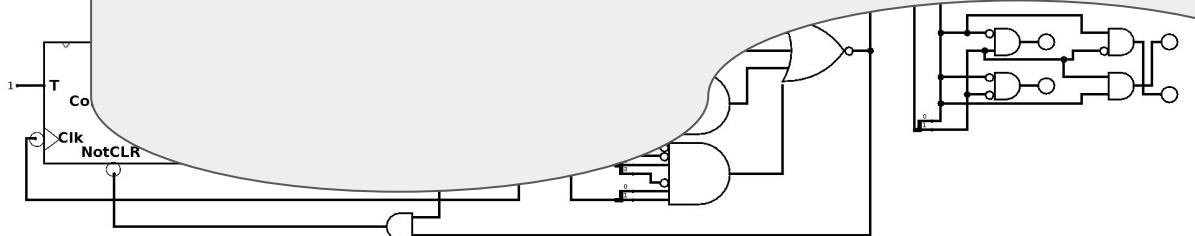
# Conclusiones



## Conclusiones



**¿Que pasa si queremos cambiar los tiempos?  
¿Existen máquinas de estado que cambien la lógica manteniendo entradas y salidas?**



## Conclusiones

Si lo pensamos un poco, una computadora también tiene entradas, salidas, y muchos circuitos combinacionales y secuenciales. Podemos pensar en una computadora como una máquina de estados, solo que las transiciones entre los estados se cargan en una serie de pasos (instrucciones) que ejecutan movimientos de señales fijos (según la instrucción). El conjunto de estos pasos son los programas. En la unidad 2 veremos como funciona una Unidad central de procesos.

