Circuitos Secuenciales

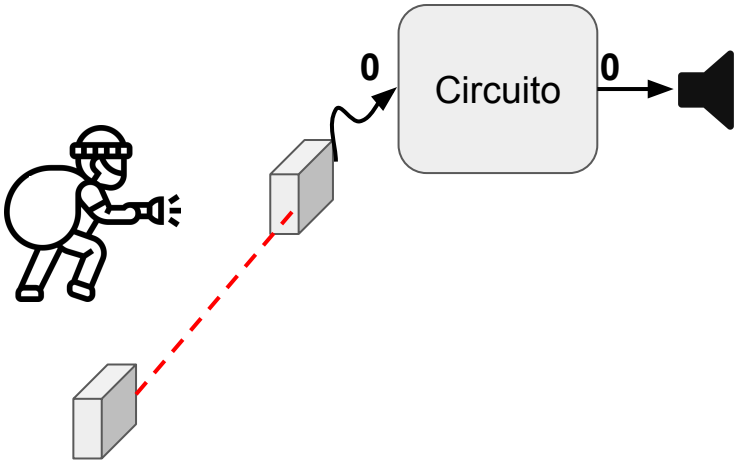
Unidad 4.0
Flip Flops

Versión 1.0.0

Jaír Hnatiuk, Carlos Maidana, Carlos Rodríguez, Edgardo Gho, Martín Ferreyra Biron

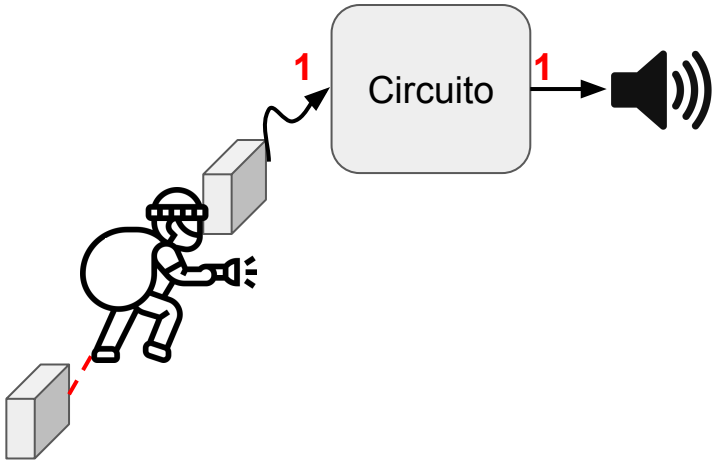
Un sistema de alarma

Deseamos crear un circuito que se conecte a una barrera infrarroja, de forma tal que si alguien interfiere la barrera suene una sirena. Mientras la barrera no se encuentra bloqueada su salida es cero y el circuito genera 0 para la sirena.



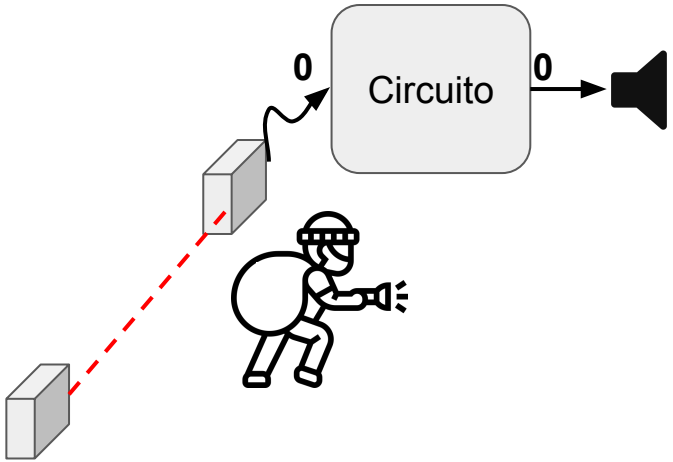
Un sistema de alarma

Cuando la barrera se interfiere produce un uno, y el circuito genera un uno encendiendo la sirena.



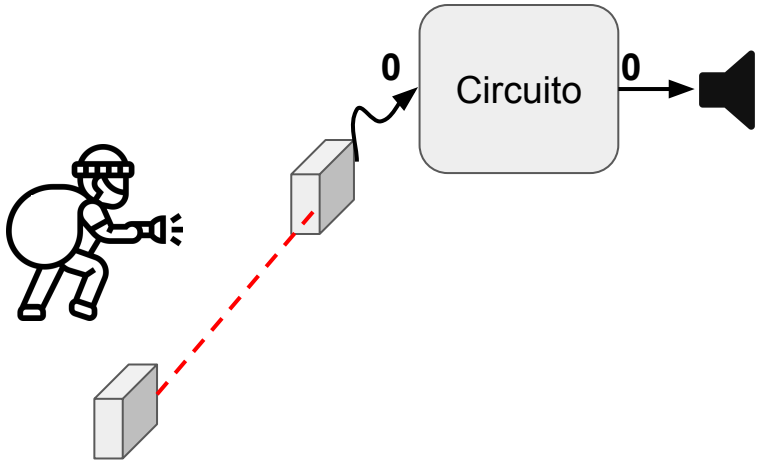
Un sistema de alarma

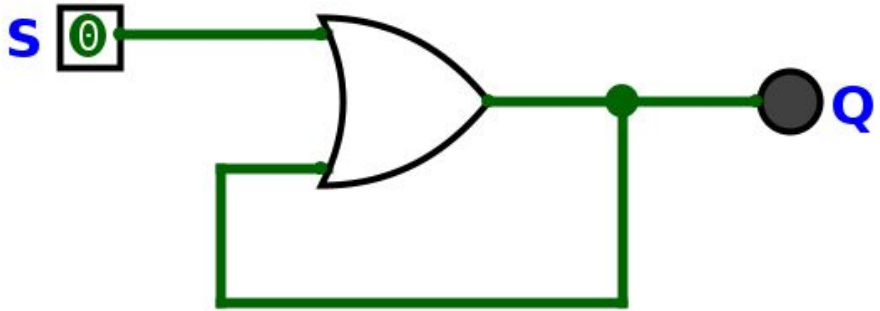
Pero el problema sucede cuando la barrera vuelve a cero, el circuito no tiene memoria y se silencia nuevamente la sirena. Este sistema de alarma no sirve, necesitamos que el mismo tenga memoria.



Un sistema de alarma

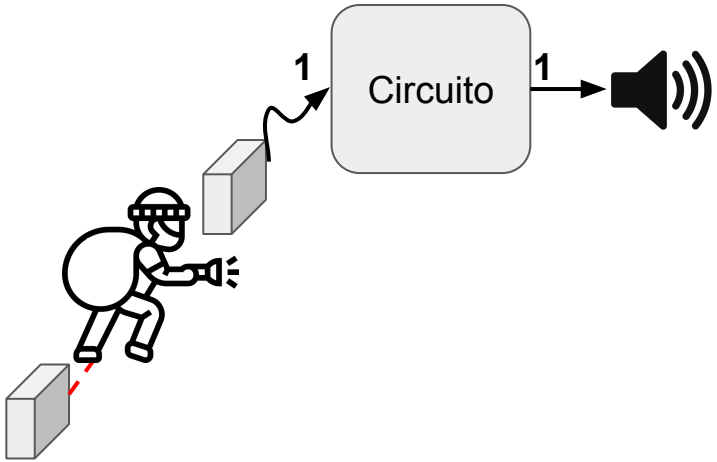
Un circuito posible sería una compuerta OR realimentada. En este caso asumiendo que podemos forzar un cero en la realimentación, cuando el circuito recibe un uno en S produce un uno en Q que se alimenta en la OR.

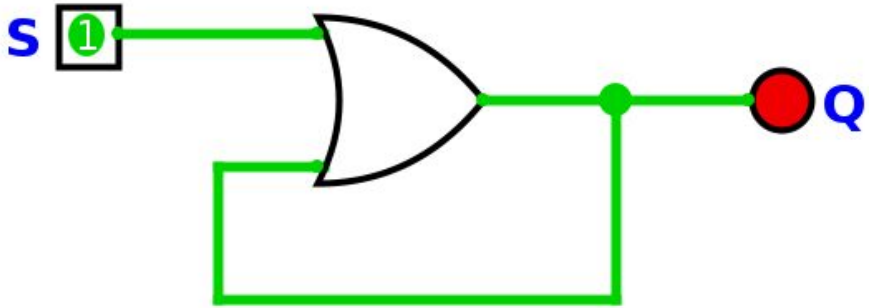




Un sistema de alarma

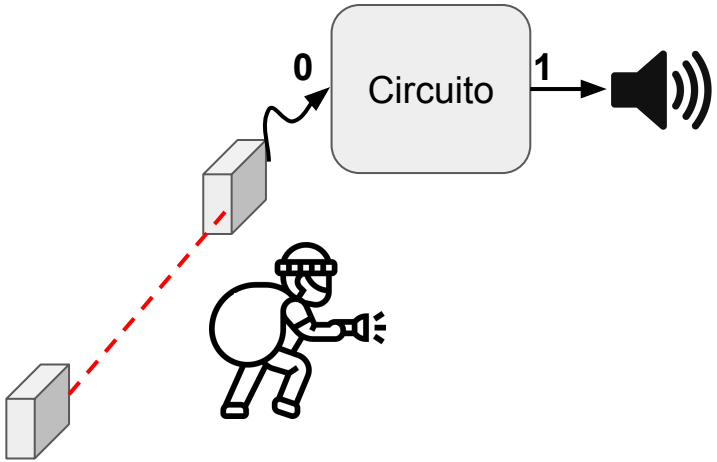
Ahora el circuito detecta al ladrón…

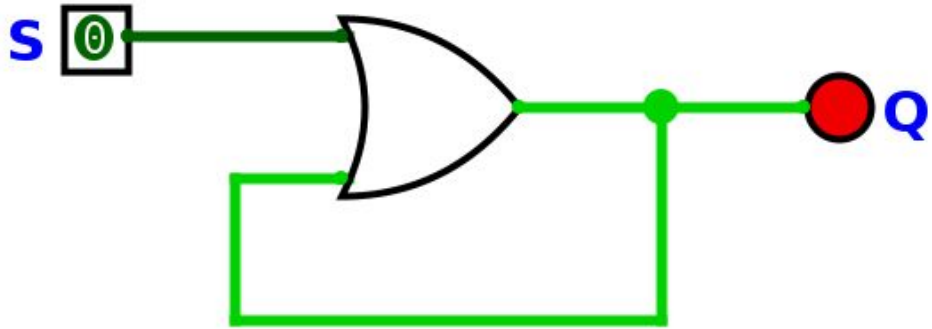




Un sistema de alarma

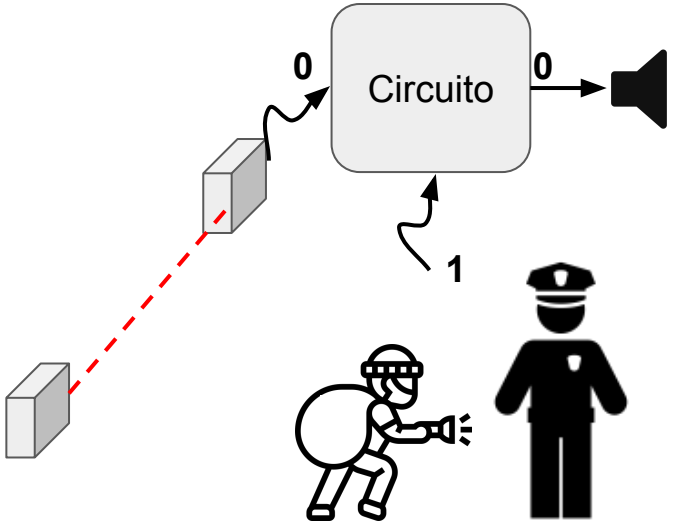
Y la sirena queda encendida aun cuando la barrera se encuentra desactivada. El problema que tenemos ahora es que no tenemos forma de reiniciar el circuito… o sea… no podemos apagar la sirena.

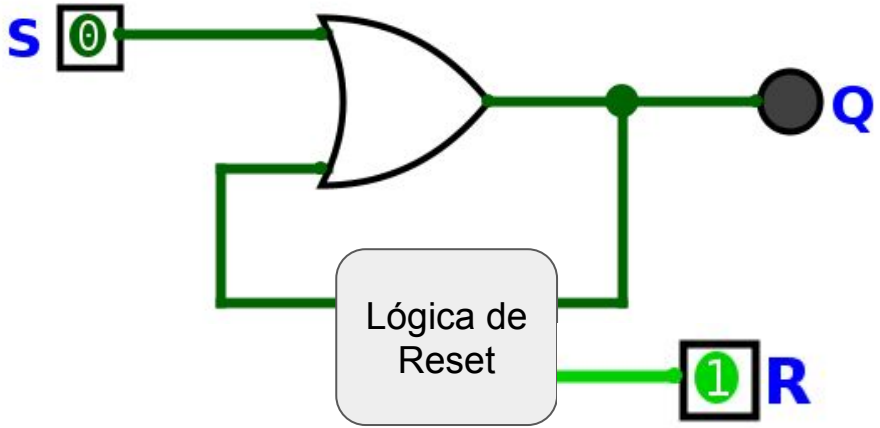




Un sistema de alarma

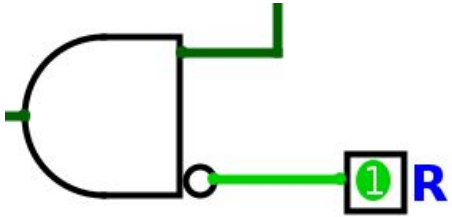
Lo que necesitamos ahora es de alguna forma controlar esa realimentación de forma tal que podamos generar un cero en la OR. Para ello vamos a usar otro circuito con una entrada extra (R) que genere un cero si R=1.



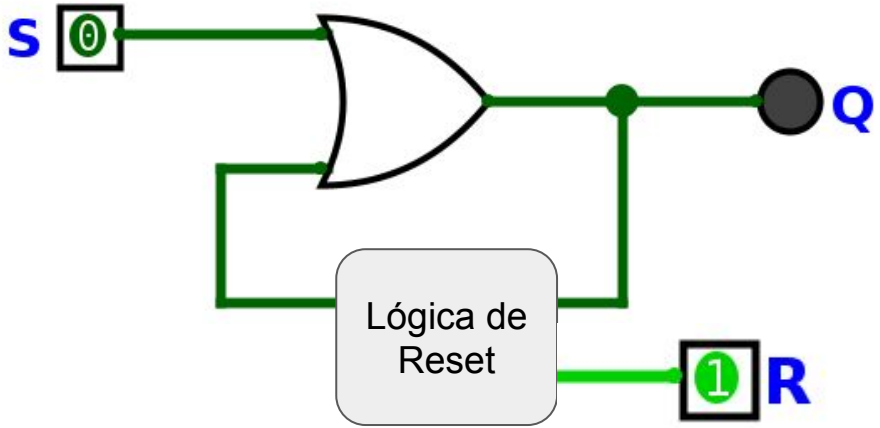


Un sistema de alarma

Armemos entonces la lógica de Reset. Planteamos la tabla de verdad donde las entradas son Q y R. Se debe generar un cero a la salida si R=1. En el caso que R=0 y Q=1 entonces la salida es uno. En el caso que R=0 y Q=0 la salida es cero. Vemos que si lo resolvemos por los unos, tenemos la función: $\overline{R}\,Q$



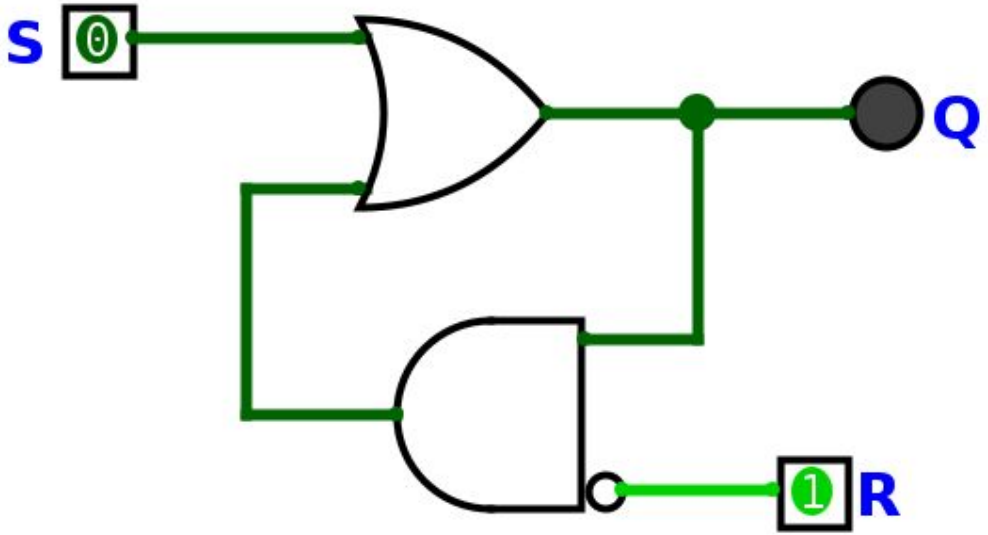
R	Q	out
0	0	0
0	1	1
1	0	0
1	1	0



Un sistema de alarma

Este circuito funciona como memoria. Almacena un estado (cero o uno). Se logra por la realimentación (controlada). Pero como vimos en la unidad 1.2, las compuertas generalmente vienen en circuitos integrados que incluyen varias compuertas del mismo tipo. Así que tratemos de convertir esto a OR…

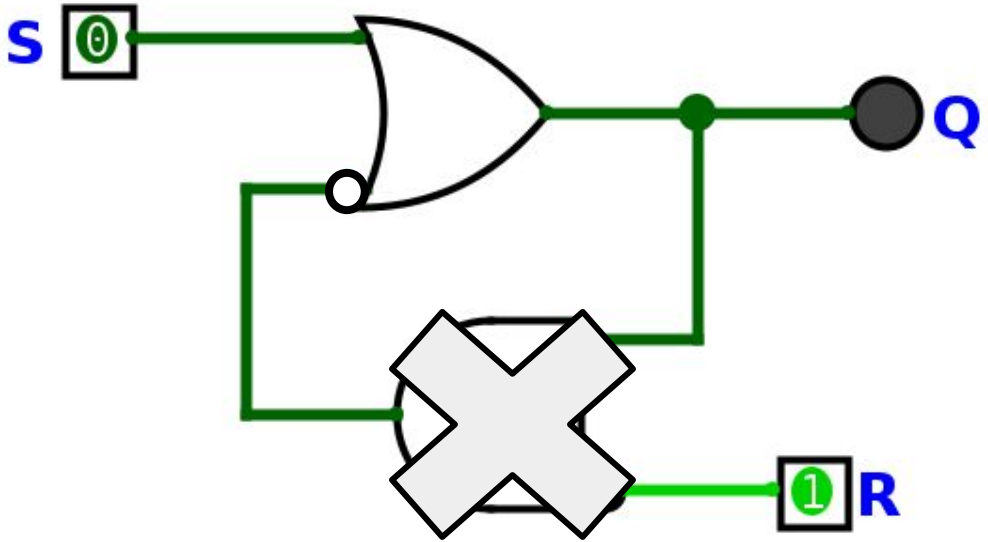
R	Q	out
0	0	0
0	1	1
1	0	0
1	1	0



Un sistema de alarma

Si negamos la salida out, obtenemos ahora un circuito que solo tiene un cero, por ende podemos implementarlo con una OR.. pero tenemos que ahora NEGAR la entrada en la OR original, ya que tenemos la inversa. La nueva función ahora es $(R+\overline{Q})$

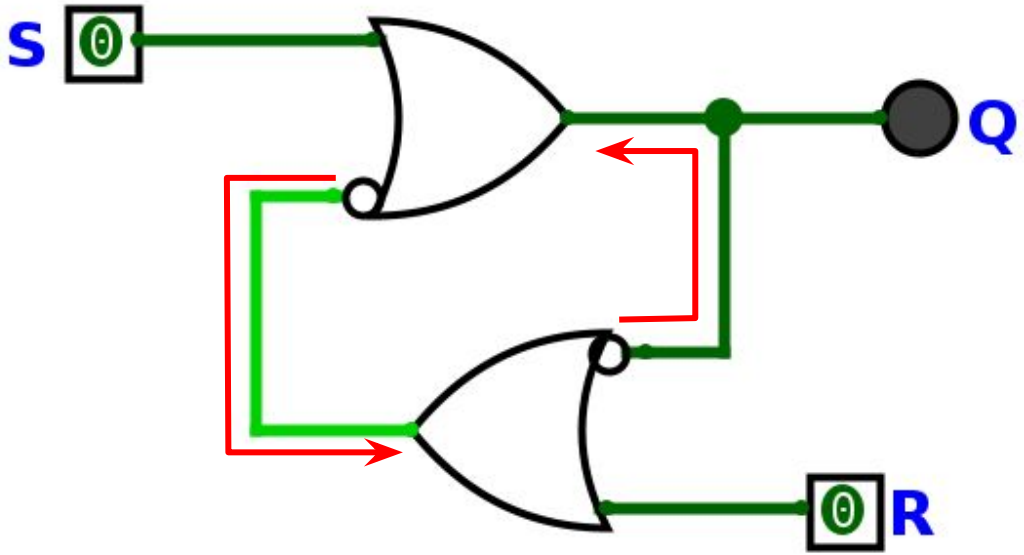
R	Q	out	!out
0	0	0	1
0	**1**	1	**0**
1	0	0	1
1	1	0	1



Un sistema de alarma

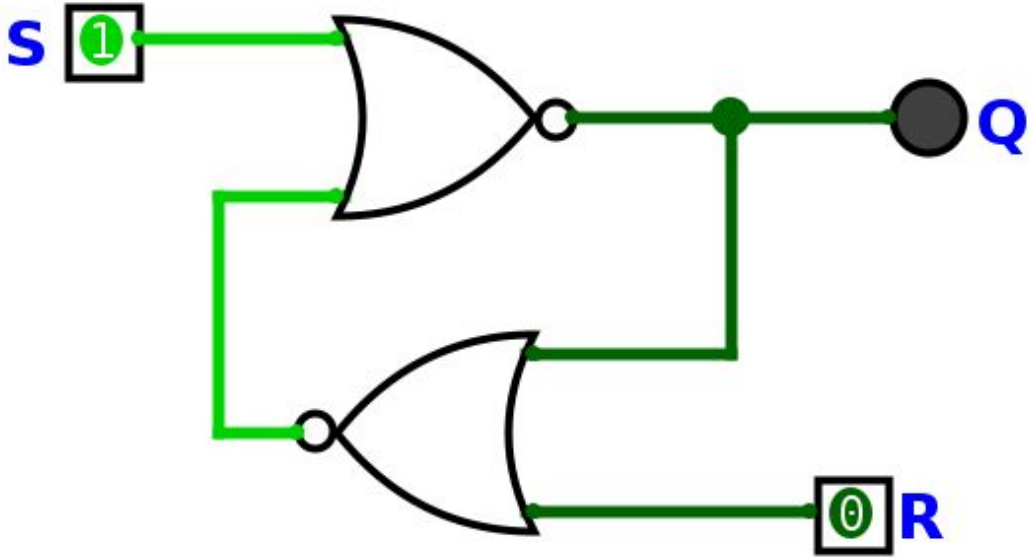
Hemos convertido el circuito a un único tipo de compuertas…. pero no del todo, ya que tenemos compuertas OR pero también tenemos los negadores…
Haciendo un poco de “ingeniería creativa” vamos a mover los negadores a la salida de las OR…

R	Q	out	!out
0	0	0	1
0	**1**	1	**0**
1	0	0	1
1	1	0	1



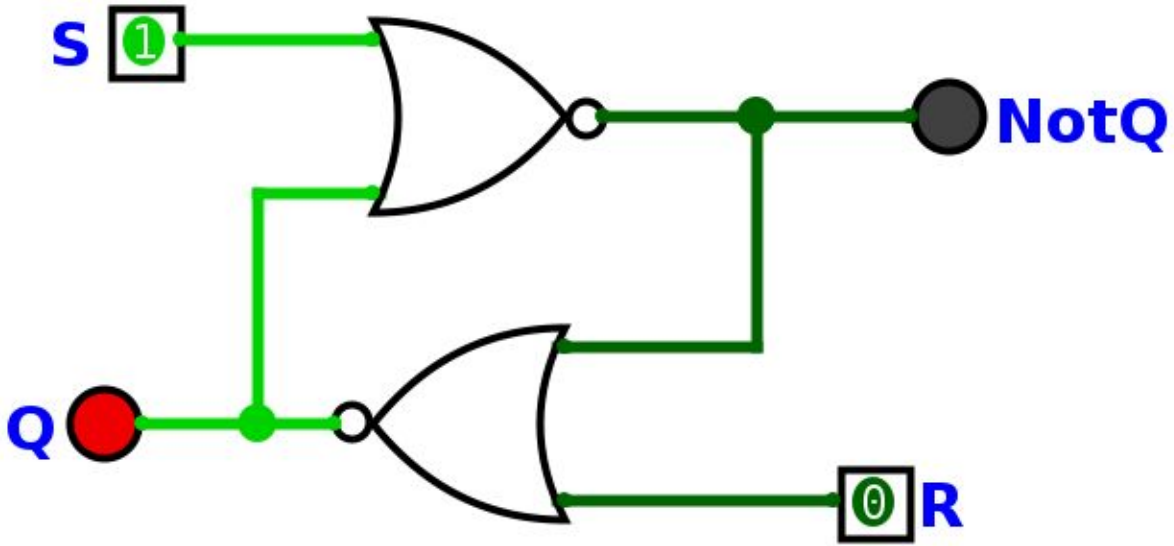
Un sistema de alarma

Y ahora tenemos un solo tipo de compuerta (todas NOR). Pero, si miramos la salida Q, ahora no coincide con el caso original. Cuando S=1 y R=0, el valor de Q deberia ser uno, pero es cero. Vemos que el cable de la OR de abajo ahora si es uno.. así que vamos a definir eso como Q, y la salida original como NotQ.



Un sistema de alarma

Y ahora si tenemos un Latch (cerrojo) RS.. básicamente un circuito con memoria donde tenemos una entrada para forzar la memoria en uno, y otra entrada para forzar la memoria en cero.



Un sistema de alarma

Dado que nos gusta ver las entradas en la izquierda y las salidas a la derecha, entonces lo dibujamos de otra forma, pero es el mismo circuito que antes. Este es un Latch RS implementado con NOR, conocido también como FF RS NOR Asincrónico. El problema es que ahora para estudiar este circuito necesitamos considerar el paso del tiempo… utilizamos un diagrama de tiempos.

R 0
S 0
Q 0
!Q 1

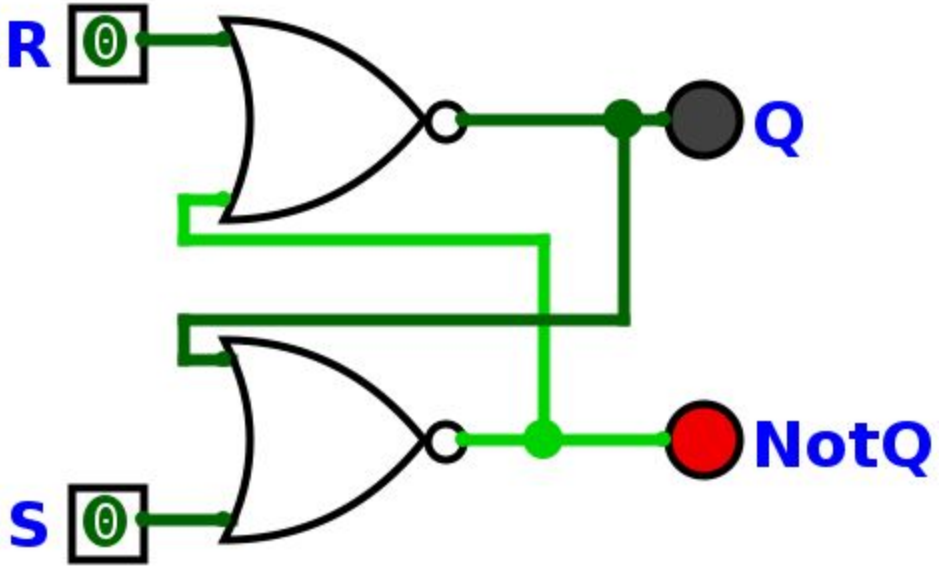


Diagrama de tiempos

Comenzamos el paso del tiempo… en cada renglón dibujamos una línea indicando el valor de la entrada o la salida.. luego mantenemos este estado hasta que se produce un cambio.

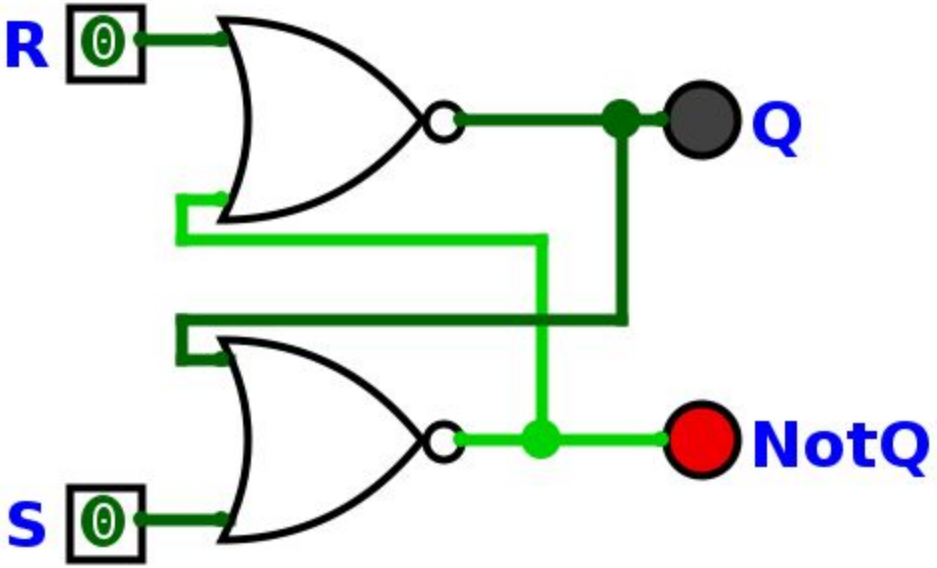
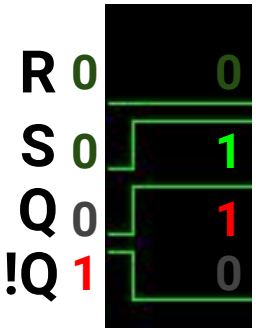


Diagrama de tiempos

Si se produce un cambio tal que S=1, vemos que (luego de un tiempo de propagación) la salida Q pasa a 1 y !Q pasa a 0. El diagrama de tiempo representa estos cambios en cada renglón.



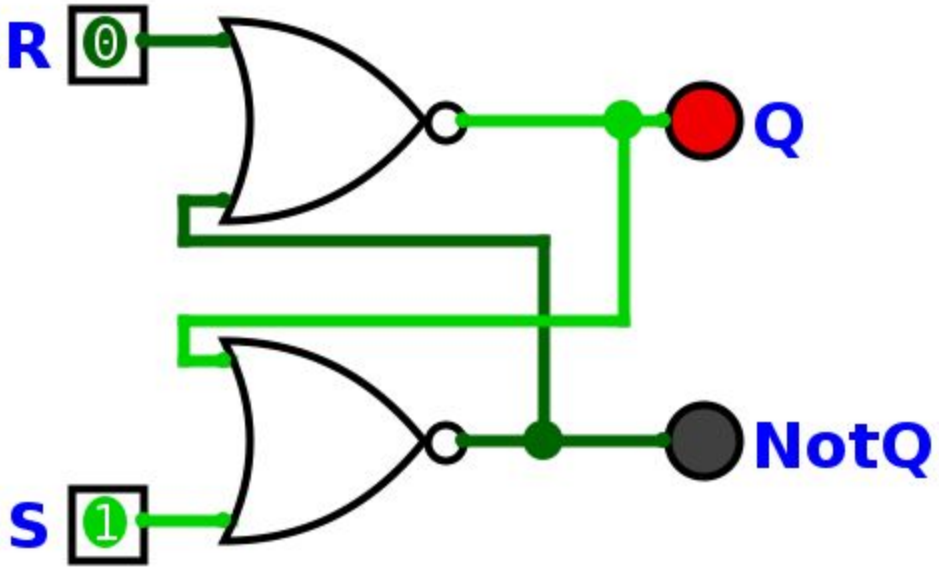
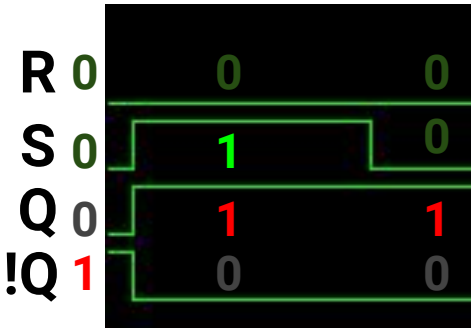


Diagrama de tiempos

Si la entrada S vuelve a cero, el “estado” en Q quedó memorizado. Vemos que esa salida se mantiene en uno.



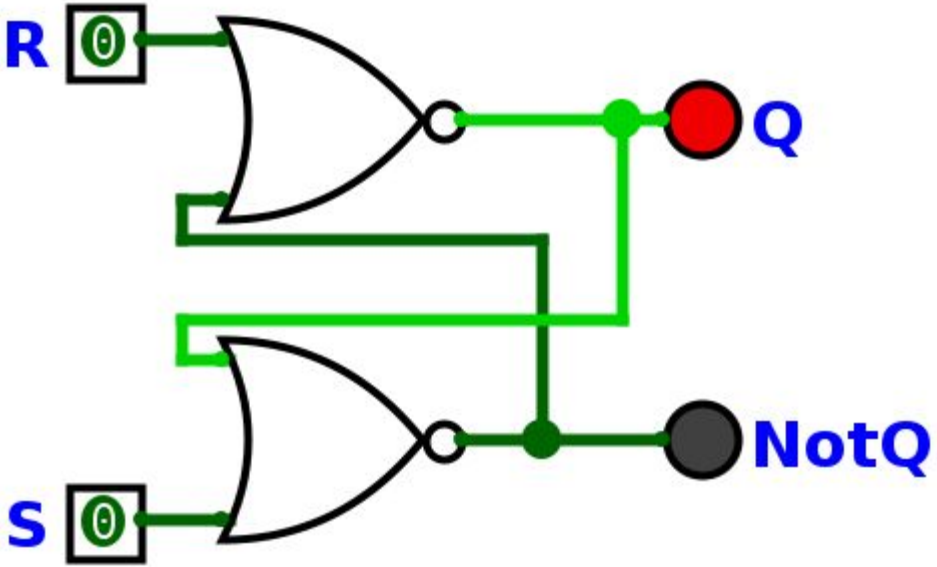
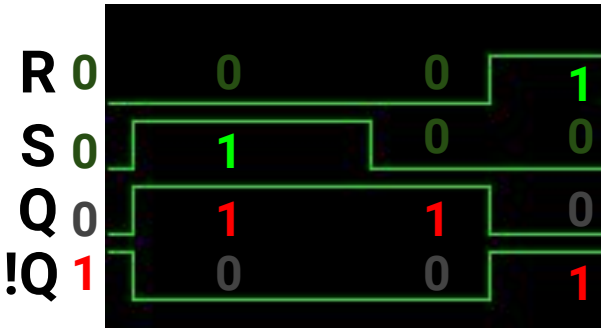


Diagrama de tiempos

Si la entrada R=1, ahora “borramos” el uno en Q, forzando esta salida a cero, y haciendo que !Q sea 1.



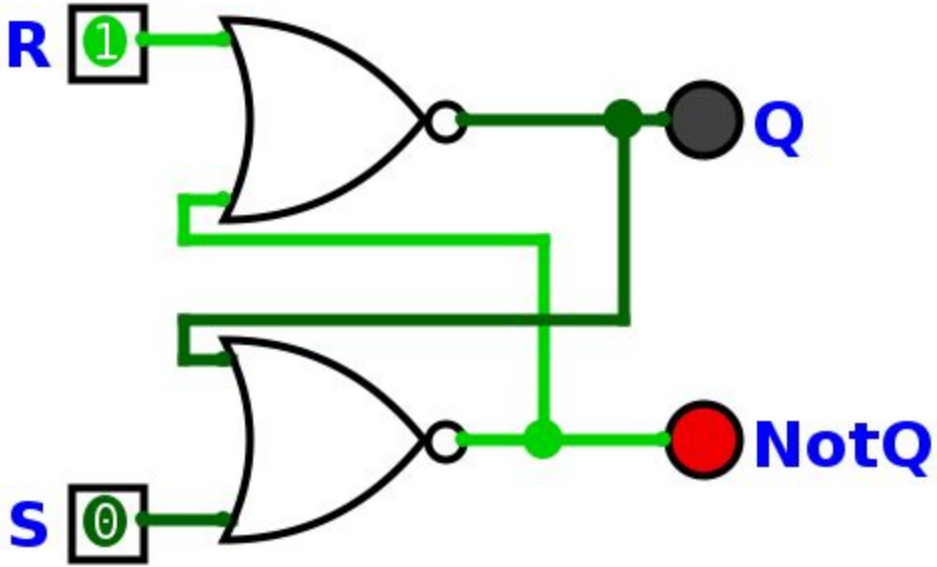
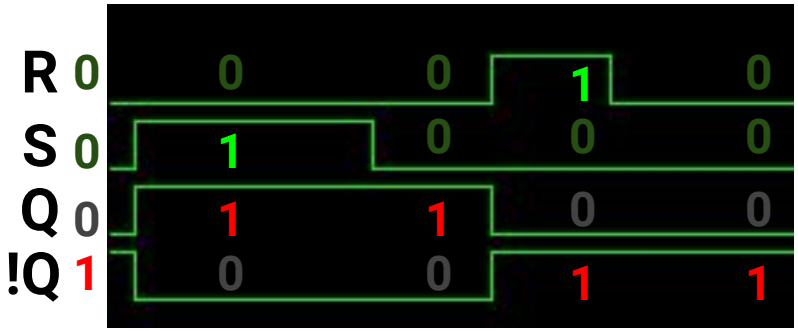


Diagrama de tiempos

Si quitamos ese uno en R haciendo R=0, vemos que el estado queda memorizado, y la salida Q sigue siendo cero.



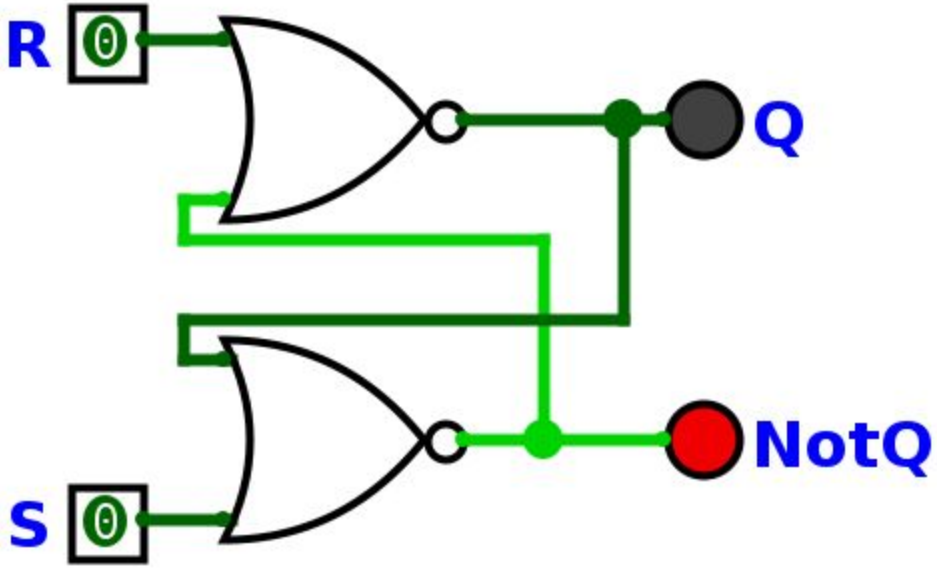
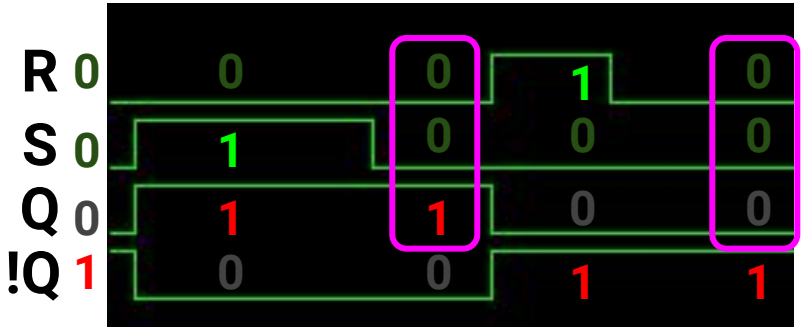


Diagrama de tiempos

Notemos que a lo largo del tiempo.. hubo dos instantes en donde S=0 y R=0.. pero la salida Q no siempre tuvo el mismo valor... En el primer caso Q vale 1 y en el segundo caso Q vale 0. Estos valores son los valores “anteriores” que tenía Q. Podemos plantear una tabla de verdad....



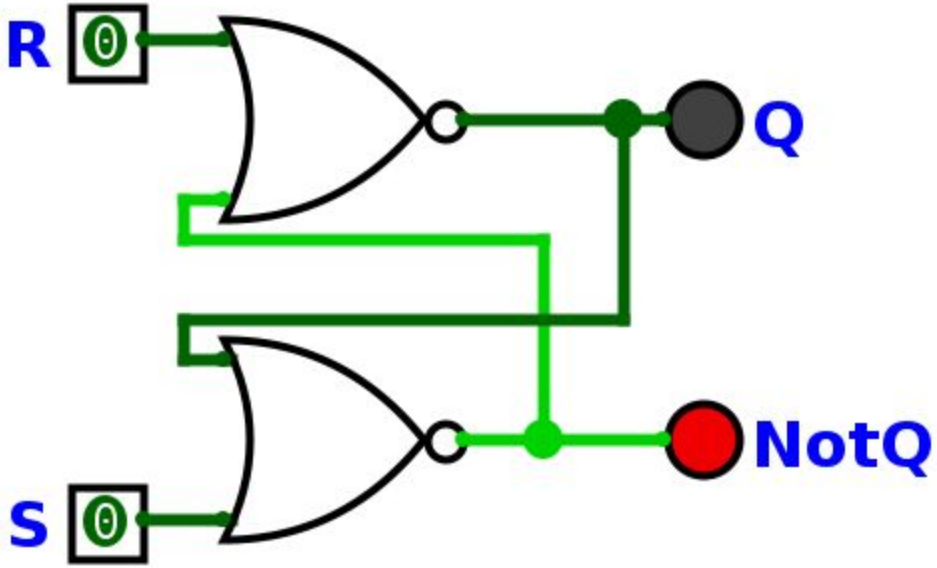
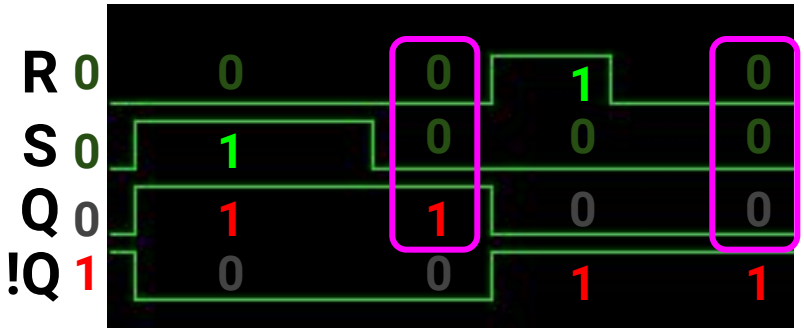
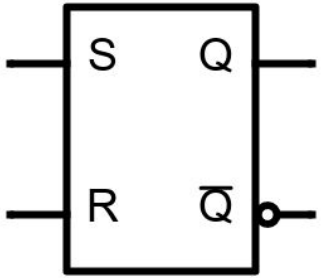


Diagrama de tiempos

En esta tabla de verdad representamos las entradas S y R, pero ahora la salida Q depende del momento del tiempo (Q_N). Vemos que esa salida se encuentra definida como 0 cuando S=0 y R=1. Luego se define como 1 cuando S=1 y R=0. En el caso donde S=0 y R=0, el valor de Q_N es igual al valor que tenía anteriormente (o sea, Q_{N-1}). Nos queda el estado S=1 y R=1.. en este caso lo que ocurre es que Q=!Q=0. Esto quiere decir que este Latch no puede memorizar ya que sus salidas complementarias son iguales. A este estado lo llamamos prohibido.



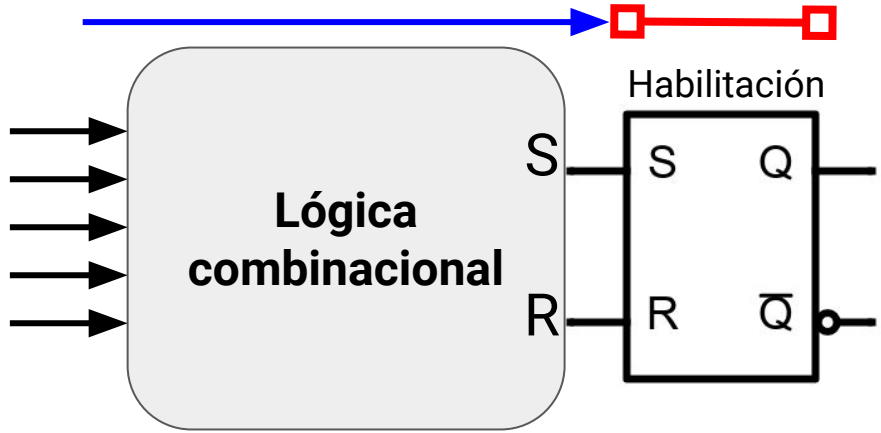
S	R	Q_N
0	0	Q_{N-1}
0	1	0
1	0	1
1	1	!!!



Circuitos Sincrónicos

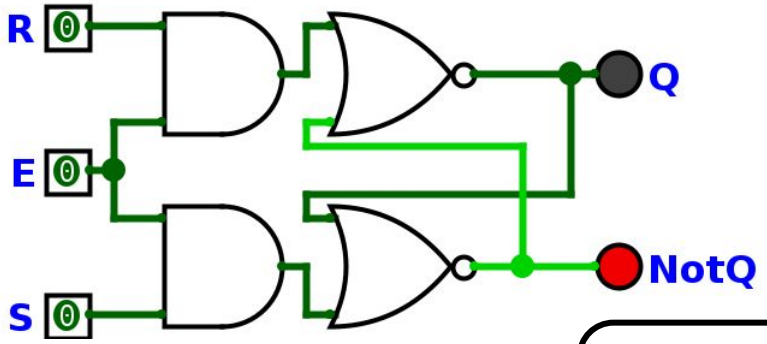
Almacenando resultados

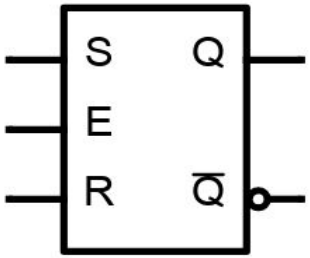
Los circuitos secuenciales suelen conectarse a la salida de un circuito combinatorio para por ejemplo almacenar un resultado. Esta lógica combinacional va a detectar cambios en sus entradas y generar valores para S y para R en caso de tener que almacenar unos o ceros. El problema es que mientras esta lógica propaga los cambios probablemente se generen valores intermedios (glitches) en las salidas S y R. Es por esto que tenemos que controlar en qué momento exacto el Latch se actualiza.

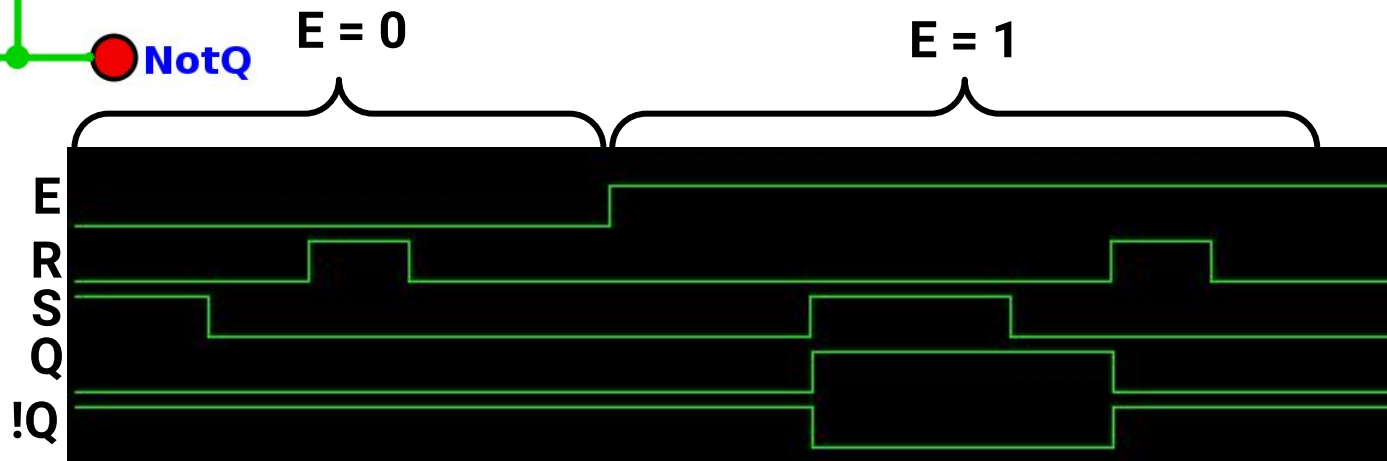


Señal de habilitación (enable)

Agregamos una nueva entrada de Habilitación - Enable (E). Utilizamos dos compuertas AND (una para R y otra para S) de forma tal que si E no vale 1, entonces no deja pasar los valores de R y S , por ende el Latch mantiene su estado. A este circuito lo llamamos **Flip Flop RS Sincrónico**.

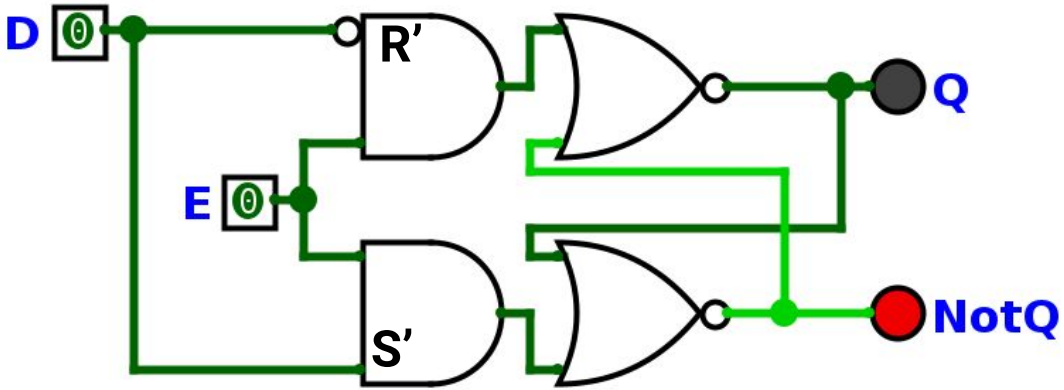


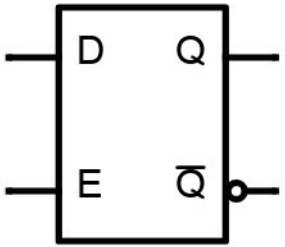




Simplificando en una sola entrada de datos (D)

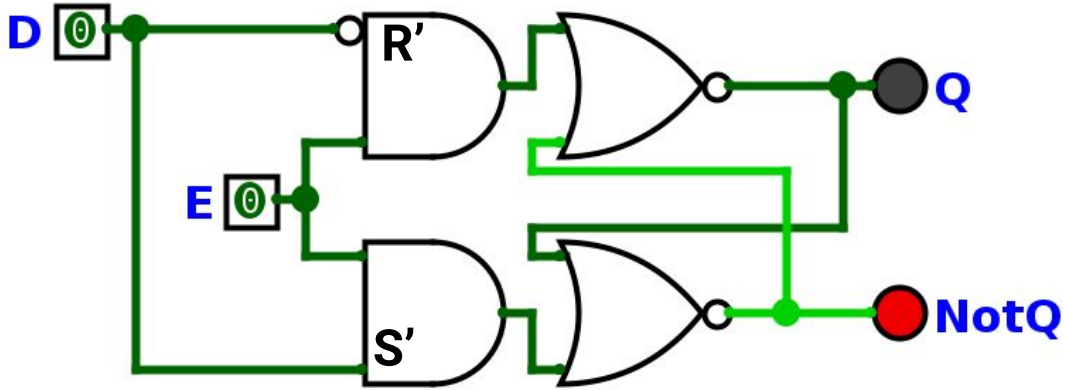
Tener una entrada para forzar un uno (S) y otra para forzar un cero (R) complica los diseños. Mucho más simple es unificar la entrada de datos en una sola, y utilizar la señal de Enable (E) para decidir en qué momento se guarda el valor de D. De esta forma evitamos el estado S=1 y R=1 prohibido, ya que la entrada D se niega en R’ y entra sin negar en S’. A este circuito lo conocemos como **FF D sincrónico por nivel**.

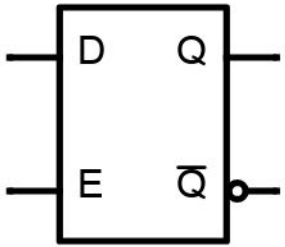


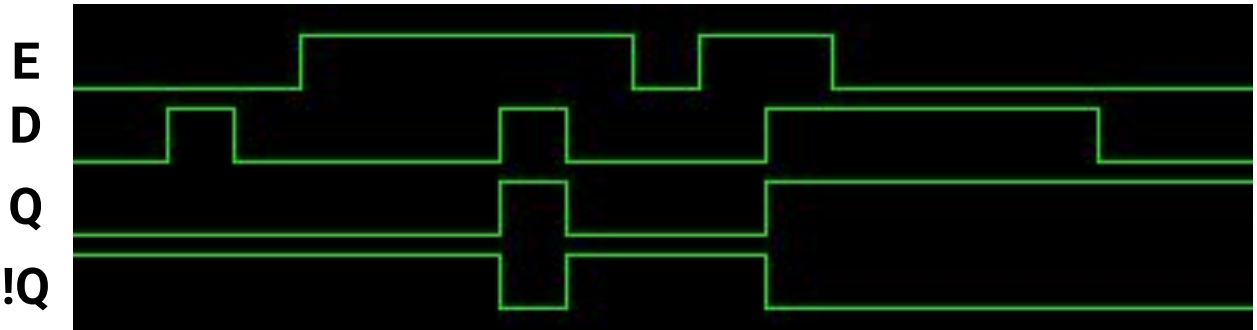


Simplificando en una sola entrada de datos (D)

Vemos el funcionamiento del mismo en un diagrama de tiempos…

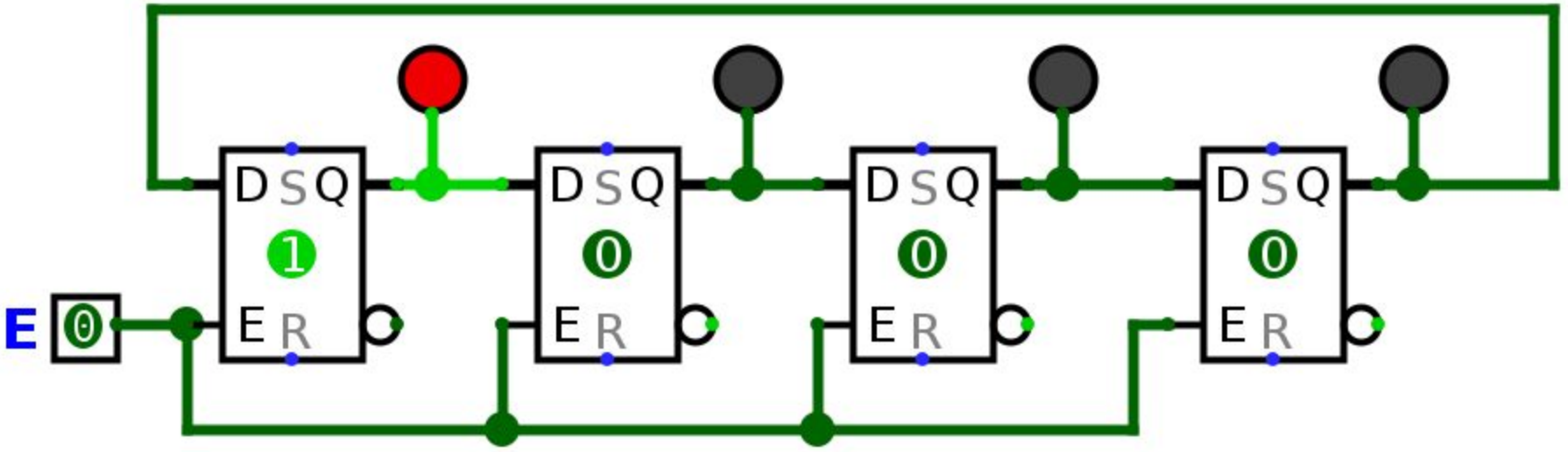






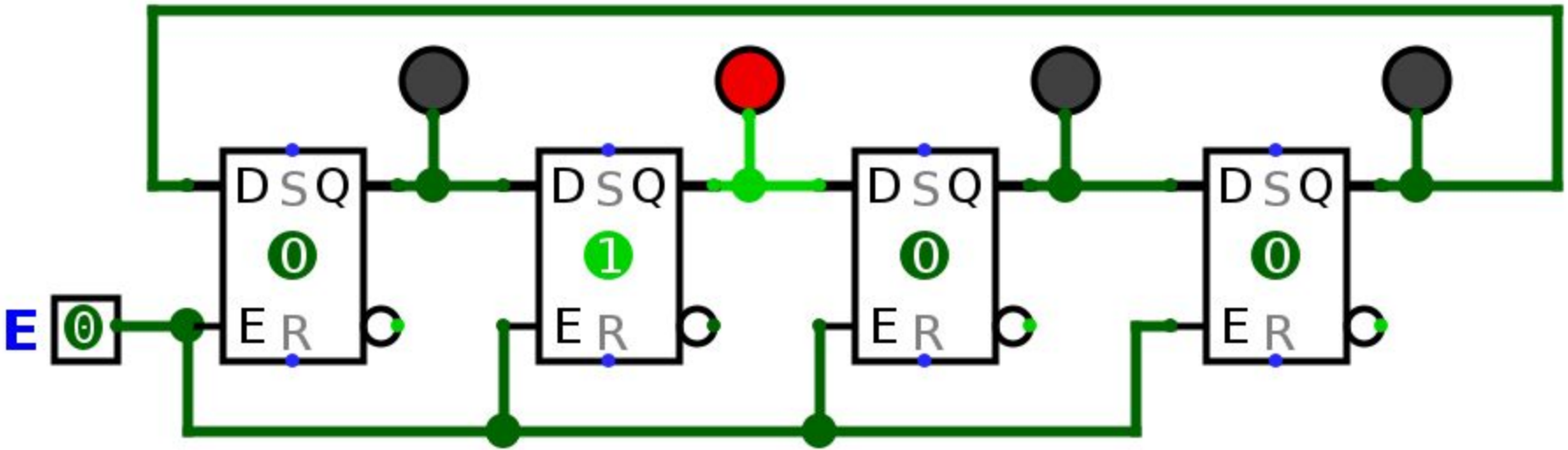
Un uno que se mueve

En este circuito el primer FF tiene un uno, que se conecta al segundo, que a su vez al tercero y así.. vemos que el último FF realimenta la salida con el primer FF. Comenzamos con el primero encendido…



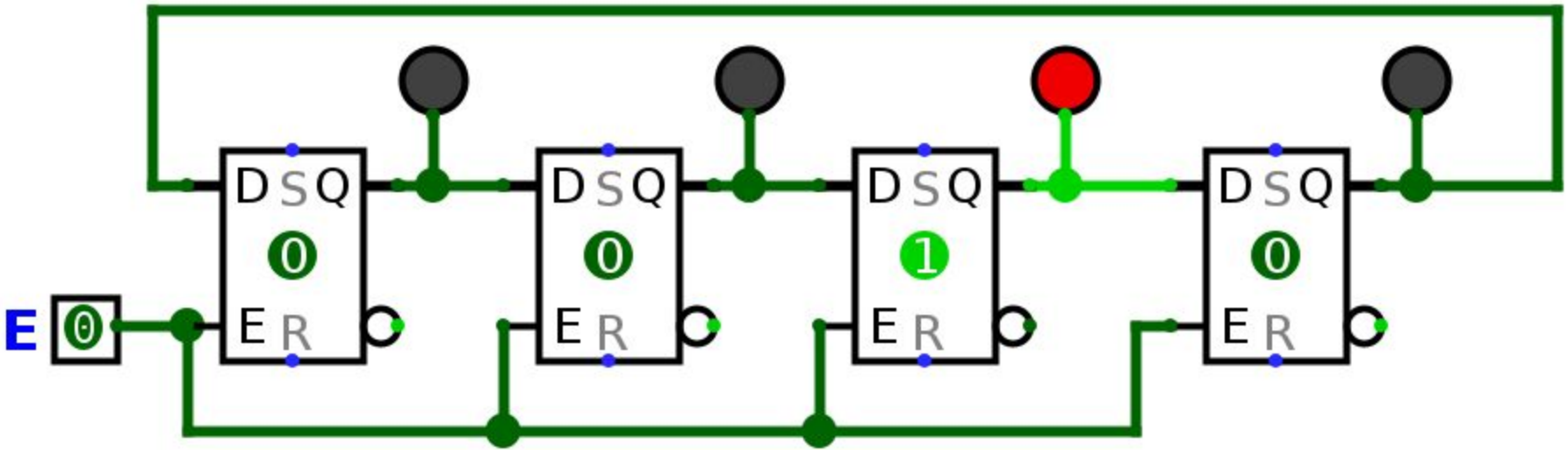
Un uno que se mueve

Luego el 1 se pasa al segundo…



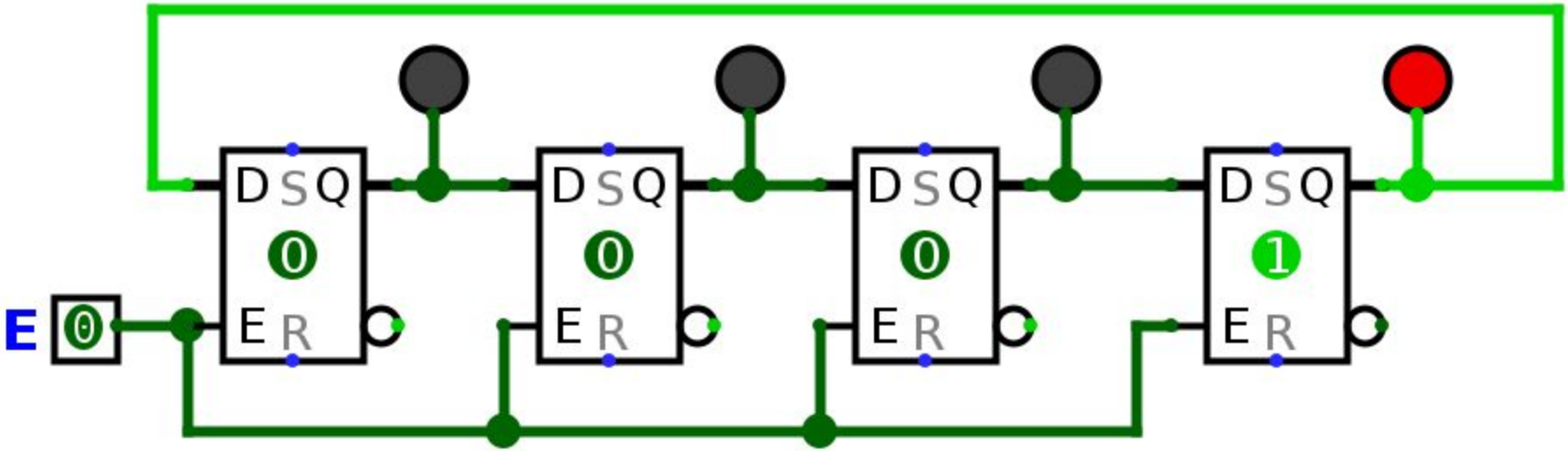
Un uno que se mueve

Luego el 1 se pasa al tercero…



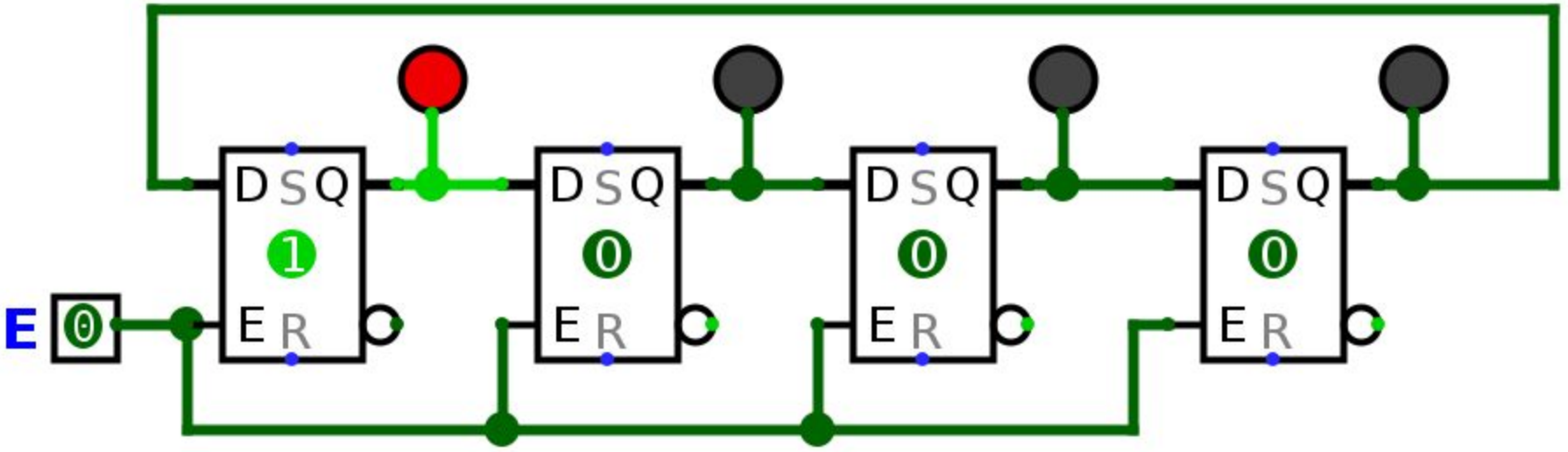
Un uno que se mueve

Y por último al cuarto.. para luego quedar en la entrada del primero listo para comenzar la secuencia nuevamente…



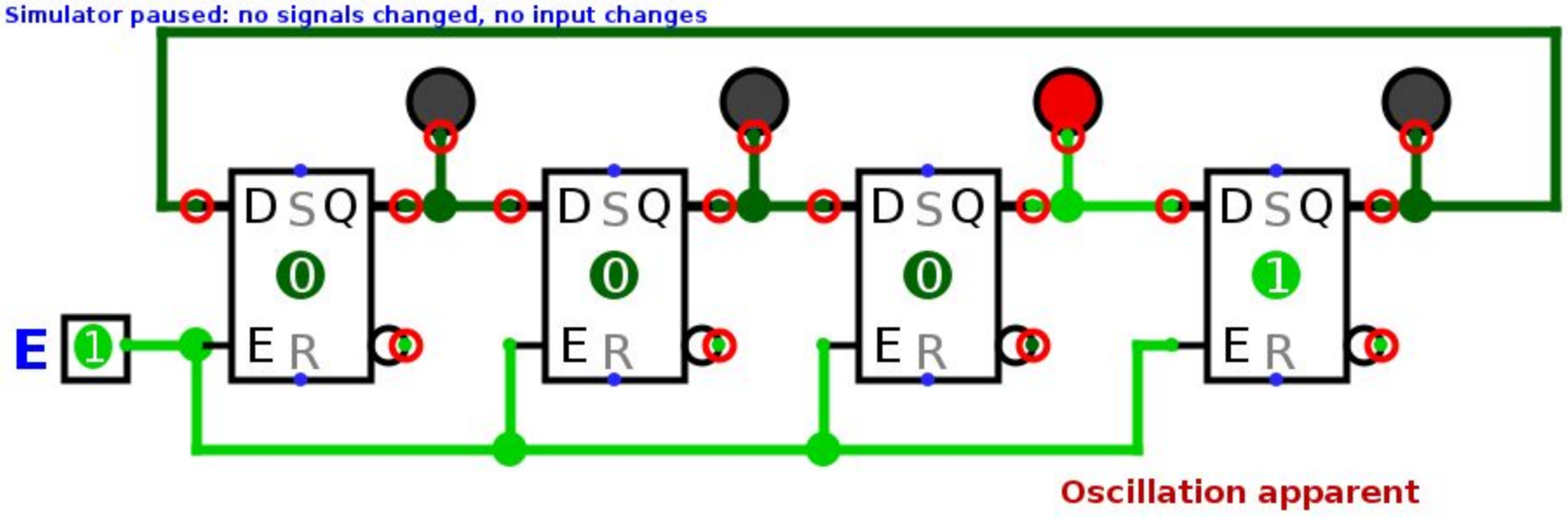
Un uno que se mueve

Todo parece estar bien… pero cuando damos el pulso en Enable….



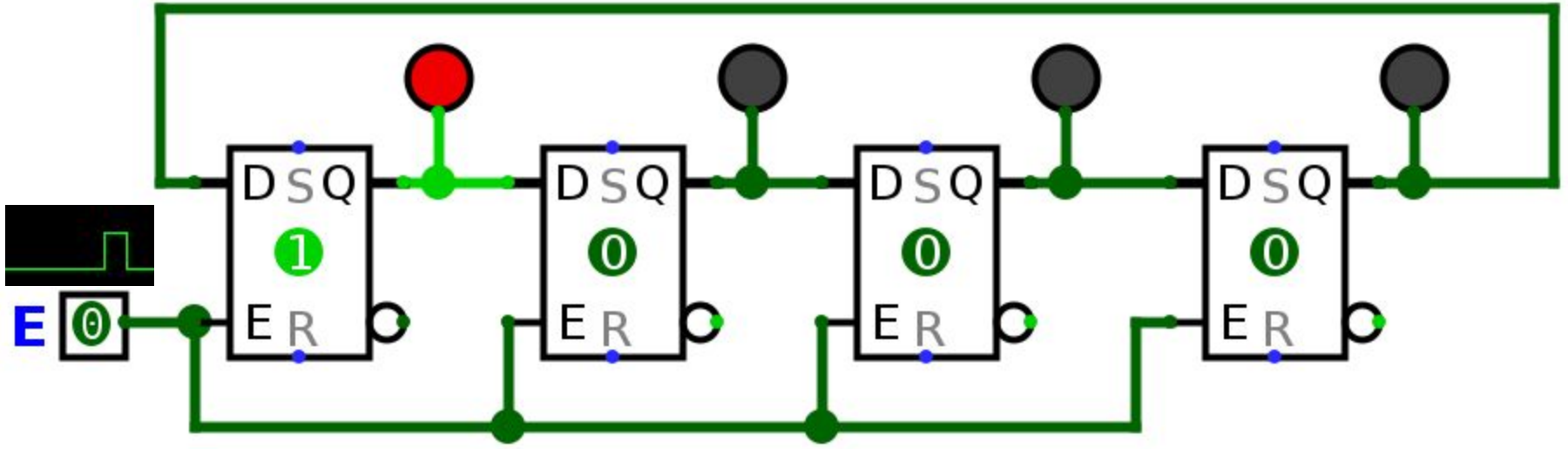
Un uno que se mueve

EXPLOTA TODO!. El simulador detiene la simulación e indica que existe oscilaciones. Eso quiere decir que el circuito nunca llega a un estado estable. Si lo pensamos… en el momento que damos E=1, ese valor ingresa a todos los FF, por ende se actualizan todos a la vez, y mientras E=1 siguen actualizando.



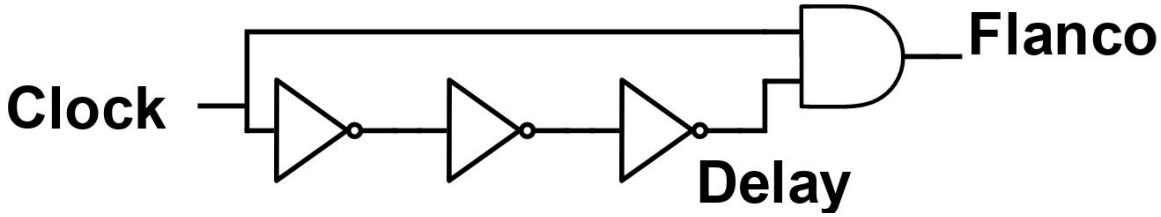
Necesitamos que la señal de Enable sea instantánea

Si logramos que el pulso de Enable sea del tiempo justo que le permita a un FF actualizar una única vez, este circuito funcionará correctamente. Para lograr eso vamos a utilizar un circuito detector de flanco.



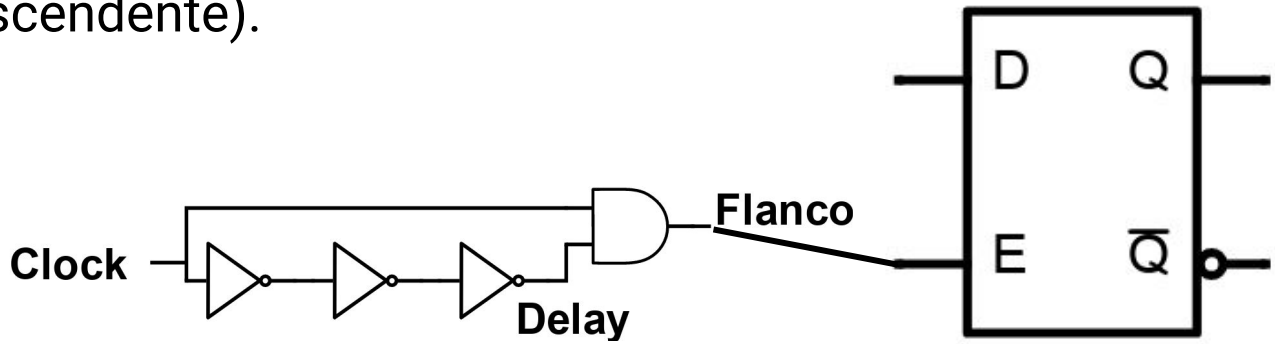
Detector de flanco ascendente

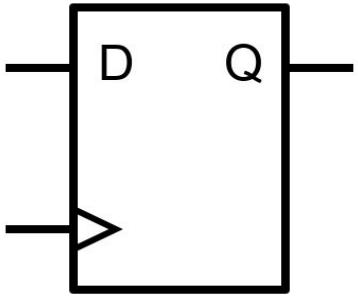
Vemos que este circuito tiene una entrada (Clock) y una salida (Flanco). Cuando la señal de Clock pasa de 0 a 1, una de las entradas de la AND pasa a 1, pero ese cambio llega 3 tiempos de propagación más tarde (delay) a la otra entrada. Durante este pequeño tiempo, el valor de la AND es uno (Flanco). Por ende, cuando se produce un cambio en Clock de 0 a 1 (flanco ascendente) el circuito genera un pulso en su salida. Solo ocurre en la transición de 0 a 1. Así como armamos un detector de flanco ascendente, de manera similar podemos generar un detector de flanco descendente.



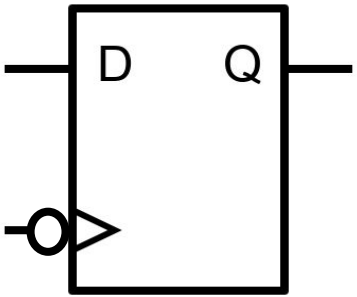
Detector de flanco ascendente

Si conectamos este circuito al enable del FF, obtenemos un nuevo FF D sincrónico por flanco (ascendente).





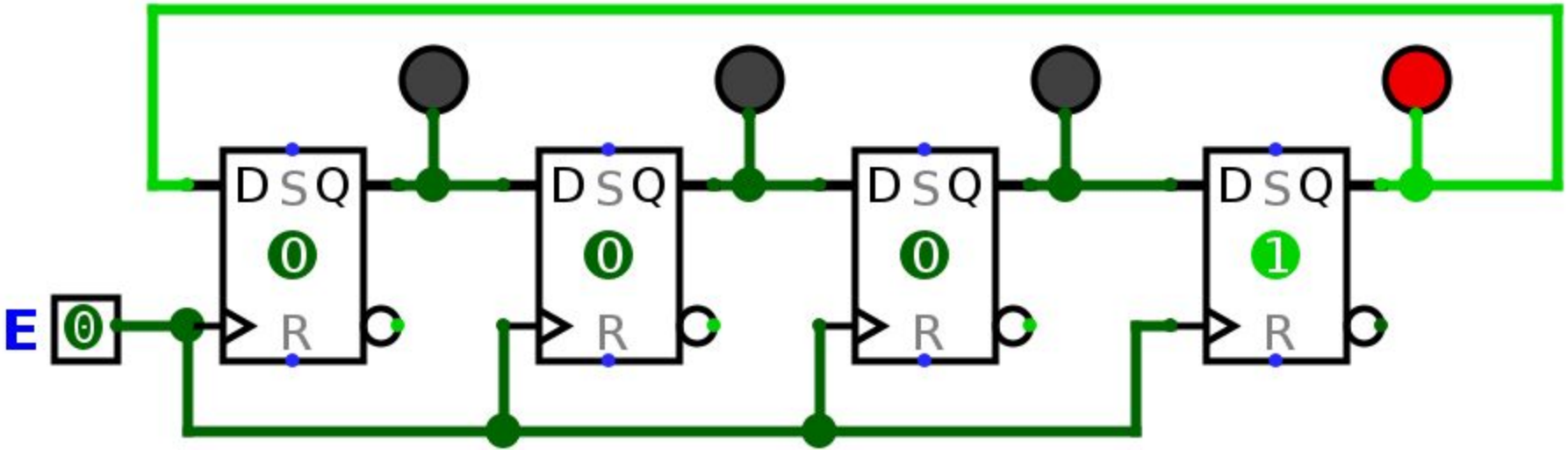
FF D Flanco Ascendente



FF D Flanco Descendente

Contador one hot

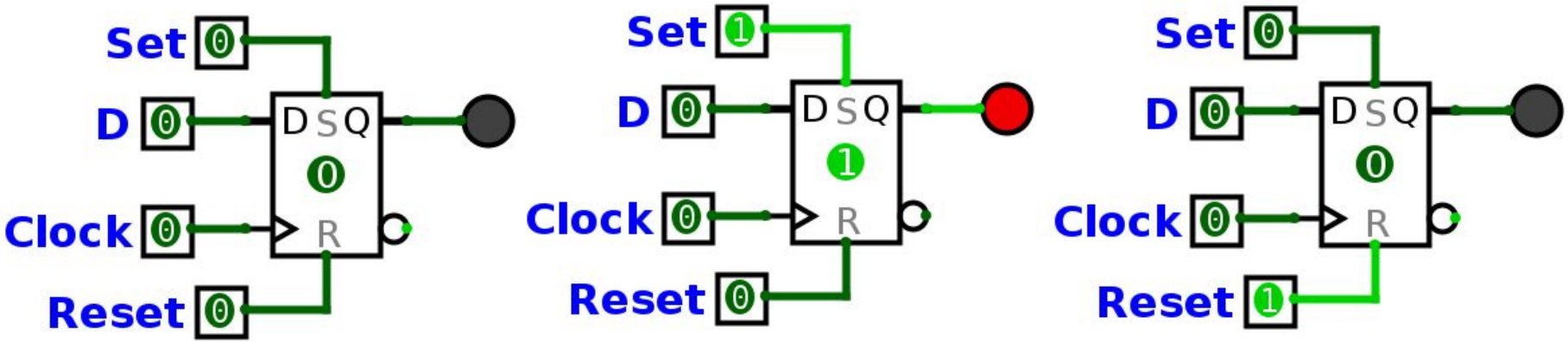
Utilizando FF D por flanco (en este caso ascendente) logramos fabricar el circuito sin problemas.



Tipos de Flip Flop

FF D Flanco ascendente - Entradas forzadas

Los FF D vienen acompañados de entradas forzadas (Set y Reset). Estas entradas sirven para forzar su estado inicial. En estos casos estas entradas son Asincrónicas, ya que sin importar la señal de Clock, van a forzar su valor.

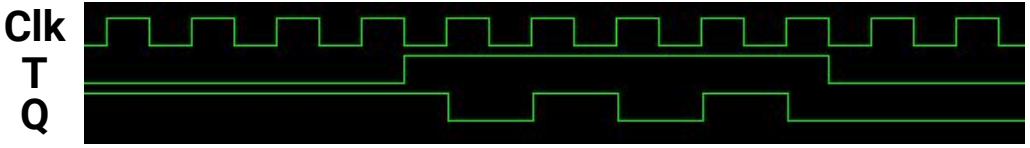


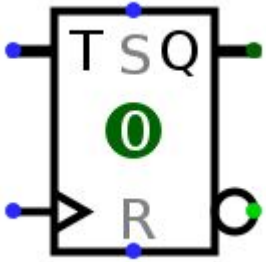
FF T - Toggle

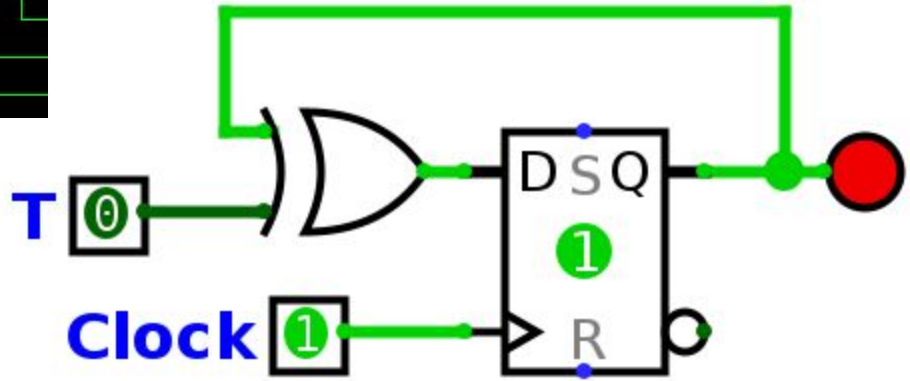
Utilizando un FF D podemos realimentar la salida mediante una XOR. Lo que logramos es un circuito con la siguiente tabla de verdad.
La misma indica que si T=0 y tenemos un flanco ascendente, entonces Q no cambia su valor, mientras que si T=1, al producirse un flanco ascendente la salida Q toma su valor inverso.

T	Clk	Q_N
0	↑	Q_{N-1}
1	↑	$!Q_{N-1}$

Vemos en el diagrama de tiempos que si dejamos T=1 la salida producida tiene el doble de periodo del clock.







Circuitos Secuenciales

Unidad 4.1
Registros y Contadores

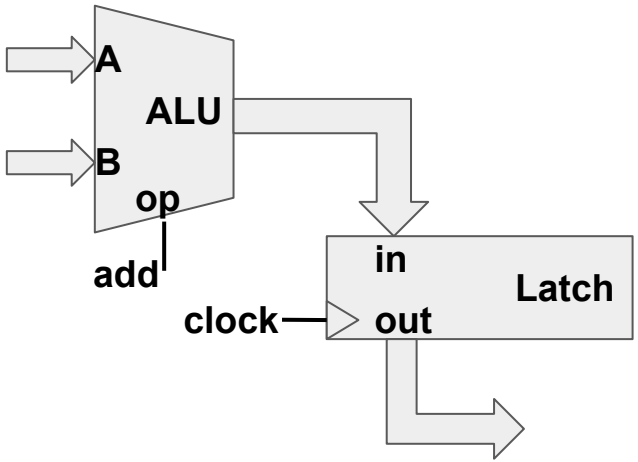
Versión 1.0.0

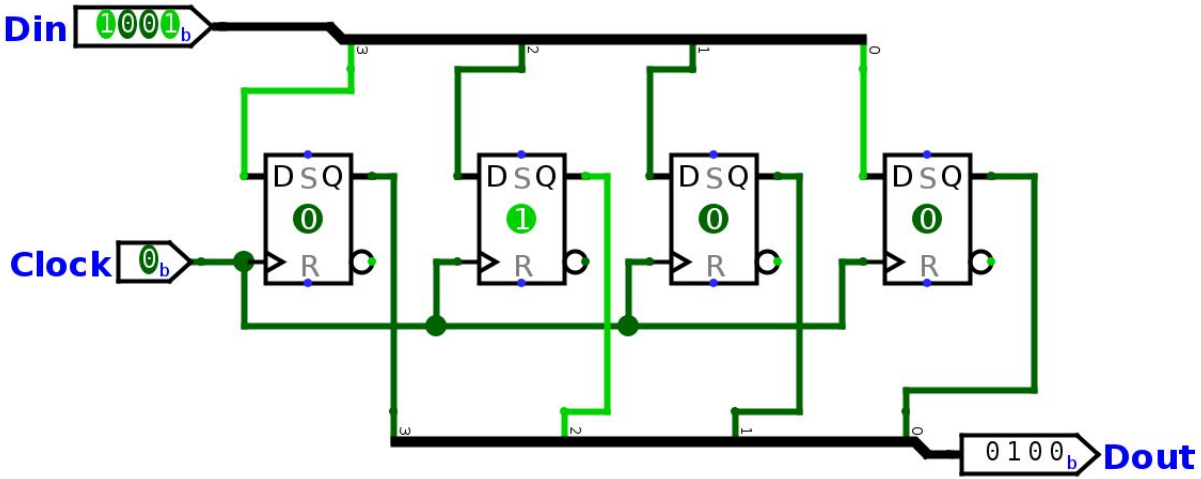
Jaír Hnatiuk, Carlos Maidana, Carlos Rodríguez, Edgardo Gho, Martín Ferreyra Biron

Registros

Registro Latch (Paralelo - Paralelo)

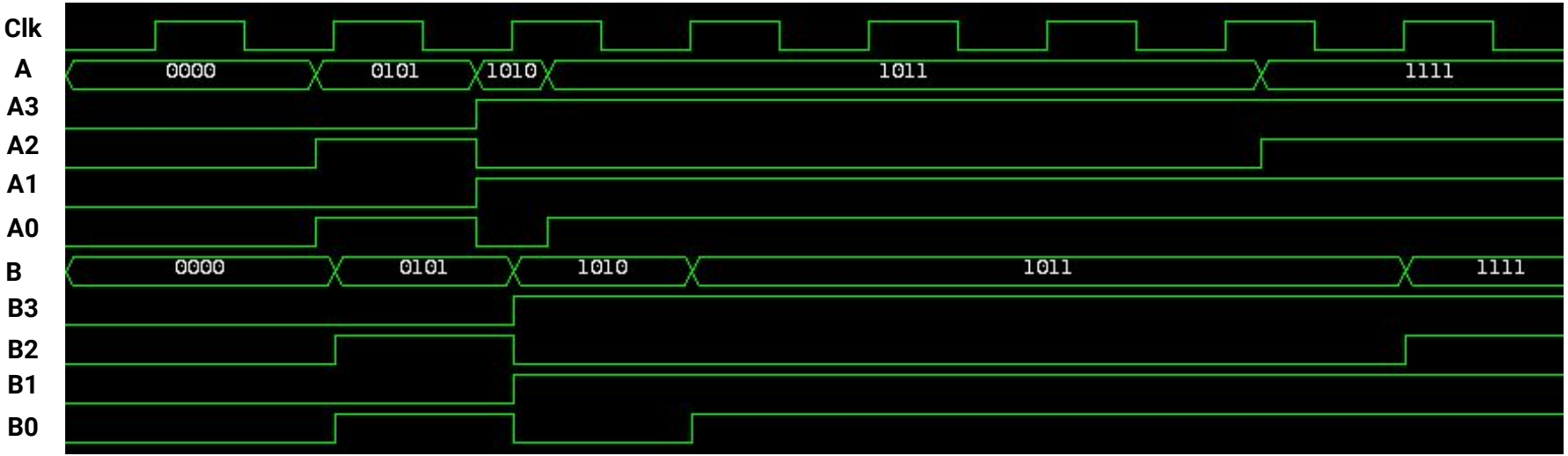
Cuando trabajamos con circuitos aritméticos, es común almacenar el resultado de una operación. Para esto usamos un circuito que memorice los bits del resultado. Dado que usamos generalmente operandos de 4,8,16,32,64 bits , vamos a necesitar tantos FF como bits tengan los operandos.





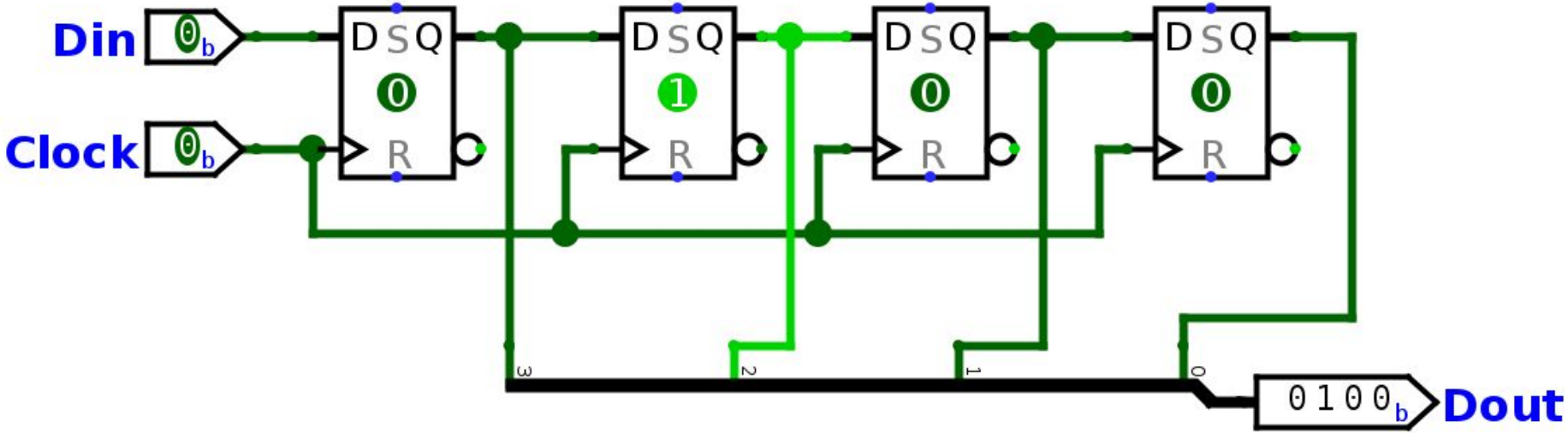
Registro Latch (Paralelo - Paralelo)

Cuando trabajamos con operandos (palabras) conviene representarlos con su respectivo peso en el diagrama de tiempos.

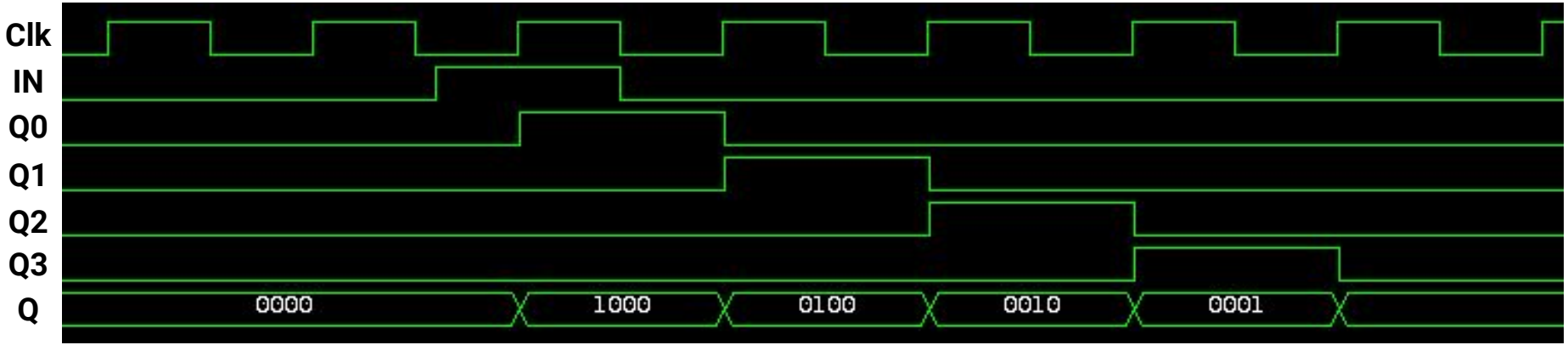


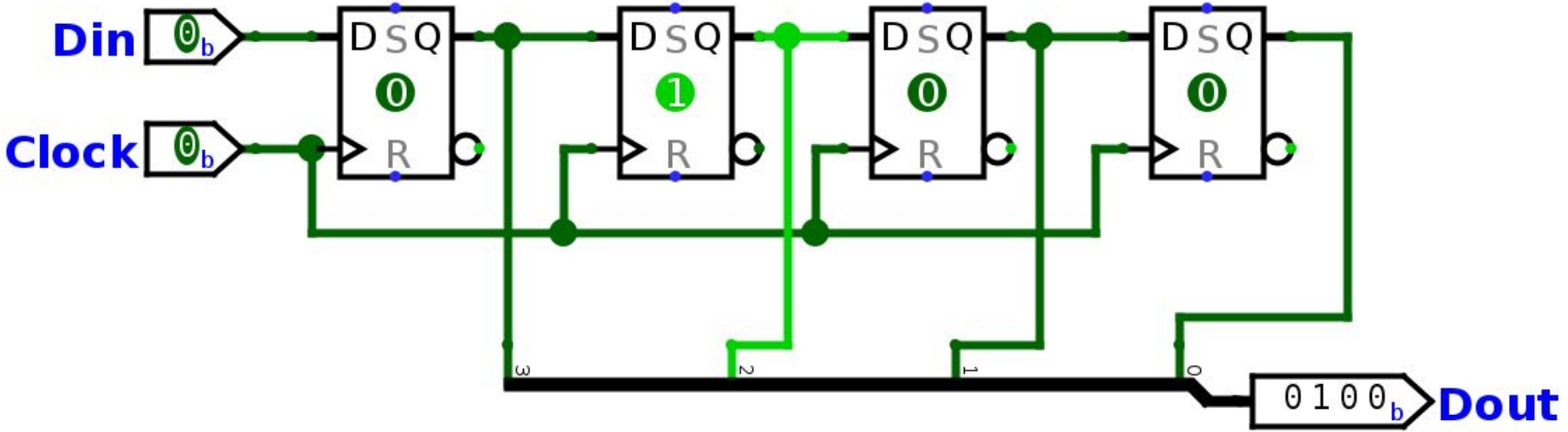
Registro Shift (Serie - Paralelo)

Podemos ingresar datos a un registro, un bit a la vez (serie), y que la salida del registro sea paralelo.



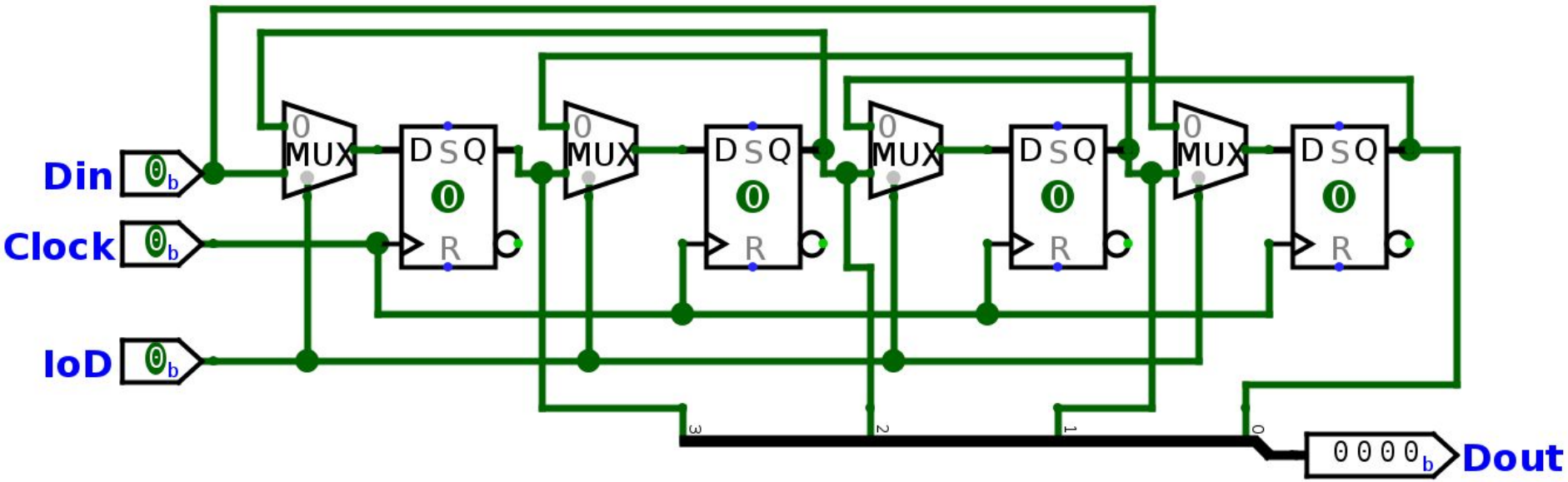
Registro Shift (Serie - Paralelo)





Registro Shift Izquierda o Derecha (Serie - Paralelo)

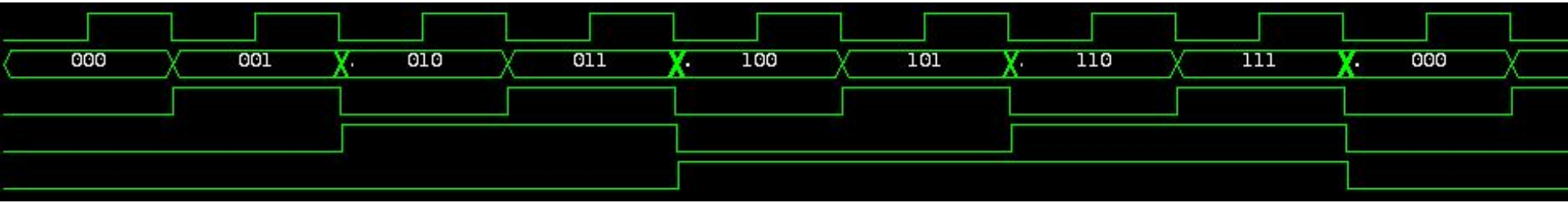
Utilizando MUXes podemos elegir el sentido para el cual se desplazan los bits.
La entrada IoD (izquierda o derecha) elige dato que ingresa en cada FF.



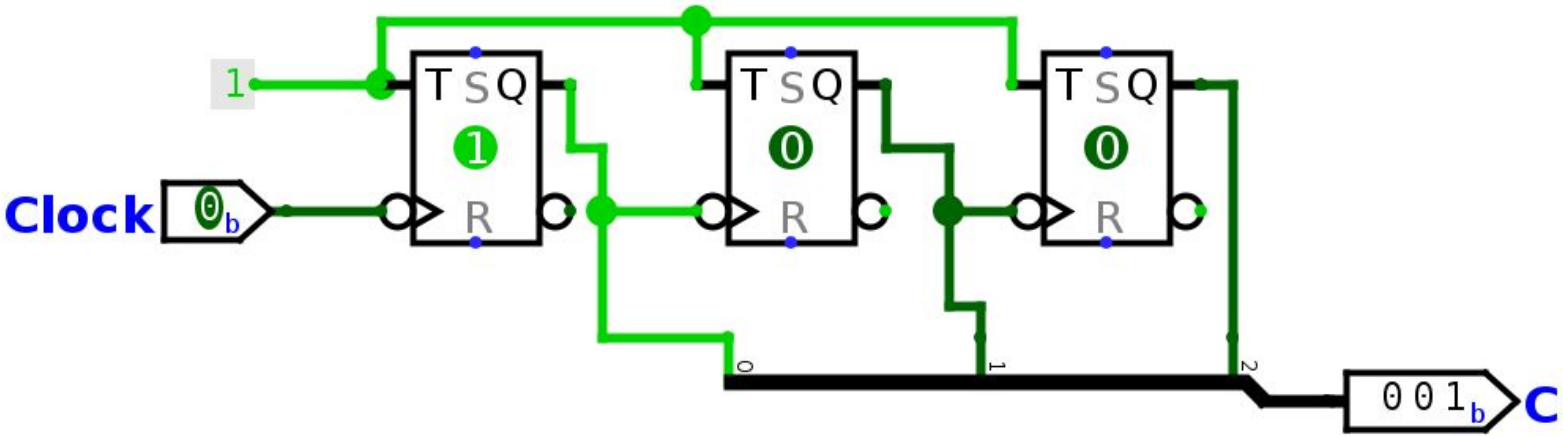
Contadores Binarios

Contador Asincrónico ascendente (2^N)

Los circuitos contadores generan una secuencia que se repite. En cada pulso de clock el circuito cuenta.

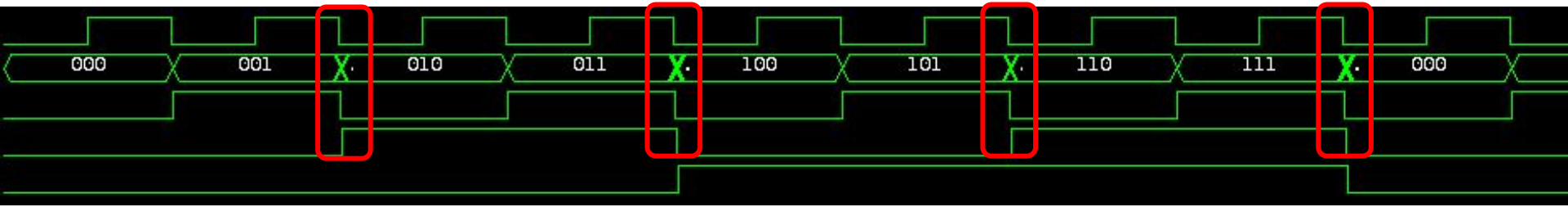


Clk	C_2	C_1	C_0
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

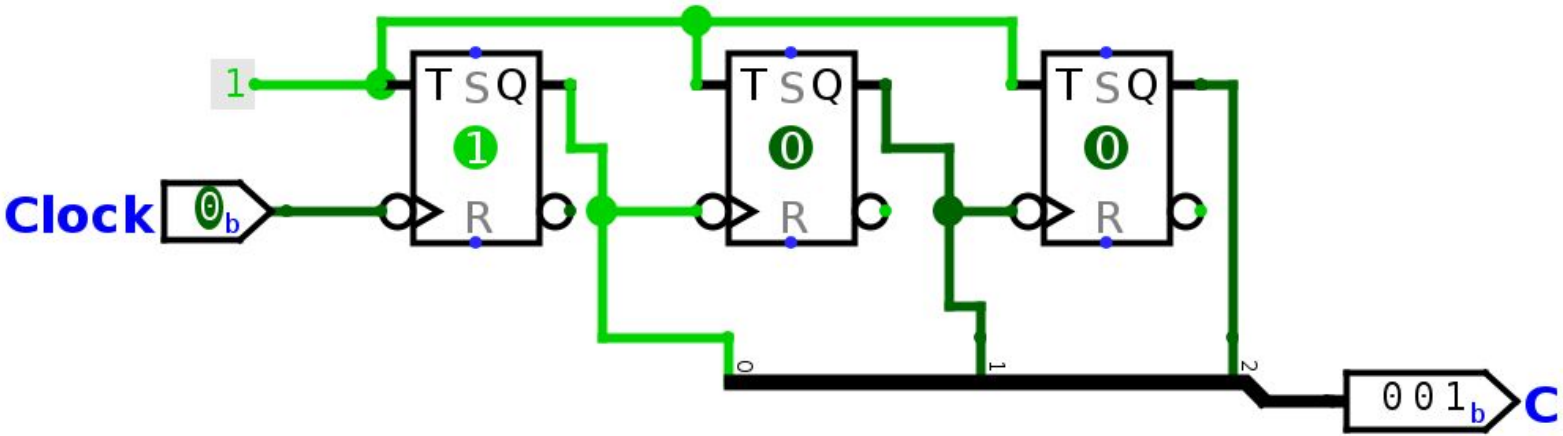


Contador Asincrónico ascendente (2^N)

Los circuitos contadores generan una secuencia que se repite. En cada pulso de clock el circuito cuenta.

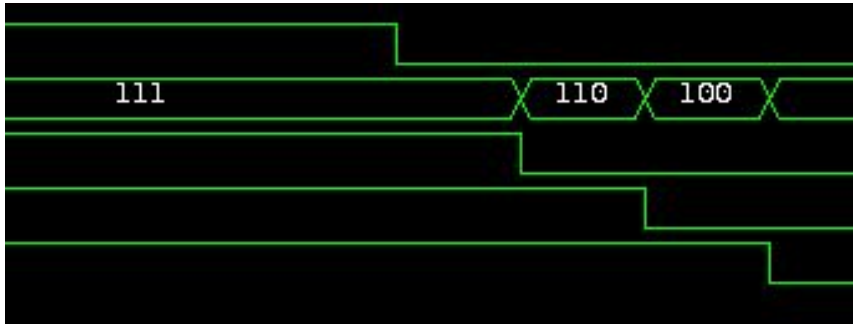


Clk	C_2	C_1	C_0
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

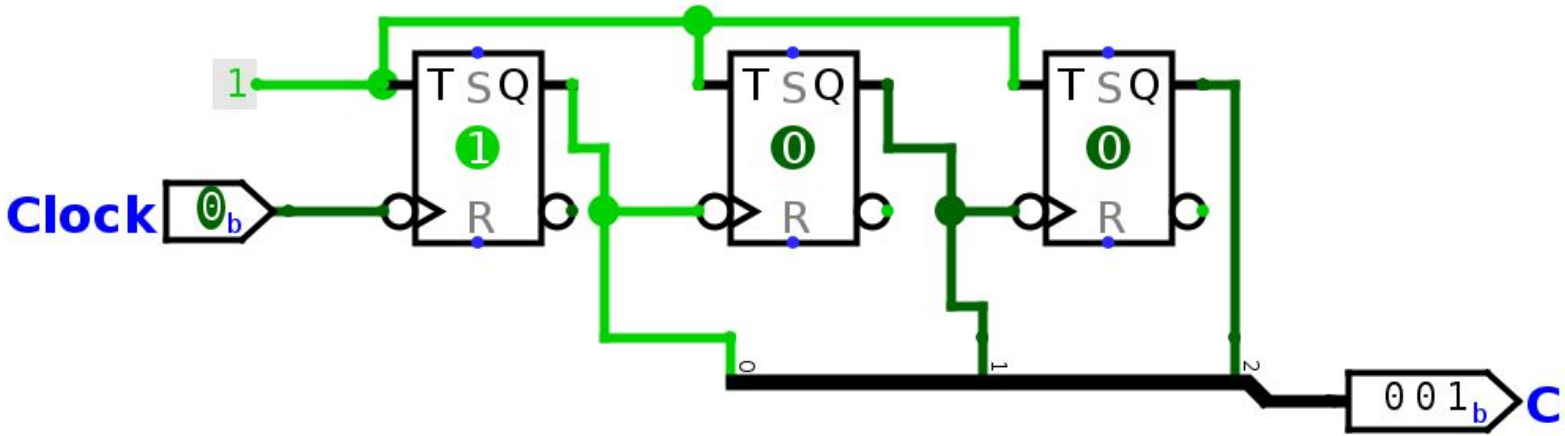


Contador Asincrónico ascendente (2^N)

El problema de este tipo de contadores es que se producen estados intermedios mientras los FF cambian sus valores.

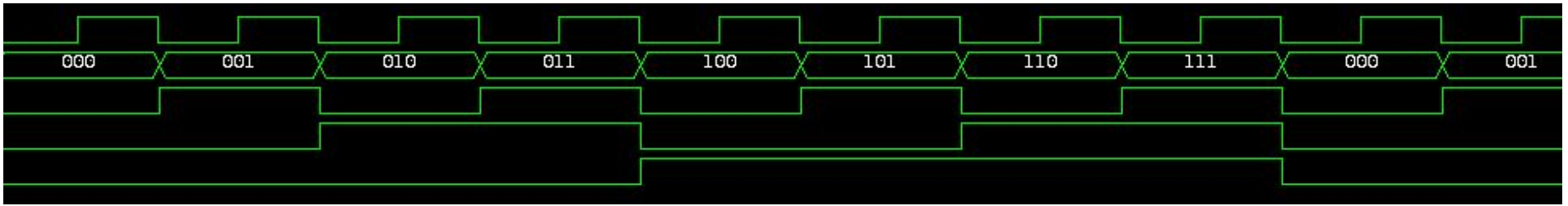


Clk	C_2	C_1	C_0
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

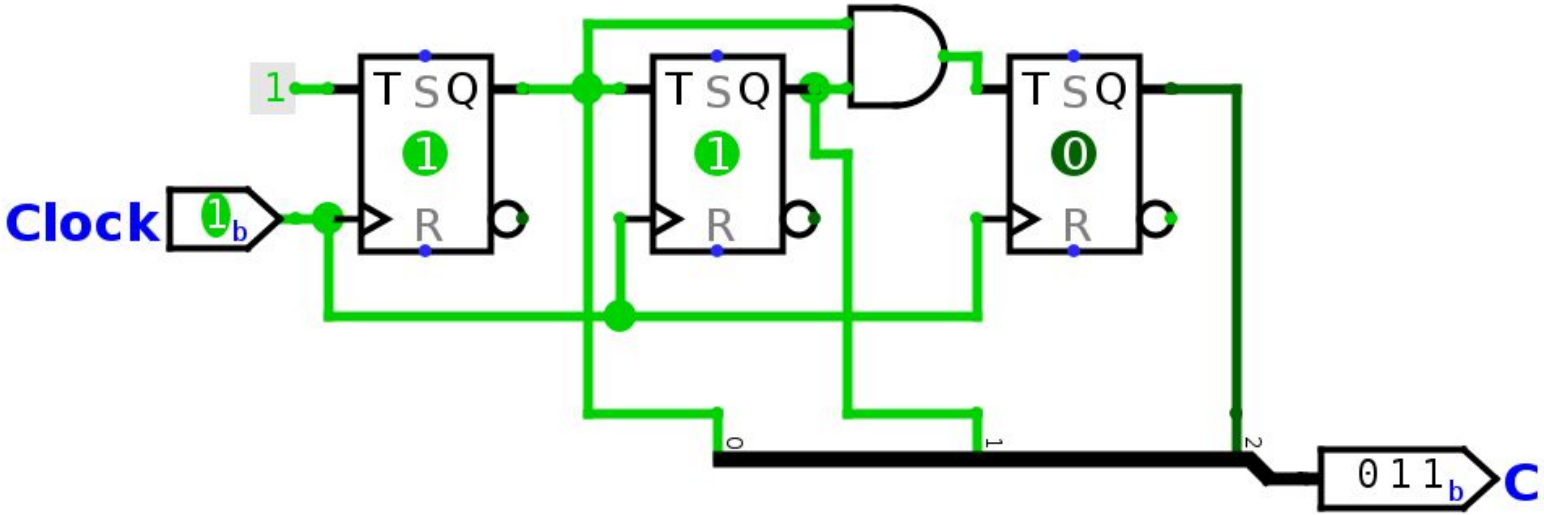


Contador Sincrónico ascendente (2^N)

Este contador es sincrónico ya que el clock es el mismo para todos los FF.
Vemos que cada FF mira los FF a su izquierda para decidir si cambia o no.

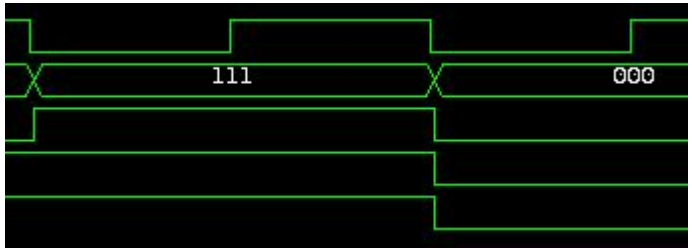


Clk	C_2	C_1	C_0
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

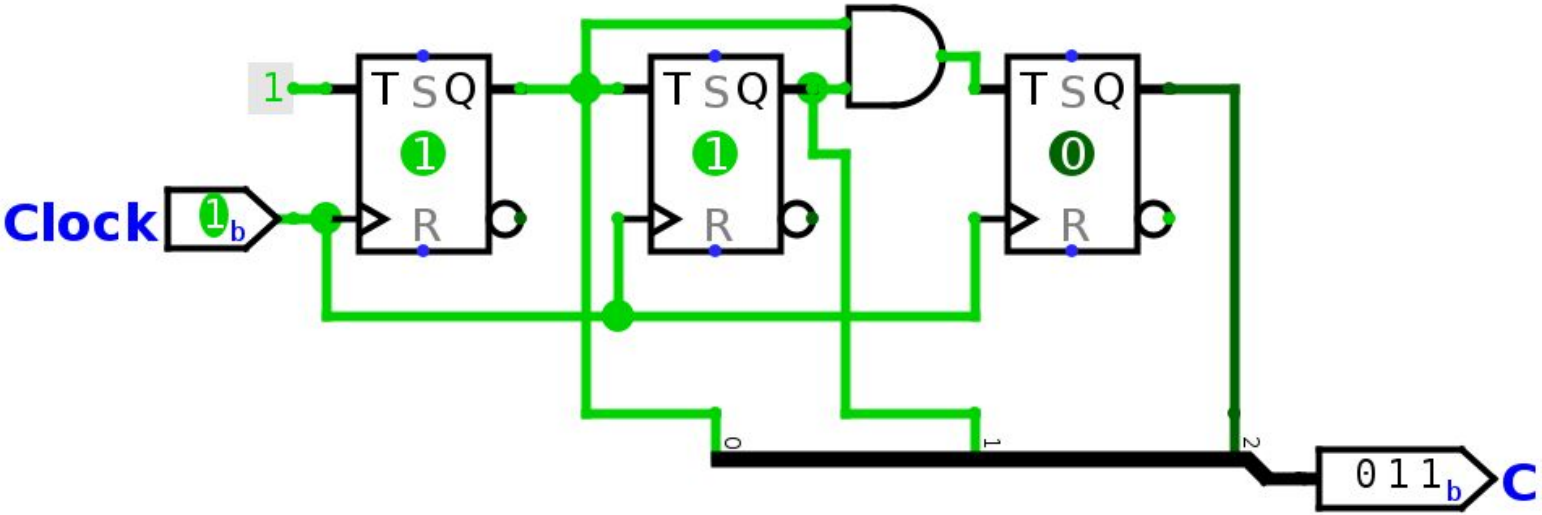


Contador Sincrónico ascendente (2^N)

No existen estados intermedios con este tipo de contadores cuando cuentan su módulo completo (en este caso 8)

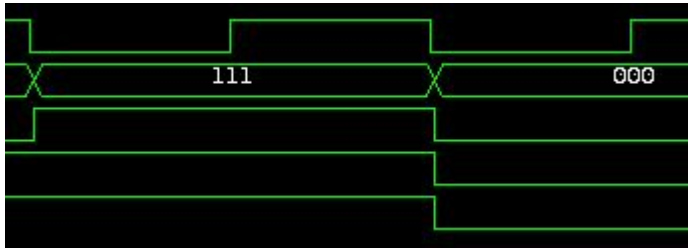


Clk	C_2	C_1	C_0
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

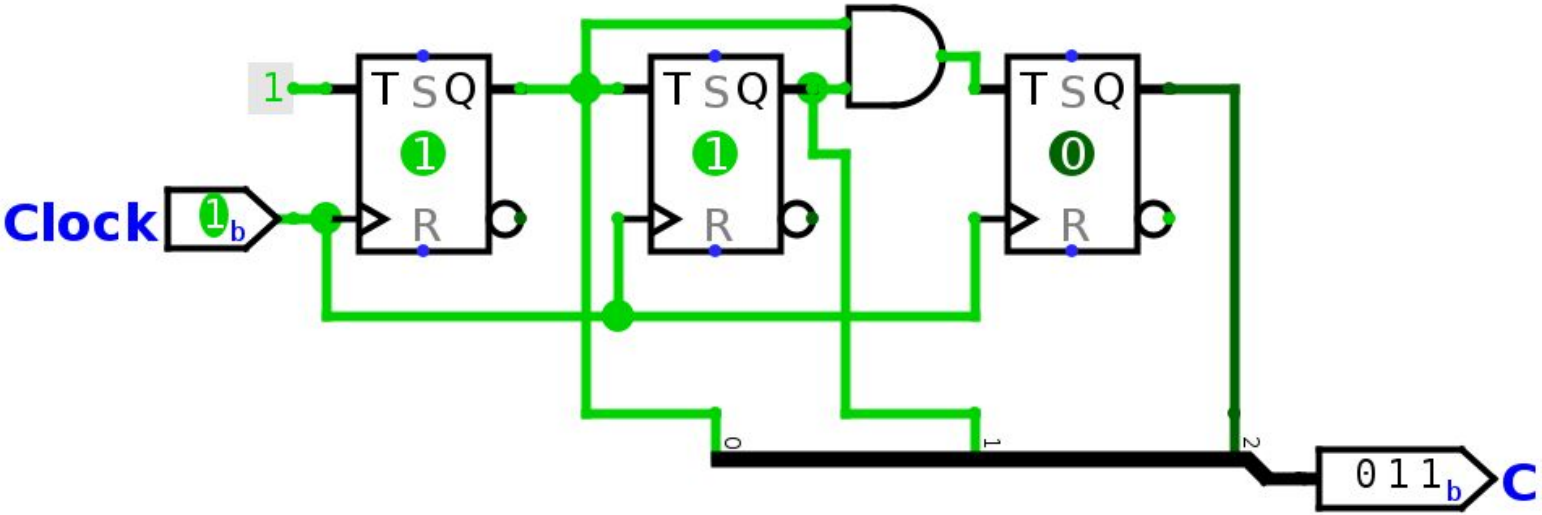


Contador Sincrónico ascendente (2^N)

No existen estados intermedios con este tipo de contadores cuando cuentan su módulo completo (en este caso 8)

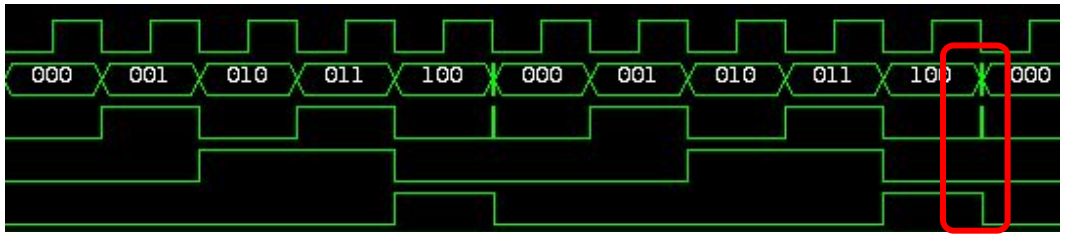


Clk	C_2	C_1	C_0
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

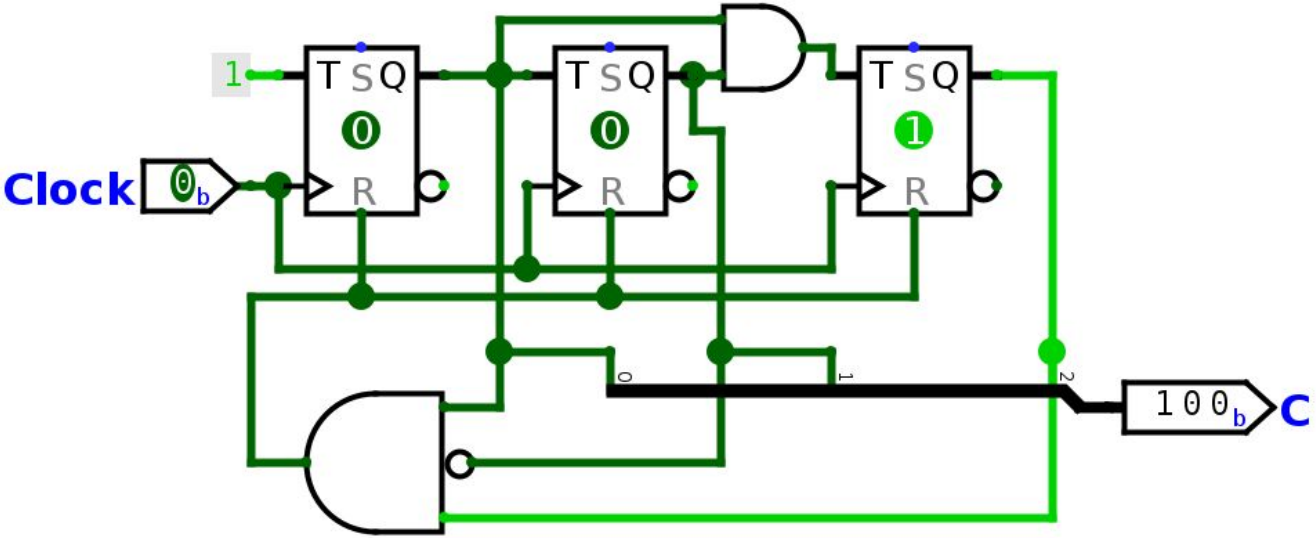


Contador Sincrónico ascendente reset asincrónico (módulo N)

Podemos detectar una combinación de salida y fabricar un uno para el reset.
Eso nos deja crear un módulo a medida, en este caso 5.

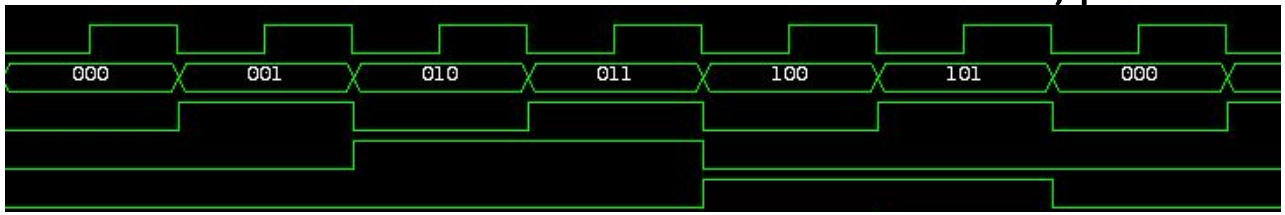


Clk	C_2	C_1	C_0
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
~~5~~	~~1~~	~~0~~	~~1~~
~~6~~	~~1~~	~~1~~	~~0~~
~~7~~	~~1~~	~~1~~	~~1~~

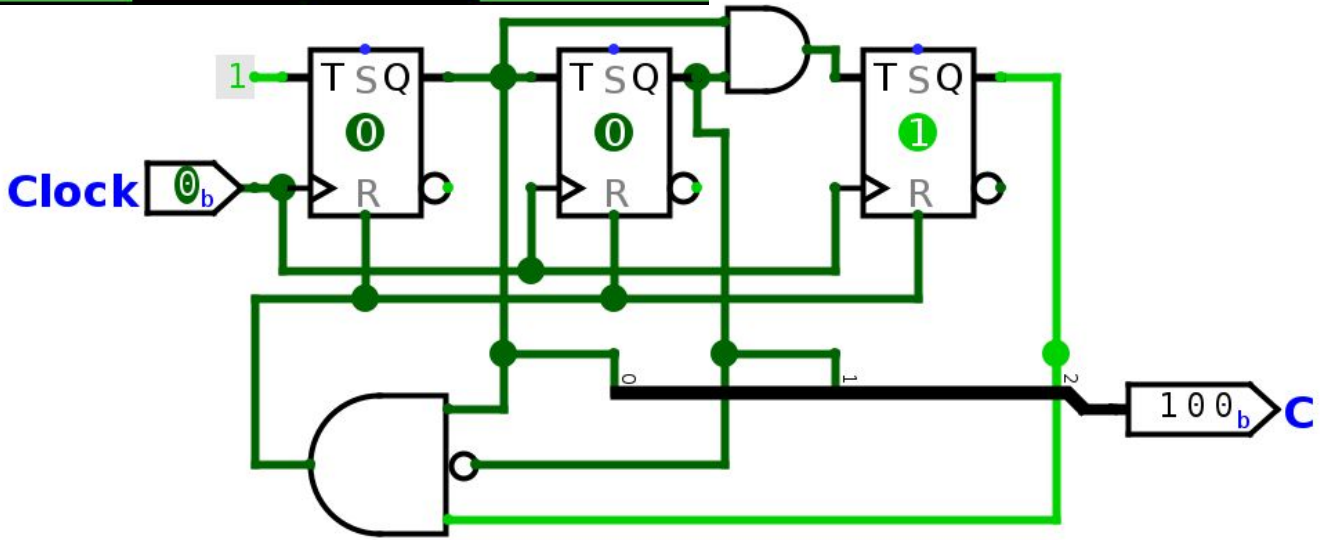


Contador Sincrónico ascendente reset sincrónico (módulo N')

Si los FF se resetean con Resets sincrónicos (es decir que funcionan solo si hay un flanco ascendente clock), entonces no se produce el estado extra. Pero ahora cuenta un elemento mas en el modulo, por ende N' = N+1.



Clk	C_2	C_1	C_0
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	**1**	**0**	**1**
~~6~~	~~1~~	~~1~~	~~0~~
~~7~~	~~1~~	~~1~~	~~1~~

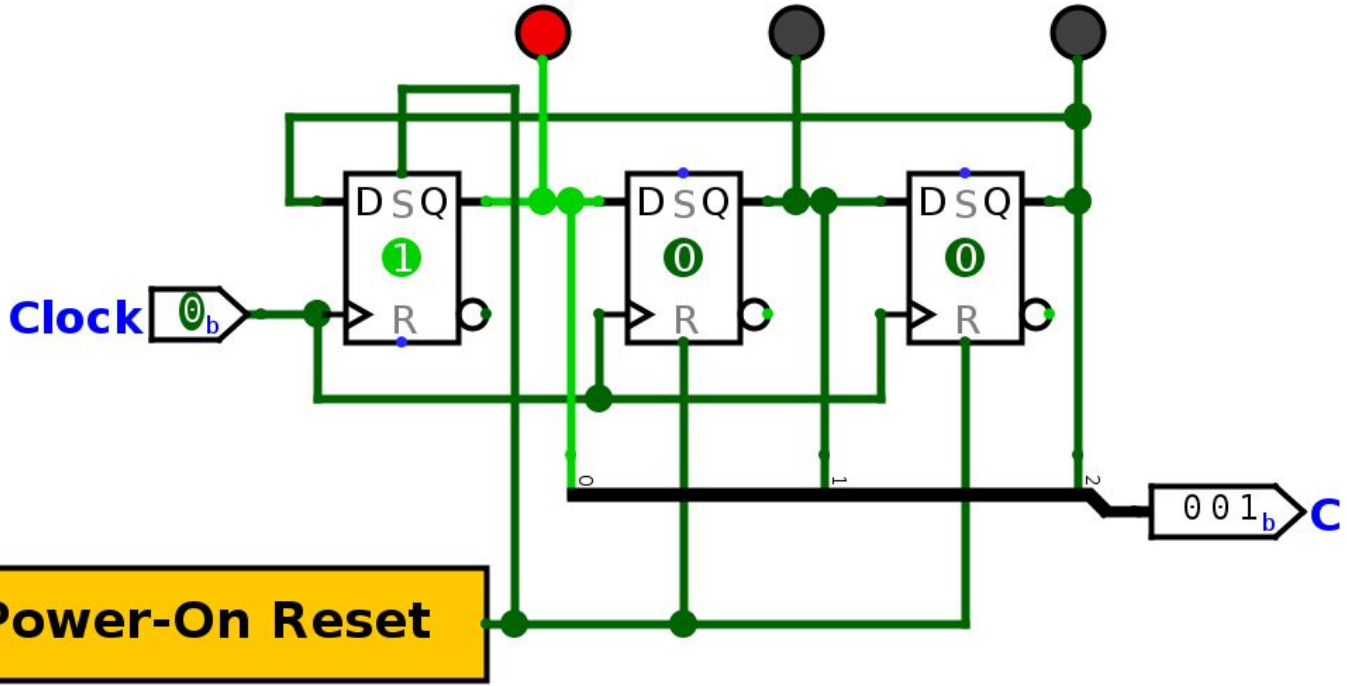


Contadores NO Binarios

Contador en anillo (one-hot) módulo N

Este contador requiere que durante el inicio (Power-On Reset) el valor de un FF tome 1 mientras el resto tomen 0. A partir de ahí, cada pulso va a “correr” el uno al siguiente FF y se genera un código One-Hot. En logisim-evolution usamos Simulate>Reset Simulation.

Clk	C_2	C_1	C_0
0	0	0	1
1	0	1	0
2	1	0	0



Contador Johnson (módulo 2N)

Este contador genera el código Johnson.

Clk	C_2	C_1	C_0
0	0	0	0
1	0	0	1
2	0	1	1
3	1	1	1
4	1	1	0
5	1	0	0