Práctica 2: Lógica Digital - Secuenciales Parte 2

Primer Cuatrimestre 2023

Organización del Computador I
DC - UBA

Introducción

Sobre la clase de hoy

Hoy vamos a ver los principios de diseño, práctica y ejemplos de circuitos secuenciales, la estructura de la clase va ser la siguiente:

- **Circuitos combinatorios - Timing**
- **Retroalimentación y cambio de modelo**
- **Circuitos secuenciales asincrónicos**
- **Circuitos secuenciales sincrónicos**
- **Latchs - Flip-flops, registros y memorias**
- **Máquinas de estado**

Sobre la clase de hoy

Hoy vamos a ver los principios de diseño, práctica y ejemplos de circuitos secuenciales, la estructura de la clase va ser la siguiente:

- **Circuitos combinatorios - Timing**
- **Retroalimentación y cambio de modelo**
- **Circuitos secuenciales asincrónicos**
- **Circuitos secuenciales sincrónicos**
- **Latchs - Flip-flops, registros y memorias**
- **Máquinas de estado**

Sobre la clase de hoy

Hoy vamos a ver los principios de diseño, práctica y ejemplos de circuitos secuenciales, la estructura de la clase va ser la siguiente:

- **Circuitos combinatorios - Timing**
- **Retroalimentación y cambio de modelo**
- **Circuitos secuenciales asincrónicos**
- **Circuitos secuenciales sincrónicos**
- **Latchs - Flip-flops, registros y memorias**
- **Máquinas de estado**

Latchs - Flip-flops

Latchs

Son circuitos que permiten *trabar o asegurar* el valor de su salida

Latchs

Son circuitos que permiten *trabar o asegurar* el valor de su salida

- Permiten el cambio de sus salidas según el **nivel** de las entradas.

Latchs

Son circuitos que permiten *trabar o asegurar* el valor de su salida

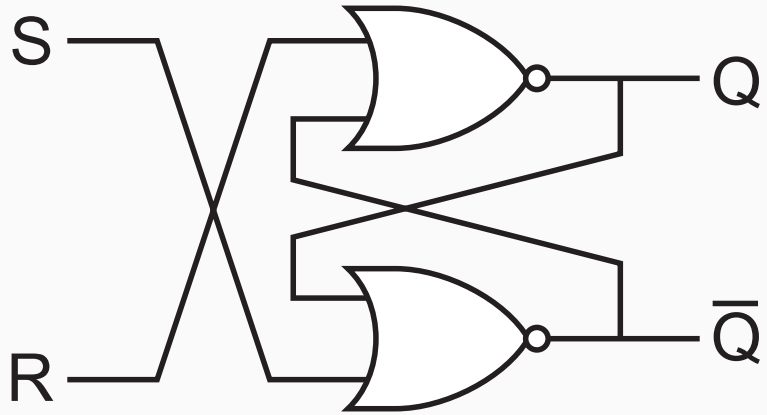
- Permiten el cambio de sus salidas según el **nivel** de las entradas.
- Utilizan **realimentación**

Latchs

Son circuitos que permiten *trabar o asegurar* el valor de su salida

- Permiten el cambio de sus salidas según el **nivel** de las entradas.
- Utilizan **realimentación**

Ejemplo:



Latch RS (Reset-Set)

Analicemos el ejemplo anterior:

Latch RS implementado con NOR:

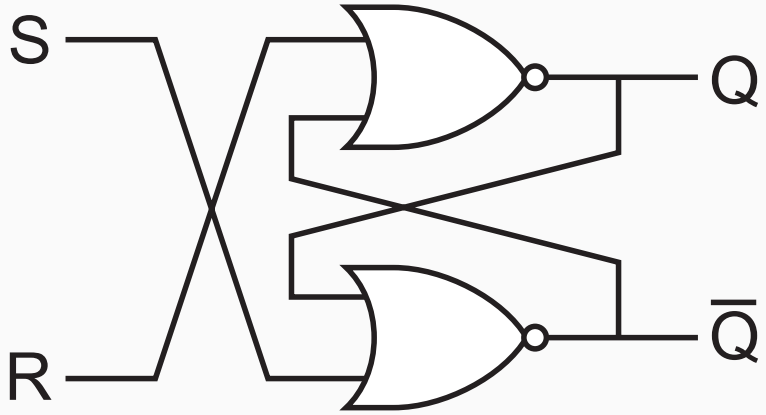


Tabla de verdad:

S	R	Q	$\overline{Q}$
1	0		
0	1		
0	0		
1	1		

Latch RS (Reset-Set)

Analicemos el ejemplo anterior:

Latch RS implementado con NOR:

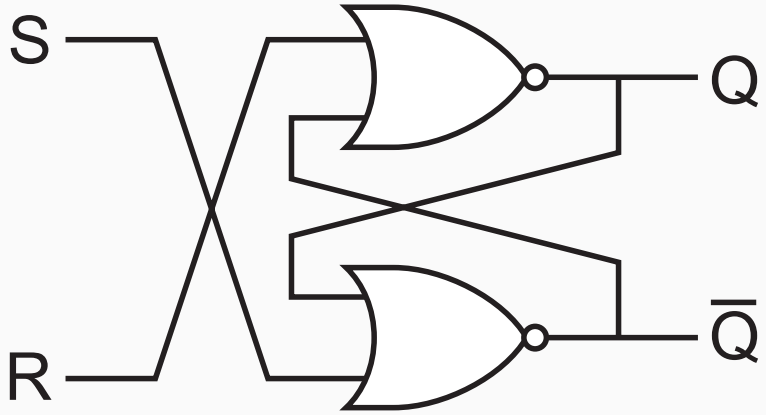


Tabla de verdad:

S	R	Q	$\overline{Q}$
1	0		0
0	1		
0	0		
1	1		

Latch RS (Reset-Set)

Analicemos el ejemplo anterior:

Latch RS implementado con NOR:

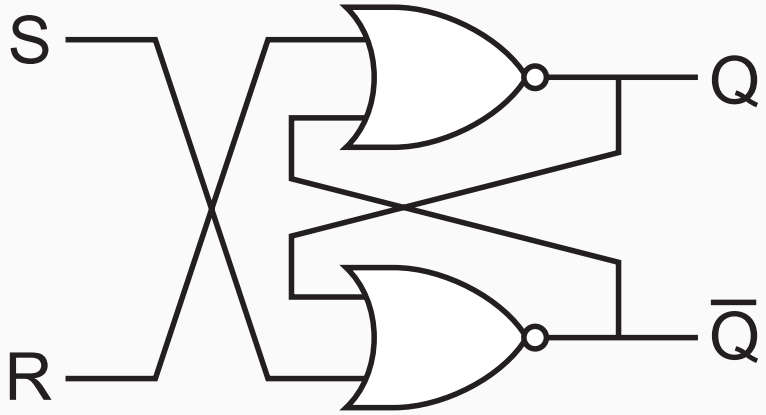


Tabla de verdad:

S	R	Q	$\overline{Q}$
1	0	1	0
0	1		
0	0		
1	1		

Latch RS (Reset-Set)

Analicemos el ejemplo anterior:

Latch RS implementado con NOR:

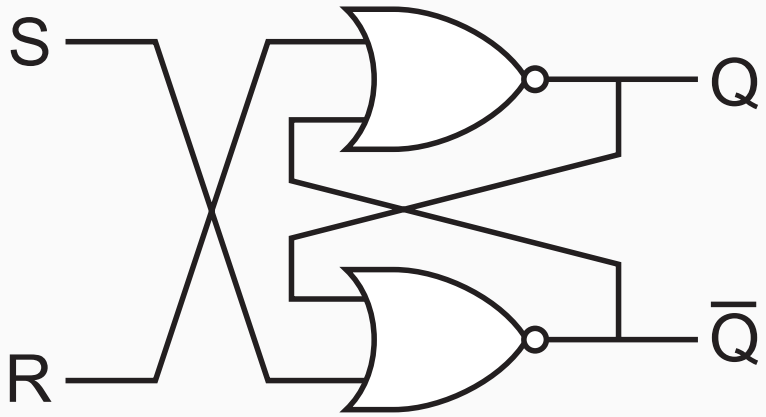


Tabla de verdad:

S	R	Q	$\overline{Q}$
1	0	1	0
0	1	0	1
0	0		
1	1		

Latch RS (Reset-Set)

Analicemos el ejemplo anterior:

Latch RS implementado con NOR:

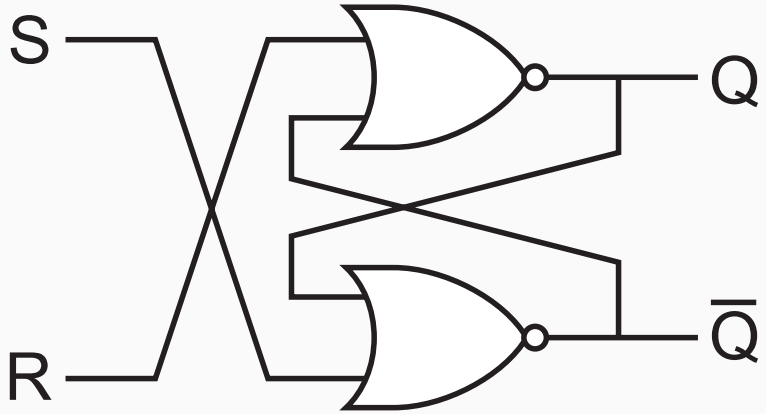


Tabla de verdad:

S	R	Q	$\overline{Q}$
1	0	1	0
0	1	0	1
0	0	$Q*$	$\overline{Q}*$ [1]
1	1		

[1] $Q*$ o $\overline{Q}*$ refiere al *estado* anterior de la salida

Latch RS (Reset-Set)

Analicemos el ejemplo anterior:

Latch RS implementado con NOR:

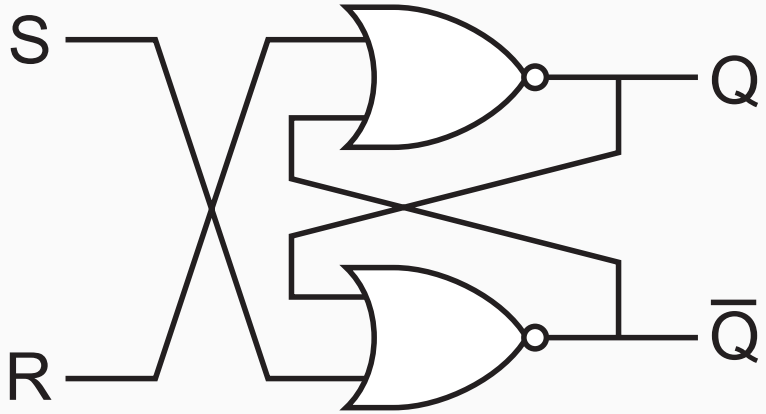


Tabla de verdad:

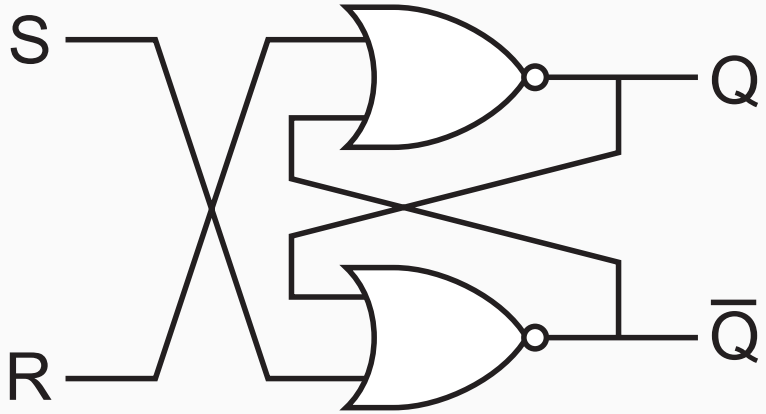
S	R	Q	$\overline{Q}$
1	0	1	0
0	1	0	1
0	0	$Q*$	$\overline{Q}*$ [1]
1	1	0	0

[1] $Q*$ o $\overline{Q}*$ refiere al *estado* anterior de la salida

Latch RS (Reset-Set)

Analicemos el ejemplo anterior:

Latch RS implementado con NOR:



Con $S, R = (1, 1)$:

Tabla de verdad:

S	R	Q	$\overline{Q}$
1	0	1	0
0	1	0	1
0	0	$Q*$	$\overline{Q}*$ [1]
1	1	0	0

[1] $Q*$ o $\overline{Q}*$ refiere al *estado* anterior de la salida

Latch RS (Reset-Set)

Analicemos el ejemplo anterior:

Latch RS implementado con NOR:

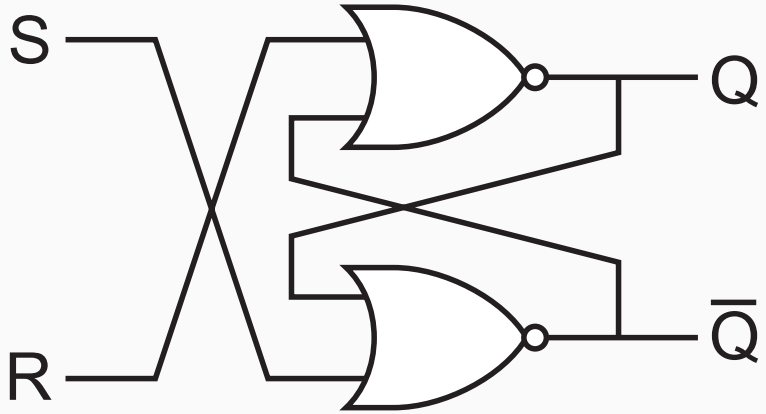


Tabla de verdad:

S	R	Q	$\overline{Q}$
1	0	1	0
0	1	0	1
0	0	$Q*$	$\overline{Q}*$ [1]
1	1	0	0

Con $S, R = (1, 1)$:

- El valor de las salidas es inconsistente con la especificación

[1] $Q*$ o $\overline{Q}*$ refiere al *estado* anterior de la salida

Latch RS (Reset-Set)

Analicemos el ejemplo anterior:

Latch RS implementado con NOR:

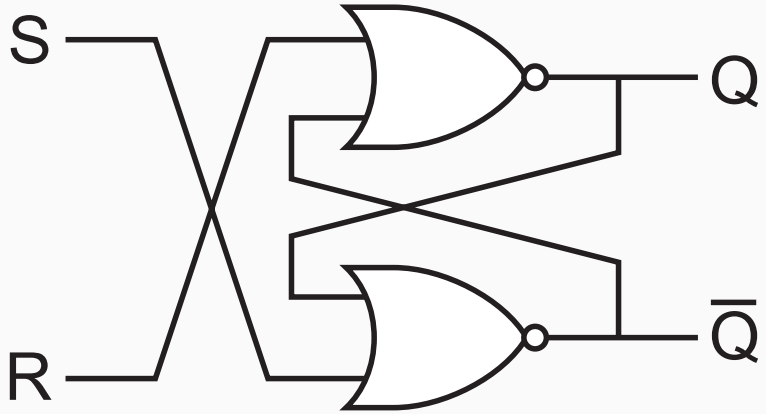


Tabla de verdad:

S	R	Q	$\overline{Q}$
1	0	1	0
0	1	0	1
0	0	$Q*$	$\overline{Q}*$ [1]
1	1	0	0

Con $S, R = (1, 1)$:

- El valor de las salidas es inconsistente con la especificación
- El valor de las salidas depende de la implementación. **Tarea:** implementar con NANDs

[1] $Q*$ o $\overline{Q}*$ refiere al *estado* anterior de la salida

Latch JK

Tratemos de modificar el comportamiento para el caso cuando las entradas son $(1, 1)$:

Latch JK:

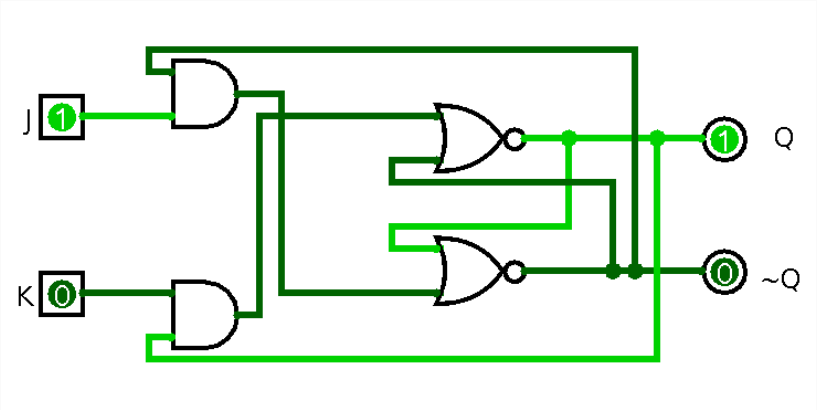


Tabla de verdad:

J	K	Q	$\overline{Q}$
1	0		
0	1		
0	0		
1	1		

Latch JK

Tratemos de modificar el comportamiento para el caso cuando las entradas son $(1, 1)$:

Latch JK:

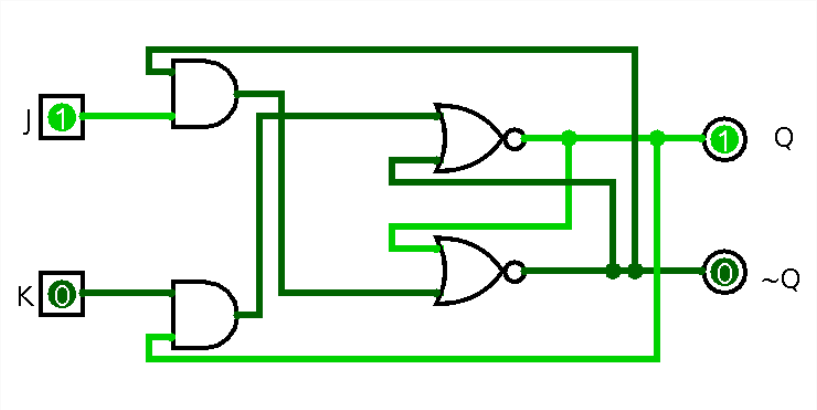


Tabla de verdad:

J	K	Q	$\overline{Q}$
1	0		0
0	1		
0	0		
1	1		

Latch JK

Tratemos de modificar el comportamiento para el caso cuando las entradas son $(1, 1)$:

Latch JK:

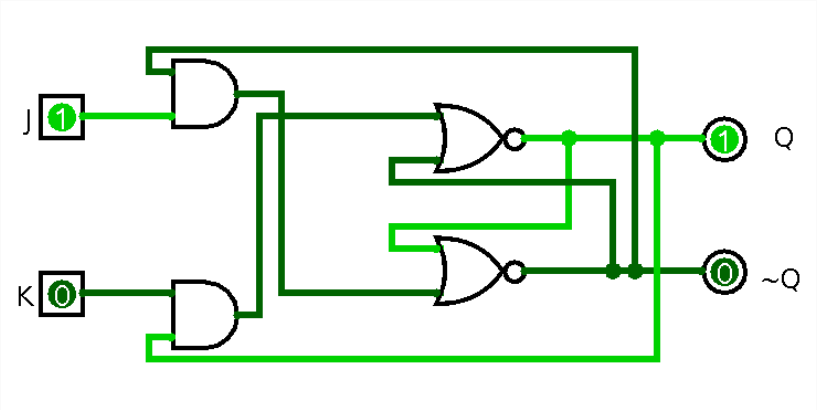


Tabla de verdad:

J	K	Q	$\overline{Q}$
1	0	1	0
0	1		
0	0		
1	1		

Latch JK

Tratemos de modificar el comportamiento para el caso cuando las entradas son $(1, 1)$:

Latch JK:

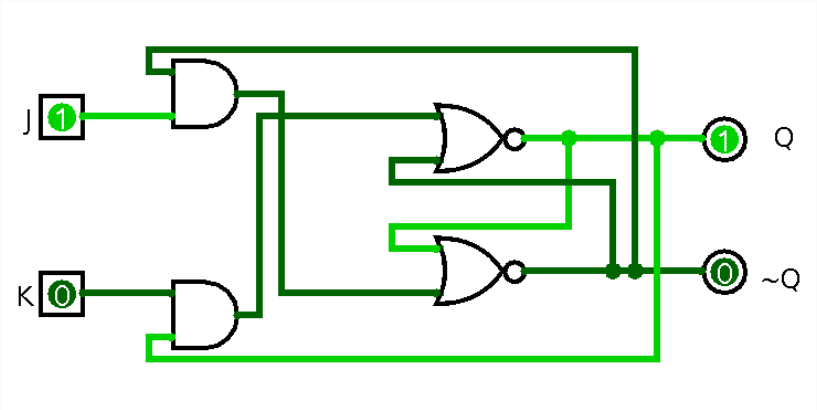


Tabla de verdad:

J	K	Q	$\overline{Q}$
1	0	1	0
0	1	0	1
0	0		
1	1		

Latch JK

Tratemos de modificar el comportamiento para el caso cuando las entradas son $(1, 1)$:

Latch JK:

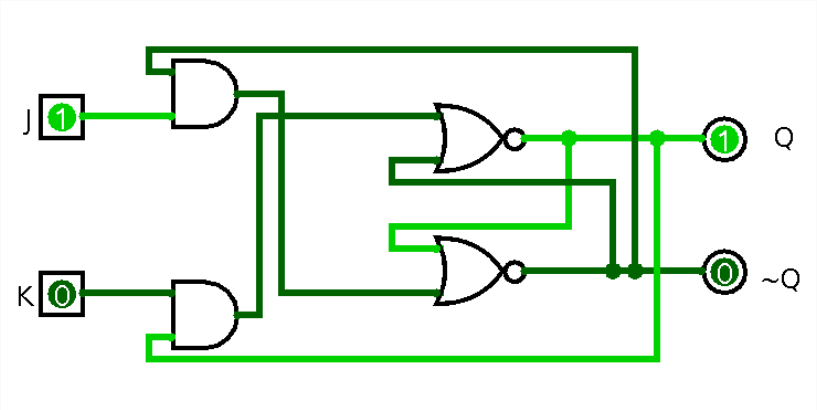


Tabla de verdad:

J	K	Q	$\overline{Q}$
1	0	1	0
0	1	0	1
0	0	$Q*$	$\overline{Q}*$
1	1		

Latch JK

Tratemos de modificar el comportamiento para el caso cuando las entradas son $(1, 1)$:

Latch JK:

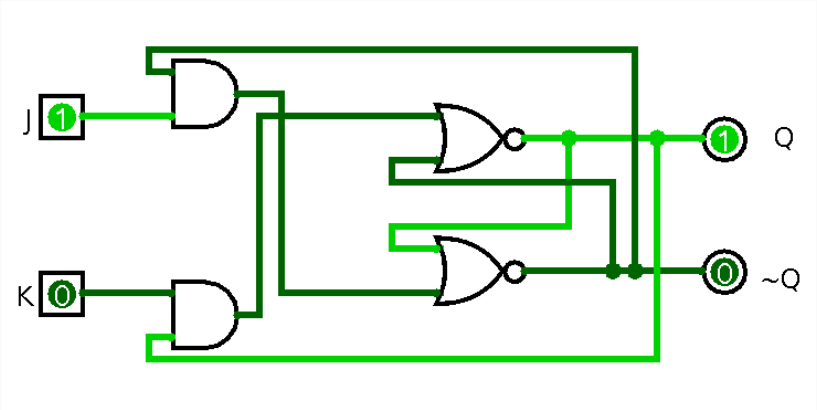


Tabla de verdad:

J	K	Q	$\overline{Q}$
1	0	1	0
0	1	0	1
0	0	$Q*$	$\overline{Q}*$
1	1	$\overline{Q}*$	$Q*$

Latch JK

Tratemos de modificar el comportamiento para el caso cuando las entradas son $(1, 1)$:

Latch JK:

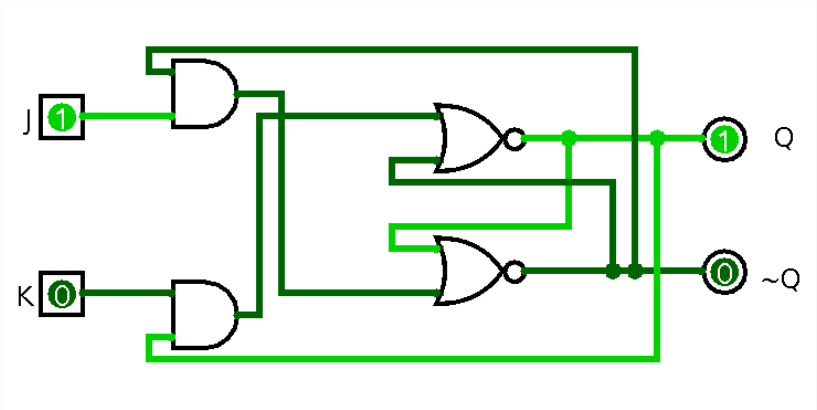


Tabla de verdad:

J	K	Q	$\overline{Q}$
1	0	1	0
0	1	0	1
0	0	$Q*$	$\overline{Q}*$
1	1	$\overline{Q}*$	$Q*$

Con $J, K = (1, 1)$:

- El valor de las salidas está ahora definido.

Latch JK

Tratemos de modificar el comportamiento para el caso cuando las entradas son $(1, 1)$:

Latch JK:

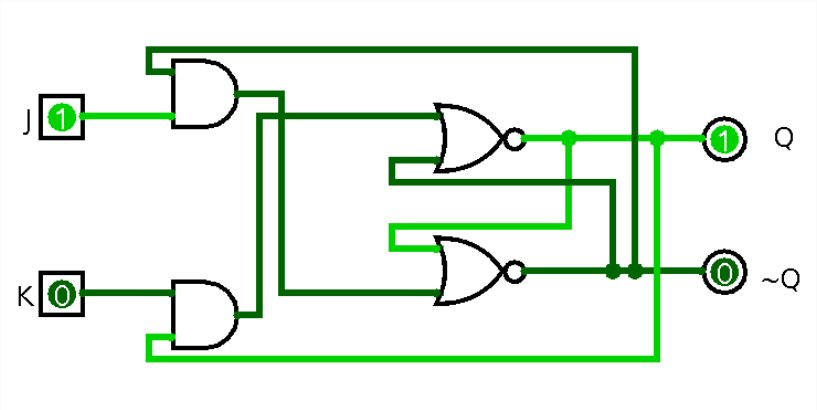


Tabla de verdad:

J	K	Q	$\overline{Q}$
1	0	1	0
0	1	0	1
0	0	$Q*$	$\overline{Q}*$
1	1	$\overline{Q}*$	$Q*$

Con $J, K = (1, 1)$:

- El valor de las salidas está ahora definido.
- El circuito oscila (estado inestable).

Latch D

- Nos permite almacenar 1 bit
- Tiene una entrada de **datos** y una de **control**

Latch D:

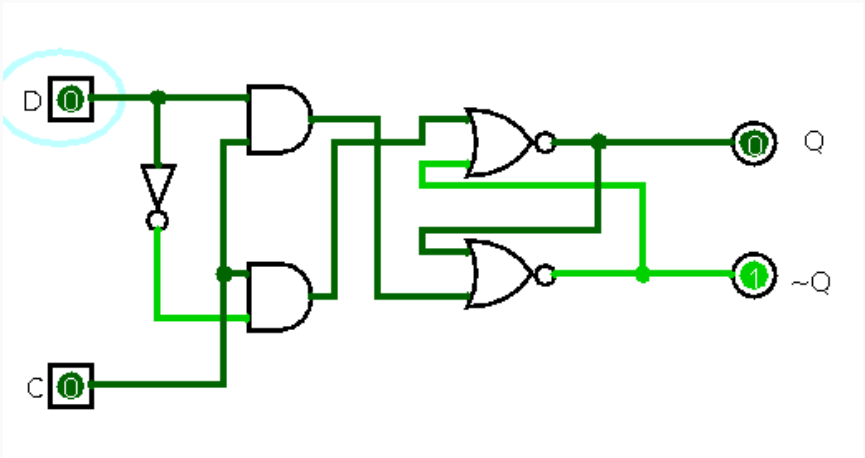


Tabla de verdad:

D	C	Q	$\overline{Q}$
1	0		
0	1		
0	0		
1	1		

Latch D

- Nos permite almacenar 1 bit
- Tiene una entrada de **datos** y una de **control**

Latch D:

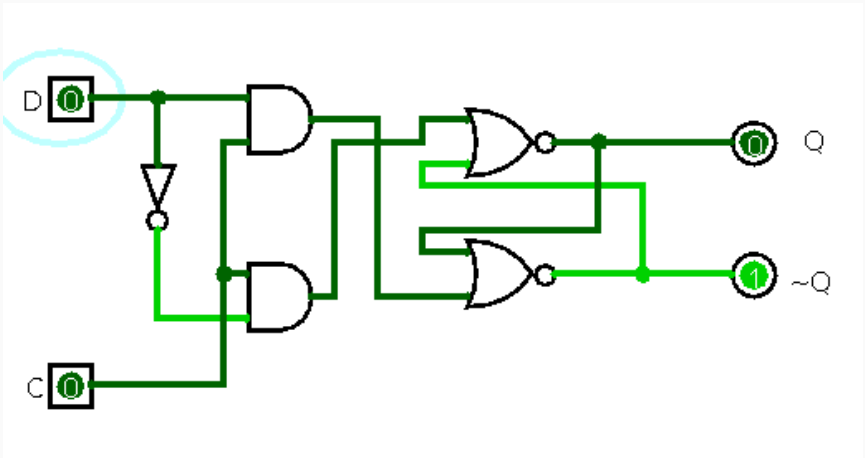


Tabla de verdad:

D	C	Q	$\overline{Q}$
1	0		$\overline{Q}*$
0	1		
0	0		
1	1		

Latch D

- Nos permite almacenar 1 bit
- Tiene una entrada de **datos** y una de **control**

Latch D:

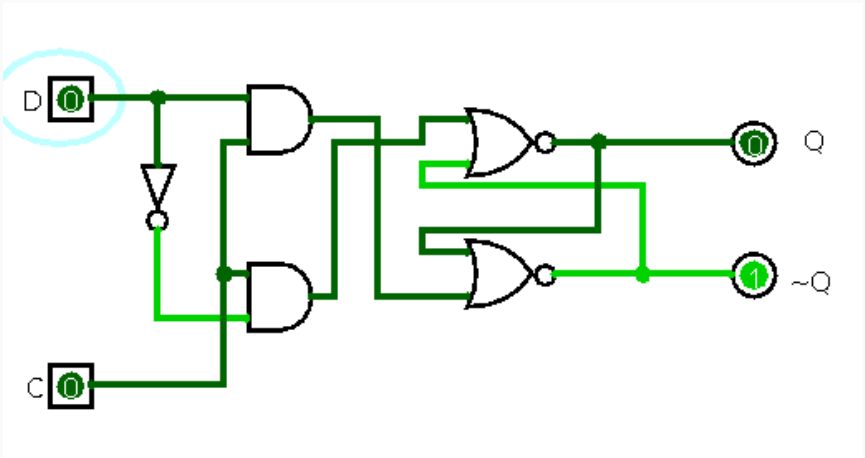


Tabla de verdad:

D	C	Q	$\overline{Q}$
1	0	$Q*$	$\overline{Q}*$
0	1		
0	0		
1	1		

Latch D

- Nos permite almacenar 1 bit
- Tiene una entrada de **datos** y una de **control**

Latch D:

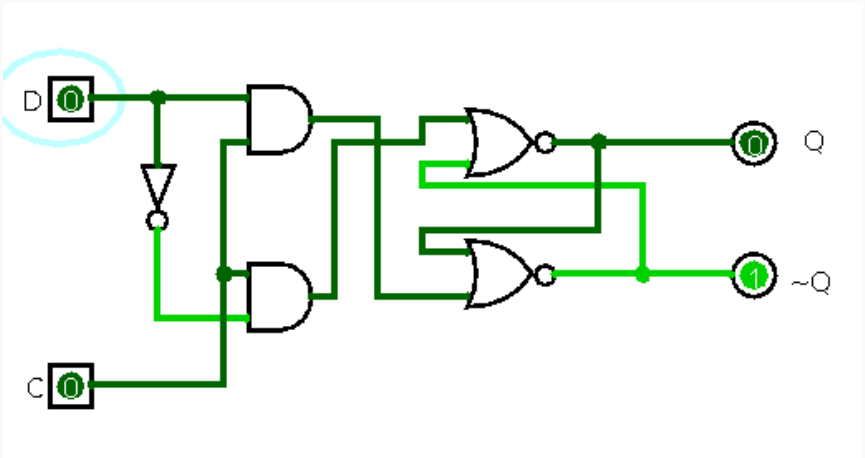


Tabla de verdad:

D	C	Q	$\overline{Q}$
1	0	$Q*$	$\overline{Q}*$
0	1	0	1
0	0		
1	1		

Latch D

- Nos permite almacenar 1 bit
- Tiene una entrada de **datos** y una de **control**

Latch D:

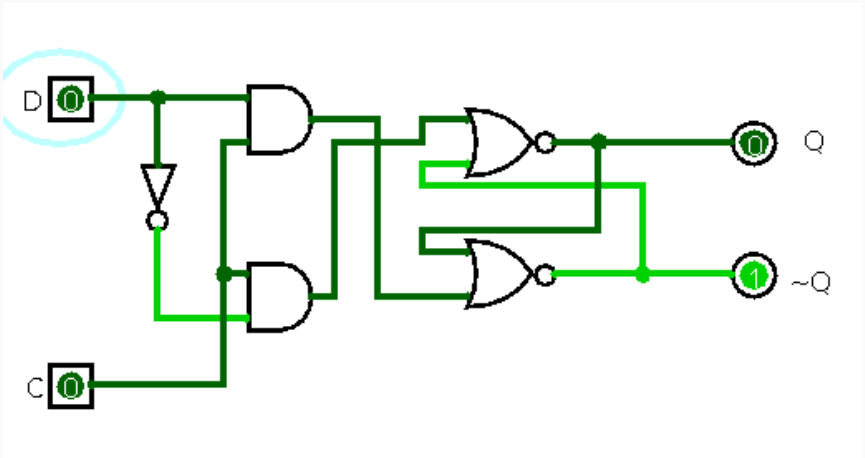


Tabla de verdad:

D	C	Q	$\overline{Q}$
1	0	$Q*$	$\overline{Q}*$
0	1	0	1
0	0	$Q*$	$\overline{Q}*$
1	1		

Latch D

- Nos permite almacenar 1 bit
- Tiene una entrada de **datos** y una de **control**

Latch D:

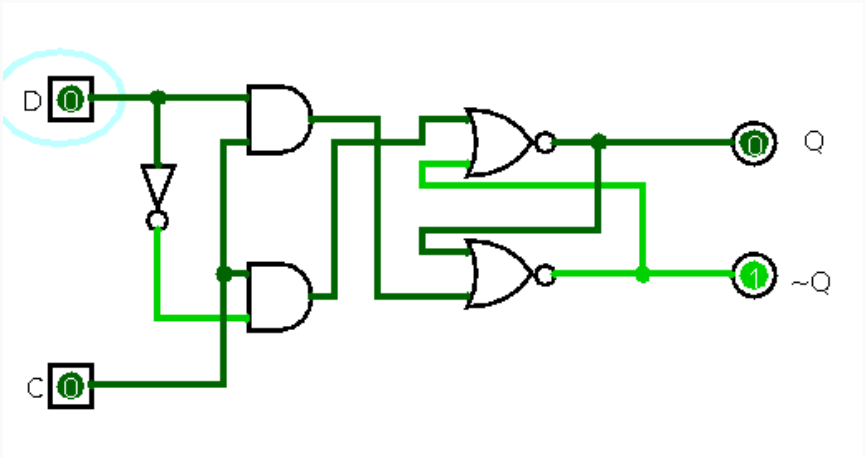


Tabla de verdad:

D	C	Q	$\overline{Q}$
1	0	$Q*$	$\overline{Q}*$
0	1	0	1
0	0	$Q*$	$\overline{Q}*$
1	1	1	0

Latch D

- Nos permite almacenar 1 bit
- Tiene una entrada de **datos** y una de **control**

Latch D:

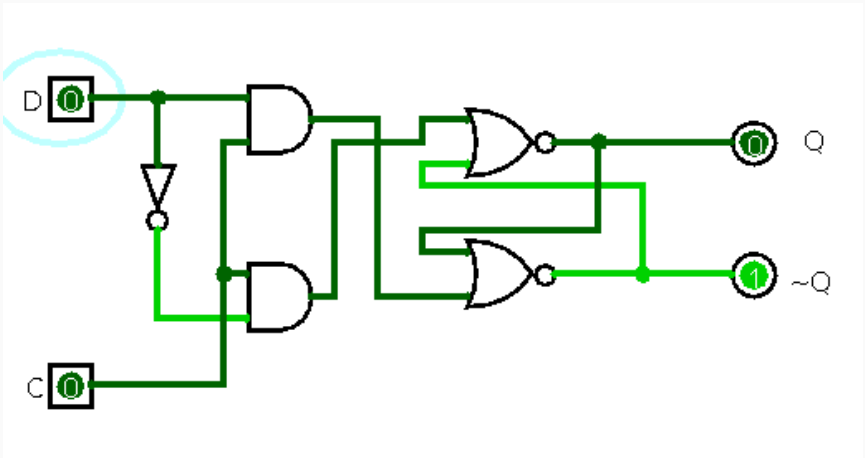


Tabla de verdad:

D	C	Q	$\overline{Q}$
1	0	$Q*$	$\overline{Q}*$
0	1	0	1
0	0	$Q*$	$\overline{Q}*$
1	1	1	0

En este caso el circuito es estable en todos los estados. Sin embargo:

Latch D

- Nos permite almacenar 1 bit
- Tiene una entrada de **datos** y una de **control**

Latch D:

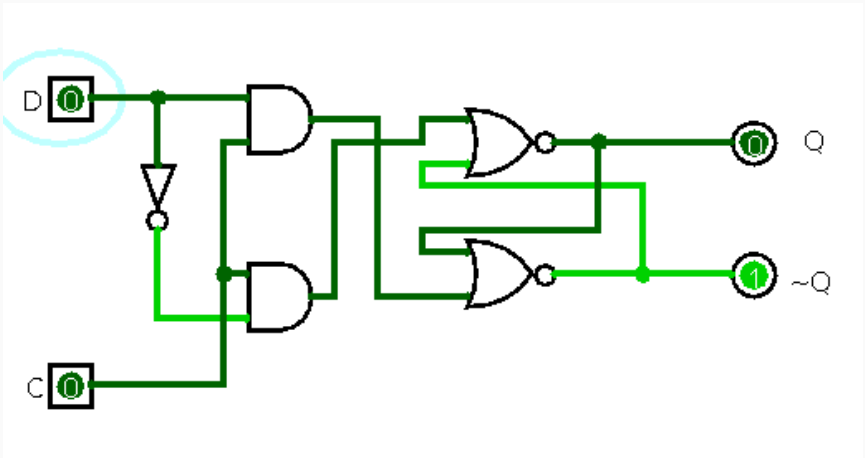


Tabla de verdad:

D	C	Q	$\overline{Q}$
1	0	$Q*$	$\overline{Q}*$
0	1	0	1
0	0	$Q*$	$\overline{Q}*$
1	1	1	0

En este caso el circuito es estable en todos los estados. Sin embargo:

- Los tiempos no se pueden predecir (dependen de C)

Latch D

- Nos permite almacenar 1 bit
- Tiene una entrada de **datos** y una de **control**

Latch D:

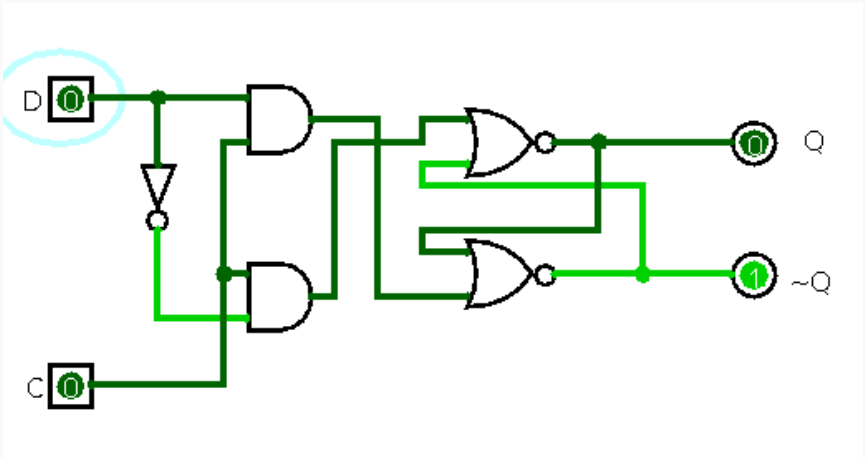


Tabla de verdad:

D	C	Q	$\overline{Q}$
1	0	$Q*$	$\overline{Q}*$
0	1	0	1
0	0	$Q*$	$\overline{Q}*$
1	1	1	0

En este caso el circuito es estable en todos los estados. Sin embargo:

- Los tiempos no se pueden predecir (dependen de C)
- Puede causar carreras si existe un lazo en el circuito externo.

Sincronizando...

Como vimos en la primer parte, nos interesa poder tener un control de los momentos de transición de estados $\Rightarrow$ **CLOCK**

Sincronizando...

Como vimos en la primer parte, nos interesa poder tener un control de los momentos de transición de estados $\Rightarrow$ **CLOCK**

Vimos también que ser reactivo al nivel de una señal no es conveniente $\Rightarrow$ **Sensibilidad al flanco**

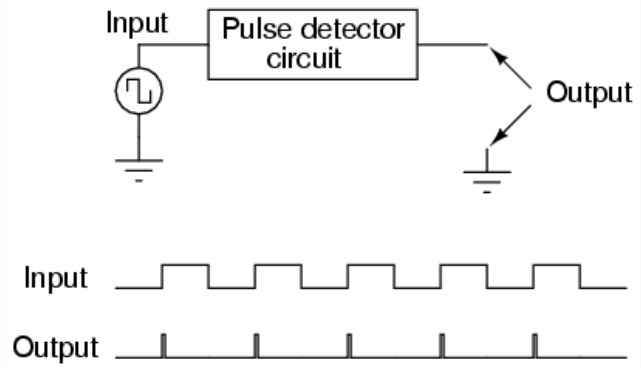
Sincronizando...

Como vimos en la primer parte, nos interesa poder tener un control de los momentos de transición de estados $\Rightarrow$ **CLOCK**

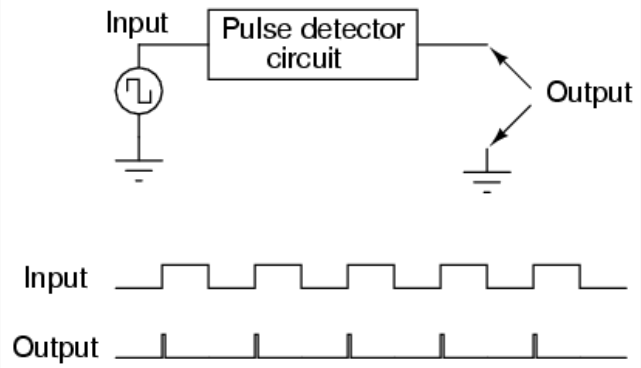
Vimos también que ser reactivo al nivel de una señal no es conveniente $\Rightarrow$ **Sensibilidad al flanco**

Necesitamos definir un circuito que sea capaz de detectar los flancos de una señal

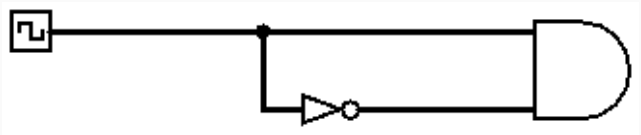
Detector de flanco

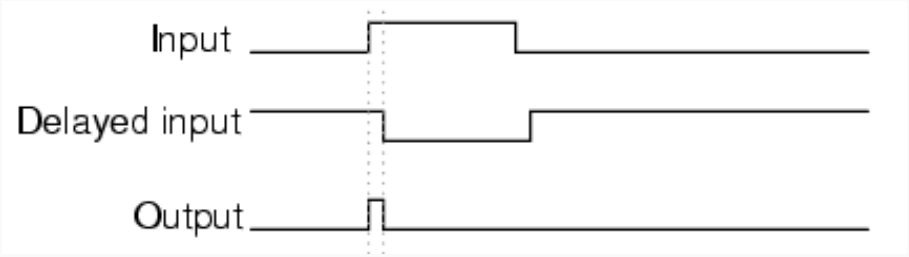


Detector de flanco



Entonces, aprovechando los tiempos de propagación:





Flip-Flop D (Delay)

Ahora nuestro latch es sólo sensible a los flancos ascendentes del clock, entonces:

Lo podemos representar:

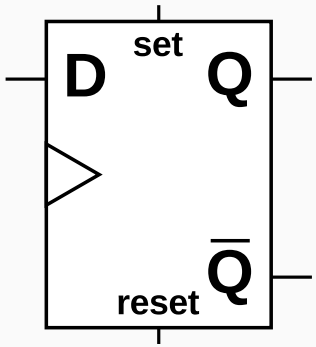


Tabla de verdad:

D	clk	Q_{T+1}	$\overline{Q_{T+1}}$
1	0		
0	$1\uparrow$		
0	0		
1	$1\uparrow$		

Flip-Flop D (Delay)

Ahora nuestro latch es sólo sensible a los flancos ascendentes del clock, entonces:

Lo podemos representar:

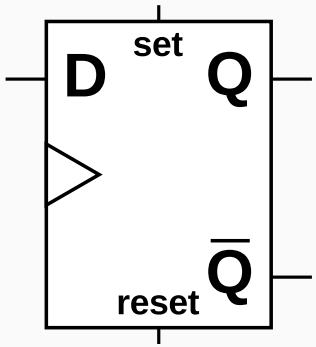


Tabla de verdad:

D	clk	Q_{T+1}	$\overline{Q_{T+1}}$
1	0		$\overline{Q_T}$
0	$1\uparrow$		
0	0		
1	$1\uparrow$		

Flip-Flop D (Delay)

Ahora nuestro latch es sólo sensible a los flancos ascendentes del clock, entonces:

Lo podemos representar:

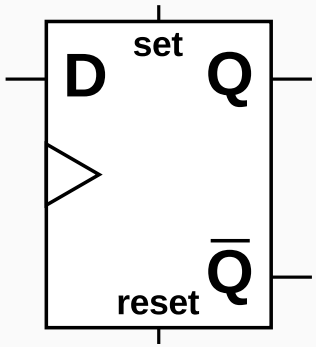


Tabla de verdad:

D	clk	Q_{T+1}	$\overline{Q_{T+1}}$
1	0	Q_T	$\overline{Q_T}$
0	$1\uparrow$		
0	0		
1	$1\uparrow$		

Flip-Flop D (Delay)

Ahora nuestro latch es sólo sensible a los flancos ascendentes del clock, entonces:

Lo podemos representar:

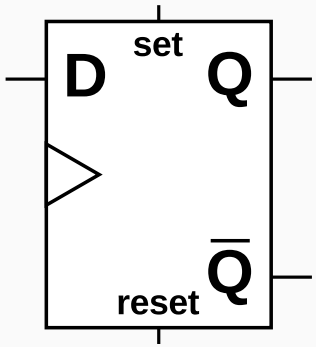


Tabla de verdad:

D	clk	Q_{T+1}	$\overline{Q_{T+1}}$
1	0	Q_T	$\overline{Q_T}$
0	$1\uparrow$	0	1
0	0		
1	$1\uparrow$		

Flip-Flop D (Delay)

Ahora nuestro latch es sólo sensible a los flancos ascendentes del clock, entonces:

Lo podemos representar:

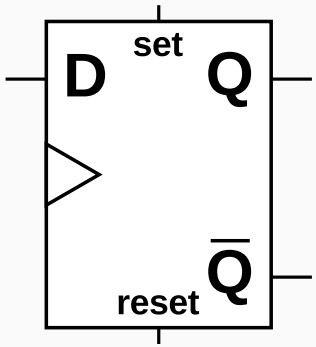


Tabla de verdad:

D	clk	Q_{T+1}	$\overline{Q_{T+1}}$
1	0	Q_T	$\overline{Q_T}$
0	$1\uparrow$	0	1
0	0	Q_T	$\overline{Q_T}$
1	$1\uparrow$		

Flip-Flop D (Delay)

Ahora nuestro latch es sólo sensible a los flancos ascendentes del clock, entonces:

Lo podemos representar:

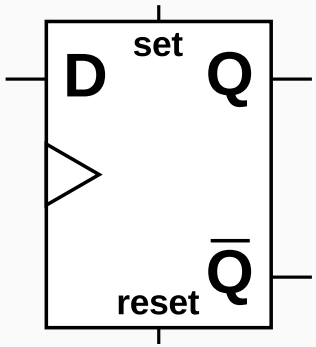


Tabla de verdad:

D	clk	Q_{T+1}	$\overline{Q_{T+1}}$
1	0	Q_T	$\overline{Q_T}$
0	$1\uparrow$	0	1
0	0	Q_T	$\overline{Q_T}$
1	$1\uparrow$	1	0

Flip-Flop D (Delay)

Ahora nuestro latch es sólo sensible a los flancos ascendentes del clock, entonces:

Lo podemos representar:

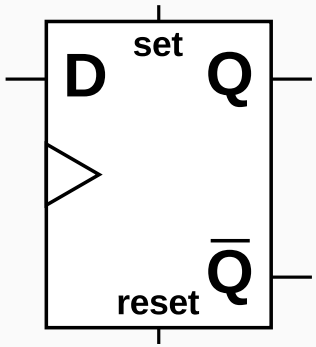


Tabla de verdad:

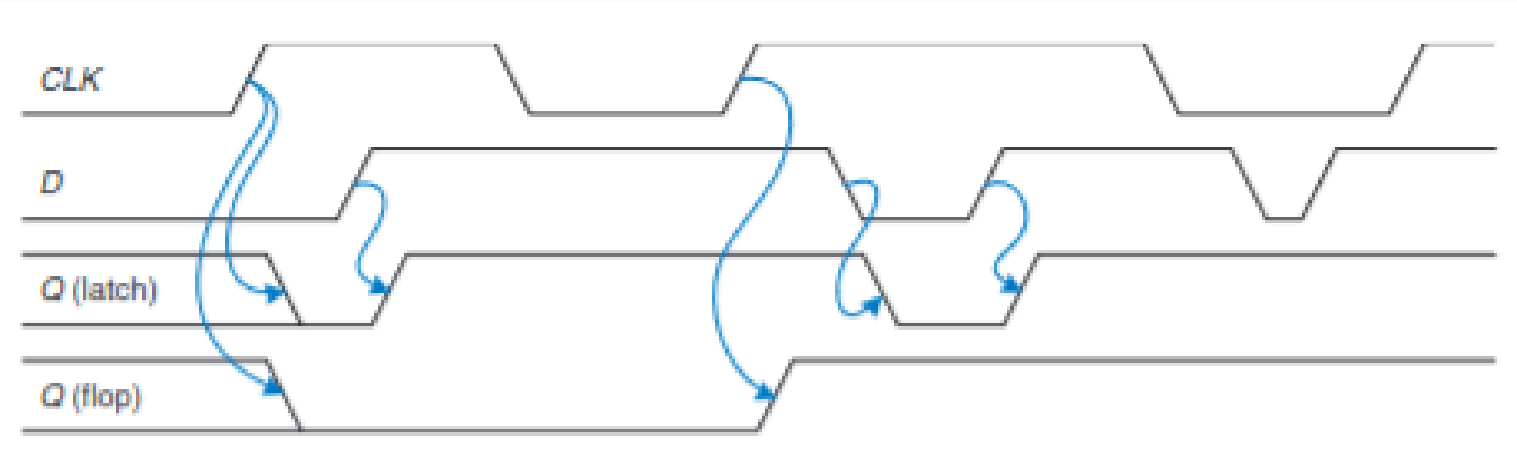
D	clk	Q_{T+1}	$\overline{Q_{T+1}}$
1	0	Q_T	$\overline{Q_T}$
0	1↑	0	1
0	0	Q_T	$\overline{Q_T}$
1	1↑	1	0

Siendo $T = n.T_{clock}$ y $T + 1 = (n + 1)T_{clock}$, donde:

- T_clock es el período del clock (tiempo que dura un ciclo)
- n es una cierta cantidad de pulsos de clock

Latch vs Flip-Flop

Ahora podemos entender bien las diferencias:



Flip-Flop J-K

Volviendo al latch J-K, ahora con detección de flanco podemos obtener un comportamiento más *adecuado*:

Ahora lo podemos representar como:

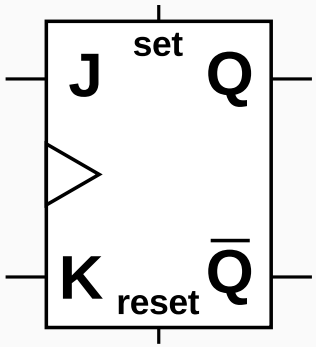


Tabla de verdad:

J	K	clk	Q_{T+1}	$\overline{Q}_{T+1}$
1	0	$1\uparrow$		
0	1	$1\uparrow$		
0	0	$1\uparrow$		
1	1	$1\uparrow$		
x	x	0		

Flip-Flop J-K

Volviendo al latch J-K, ahora con detección de flanco podemos obtener un comportamiento más *adecuado*:

Ahora lo podemos representar como:

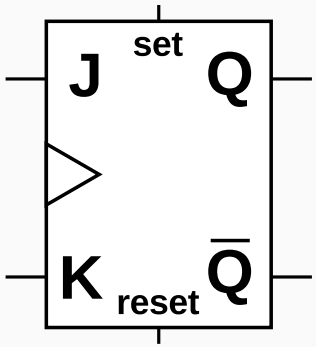


Tabla de verdad:

J	K	clk	Q_{T+1}	$\overline{Q}_{T+1}$
1	0	$1\uparrow$		0
0	1	$1\uparrow$		
0	0	$1\uparrow$		
1	1	$1\uparrow$		
x	x	0		

Flip-Flop J-K

Volviendo al latch J-K, ahora con detección de flanco podemos obtener un comportamiento más *adecuado*:

Ahora lo podemos representar como:

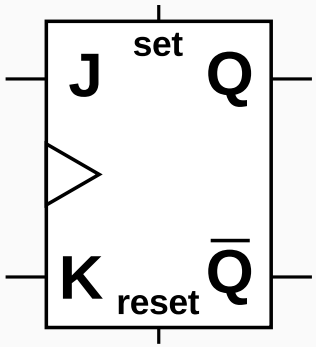


Tabla de verdad:

J	K	clk	Q_{T+1}	$\overline{Q}_{T+1}$
1	0	$1\uparrow$	1	0
0	1	$1\uparrow$		
0	0	$1\uparrow$		
1	1	$1\uparrow$		
x	x	0		

Flip-Flop J-K

Volviendo al latch J-K, ahora con detección de flanco podemos obtener un comportamiento más *adecuado*:

Ahora lo podemos representar como:

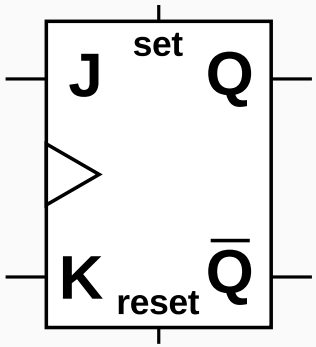


Tabla de verdad:

J	K	clk	Q_{T+1}	$\overline{Q}_{T+1}$
1	0	$1\uparrow$	1	0
0	1	$1\uparrow$	0	1
0	0	$1\uparrow$		
1	1	$1\uparrow$		
x	x	0		

Flip-Flop J-K

Volviendo al latch J-K, ahora con detección de flanco podemos obtener un comportamiento más *adecuado*:

Ahora lo podemos representar como:

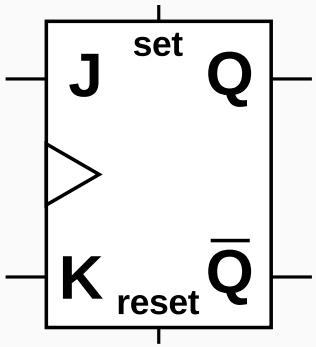


Tabla de verdad:

J	K	clk	Q_{T+1}	$\overline{Q}_{T+1}$
1	0	$1\uparrow$	1	0
0	1	$1\uparrow$	0	1
0	0	$1\uparrow$	Q_T	$\overline{Q}_T$
1	1	$1\uparrow$		
x	x	0		

Flip-Flop J-K

Volviendo al latch J-K, ahora con detección de flanco podemos obtener un comportamiento más *adecuado*:

Ahora lo podemos representar como:

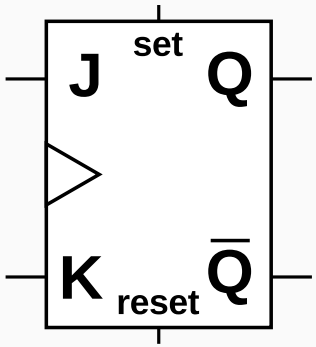


Tabla de verdad:

J	K	clk	Q_{T+1}	$\overline{Q}_{T+1}$
1	0	1↑	1	0
0	1	1↑	0	1
0	0	1↑	Q_T	$\overline{Q}_T$
1	1	1↑	$\overline{Q}_T$	Q_T
x	x	0		

Flip-Flop J-K

Volviendo al latch J-K, ahora con detección de flanco podemos obtener un comportamiento más *adecuado*:

Ahora lo podemos representar como:

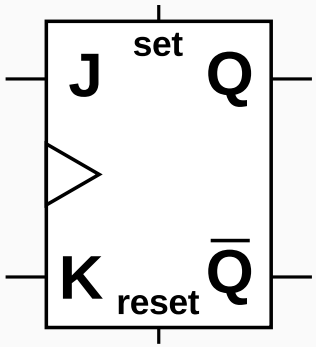


Tabla de verdad:

J	K	clk	Q_{T+1}	$\overline{Q}_{T+1}$
1	0	$1\uparrow$	1	0
0	1	$1\uparrow$	0	1
0	0	$1\uparrow$	Q_T	$\overline{Q}_T$
1	1	$1\uparrow$	$\overline{Q}_T$	Q_T
x	x	0	$\overline{Q}_T$	Q_T

Flip-Flop J-K

Volviendo al latch J-K, ahora con detección de flanco podemos obtener un comportamiento más *adecuado*:

Ahora lo podemos representar como:

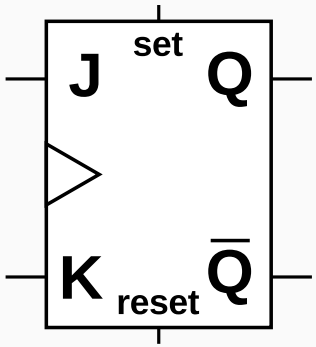


Tabla de verdad:

J	K	clk	Q_{T+1}	$\overline{Q}_{T+1}$
1	0	$1\uparrow$	1	0
0	1	$1\uparrow$	0	1
0	0	$1\uparrow$	Q_T	$\overline{Q}_T$
1	1	$1\uparrow$	$\overline{Q}_T$	Q_T
x	x	0	$\overline{Q}_T$	Q_T

Ahora en el caso crítico donde $J, K = (1, 1)$ la salida tiene un estado y un tiempo de cambio bien definido:

Se niega el valor anterior cada 1 colck

Registros y memorias

Registros

Ya vimos como un FF D puede almacenar un bit... ¡pero sólo durante un clock!

Registros

Ya vimos como un FF D puede almacenar un bit... ¡pero sólo durante un clock!

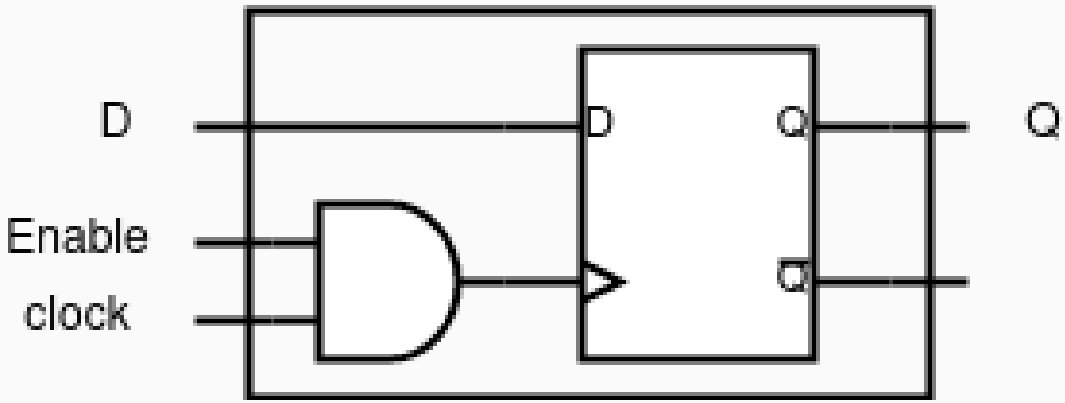
- Debemos poder elegir con una entrada adicional de control por cuanto tiempo queremos almacenar $\Rightarrow$ **enable**.

Registros

Ya vimos como un FF D puede almacenar un bit... ¡pero sólo durante un clock!

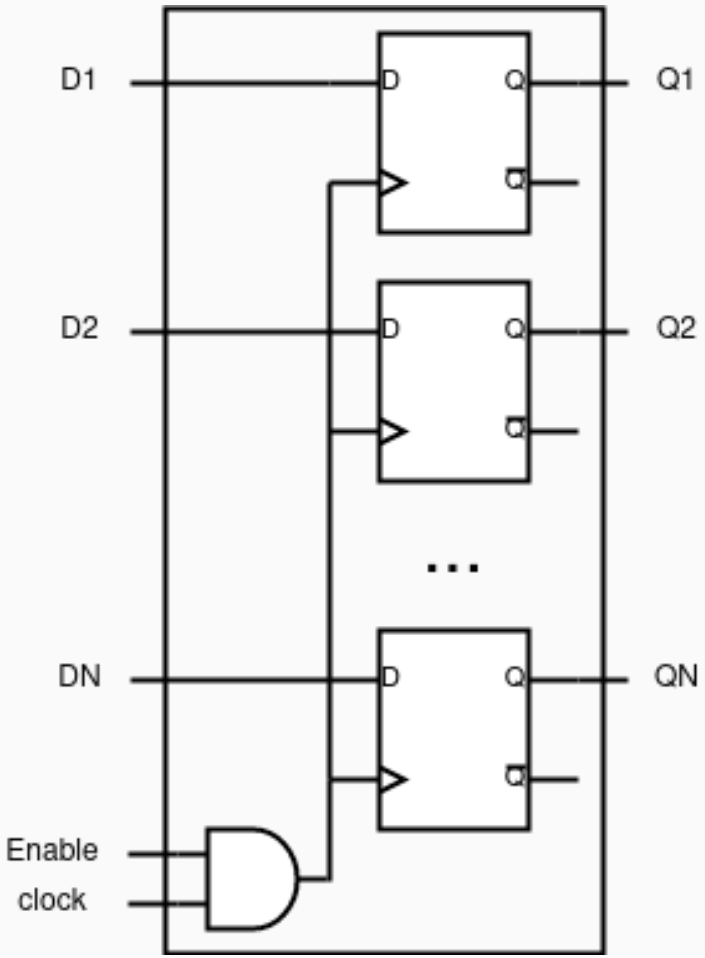
- Debemos poder elegir con una entrada adicional de control por cuanto tiempo queremos almacenar $\Rightarrow$ **enable**.

¡Sencillo!:



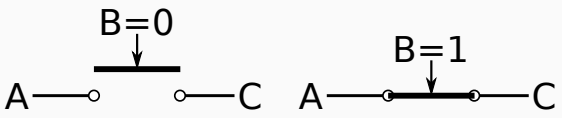
Registro de N-bits

Podemos componer la solución anterior para poder almacenar N bits:



Componentes de Tres Estados

Noción Eléctrica



Símbolo

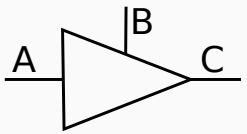
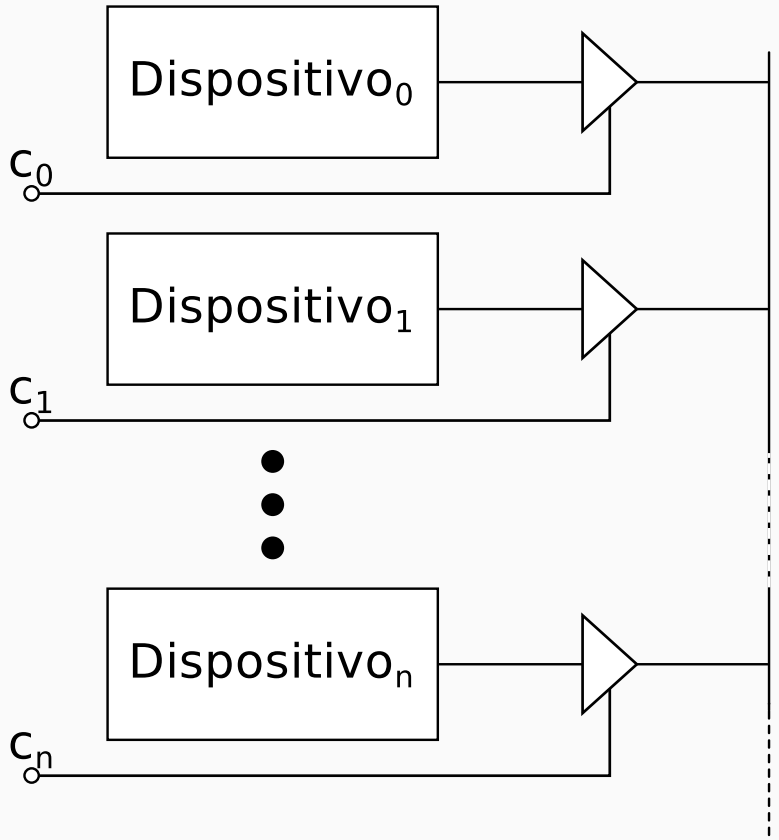


Tabla de Verdad

A	B	C
0	1	0
1	1	1
_	0	Hi-Z

Hi-Z significa "alta impedancia", es decir, que tiene una resistencia alta al pasaje de corriente. Como consecuencia de esto, podemos considerar al pin C como desconectado del circuito.

Componentes de Tres Estados



IMPORTANTE: Sólo deben ser usados a la salida de componentes para permitirles conectarse a un medio compartido (bus).

Ejercicio 0

a) Diseñar un registro de 3 *bits*. El mismo debe contar con 3 entradas $e_0, \ldots, e_2$ para ingresar el dato a almacenar, 3 salidas $s_0, \ldots, s_2$ para ver el dato almacenado y las señales de control CLK, RESET y WRITEENABLE.

Ejercicio 0

a) Diseñar un registro de 3 *bits*. El mismo debe contar con 3 entradas $e_0, \ldots, e_2$ para ingresar el dato a almacenar, 3 salidas $s_0, \ldots, s_2$ para ver el dato almacenado y las señales de control CLK, RESET y WRITEENABLE.

b) Modificar el diseño anterior agregándole componentes de 3 estados para que sólo cuando se active la señal de control ENABLEOUT muestre el dato almacenado.

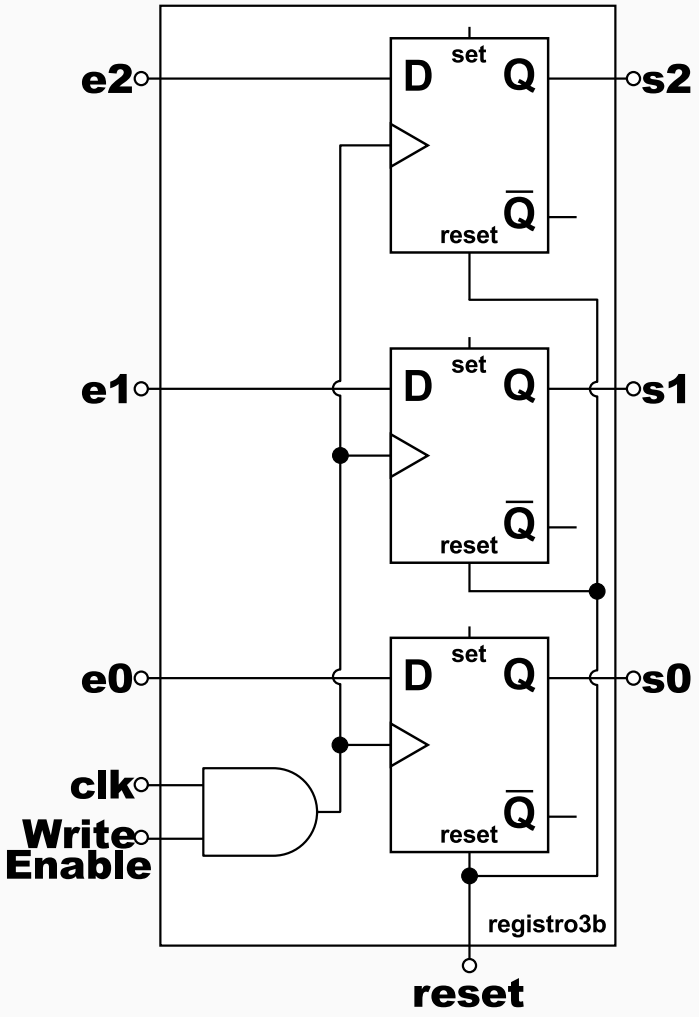
Ejercicio 0

a) Diseñar un registro de 3 *bits*. El mismo debe contar con 3 entradas $e_0, \ldots, e_2$ para ingresar el dato a almacenar, 3 salidas $s_0, \ldots, s_2$ para ver el dato almacenado y las señales de control CLK, RESET y WRITEENABLE.

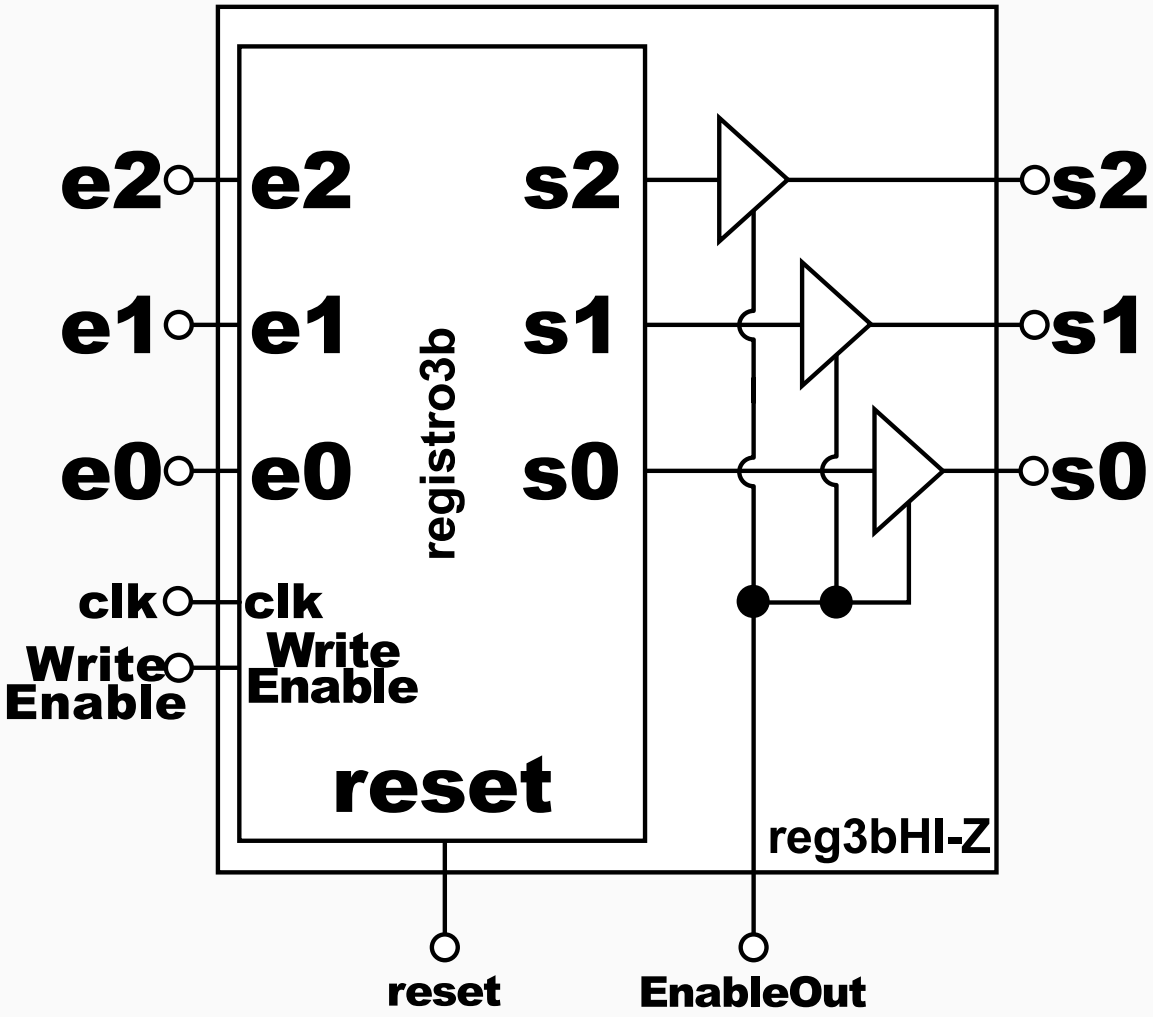
b) Modificar el diseño anterior agregándole componentes de 3 estados para que sólo cuando se active la señal de control ENABLEOUT muestre el dato almacenado.

c) Modificar nuevamente el diseño para que e_i y s_i estén conectadas entre sí al mismo tiempo teniendo en lugar de 3 entradas y 3 salidas, 3 entrada-salidas

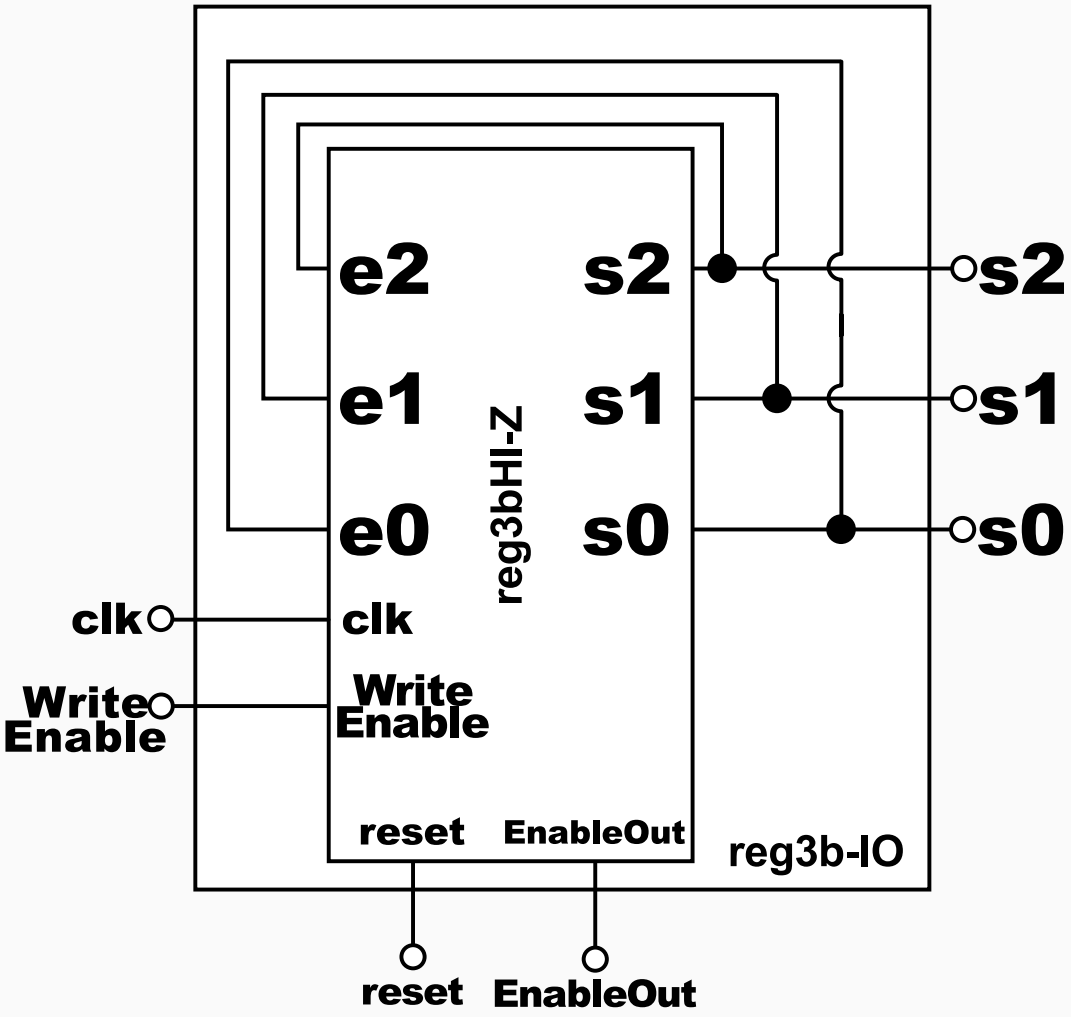
Solución - Ejercicio 0.a



Solución - Ejercicio 0.b



Solución - Ejercicio 0.c



Ejercicio 1

a) Realizar el esquema de interconexión de n registros como el diseñado

Ejercicio 1

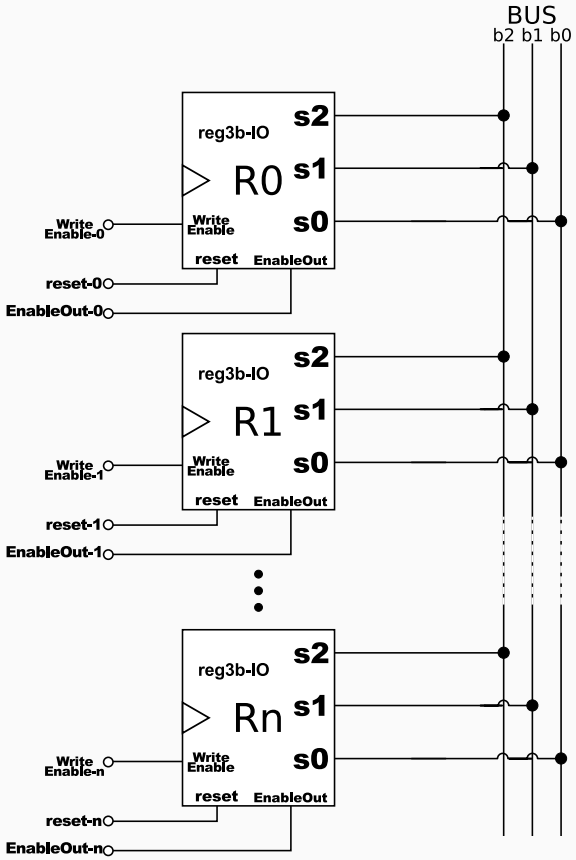
a) Realizar el esquema de interconexión de n registros como el diseñado

b) Dar una secuencia de valores de las señales de control para que se copie el dato del R1 al R0

Ejercicio 1

a) Realizar el esquema de interconexión de n registros como el diseñado

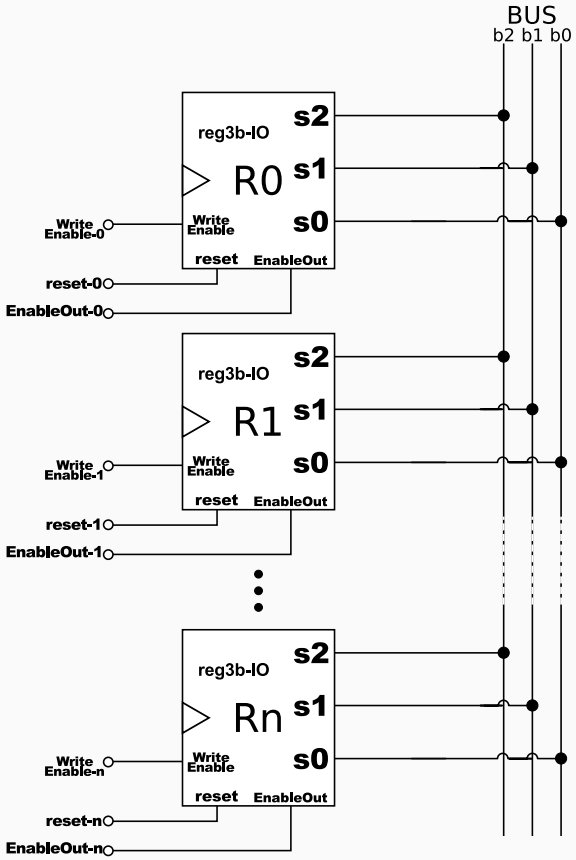
b) Dar una secuencia de valores de las señales de control para que se copie el dato del R1 al R0



Ejercicio 1

a) Realizar el esquema de interconexión de n registros como el diseñado

b) Dar una secuencia de valores de las señales de control para que se copie el dato del R1 al R0



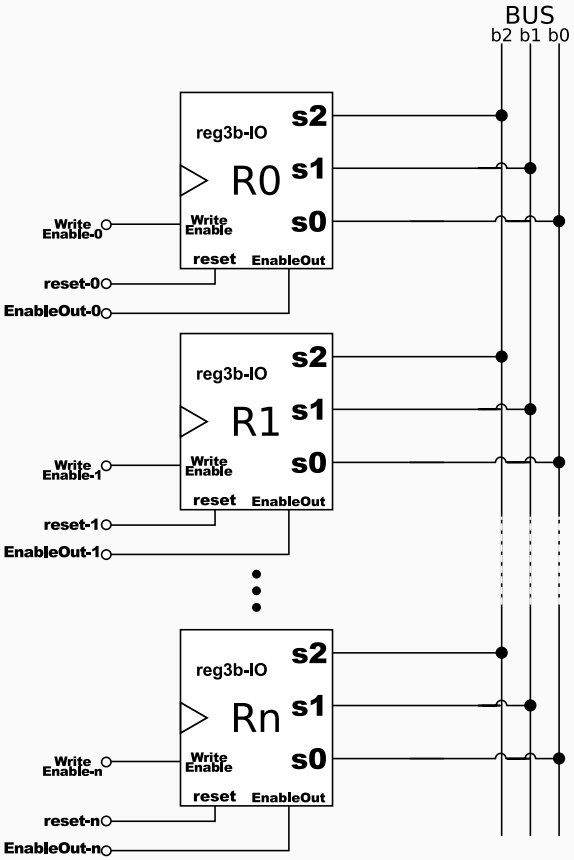
Señales de control:

R0	R1	...	Rn
WriteEnable-0	WriteEnable-1	...	WriteEnable-n
reset-0	reset-1	...	reset-n
EnableOut-0	EnableOut-1	...	EnableOut-n

Ejercicio 1

a) Realizar el esquema de interconexión de n registros como el diseñado

b) Dar una secuencia de valores de las señales de control para que se copie el dato del R1 al R0



Señales de control:

R0	R1	...	Rn
WriteEnable-0	WriteEnable-1	...	WriteEnable-n
reset-0	reset-1	...	reset-n
EnableOut-0	EnableOut-1	...	EnableOut-n

Inician todas las señales en 0. Luego se sigue la siguiente secuencia:

- EnableOut-1 $\leftarrow$ 1
- WriteEnable-0 $\leftarrow$ 1
- ...clk....
- WriteEnable-0 $\leftarrow$ 0
- EnableOut-1 $\leftarrow$ 0

Memorias (intro)

Conceptualmente podemos pensar una memoria como M posiciones de almacenamiento de N bits cada una.

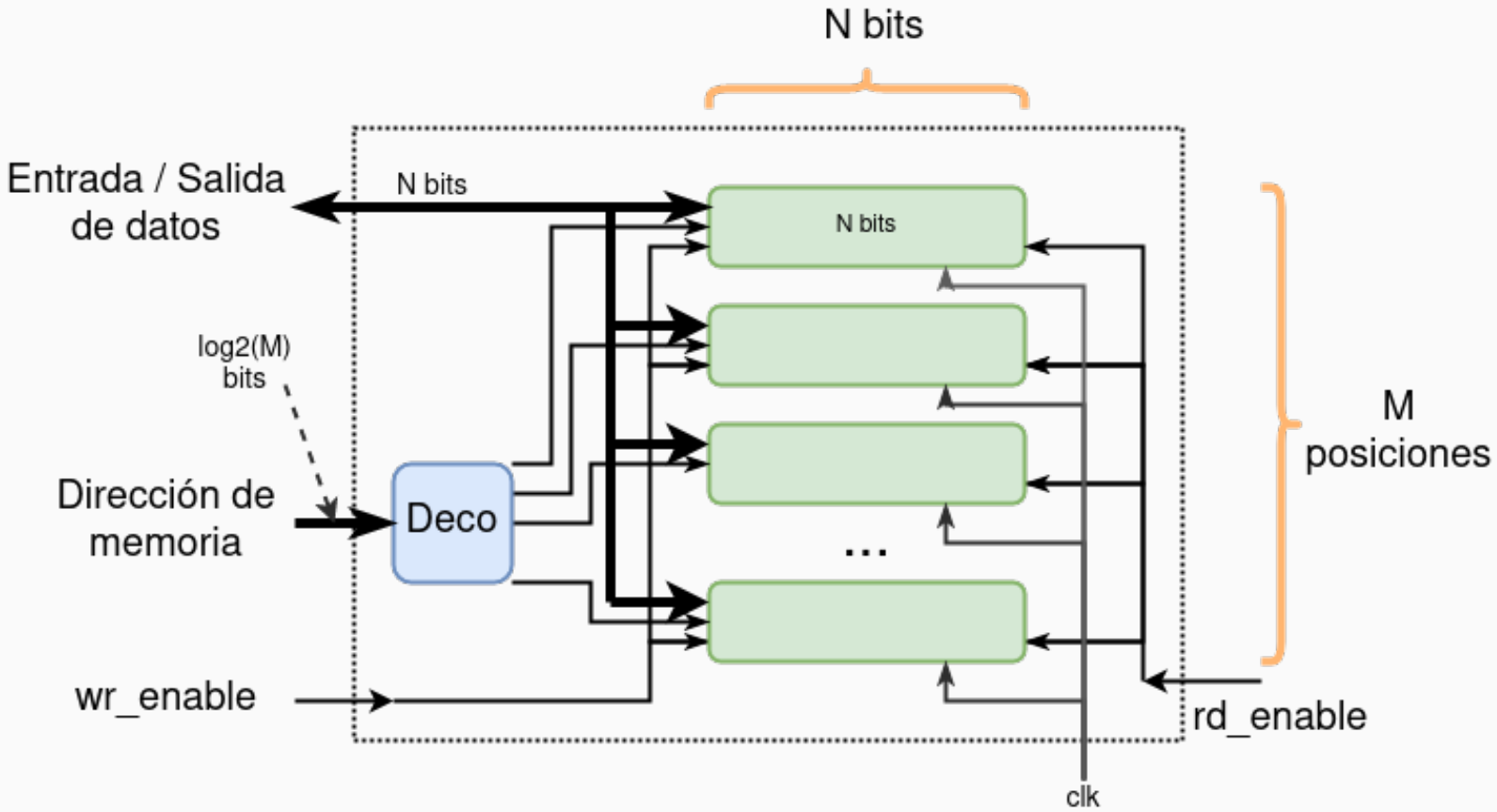
Memorias (intro)

Conceptualmente podemos pensar una memoria como M posiciones de almacenamiento de N bits cada una.

Debemos poder seleccionar a cuál queremos acceder

Memorias (intro)

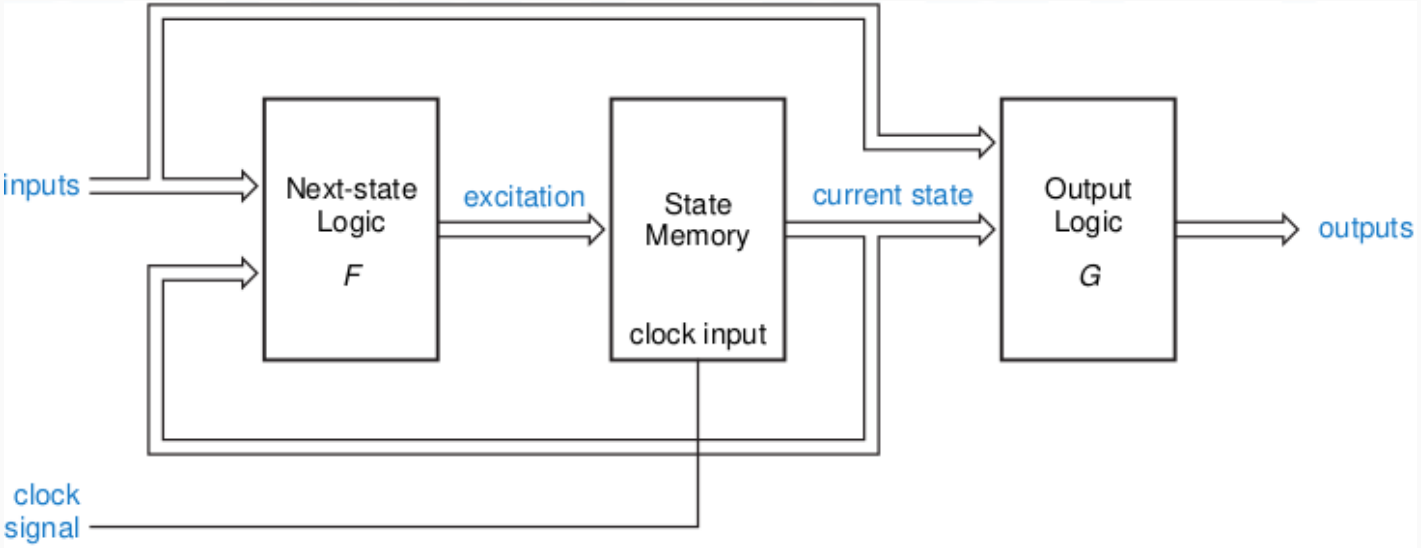
Conceptualmente podemos pensar una memoria como M posiciones de almacenamiento de N bits cada una.
Debemos poder seleccionar a cuál queremos acceder



Máquinas de estado

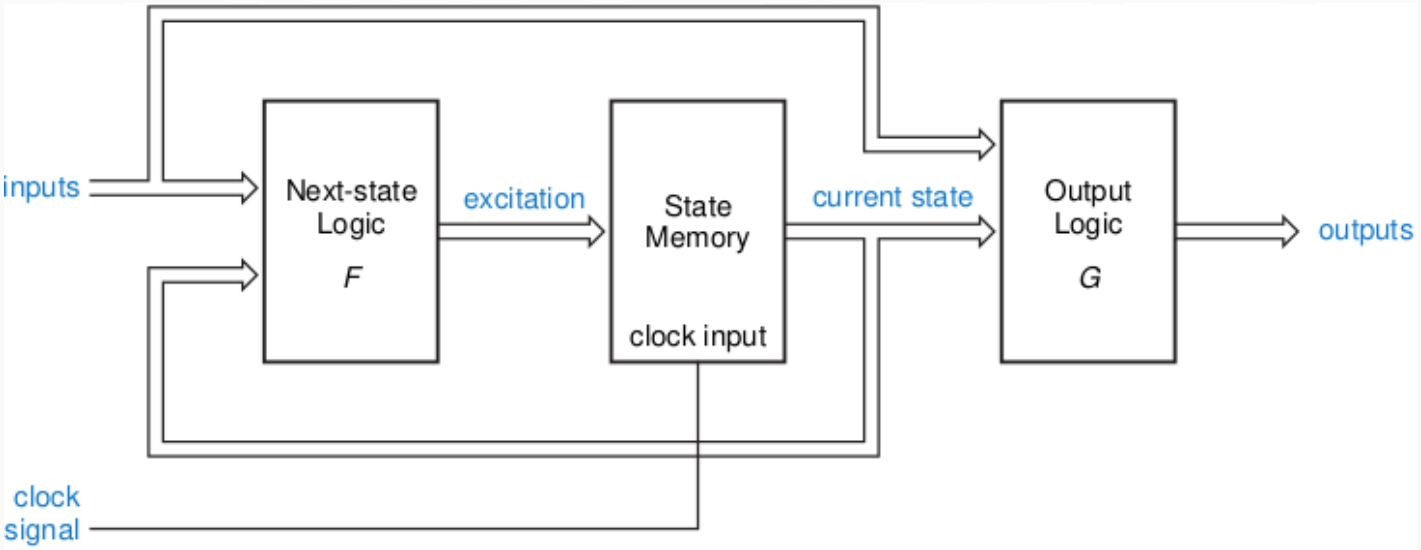
Modelo general de un circuito secuencial

Conceptualmente podemos pensar a un secuencial como:



Modelo general de un circuito secuencial

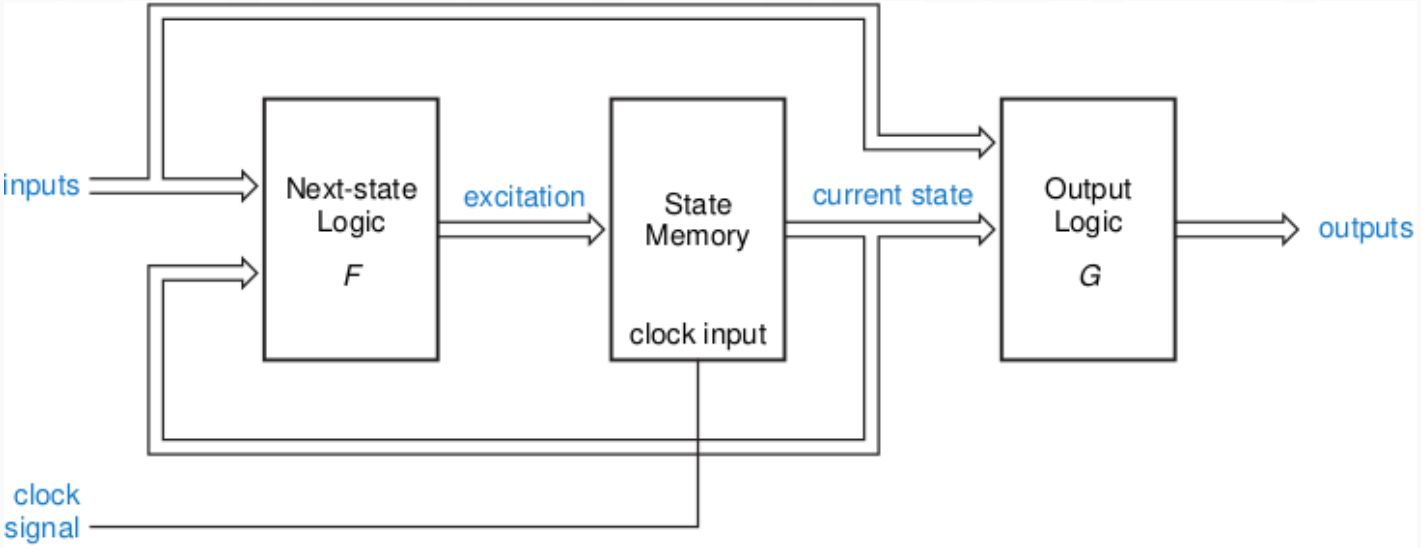
Conceptualmente podemos pensar a un secuencial como:



Compuesto por tres bloques principales:

Modelo general de un circuito secuencial

Conceptualmente podemos pensar a un secuencial como:

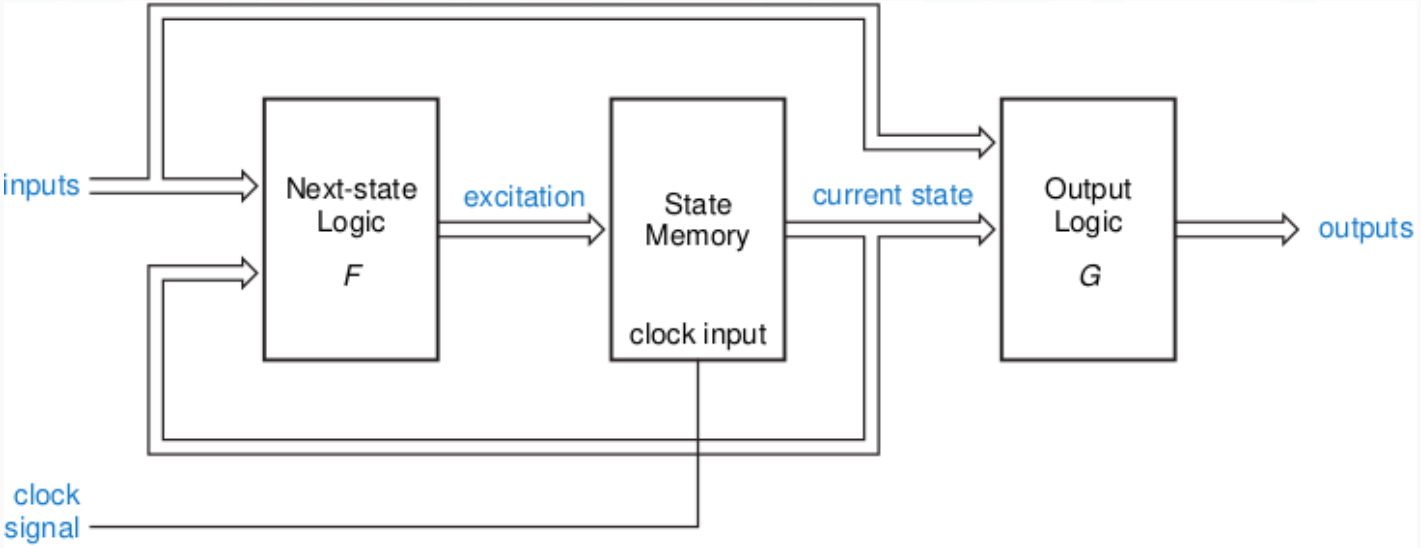


Compuesto por tres bloques principales:

- **Lógica de próximo estado**: $f(estado_{actual}, entradas)$

Modelo general de un circuito secuencial

Conceptualmente podemos pensar a un secuencial como:

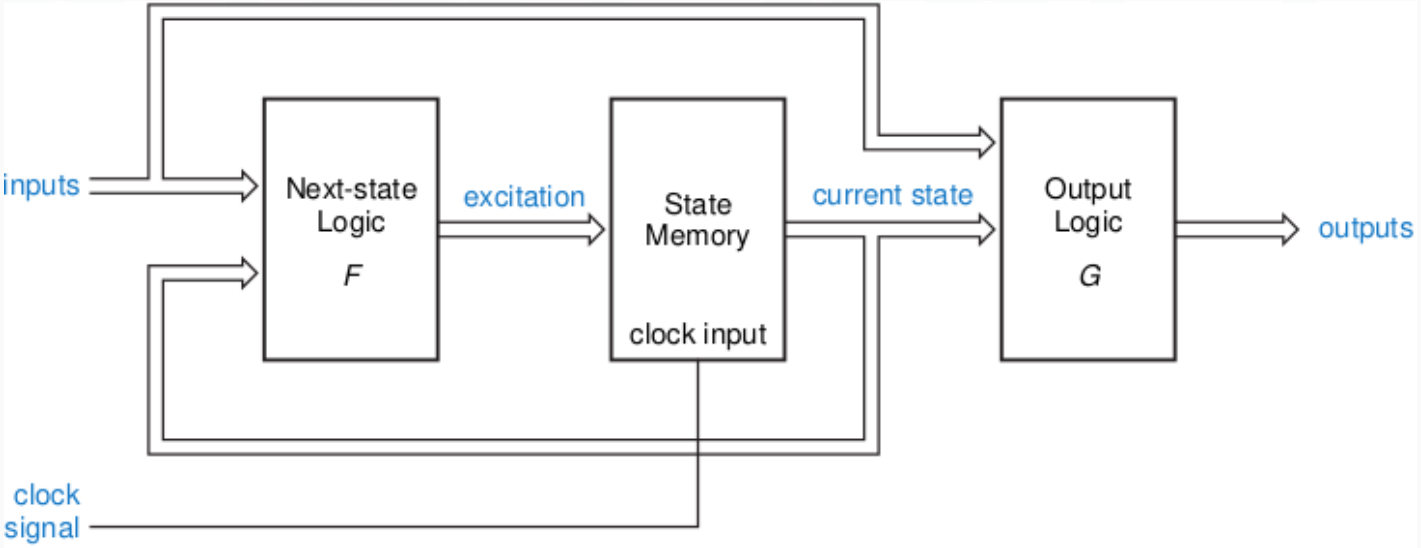


Compuesto por tres bloques principales:

- **Lógica de próximo estado**: $f(estado_{actual}, entradas)$
- **Registro (o memoria) de estado**: $f(estado_{próximo}, clk)$

Modelo general de un circuito secuencial

Conceptualmente podemos pensar a un secuencial como:



Compuesto por tres bloques principales:

- **Lógica de próximo estado**: $f(estado_{actual}, entradas)$
- **Registro (o memoria) de estado**: $f(estado_{próximo}, clk)$
- **Lógica de salida**: $f(estado_{actual}, entradas)$

Máquinas de estado

- Los circuitos secuenciales pueden ser pensados formalmente como una *Maquina de Estados Finitos* o *FSM*

Máquinas de estado

- Los circuitos secuenciales pueden ser pensados formalmente como una *Maquina de Estados Finitos* o *FSM*
- Las FSM son el siguiente nivel en cuanto a capacidad de computo luego de la lógica combinacional.

Máquinas de estado

- Los circuitos secuenciales pueden ser pensados formalmente como una *Maquina de Estados Finitos* o *FSM*
- Las FSM son el siguiente nivel en cuanto a capacidad de computo luego de la lógica combinacional.

Máquinas de estado

- Los circuitos secuenciales pueden ser pensados formalmente como una *Maquina de Estados Finitos* o *FSM*
- Las FSM son el siguiente nivel en cuanto a capacidad de computo luego de la lógica combinacional.

Una máquina de estados queda definida por:

Máquinas de estado

- Los circuitos secuenciales pueden ser pensados formalmente como una *Maquina de Estados Finitos* o *FSM*
- Las FSM son el siguiente nivel en cuanto a capacidad de computo luego de la lógica combinacional.

Una máquina de estados queda definida por:

- Una lista de estados

Máquinas de estado

- Los circuitos secuenciales pueden ser pensados formalmente como una *Maquina de Estados Finitos* o *FSM*
- Las FSM son el siguiente nivel en cuanto a capacidad de computo luego de la lógica combinacional.

Una máquina de estados queda definida por:

- Una lista de estados
- Un estado inicial

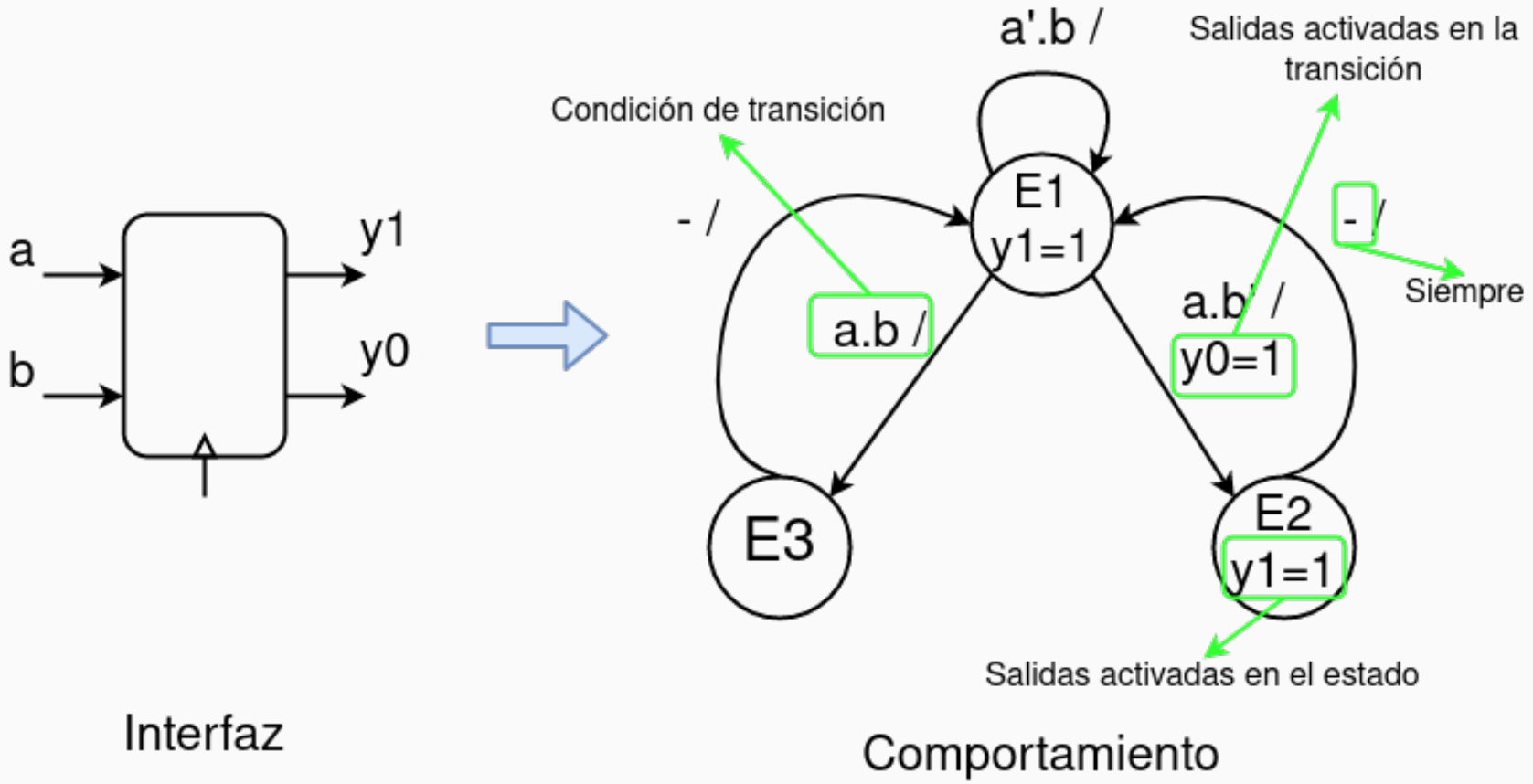
Máquinas de estado

- Los circuitos secuenciales pueden ser pensados formalmente como una *Maquina de Estados Finitos* o *FSM*
- Las FSM son el siguiente nivel en cuanto a capacidad de computo luego de la lógica combinacional.

Una máquina de estados queda definida por:

- Una lista de estados
- Un estado inicial
- Una lista de funciones disparan las transiciones en función de las entradas

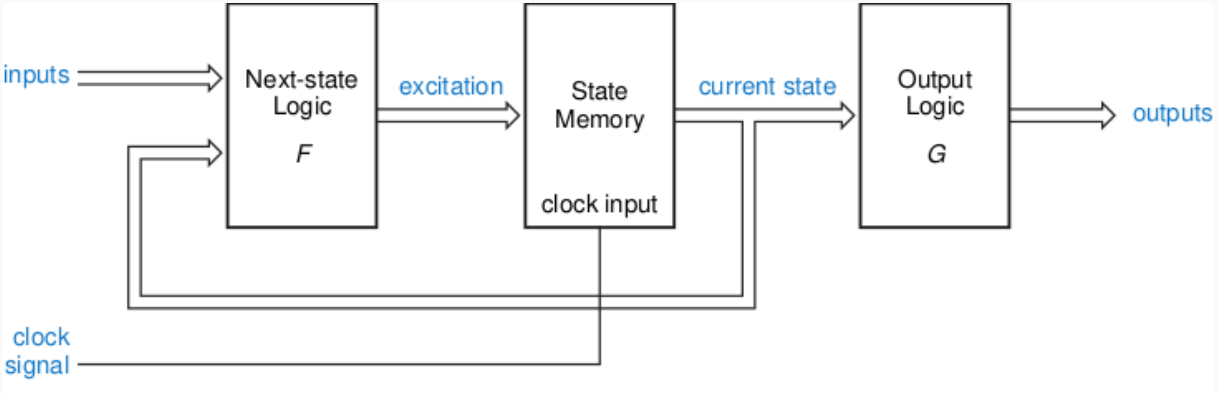
Diagramas de estado



Todas las salidas en '0' salvo que se explicite lo contrario

FSM - Moore

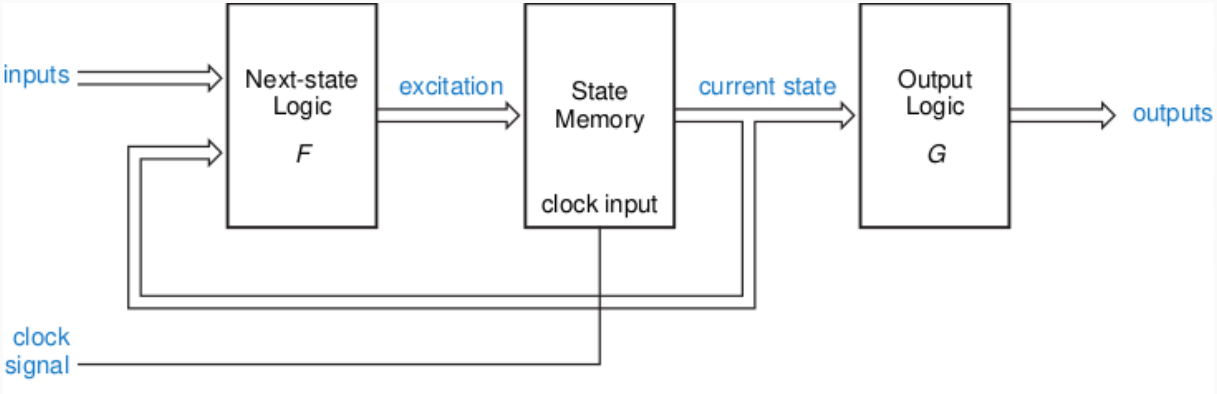
Si la salida depende sólo del estado actual la llameremos como *FSM de Moore*:



Características:

FSM - Moore

Si la salida depende sólo del estado actual la llameremos como *FSM de Moore*:

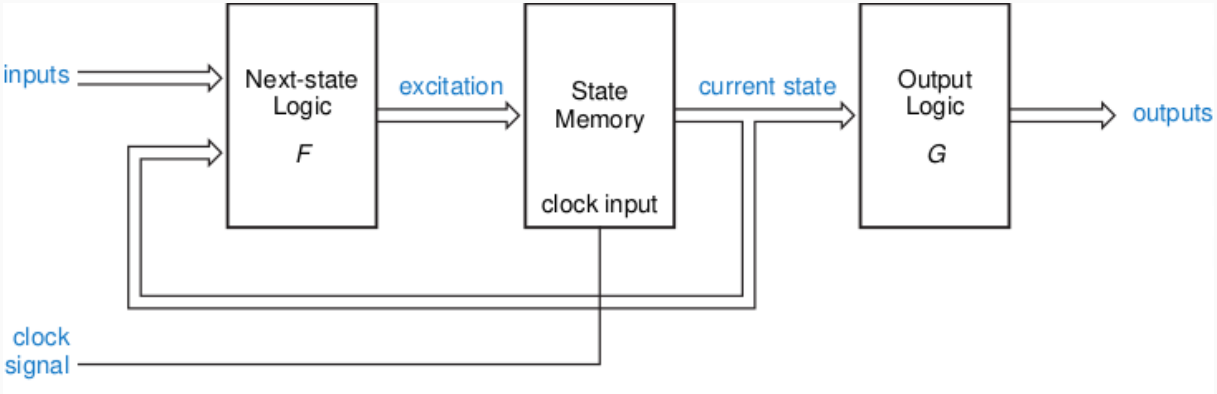


Características:

- La salida siempre cambia un clock después que se dispara la condición de transición

FSM - Moore

Si la salida depende sólo del estado actual la llameremos como *FSM de Moore*:

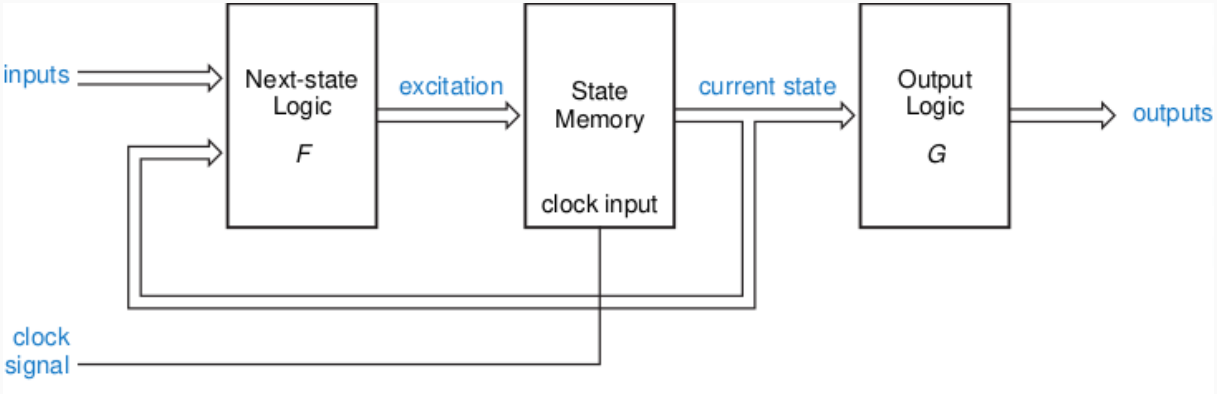


Características:

- La salida siempre cambia un clock después que se dispara la condición de transición
- No produce *glitches* a la salida

FSM - Moore

Si la salida depende sólo del estado actual la llameremos como *FSM de Moore*:

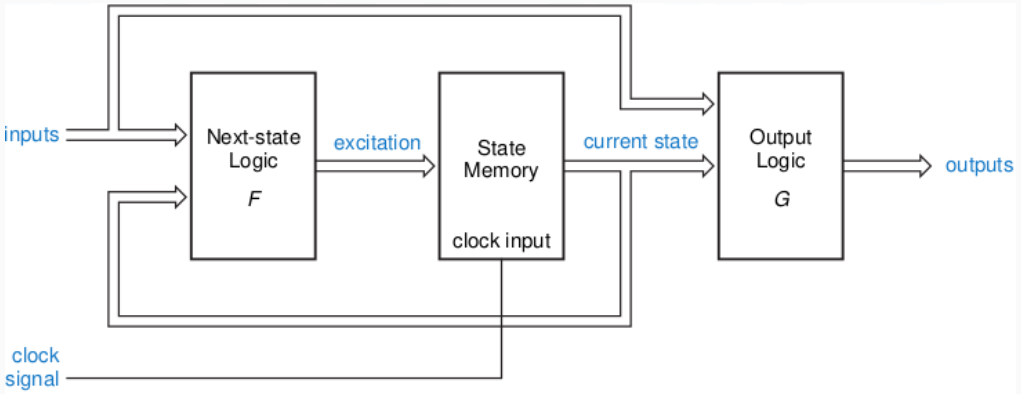


Características:

- La salida siempre cambia un clock después que se dispara la condición de transición
- No produce *glitches* a la salida
- La cantidad de estados para reproducir cierto comportamiento puede ser más grande que con otro tipo de FSM.

FSM - Mealy

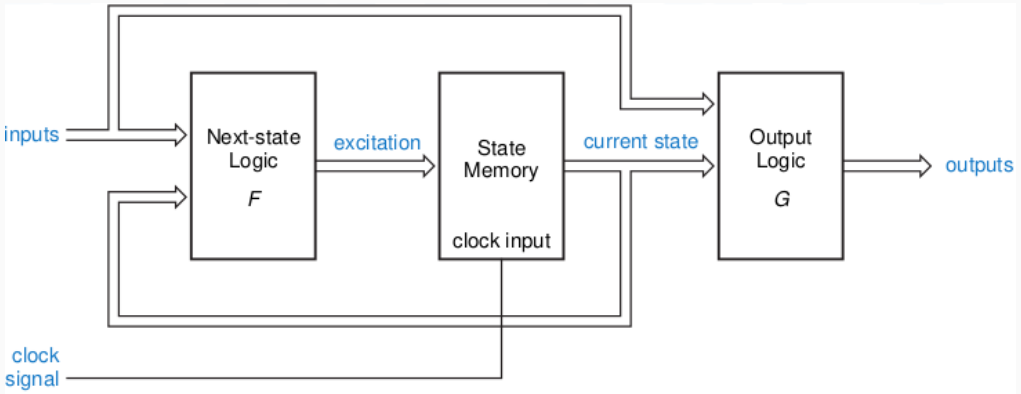
Si la salida depende tanto del estado actual cómo de las entradas, la llameremos como *FSM de Mealy*:



Características:

FSM - Mealy

Si la salida depende tanto del estado actual cómo de las entradas, la llameremos como *FSM de Mealy*:

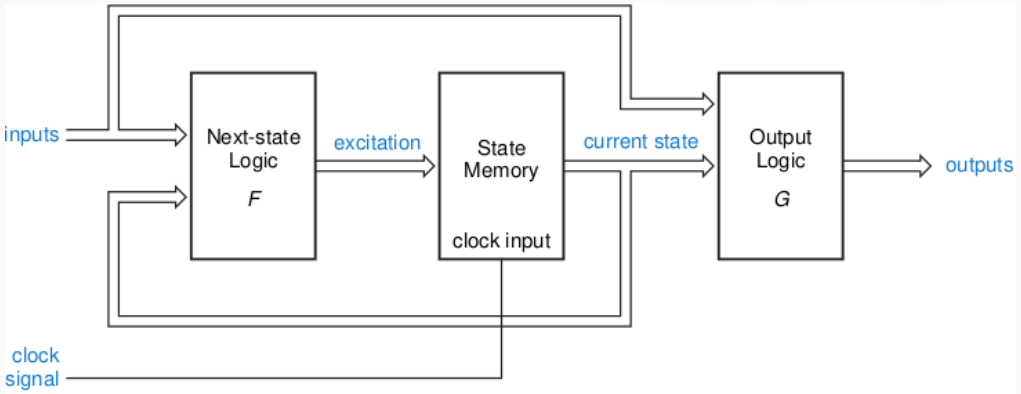


Características:

- La salida puede cambiar dentro del mismo clock en que se dispara la condición de transición

FSM - Mealy

Si la salida depende tanto del estado actual cómo de las entradas, la llameremos como *FSM de Mealy*:

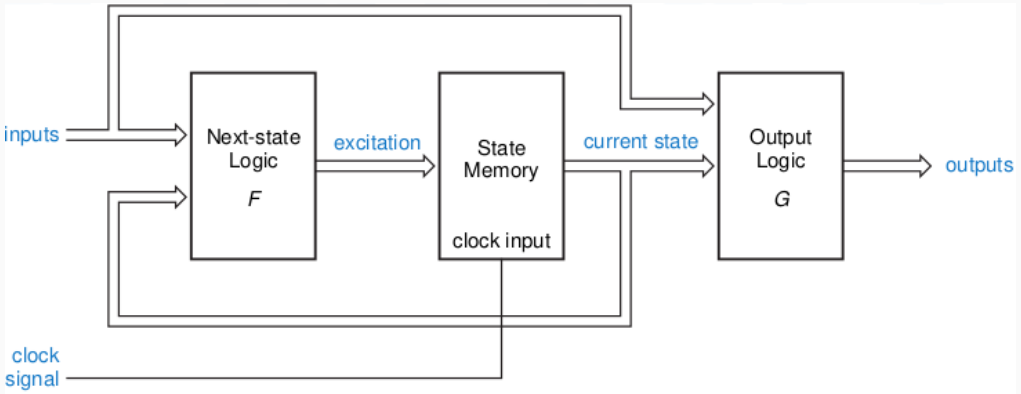


Características:

- La salida puede cambiar dentro del mismo clock en que se dispara la condición de transición
- Produce *glitches* a la salida

FSM - Mealy

Si la salida depende tanto del estado actual cómo de las entradas, la llameremos como *FSM de Mealy*:



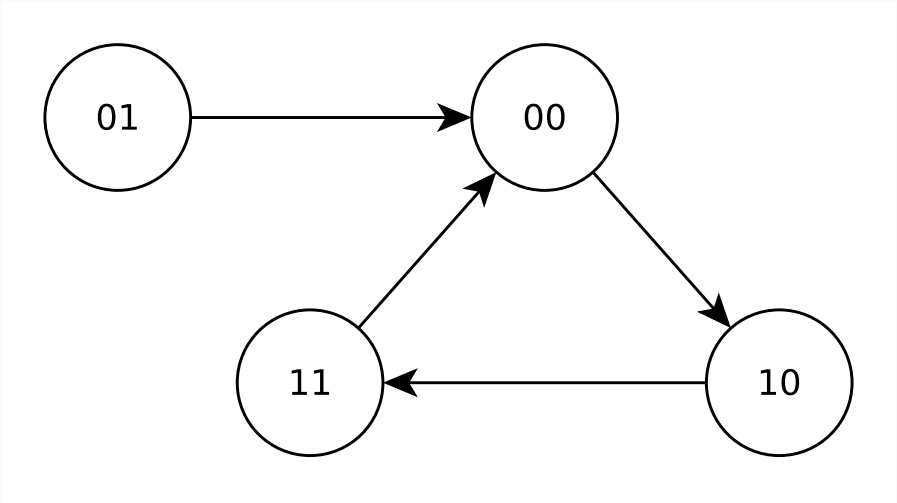
Características:

- La salida puede cambiar dentro del mismo clock en que se dispara la condición de transición
- Produce *glitches* a la salida
- La cantidad de estados para reproducir cierto comportamiento generalmente es más chica que en *Moore*

Ejercicio 1

Implementar una FSM en base a un registro de dos *bits* que siga los siguientes estados y que cada cambio se produzca al apretar un pulsador. Usando flip-flops D y compuertas básicas a elección.

Nos piden además que el componente a desarrollar cuente con una entrada de `Reset`.



Solución - Ejercicio 1

En este caso, dado un estado t definido por el valor de Q_1 y Q_0 podemos ver cuáles serán los próximos valores a almacenar:

$Q_1(t)$	$Q_0(t)$	$Q_1(t+1)$	$Q_0(t+1)$
0	1		
0	0		
1	0		
1	1		

Solución - Ejercicio 1

En este caso, dado un estado t definido por el valor de Q_1 y Q_0 podemos ver cuáles serán los próximos valores a almacenar:

$Q_1(t)$	$Q_0(t)$	$Q_1(t+1)$	$Q_0(t+1)$
0	1	0	0
0	0		
1	0		
1	1		

Solución - Ejercicio 1

En este caso, dado un estado t definido por el valor de Q_1 y Q_0 podemos ver cuáles serán los próximos valores a almacenar:

$Q_1(t)$	$Q_0(t)$	$Q_1(t+1)$	$Q_0(t+1)$
0	1	0	0
0	0	1	0
1	0		
1	1		

Solución - Ejercicio 1

En este caso, dado un estado t definido por el valor de Q_1 y Q_0 podemos ver cuáles serán los próximos valores a almacenar:

$Q_1(t)$	$Q_0(t)$	$Q_1(t+1)$	$Q_0(t+1)$
0	1	0	0
0	0	1	0
1	0	1	1
1	1		

Solución - Ejercicio 1

En este caso, dado un estado t definido por el valor de Q_1 y Q_0 podemos ver cuáles serán los próximos valores a almacenar:

$Q_1(t)$	$Q_0(t)$	$Q_1(t+1)$	$Q_0(t+1)$
0	1	0	0
0	0	1	0
1	0	1	1
1	1	0	0

Solución - Ejercicio 1

En este caso, dado un estado t definido por el valor de Q_1 y Q_0 podemos ver cuáles serán los próximos valores a almacenar:

$Q_1(t)$	$Q_0(t)$	$Q_1(t+1)$	$Q_0(t+1)$
0	1	0	0
0	0	1	0
1	0	1	1
1	1	0	0

¿Qué valores deberían tener D_1 y D_0 para obtener los valores deseados en el tiempo t+1, es decir, de $Q_1(t+1)$ y $Q_0(t+1)$? Esto es, la **lógica de próximo estado**

Solución - Ejercicio 1

Usando que el flip-flop D define su próximo valor en referencia a lo que tiene en la entrada D, vemos que la suma de productos nos define los valores de D:

Solución - Ejercicio 1

Usando que el flip-flop D define su próximo valor en referencia a lo que tiene en la entrada D, vemos que la suma de productos nos define los valores de D:

$$D_0 =$$
$$D_1 =$$
$$=$$

Solución - Ejercicio 1

Usando que el flip-flop D define su próximo valor en referencia a lo que tiene en la entrada D, vemos que la suma de productos nos define los valores de D:

$$D_0 = (Q_1.\bar{Q}_0)$$
$$D_1 =$$
$$=$$

Solución - Ejercicio 1

Usando que el flip-flop D define su próximo valor en referencia a lo que tiene en la entrada D, vemos que la suma de productos nos define los valores de D:

$$
\begin{aligned}
D_0 &= (Q_1.\bar{Q}_0) \\
D_1 &= (\bar{Q}_1.\bar{Q}_0) + (Q_1.\bar{Q}_0) \\
&=
\end{aligned}
$$

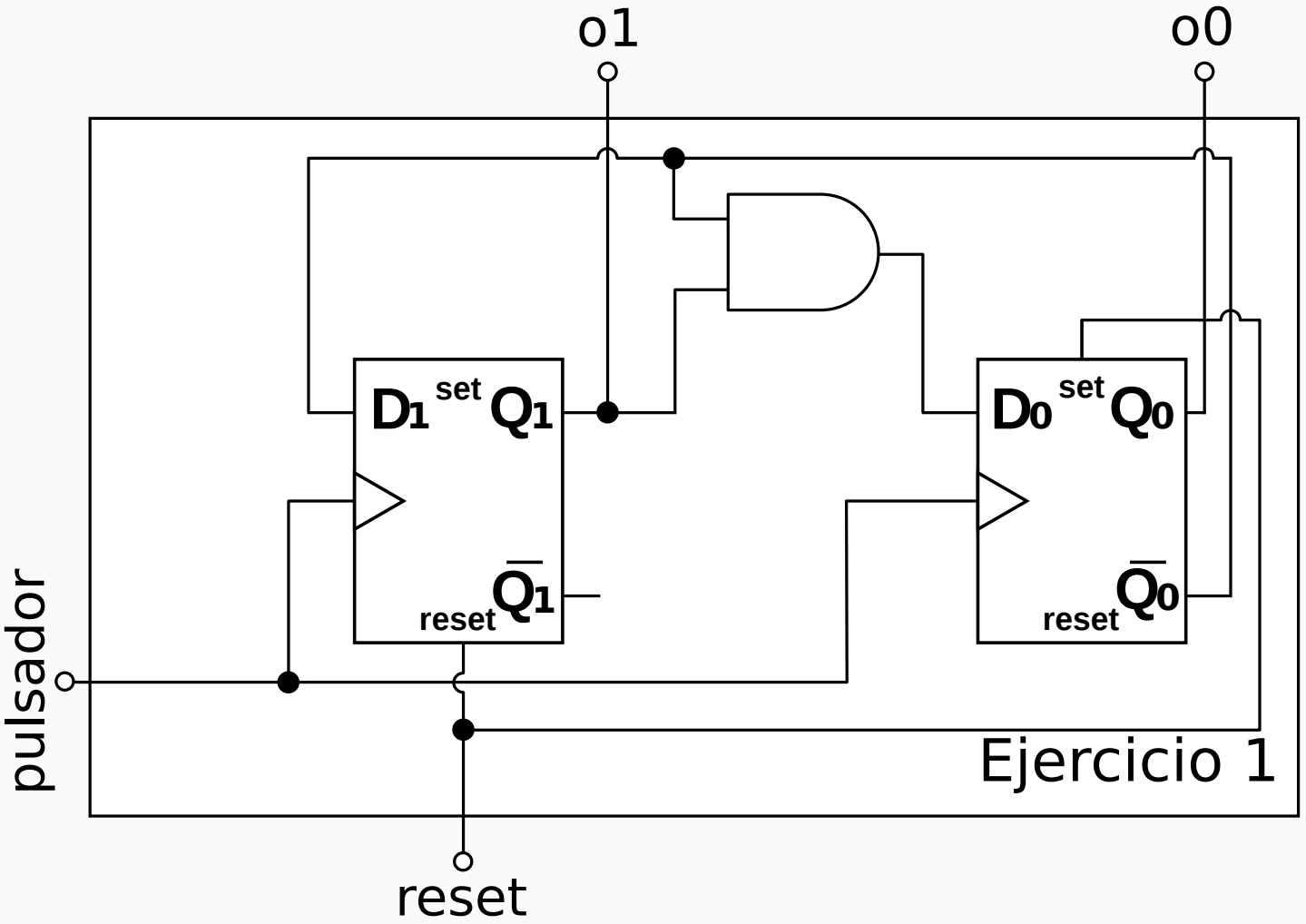
Solución - Ejercicio 1

Usando que el flip-flop D define su próximo valor en referencia a lo que tiene en la entrada D, vemos que la suma de productos nos define los valores de D:

$$
\begin{aligned}
D_0 &= (Q_1.\bar{Q}_0) \\
D_1 &= (\bar{Q}_1.\bar{Q}_0) + (Q_1.\bar{Q}_0) \\
&= (\bar{Q}_1 + Q_1).\bar{Q}_0 \\
&= 1.\bar{Q}_0 \\
&= \bar{Q}_0
\end{aligned}
$$

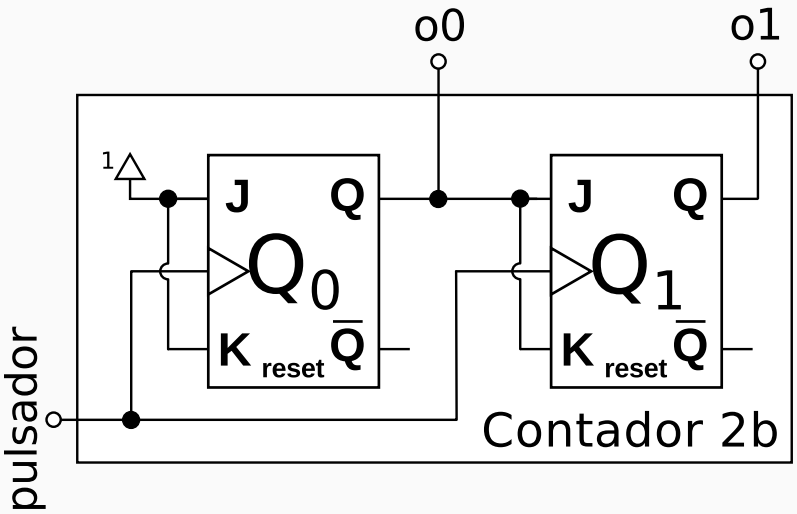
Solución - Ejercicio 1

Así se obtiene el siguiente circuito:



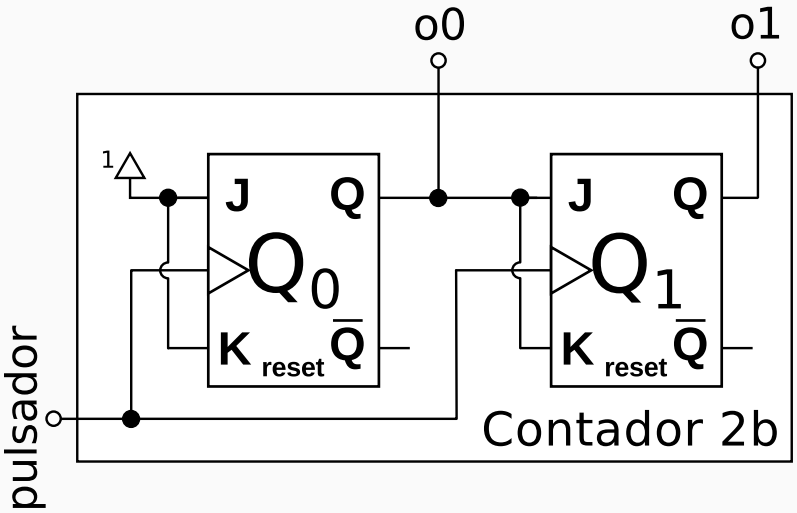
Ejercicio 2

Analizar los estados del siguiente componente:



Ejercicio 2

Analizar los estados del siguiente componente:

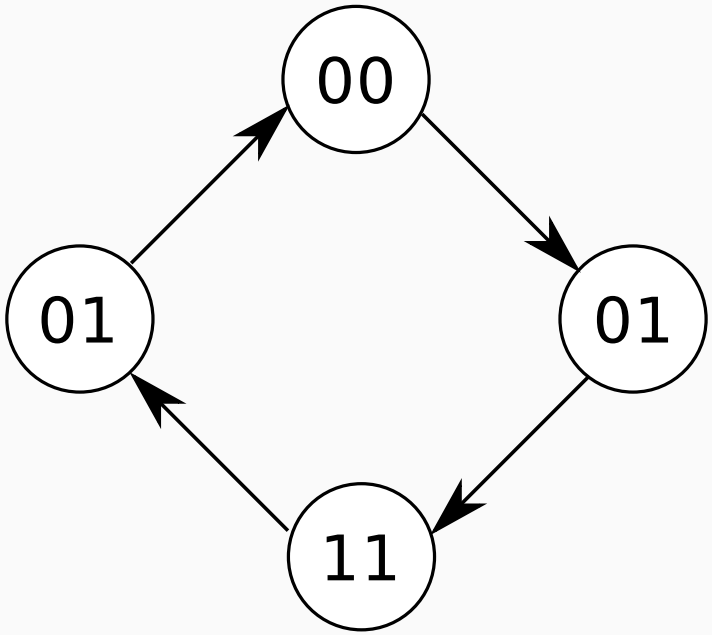


Solución:

$Q_1(t)$	$Q_0(t)$	$Q_1(t+1)$	$Q_0(t+1)$
0	0	0	1
0	1	1	0
1	0	1	1
1	1	0	0

Ejercicio 3

Implementar una FSM en base a ub registro de dos *bits* que siga los siguientes estados y que cada cambio se produzca al apretar un pulsador.



Solución - Ejercicio 3

Realizando un análisis análogo al del ejercicio anterior se obtiene:

$Q_1(t)$	$Q_0(t)$	$Q_1(t+1)$	$Q_0(t+1)$
0	0	0	1
0	1	?	?
1	0	-	-
1	1	0	1

Solución - Ejercicio 3

Realizando un análisis análogo al del ejercicio anterior se obtiene:

$Q_1(t)$	$Q_0(t)$	$Q_1(t+1)$	$Q_0(t+1)$
0	0	0	1
0	1	?	?
1	0	-	-
1	1	0	1

Lo cual no parece funcionar, ya que para el 01 no se puede determinar si es 11 ó 00 y para 10 no hay definido un próximo estado.

Solución - Ejercicio 3

En este caso, necesitamos introducir la **lógica de salida**.

¡No debemos confundir la etiqueta del estado con el valor de las salidas!

Solución - Ejercicio 3

En este caso, necesitamos introducir la **lógica de salida**.

¡No debemos confundir la etiqueta del estado con el valor de las salidas!

Renombremos a los estados con nombres únicos, por ejemplo:
$S_1 = 00, \qquad S_2 = 01, \qquad S_3 = 10, \qquad S_4 = 11$

Q_1	$Q_0 \rightarrow$	o_1	o_0
0	0 $\rightarrow$		
0	1 $\rightarrow$		
1	0 $\rightarrow$		
1	1 $\rightarrow$		

Solución - Ejercicio 3

En este caso, necesitamos introducir la **lógica de salida**.

¡No debemos confundir la etiqueta del estado con el valor de las salidas!

Renombremos a los estados con nombres únicos, por ejemplo:
$S_1 = 00, \quad S_2 = 01, \quad S_3 = 10, \quad S_4 = 11$

Q_1	$Q_0 \rightarrow$	o_1	o_0
0	0 $\rightarrow$	0	0
0	1 $\rightarrow$		
1	0 $\rightarrow$		
1	1 $\rightarrow$		

Solución - Ejercicio 3

En este caso, necesitamos introducir la **lógica de salida**.

¡No debemos confundir la etiqueta del estado con el valor de las salidas!

Renombremos a los estados con nombres únicos, por ejemplo:
$S_1 = 00, \quad S_2 = 01, \quad S_3 = 10, \quad S_4 = 11$

Q_1	$Q_0 \rightarrow$	o_1	o_0
0	0 $\rightarrow$	0	0
0	1 $\rightarrow$	0	1
1	0 $\rightarrow$		
1	1 $\rightarrow$		

Solución - Ejercicio 3

En este caso, necesitamos introducir la **lógica de salida**.

¡No debemos confundir la etiqueta del estado con el valor de las salidas!

Renombremos a los estados con nombres únicos, por ejemplo:
$S_1 = 00, \quad S_2 = 01, \quad S_3 = 10, \quad S_4 = 11$

Q_1	$Q_0 \rightarrow$	o_1	o_0
0	0 $\rightarrow$	0	0
0	1 $\rightarrow$	0	1
1	0 $\rightarrow$	1	1
1	1 $\rightarrow$		

Solución - Ejercicio 3

En este caso, necesitamos introducir la **lógica de salida**.

¡No debemos confundir la etiqueta del estado con el valor de las salidas!

Renombremos a los estados con nombres únicos, por ejemplo:
$S_1 = 00, \quad S_2 = 01, \quad S_3 = 10, \quad S_4 = 11$

Q_1	$Q_0 \rightarrow$	o_1	o_0
0	0 $\rightarrow$	0	0
0	1 $\rightarrow$	0	1
1	0 $\rightarrow$	1	1
1	1 $\rightarrow$	0	1

Solución - Ejercicio 3

Con lo cual podemos decir que:

$$o_0 = Q_1 + Q_0 \quad \text{por producto de sumas}$$

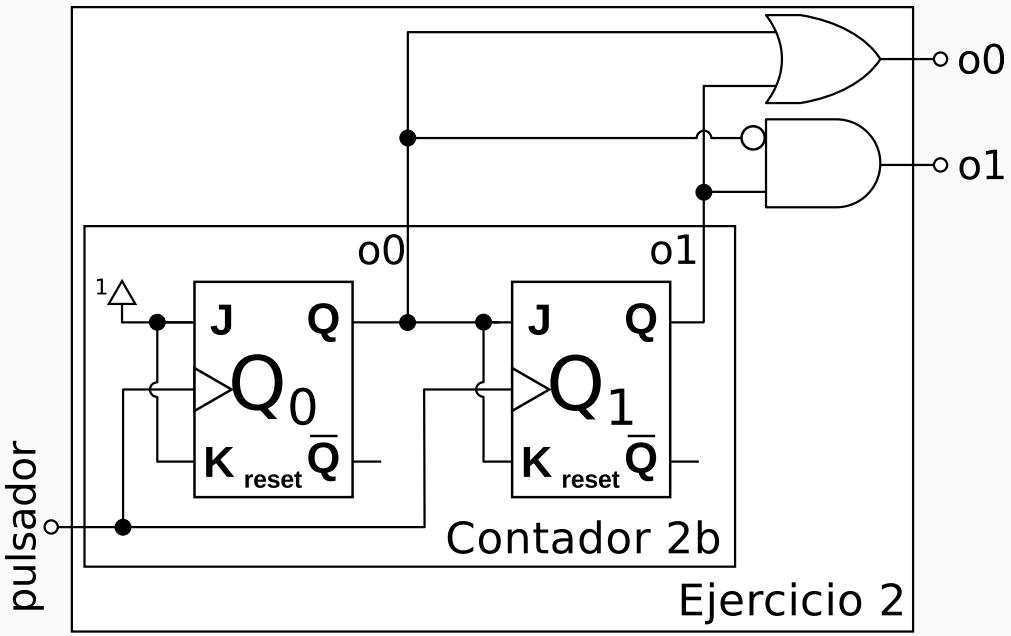
$$o_1 = Q_1 \,.\, \bar{Q}_0 \quad \text{por suma de productos}$$

Solución - Ejercicio 3

Con lo cual podemos decir que:

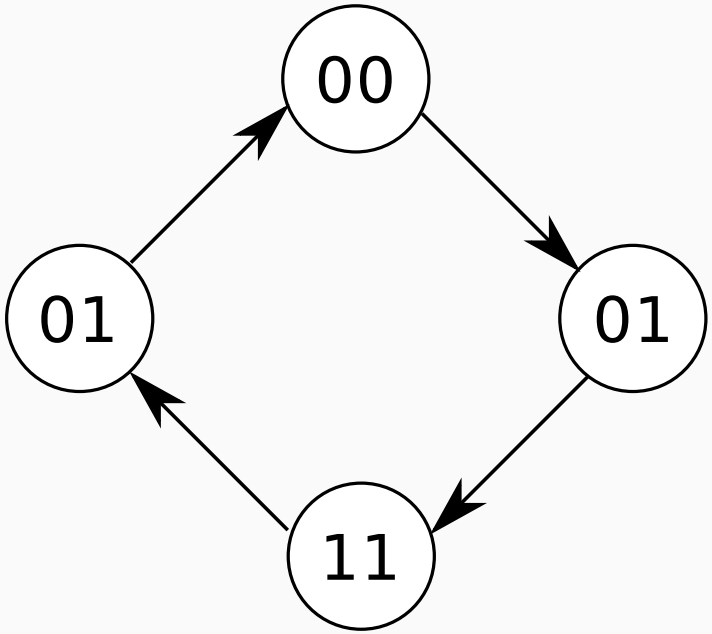
$$o_0 = Q_1 + Q_0 \quad \text{por producto de sumas}$$

$$o_1 = Q_1 \,.\, \bar{Q}_0 \quad \text{por suma de productos}$$

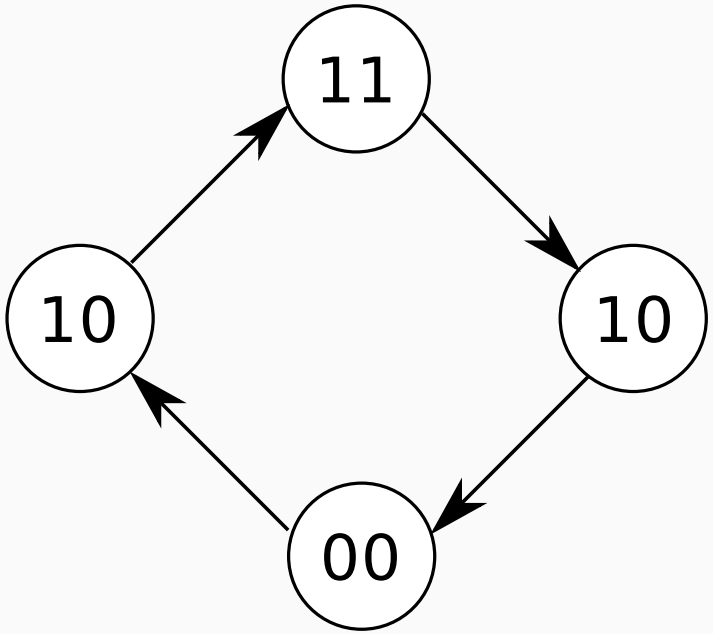


Ejercicio 3- bis

Implementar una FSM en base a un registro de dos *bits* que siga los siguientes estados y que cada cambio se produzca al apretar un pulsador. Con el agregado de que tenga una entrada llamada NEG que genera los siguientes comportamientos:



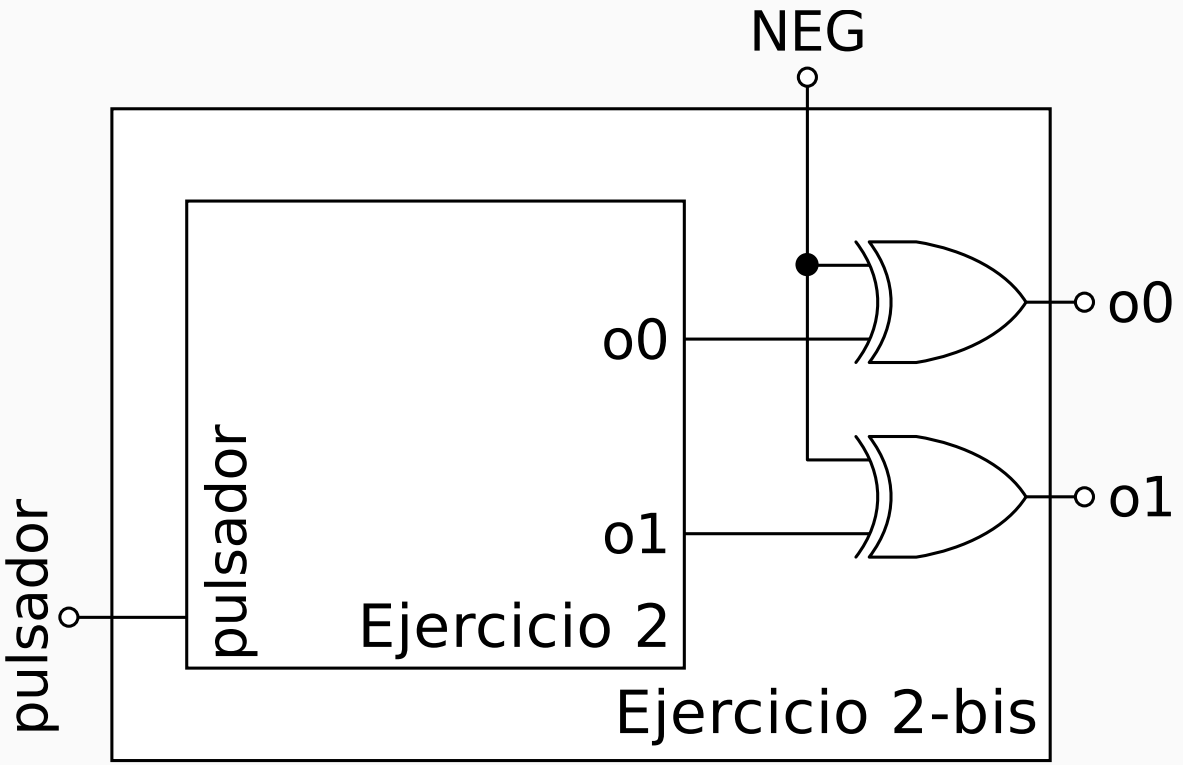
Si NEG vale 0



Si NEG vale 1

Solución - Ejercicio 3

En este caso transformamos la máquina en una **FSM de Mealy**:



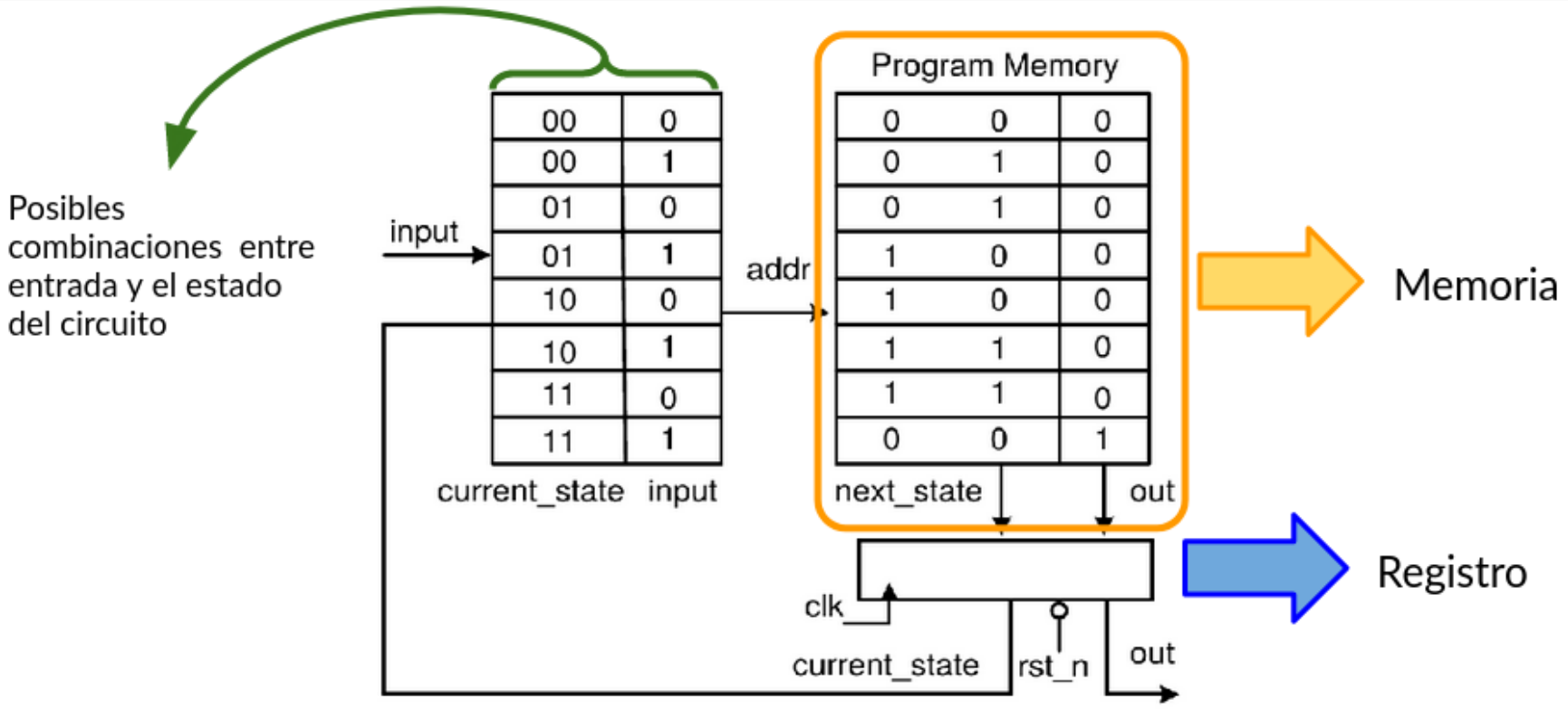
FSM Microprogramada

Ejercicio para pensar:

- Diseñar un circuito sincrónico con una entrada de un bit y una salida de un bit que detecte la ocurrencia de 4 bits en '1' a la entrada. El mismo pondrá un '1' en su salida durante un ciclo de reloj luego de contar cuatro '1's en su entrada. El ciclo se repite indefinidamente.
- Se puede utilizar un registro de 3 bits y una "Memoria" de 3 bits y 8 posiciones.
- ¿Cómo se cambiaría el comportamiento del circuito sin cambiar el hardware?

FSM Microprogramada

Resolución



Conclusiones

Conclusiones

- Estudiamos la realimentación en circuitos para mantener un dato en el tiempo y vimos implementaciones de *latchs*
- Estudiamos los problemas asociados a los tiempos de propagación de las señales
- Analizamos el uso de un *clock* para limitar la cantidad de estados, controlar las transiciones y evitar carreras
- Vimos algunas implementaciones de flip-flops, registros y memorias
- Estudiamos con cierta formalidad el comportamiento de circuitos secuenciales en general

Conclusiones

- Estudiamos la realimentación en circuitos para mantener un dato en el tiempo y vimos implementaciones de *latchs*
- Estudiamos los problemas asociados a los tiempos de propagación de las señales
- Analizamos el uso de un *clock* para limitar la cantidad de estados, controlar las transiciones y evitar carreras
- Vimos algunas implementaciones de flip-flops, registros y memorias
- Estudiamos con cierta formalidad el comportamiento de circuitos secuenciales en general

Conclusiones

- Estudiamos la realimentación en circuitos para mantener un dato en el tiempo y vimos implementaciones de *latchs*
- Estudiamos los problemas asociados a los tiempos de propagación de las señales
- Analizamos el uso de un *clock* para limitar la cantidad de estados, controlar las transiciones y evitar carreras
- Vimos algunas implementaciones de flip-flops, registros y memorias
- Estudiamos con cierta formalidad el comportamiento de circuitos secuenciales en general

Conclusiones

- Estudiamos la realimentación en circuitos para mantener un dato en el tiempo y vimos implementaciones de *latchs*
- Estudiamos los problemas asociados a los tiempos de propagación de las señales
- Analizamos el uso de un *clock* para limitar la cantidad de estados, controlar las transiciones y evitar carreras
- Vimos algunas implementaciones de flip-flops, registros y memorias
- Estudiamos con cierta formalidad el comportamiento de circuitos secuenciales en general

Conclusiones

- Estudiamos la realimentación en circuitos para mantener un dato en el tiempo y vimos implementaciones de *latchs*
- Estudiamos los problemas asociados a los tiempos de propagación de las señales
- Analizamos el uso de un *clock* para limitar la cantidad de estados, controlar las transiciones y evitar carreras
- Vimos algunas implementaciones de flip-flops, registros y memorias
- Estudiamos con cierta formalidad el comportamiento de circuitos secuenciales en general

Preguntas

¡Gracias!