

Maquinas de estado finito (FSM)

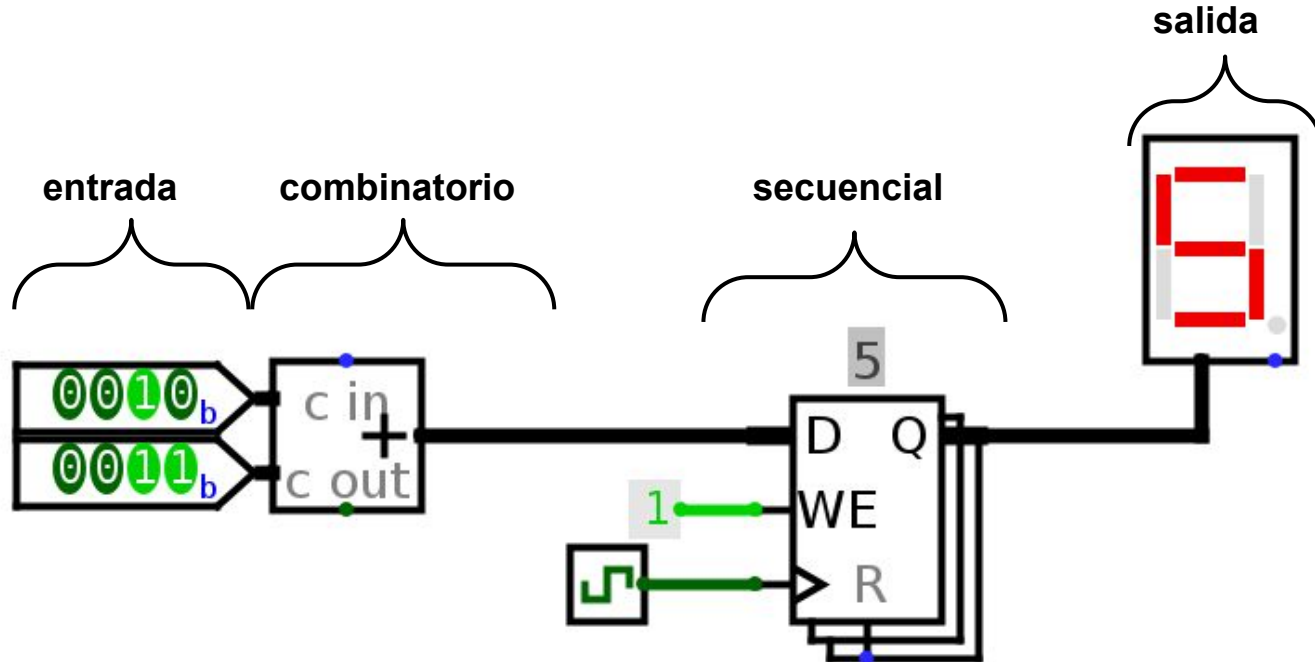
Unidad 5.0
Secuenciadores y FSM

Versión 1.0.0

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

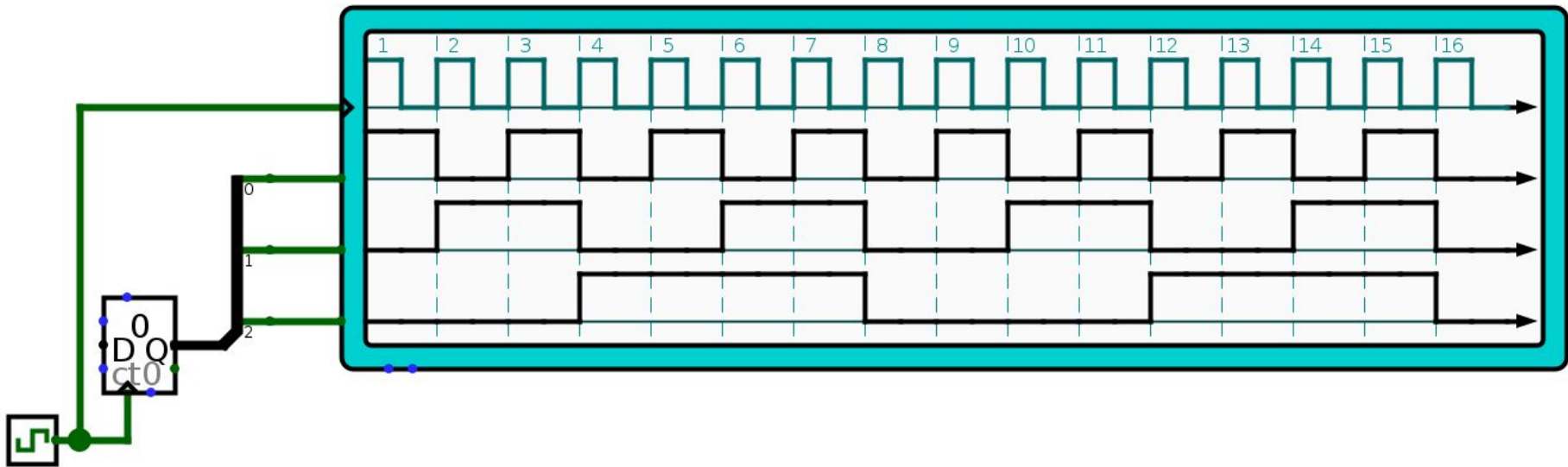
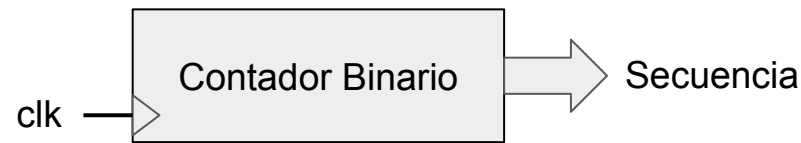
Circuitos aritméticos almacenan en registros

Hasta ahora hemos trabajado con circuitos combinacionales (ej: sumadores) y circuitos secuenciales (registros). Generalmente tenemos algún circuito combinacional (operación) y el resultado lo almacenamos en un circuito secuencial.



Contadores que generan secuencias

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



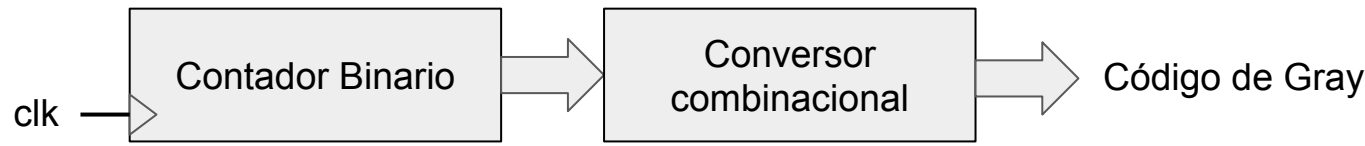
Secuenciador en código de Gray

clk	C ₂	C ₁	C ₀		G ₂	G ₁	G ₀
0	0	0	0		0	0	0
1	0	0	1		0	0	1
2	0	1	0		0	1	1
3	0	1	1	→	0	1	0
4	1	0	0		1	1	0
5	1	0	1		1	1	1
6	1	1	0		1	0	1
7	1	1	1		1	0	0

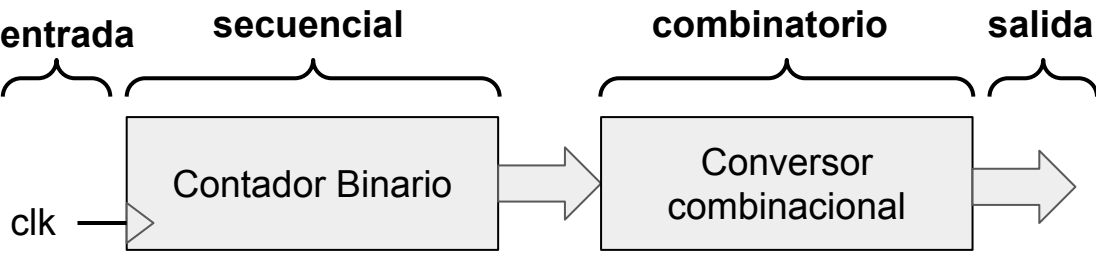
$$G_2 = C_2$$

$$G_1 = \overline{C_2} \cdot C_1 + C_2 \cdot \overline{C_1} \rightarrow C_2 \text{ xor } C_1$$

$$G_0 = C_1 \cdot \overline{C_0} + \overline{C_1} \cdot C_0 \rightarrow C_1 \text{ xor } C_0$$

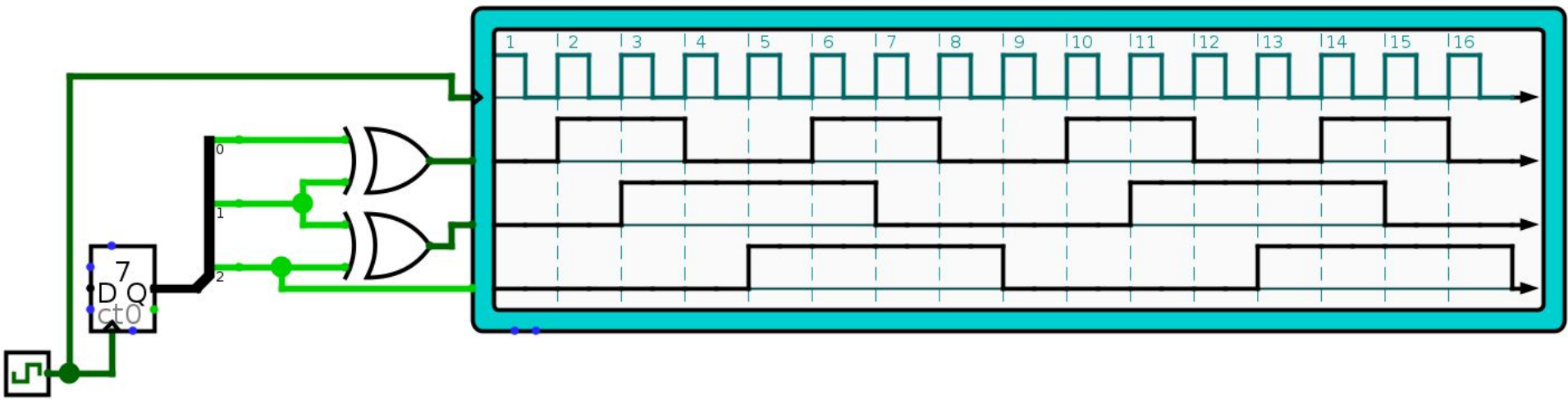


Secuenciador en código de Gray



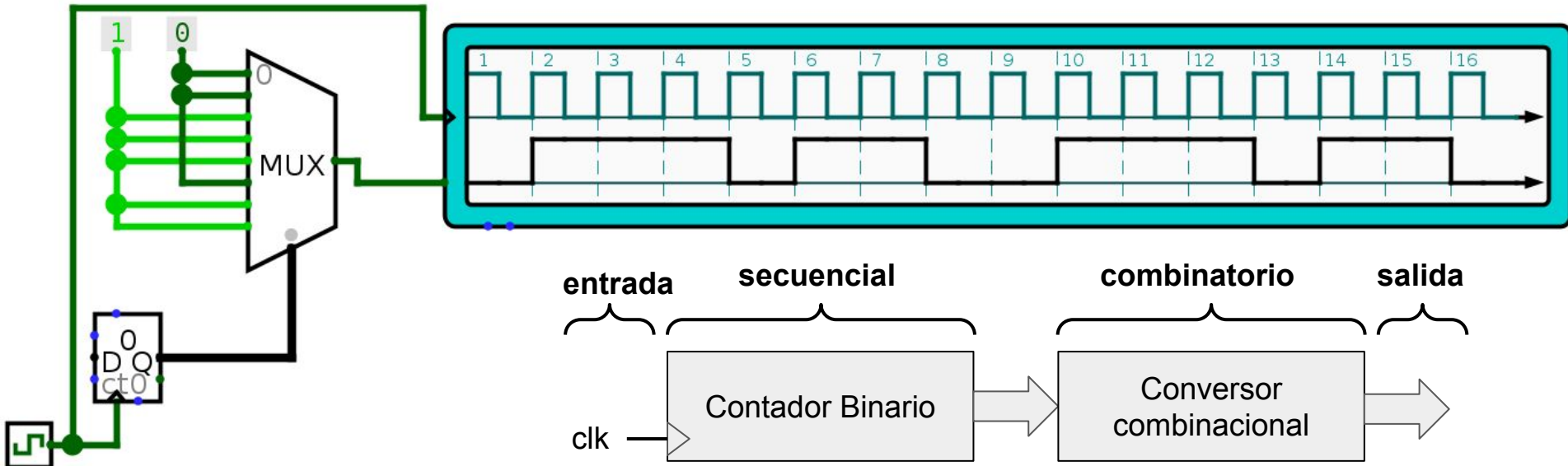
Clk	C ₂	C ₁	C ₀
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

G ₂	G ₁	G ₀
0	0	0
0	0	1
0	1	1
0	1	0
1	1	0
1	1	1
1	0	1
1	0	0



Generador de señal periódica programable

El contador genera todos los valores de selección posibles en el MUX. Podemos generar una señal periódica programable cambiando los unos y los ceros de cada entrada de datos del MUX. Podemos crear de esta forma un conversor de paralelo a serie muy fácilmente.



Máquinas de Estado Finito (FSM)

Modelo elemental de la teoría de la computación

Vamos a ver un diseño elemental que permite resolver algunos problemas típicos de la teoría de la computación. Hasta ahora hemos visto circuitos que resuelven operaciones (ej: Aritméticas como sumadores), circuitos que memorizan (ej: Latch) y hemos conectado estos dos tipos para convertir códigos y generar secuencias periódicas.

Con una FSM vamos a validar una secuencia de datos de entrada que ocurre a lo largo del tiempo.

Ejemplo: Máquina expendedora de gaseosa.

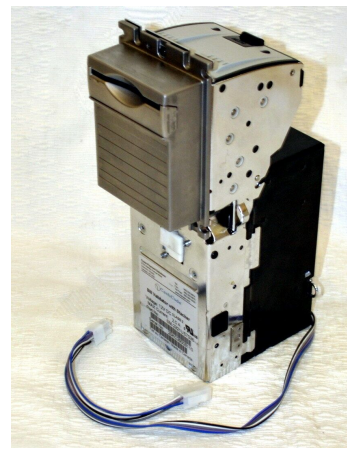
Vamos a diseñar un circuito que habilite la entrega de una gaseosa cuando la cantidad de dinero ingresado sea el suficiente. Esta máquina NO da vuelto. Posee un detector de billetes que detecta billetes de \$50, \$100 y \$200. Para entregar la gaseosa genera un 1 en la traba de puerta para destrabar. Cuando se retira la gaseosa se genera un 1 en reset para volver al estado inicial.

Máquina de gaseosa

Tenemos por un lado el detector de billetes. Cuando ingresa un billete, el detector lo valida y genera una salida binaria (A1A0) que identifica al billete:

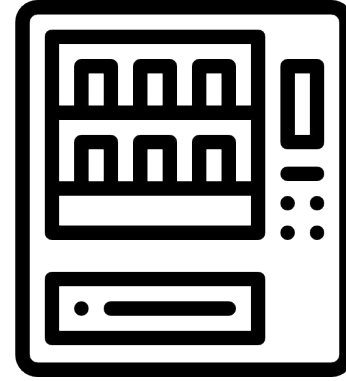
- 00 = Inválido
- 01 = \$50
- 10 = \$100
- 11 = Inválido

También genera un pulso positivo llamado Clk cuando el valor de A1A0 cambia. De esta forma podemos detectar que se ha insertado un billete nuevo.



Máquina de gaseosa

Cuando validamos que se ha ingresado el total se entrega la gaseosa generando un uno en la salida G. Esto activa un motor que deja caer la gaseosa en el canasto para retirar. La salida se mantiene en uno permitiendo al usuario ingresar su mano retirar la misma. En el momento que se retira la gaseosa existe una entrada R de reset que genera un pulso. Así que nuestra máquina va a estar compuesta por un circuito cuya entrada va a ser A1,A0, Clk, R, y la salida va a ser G.



Vamos a diseñar este circuito para que cuando el usuario haya ingresado \$150 la máquina habilite el retiro de una gaseosa. No da vuelta.

Estado a lo largo del tiempo

El circuito debe detectar que ingresaron \$150. El problema es que el ingreso se va realizando a lo largo del tiempo. Existen varias formas en las cuales se pueden llegar a \$150 utilizando billetes de \$50 y \$100

- Se ingresan 3 billetes de \$50
- Se ingresa primero uno de \$50 y luego uno de \$100
- Se ingresa primero uno de \$100 y luego uno de \$50
- Se ingresan dos de \$100 (no da vuelto)

Vamos a representar todas las combinaciones en un diagrama...

Diagrama de transición de estados

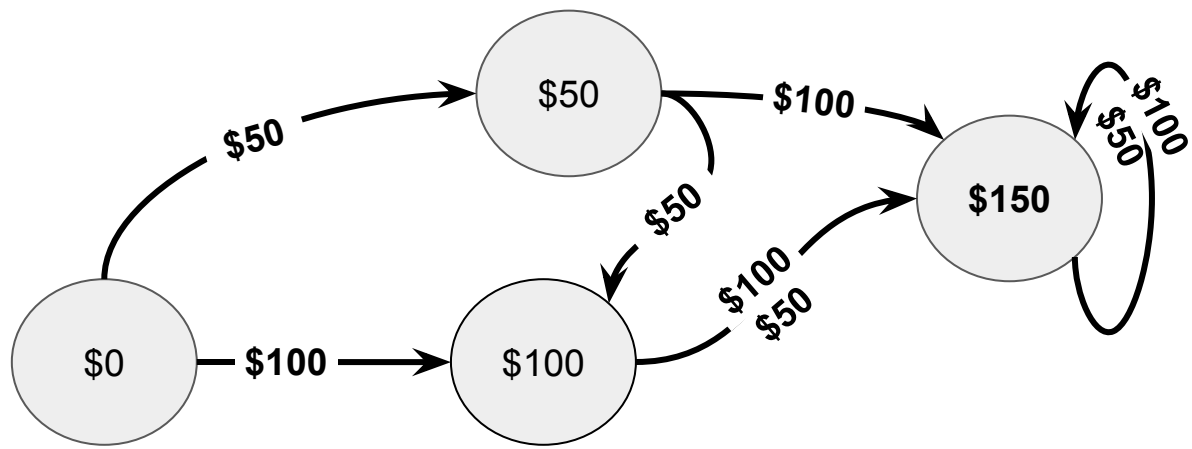
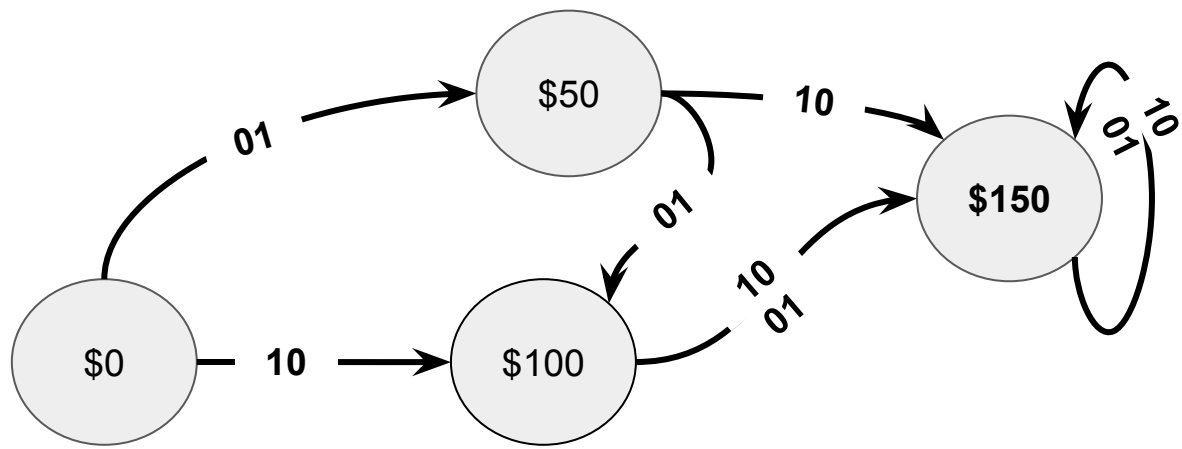
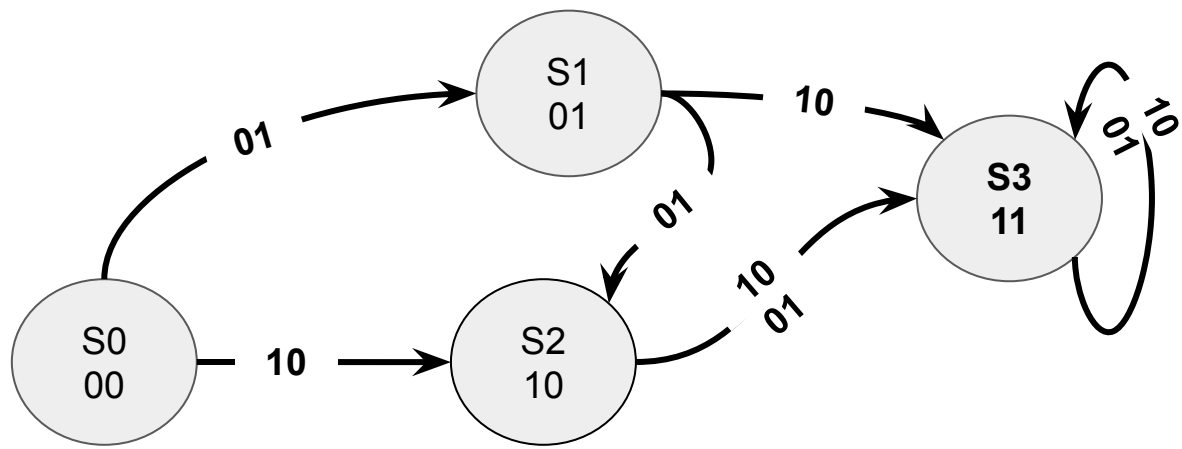


Diagrama de transición de estados



A ₁	A ₀	\$
0	0	--
0	1	50
1	0	100
1	1	--

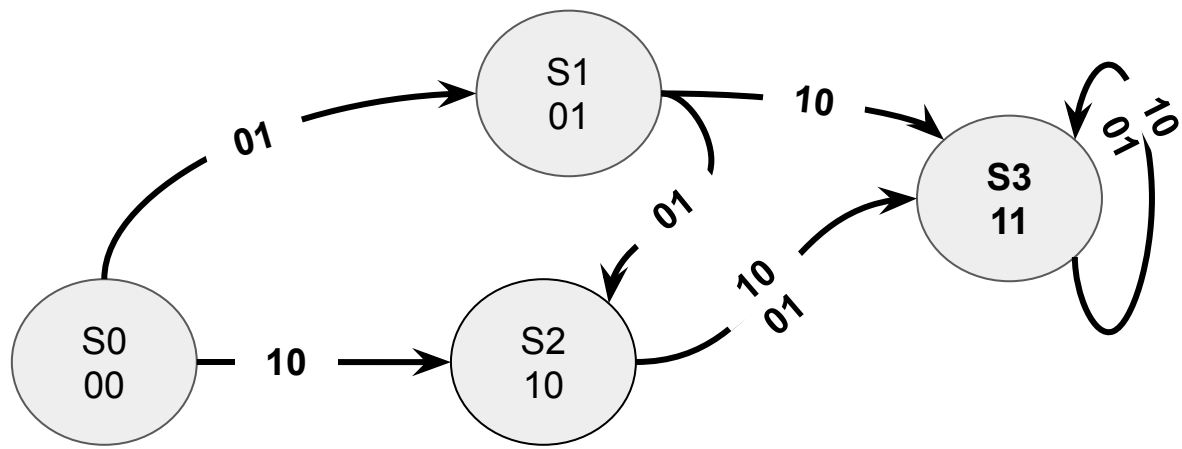
Diagrama de transición de estados



A ₁	A ₀	\$
0	0	--
0	1	50
1	0	100
1	1	--

\$	S ₁	S ₀
0	0	0
50	0	1
100	1	0
150	1	1

Diagrama de transición de estados

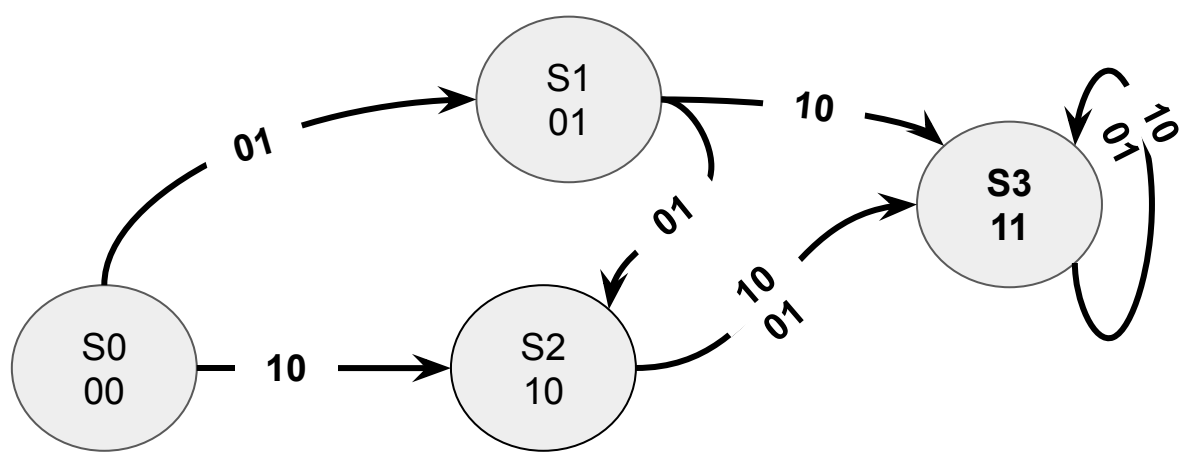


A ₁	A ₀	\$
0	0	--
0	1	50
1	0	100
1	1	--

\$	S ₁	S ₀
0	0	0
50	0	1
100	1	0
150	1	1

S1	S0	A ₁	A ₀	S1 _{n+1}	S0 _{n+1}
0	0	0	0		
0	0	0	1		
0	0	1	0		
0	0	1	1		
0	1	0	0		
0	1	0	1		
0	1	1	0		
0	1	1	1		
1	0	0	0		
1	0	0	1		
1	0	1	0		
1	0	1	1		
1	1	0	0		
1	1	0	1		
1	1	1	0		
1	1	1	1		

Diagrama de transición de estados

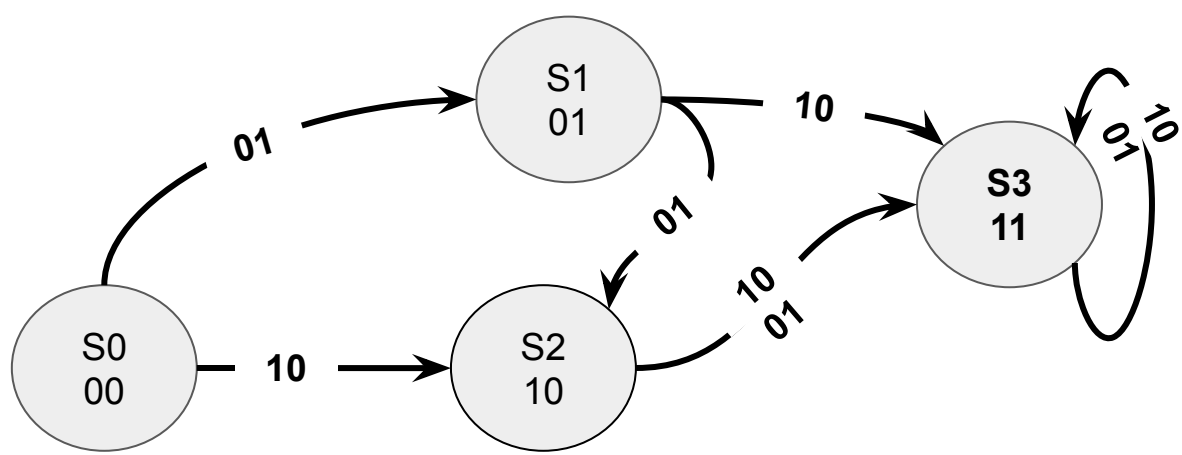


A ₁	A ₀	\$
0	0	--
0	1	50
1	0	100
1	1	--

\$	S ₁	S ₀
0	0	0
50	0	1
100	1	0
150	1	1

S1	S0	A ₁	A ₀	S1 _{n+1}	S0 _{n+1}
0	0	0	0	0	0
0	0	0	1		
0	0	1	0		
0	0	1	1		
0	1	0	0	0	1
0	1	0	1		
0	1	1	0		
0	1	1	1		
1	0	0	0	1	0
1	0	0	1		
1	0	1	0		
1	0	1	1		
1	1	0	0	1	1
1	1	0	1		
1	1	1	0		
1	1	1	1		

Diagrama de transición de estados

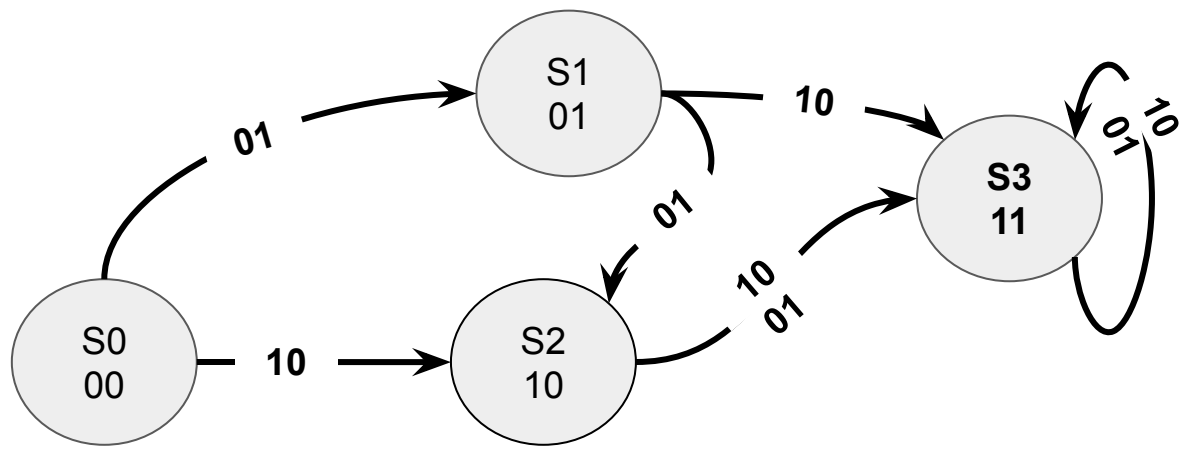


A ₁	A ₀	\$
0	0	--
0	1	50
1	0	100
1	1	--

\$	S ₁	S ₀
0	0	0
50	0	1
100	1	0
150	1	1

S1	S0	A ₁	A ₀	S1 _{n+1}	S0 _{n+1}
0	0	0	0	0	0
0	0	0	1		
0	0	1	0		
0	0	1	1	0	0
0	1	0	0	0	1
0	1	0	1		
0	1	1	0		
0	1	1	1	0	1
1	0	0	0	1	0
1	0	0	1		
1	0	1	0		
1	0	1	1	1	0
1	1	0	0	1	1
1	1	0	1		
1	1	1	0		
1	1	1	1	1	1

Diagrama de transición de estados

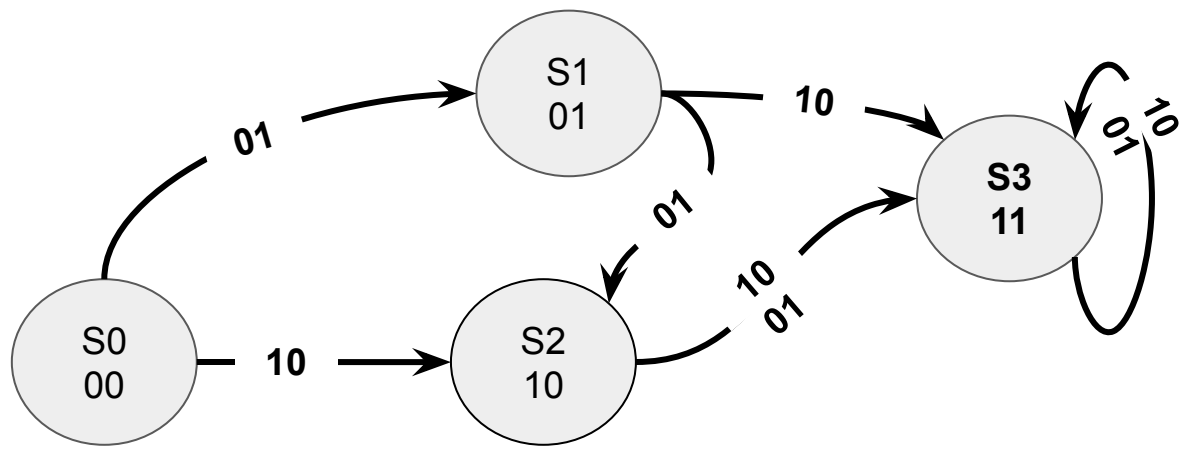


A ₁	A ₀	\$
0	0	--
0	1	50
1	0	100
1	1	--

\$	S ₁	S ₀
0	0	0
50	0	1
100	1	0
150	1	1

S1	S0	A ₁	A ₀	S1 _{n+1}	S0 _{n+1}
0	0	0	0	0	0
0	0	0	1	0	1
0	0	1	0	1	0
0	0	1	1	0	0
0	1	0	0	0	1
0	1	0	1		
0	1	1	0		
0	1	1	1	0	1
1	0	0	0	1	0
1	0	0	1		
1	0	1	0		
1	0	1	1	1	0
1	1	0	0	1	1
1	1	0	1		
1	1	1	0		
1	1	1	1	1	1

Diagrama de transición de estados

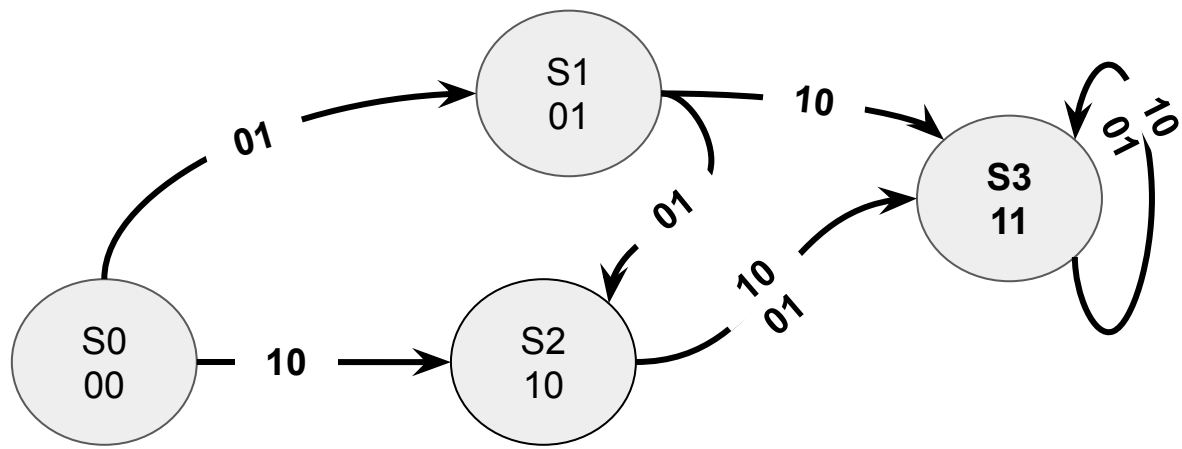


A ₁	A ₀	\$
0	0	--
0	1	50
1	0	100
1	1	--

\$	S ₁	S ₀
0	0	0
50	0	1
100	1	0
150	1	1

S1	S0	A ₁	A ₀	S1 _{n+1}	S0 _{n+1}
0	0	0	0	0	0
0	0	0	1	0	1
0	0	1	0	1	0
0	0	1	1	0	0
0	1	0	0	0	1
0	1	0	1	1	0
0	1	1	0	1	1
0	1	1	1	0	1
1	0	0	0	1	0
1	0	0	1		
1	0	1	0		
1	0	1	1	1	0
1	1	0	0	1	1
1	1	0	1		
1	1	1	0		
1	1	1	1	1	1

Diagrama de transición de estados

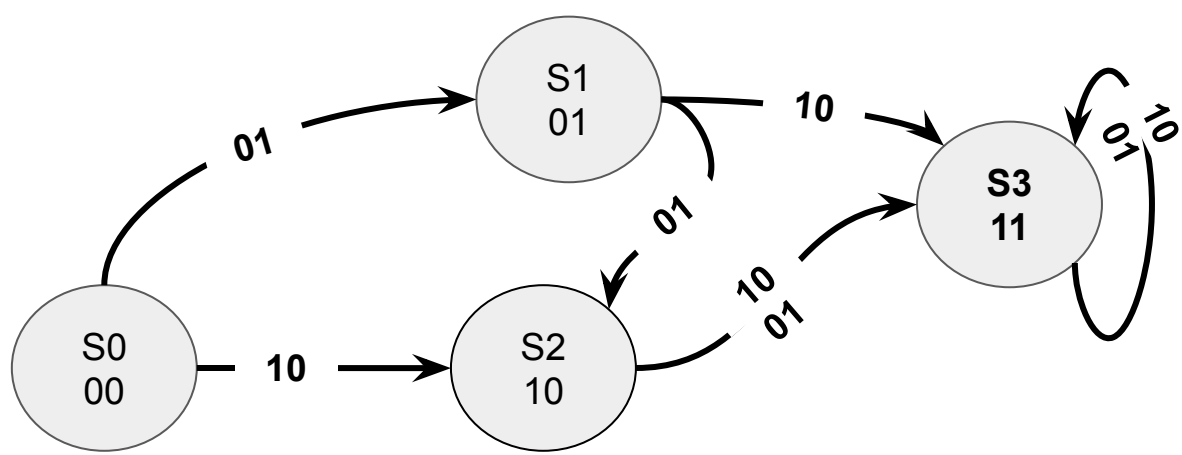


A ₁	A ₀	\$
0	0	--
0	1	50
1	0	100
1	1	--

\$	S ₁	S ₀
0	0	0
50	0	1
100	1	0
150	1	1

S1	S0	A ₁	A ₀	S1 _{n+1}	S0 _{n+1}
0	0	0	0	0	0
0	0	0	1	0	1
0	0	1	0	1	0
0	0	1	1	0	0
0	1	0	0	0	1
0	1	0	1	1	0
0	1	1	0	1	1
0	1	1	1	0	1
1	0	0	0	1	0
1	0	0	1	1	1
1	0	1	0	1	1
1	0	1	1	1	0
1	1	0	0	1	1
1	1	0	1		
1	1	1	0		
1	1	1	1	1	1

Diagrama de transición de estados

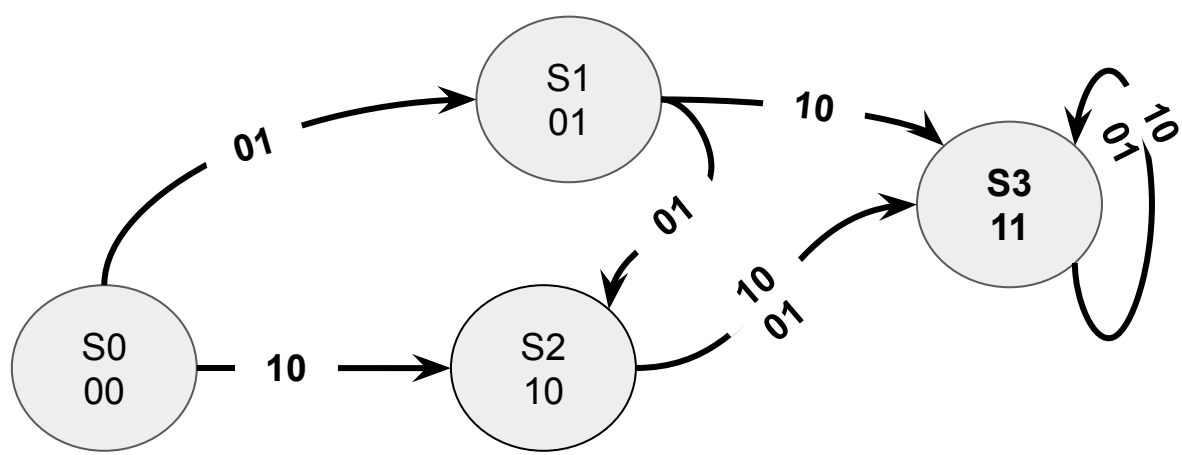


A ₁	A ₀	\$
0	0	--
0	1	50
1	0	100
1	1	--

\$	S ₁	S ₀
0	0	0
50	0	1
100	1	0
150	1	1

S1	S0	A ₁	A ₀	S1 _{n+1}	S0 _{n+1}
0	0	0	0	0	0
0	0	0	1	0	1
0	0	1	0	1	0
0	0	1	1	0	0
0	1	0	0	0	1
0	1	0	1	1	0
0	1	1	0	1	1
0	1	1	1	0	1
1	0	0	0	1	0
1	0	0	1	1	1
1	0	1	0	1	1
1	0	1	1	1	0
1	1	0	0	1	1
1	1	0	1	1	1
1	1	1	0	1	1
1	1	1	1	1	1

Diagrama de transición de estados

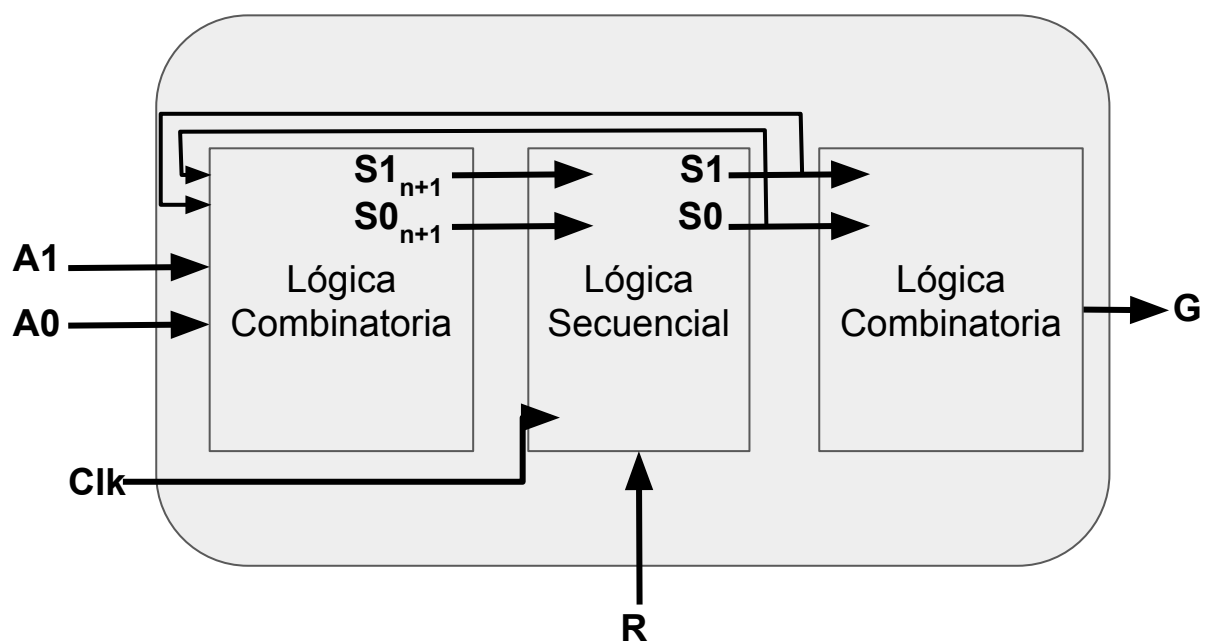


A ₁	A ₀	\$
0	0	--
0	1	50
1	0	100
1	1	--

\$	S ₁	S ₀
0	0	0
50	0	1
100	1	0
150	1	1

S1	S0	A ₁	A ₀	S1 _{n+1}	S0 _{n+1}	G
0	0	0	0	0	0	0
0	0	0	1	0	1	0
0	0	1	0	1	0	0
0	0	1	1	0	0	0
0	1	0	0	0	1	0
0	1	0	1	1	0	0
0	1	1	0	1	1	0
0	1	1	1	0	1	0
1	0	0	0	1	0	0
1	0	0	1	1	1	0
1	0	1	0	1	1	0
1	0	1	1	1	0	0
1	1	0	0	1	1	1
1	1	0	1	1	1	1
1	1	1	0	1	1	1
1	1	1	1	1	1	1

S1	S0	A ₁	A ₀	S1 _{n+1}	S0 _{n+1}	G
0	0	0	0	0	0	0
0	0	0	1	0	1	0
0	0	1	0	1	0	0
0	0	1	1	0	0	0
0	1	0	0	0	1	0
0	1	0	1	1	0	0
0	1	1	0	1	1	0
0	1	1	1	0	1	0
1	0	0	0	1	0	0
1	0	0	1	1	1	0
1	0	1	0	1	1	0
1	0	1	1	1	0	0
1	1	0	0	1	1	1
1	1	0	1	1	1	1
1	1	1	0	1	1	1
1	1	1	1	1	1	1

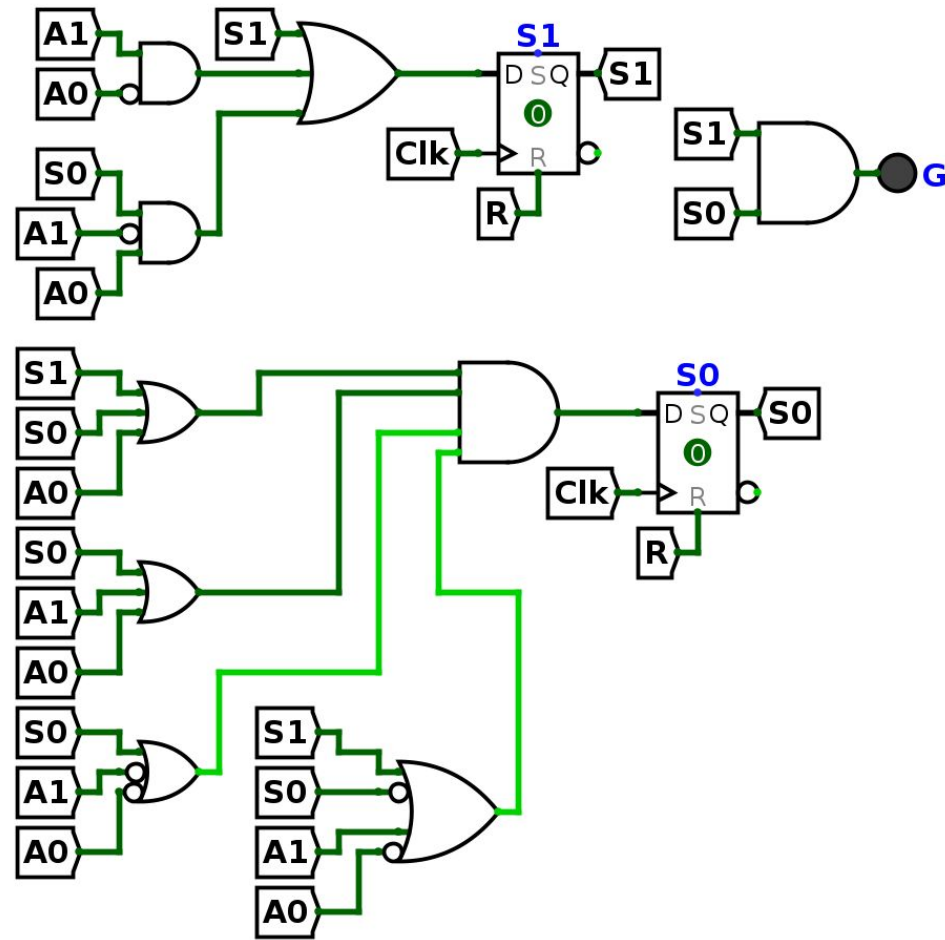
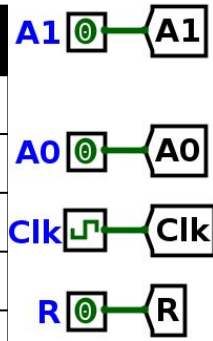


$$S1_{n+1} = S1 + A1 \cdot \overline{A0} + S0 \cdot \overline{A1} \cdot A0$$

$$S0_{n+1} = (S1 + S0 + A0) \cdot (S0 + A1 + A0) \cdot (S0 + \overline{A1} + \overline{A0}) \cdot (S1 + \overline{S0} + A1 + \overline{A0})$$

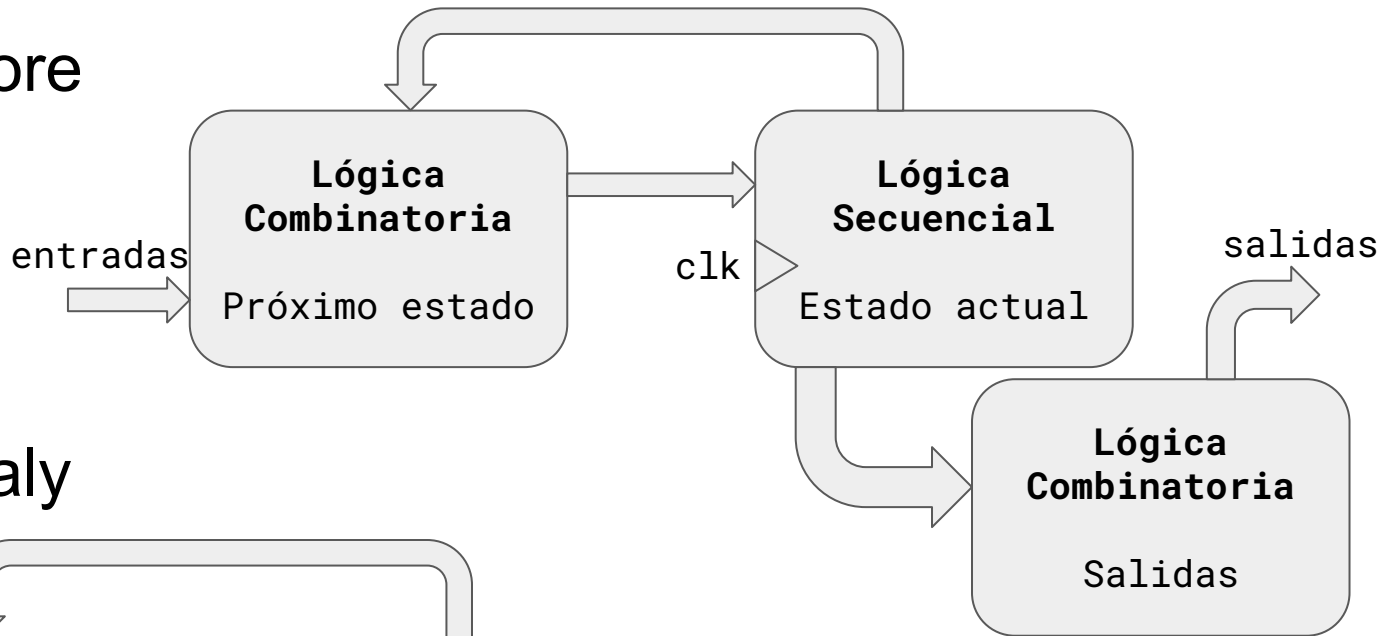
$$G = S1 \cdot S0$$

S1	S0	A ₁	A ₀	S ₁ _{n+1}	S ₀ _{n+1}	G
0	0	0	0	0	0	0
0	0	0	1	0	1	0
0	0	1	0	1	0	0
0	0	1	1	0	0	0
0	1	0	0	0	1	0
0	1	0	1	1	0	0
0	1	1	0	1	1	0
0	1	1	1	0	1	0
1	0	0	0	1	0	0
1	0	0	1	1	1	0
1	0	1	0	1	1	0
1	0	1	1	1	0	0
1	1	0	0	1	1	1
1	1	0	1	1	1	1
1	1	1	0	1	1	1
1	1	1	1	1	1	1

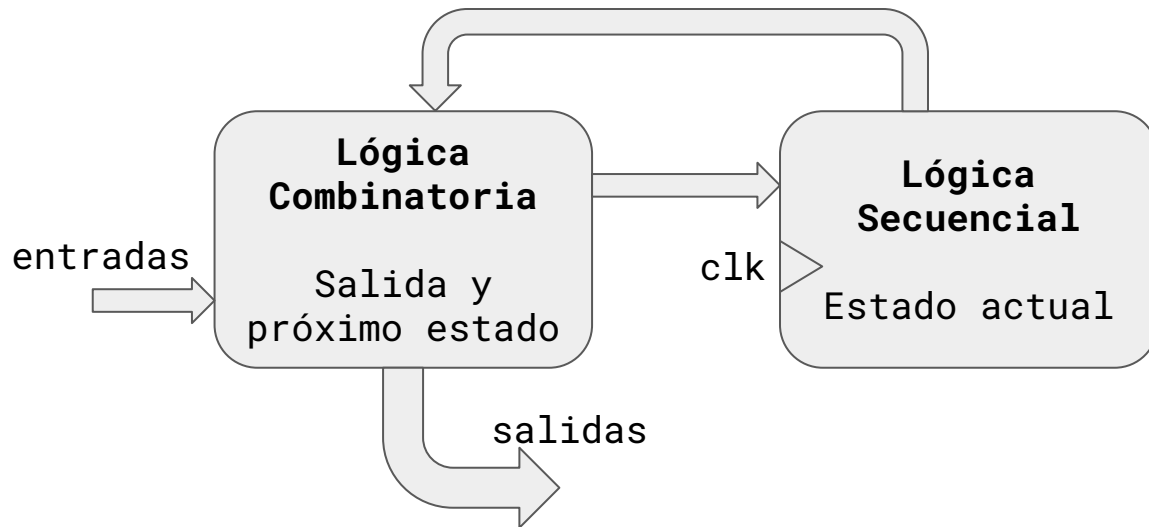


Inicio -> Ingresa Dinero -> Fin -> Reset -> Inicio...

Máquina de Moore

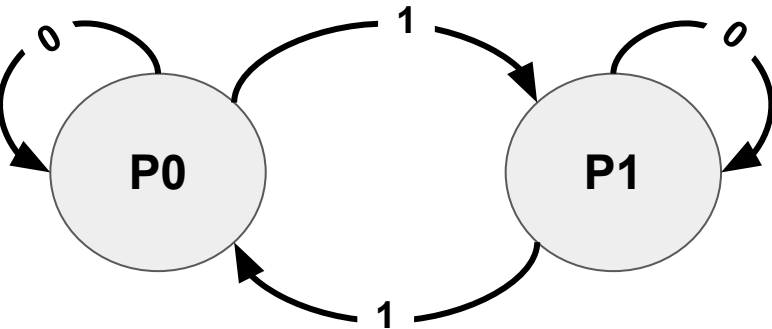


Máquina de Mealy

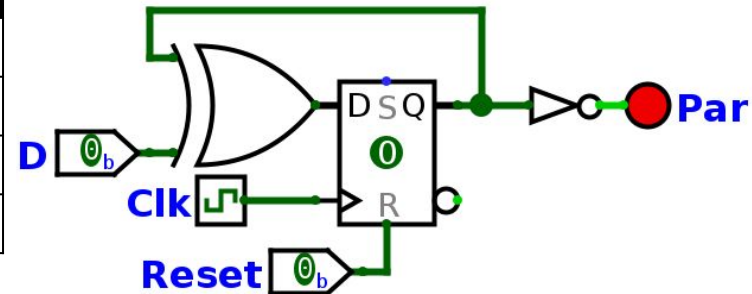


FSM sin estado final

Pensemos una máquina donde recibimos un número D de un bit (1 o 0). Queremos saber si la cantidad de unos que ingresan son par o impar. Ej: si ingresa 01101 entonces $Par=0$ ya que hay 3 unos. Si ingresa 0110 $Par=1$ ya que hay 2 unos. En este caso no existe un estado final, mientras sigan ingresando unos o ceros el estado se mantiene actualizado. Vemos que la lógica combinatoria del nuevo estado es simplemente una Xor. La salida es simplemente el estado. Si deseamos invertir la salida para indicar lo contrario podemos tomar $!Q$ (equivalente a poner una NOT en Q).

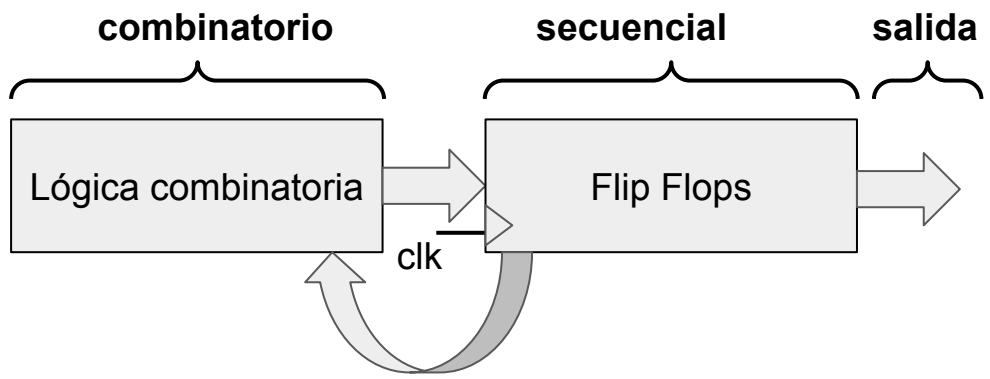
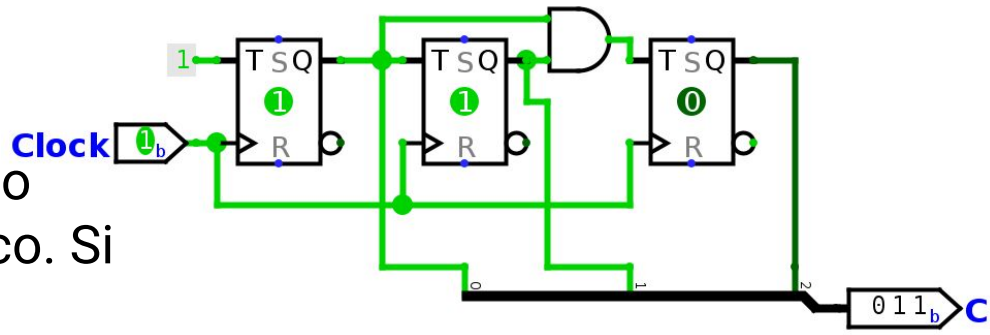


P	D	P'
0	0	0
0	1	1
1	0	1
1	1	0



Contador síncrono

En la unidad anterior aprendimos cómo funciona un contador binario síncrono. Si lo pensamos un poco, esto es tranquilamente una FSM. Tenemos todas las partes (circuitos secuenciales y combinatorios).



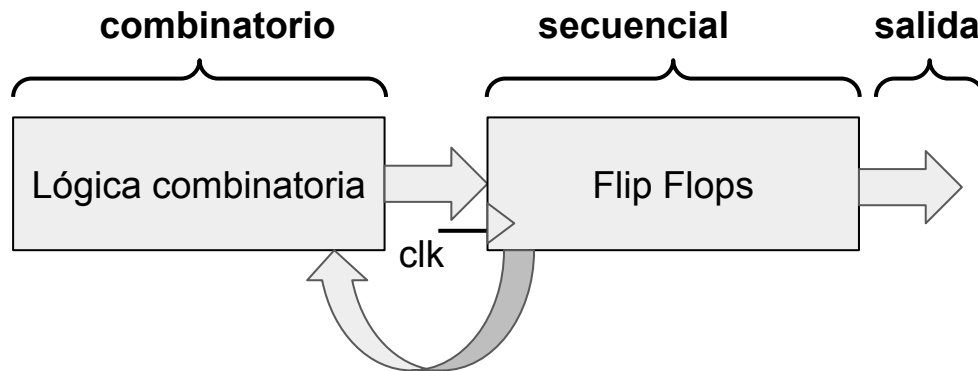
Clk	C ₂	C ₁	C ₀
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

Contador síncrono

Pensemos al estado actual como $C(C_2C_1C_0)$, el estado futuro como $C'(C'_2C'_1C'_0)$, podemos entonces plantear el circuito combinacional para bit de C' .

$$C'_0 = \overline{C_0}$$

Vemos que por inspección visual resolvemos...



Clk	C_2	C_1	C_0	C'_2	C'_1	C'_0
0	0	0	0	0	0	1
1	0	0	1	0	1	0
2	0	1	0	0	1	1
3	0	1	1	1	0	0
4	1	0	0	1	0	1
5	1	0	1	1	1	0
6	1	1	0	1	1	1
7	1	1	1	0	0	0

Contador sincrónico

Luego planteamos C'_1

$$C'_1 = \overline{C_2} \cdot \overline{C_1} \cdot C_0 + \overline{C_2} \cdot C_1 \cdot \overline{C_0} + C_2 \cdot \overline{C_1} \cdot C_0 + C_2 \cdot C_1 \cdot \overline{C_0}$$

$$C'_1 = \overline{C_1} \cdot C_0 + C_1 \cdot \overline{C_0}$$

$$C'_1 = C_1 \text{ xor } C_0$$

Clk	C_2	C_1	C_0	C'_2	C'_1	C'_0
0	0	0	0	0	0	1
1	0	0	1	0	1	0
2	0	1	0	0	1	1
3	0	1	1	1	0	0
4	1	0	0	1	0	1
5	1	0	1	1	1	0
6	1	1	0	1	1	1
7	1	1	1	0	0	0

Contador síncrono

Por último C'_2

$$C'_2 = \overline{C_2} \cdot C_1 \cdot C_0 + C_2 \cdot \overline{C_1} \cdot \overline{C_0} + C_2 \cdot \overline{C_1} \cdot C_0 + C_2 \cdot C_1 \cdot \overline{C_0}$$

$$C'_2 = \overline{C_2} \cdot C_1 \cdot C_0 + C_2 \cdot \overline{C_1} + C_2 \cdot \overline{C_0}$$

$$C'_2 = \overline{C_2} \cdot C_1 \cdot C_0 + C_2 \cdot (\overline{C_1} + \overline{C_0})$$

$$C'_2 = \overline{C_2} \cdot C_1 \cdot C_0 + C_2 \cdot (\overline{C_1 \cdot C_0})$$

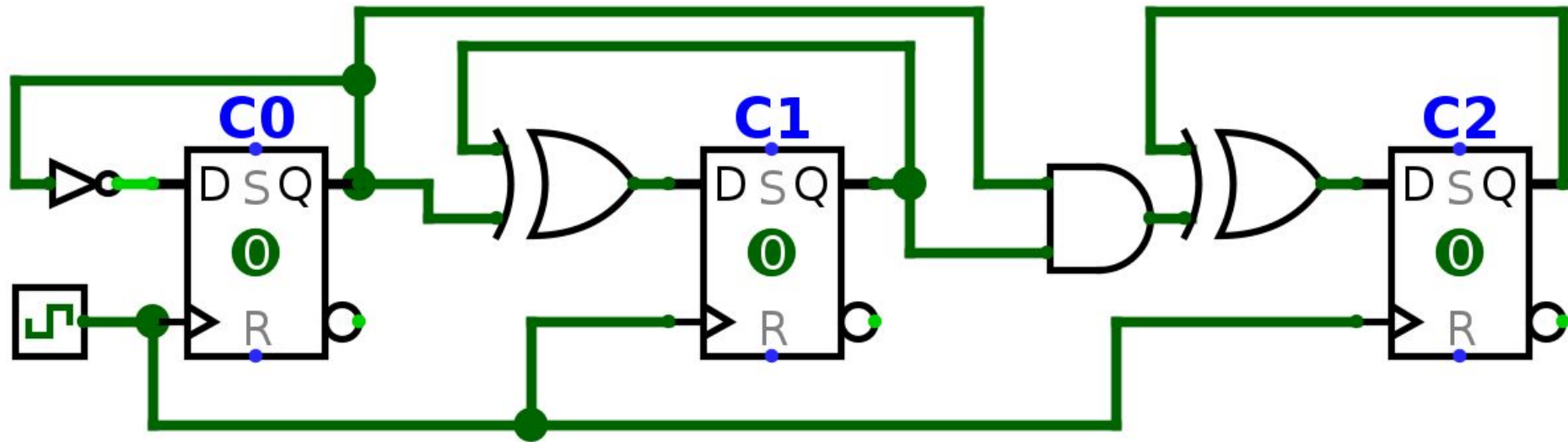
$$C'_2 = \overline{C_2} \cdot (C_1 \cdot C_0) + C_2 \cdot (\overline{C_1 \cdot C_0})$$

$$C'_2 = C_2 \text{ xor } (C_1 \cdot C_0)$$

Clk	C ₂	C ₁	C ₀	C' ₂	C' ₁	C' ₀
0	0	0	0	0	0	1
1	0	0	1	0	1	0
2	0	1	0	0	1	1
3	0	1	1	1	0	0
4	1	0	0	1	0	1
5	1	0	1	1	1	0
6	1	1	0	1	1	1
7	1	1	1	0	0	0

Contador síncrono

Todo junto... $C'_0 = \overline{C_0}$ $C'_1 = C_1 \text{ xor } C_0$ $C'_2 = C_2 \text{ xor } (C_1 \cdot C_0)$



Semáforo

Fases

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.



Verde en la fase de la avenida ex Kennedy

Vemos que en algunos momentos F1 está en verde y F2 esta en rojo. Podemos asumir por ejemplo tiempo de verde en F1 de 7 segundos (utilizamos números pequeños para simplificar la lógica, pero el diseño es escalable a cualquier número).

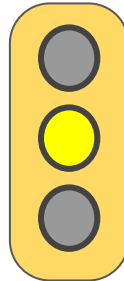


Amarillo en la fase de la avenida ex Kennedy

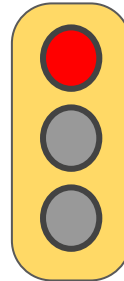
Luego F1 pasa a amarillo (mientras F2 sigue en rojo) durante un tiempo (ej: 4 segundos).



F1



F2



Rojo en la fase de la avenida ex Kennedy

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 5 segundos.

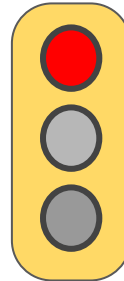


Amarillo en Florencio Varela

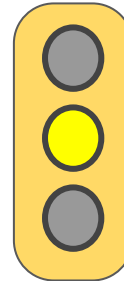
Pasado el tiempo de verde de F2 ahora tenemos amarillo en F2 y esperamos por ejemplo 3 segundos. El siguiente paso es repetir la secuencia volviendo a F1 en verde y F2 en rojo.



F1

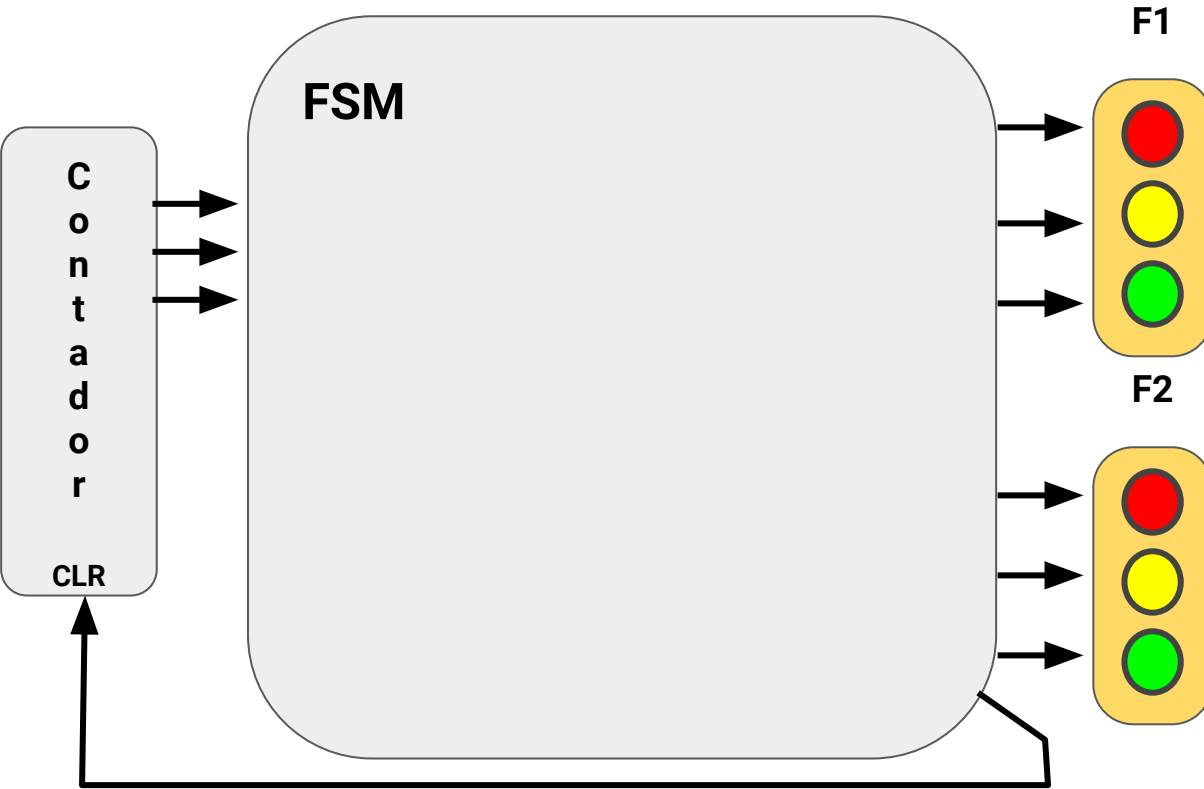


F2



FSM que controle este semáforo

Vamos a usar con contador binario de entrada el cual cuenta distintos tiempos según el estado, por ende cuando cambiemos de estado lo vamos a resetear (salida CLR). Luego vamos a controlar cada una de las lámparas de fase.



Modelo de Mealy

El mayor valor de cuenta es 7, así que con 3 bits de contador binario alcanza. Cada vez que el contador cambie, la entrada cambia y se calcula el nuevo estado y los valores de salida. Vamos a utilizar contadores con reset sincrónico (usando módulo $N+1$).

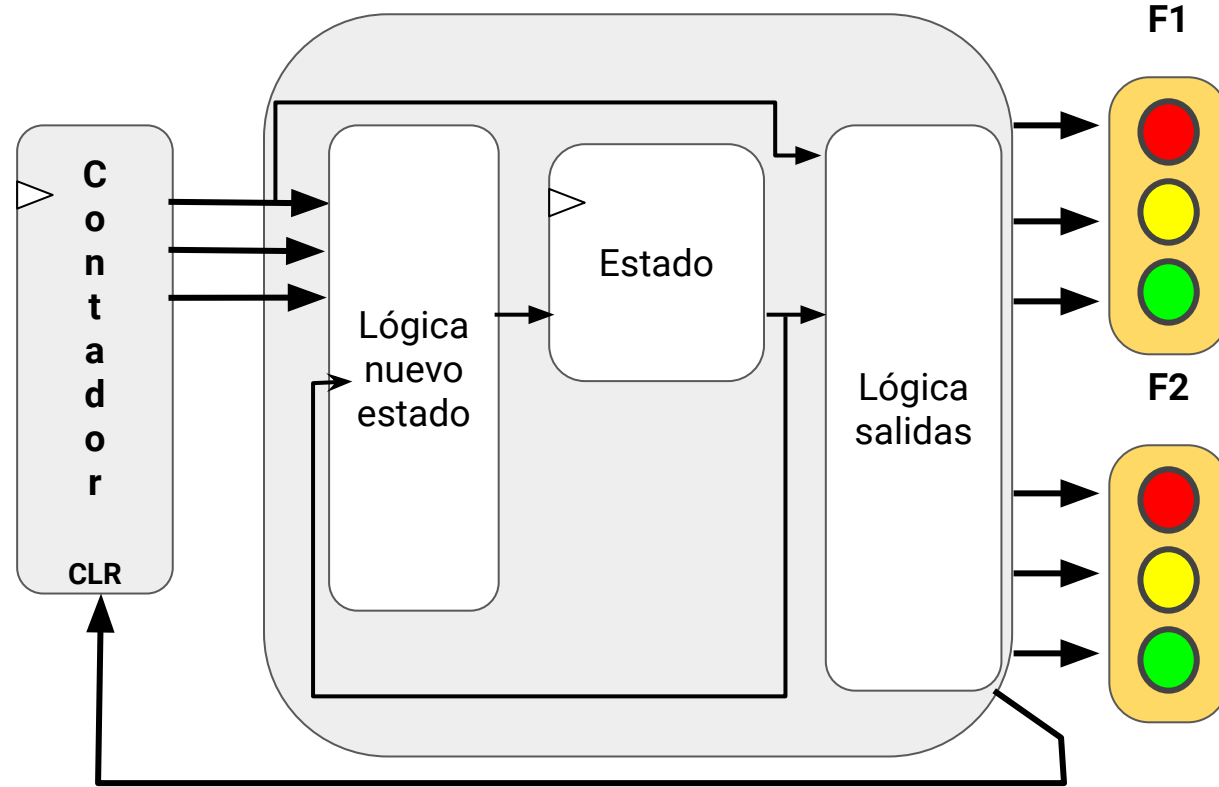
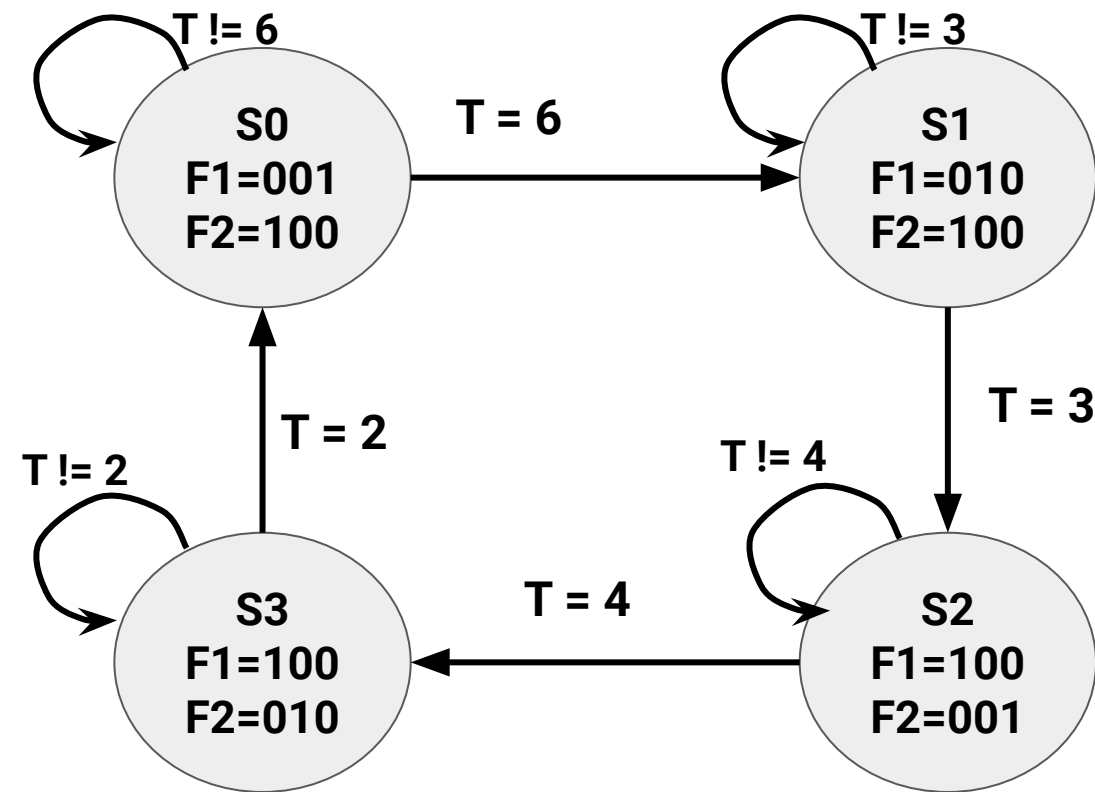


Diagrama de transición de estados

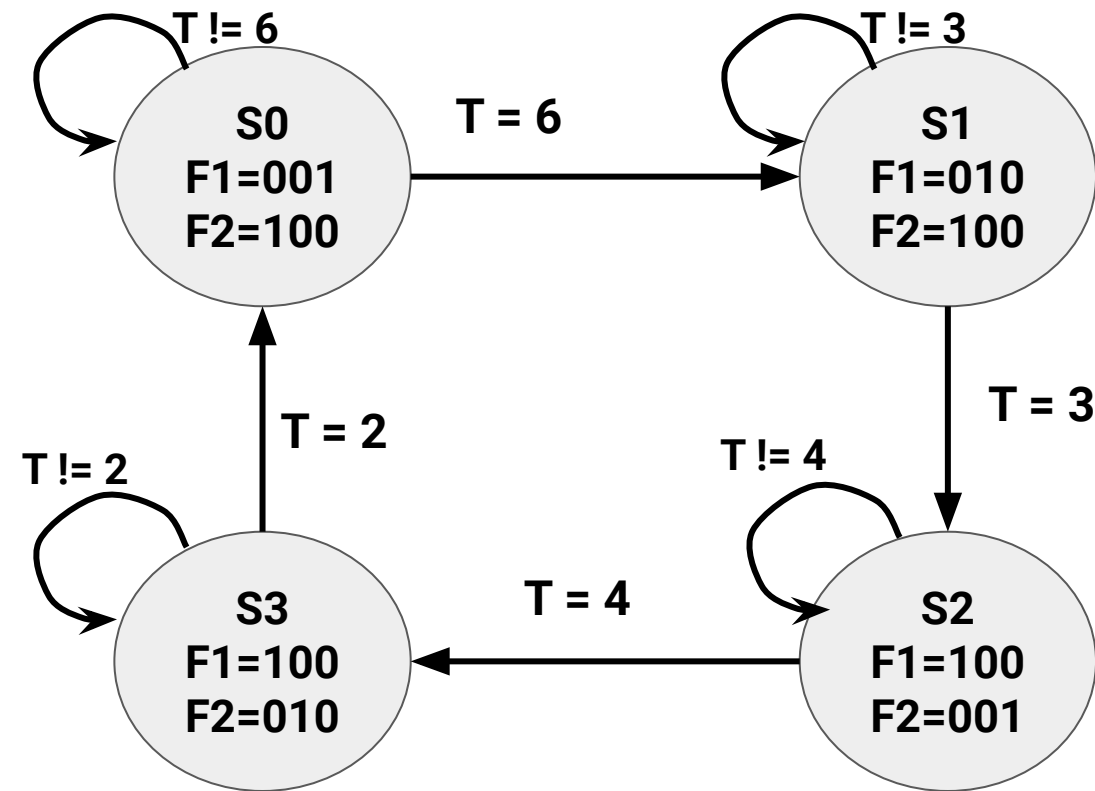
Utilizamos 4 estados. Los valores de Fx son RAV, ej: F1=001 es Verde encendido solamente. El valor de T es la entrada (proveniente del contador binario).



Si bien queremos por ejemplo que se mantenga en el estado S0 durante 7 segundos, vamos a usar contadores con reset sincrónico, por ende 7 segundos son la cuenta desde 0 hasta 6.

Lógica de nuevo estado

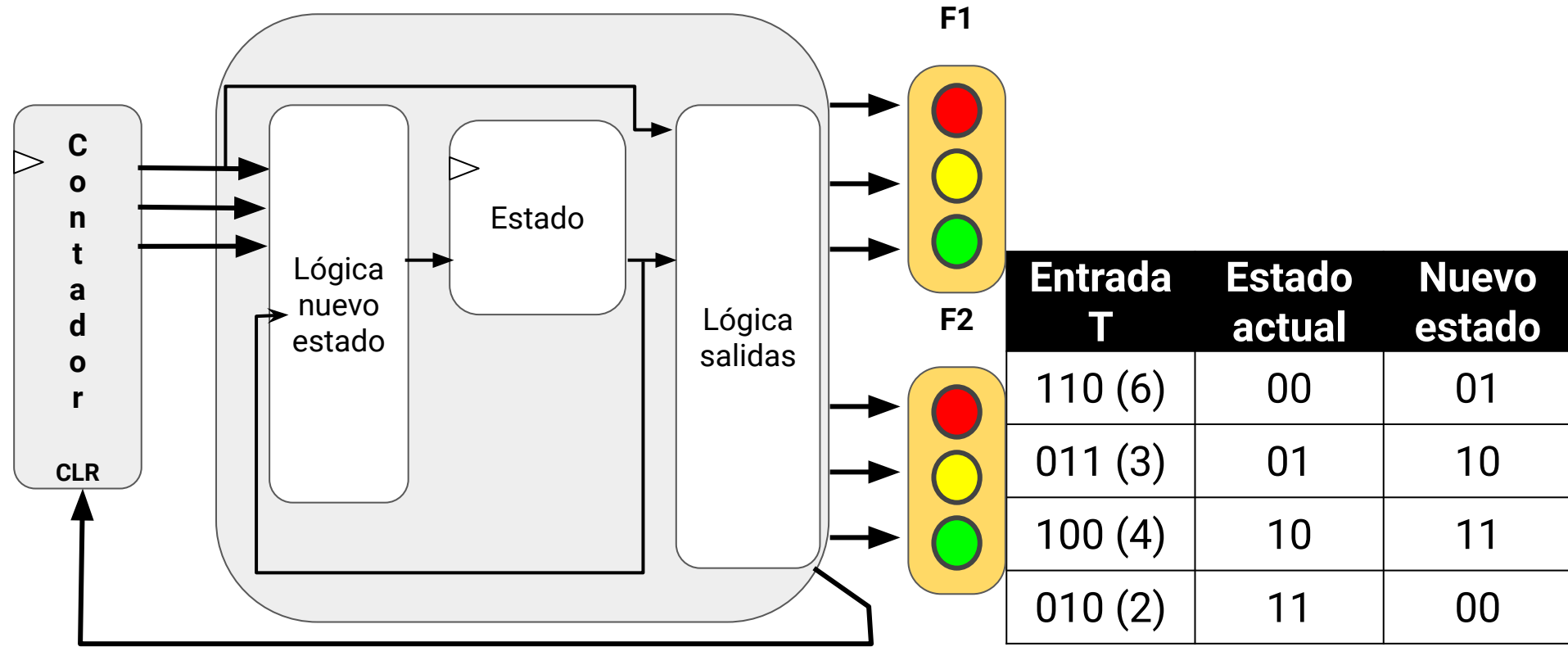
Utilizamos 4 estados. Los valores de Fx son RAV, ej: F1=001 es Verde encendido solamente. El valor de T es la entrada (proveniente del contador binario).



Entrada T	Estado actual	Nuevo estado
110 (6)	00	01
011 (3)	01	10
100 (4)	10	11
010 (2)	11	00

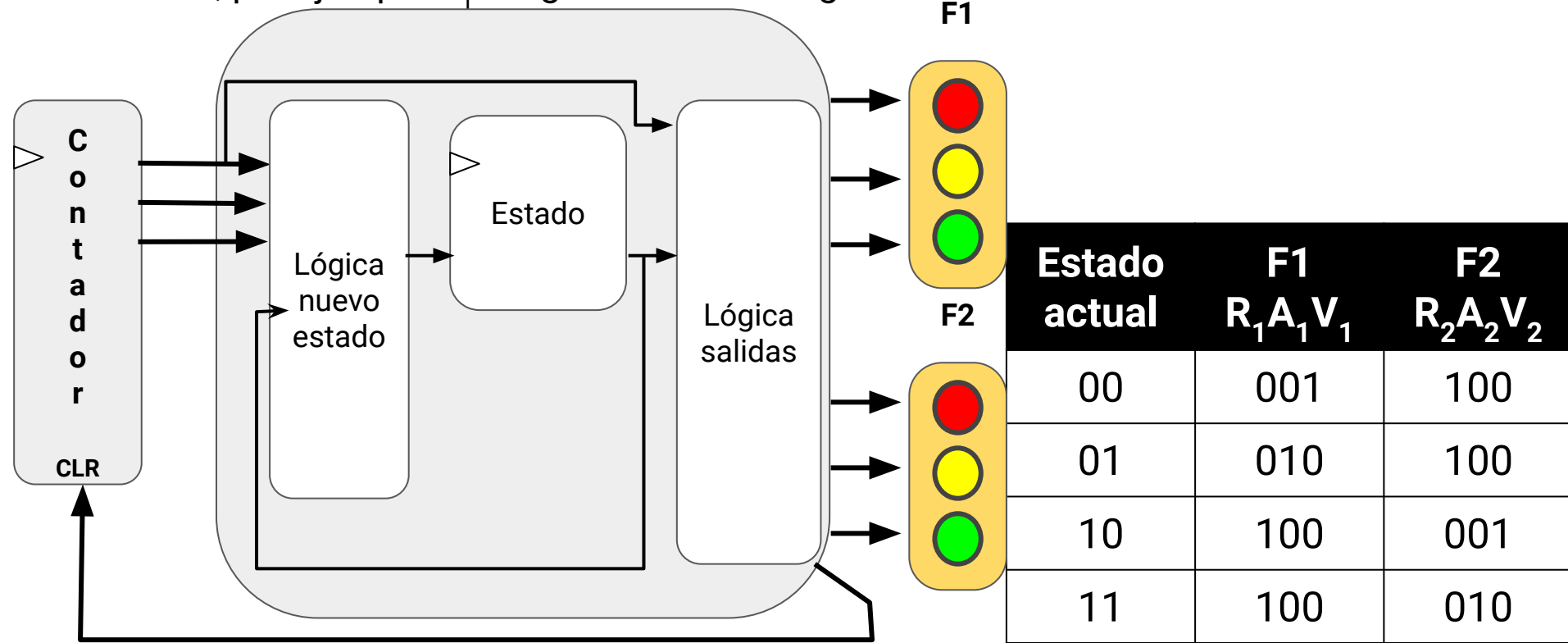
Circuito combinacional

Este diagrama de transición (convertido a una tabla de verdad) se implementa como un circuito combinatorio.



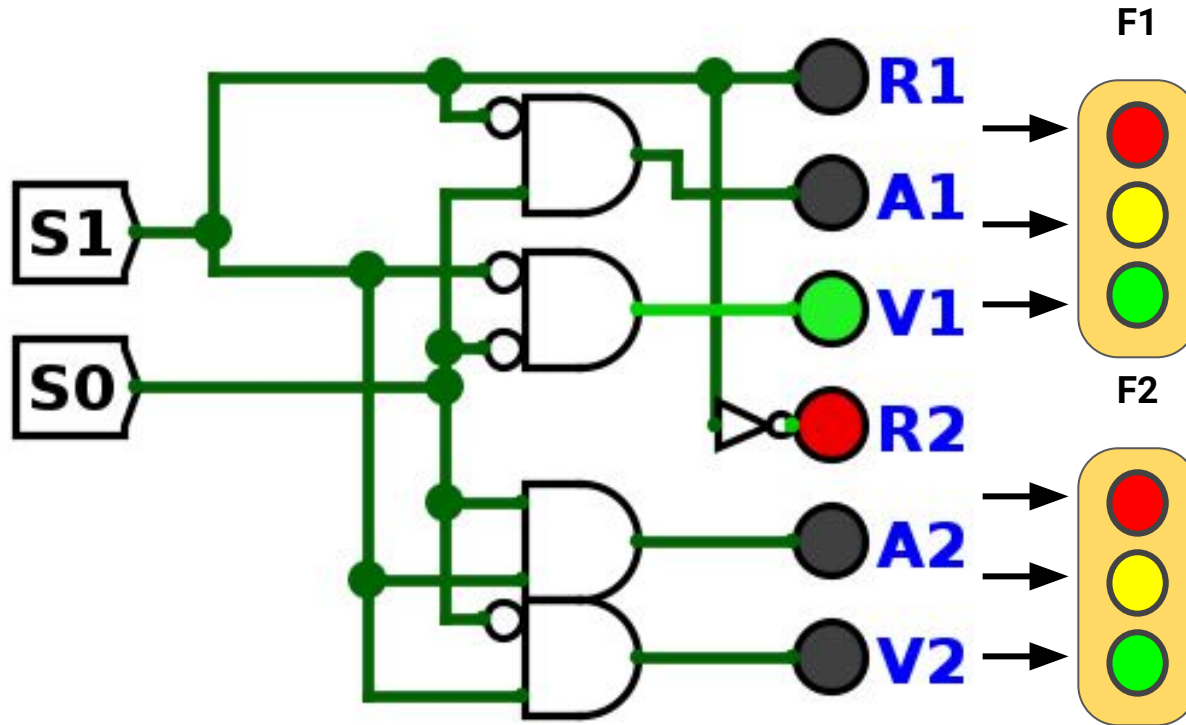
Lógica de control de las lámparas

Para las lámparas, podemos ignorar la entrada y simplemente usar el estado. Tenemos entonces 6 circuitos que dependen del estado solamente ($R_1 A_1 V_1 R_2 A_2 V_2$). Algunos son elementales, por ejemplo R_1 es igual al bit más significativo de estado.



Lógica de control de las lámparas

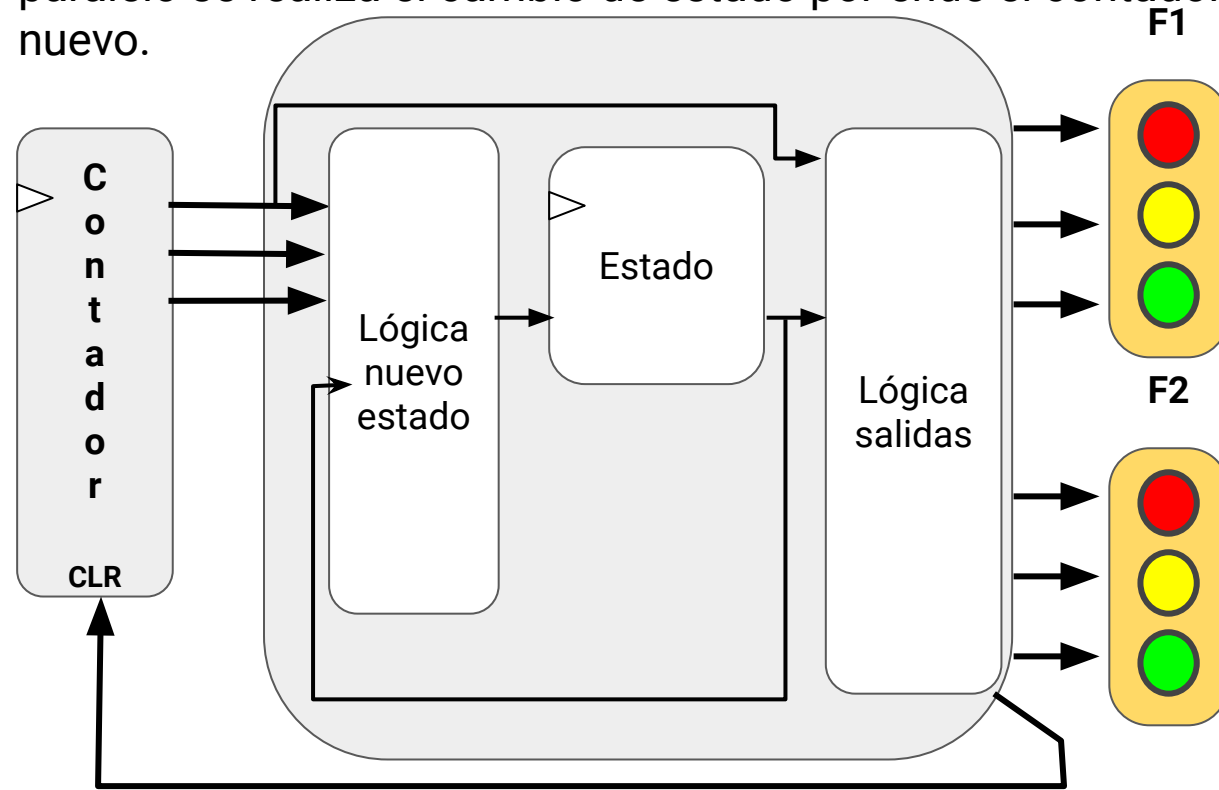
Termina siendo un circuito bastante trivial.



S_1	S_0	F1			F2		
		R_1	A_1	V_1	R_2	A_2	V_2
0	0	0	0	1	1	0	0
0	1	0	1	0	1	0	0
1	0	1	0	0	0	0	1
1	1	1	0	0	0	1	0

Lógica de reset del contador

El contador se resetea en cada cambio de estado, por ende, cuando la entrada fuerza el cambio de estado (ej: llegamos a 6 y estamos en el estado 00) el contador se resetea. En paralelo se realiza el cambio de estado por ende el contador comienza de cero en el estado nuevo.

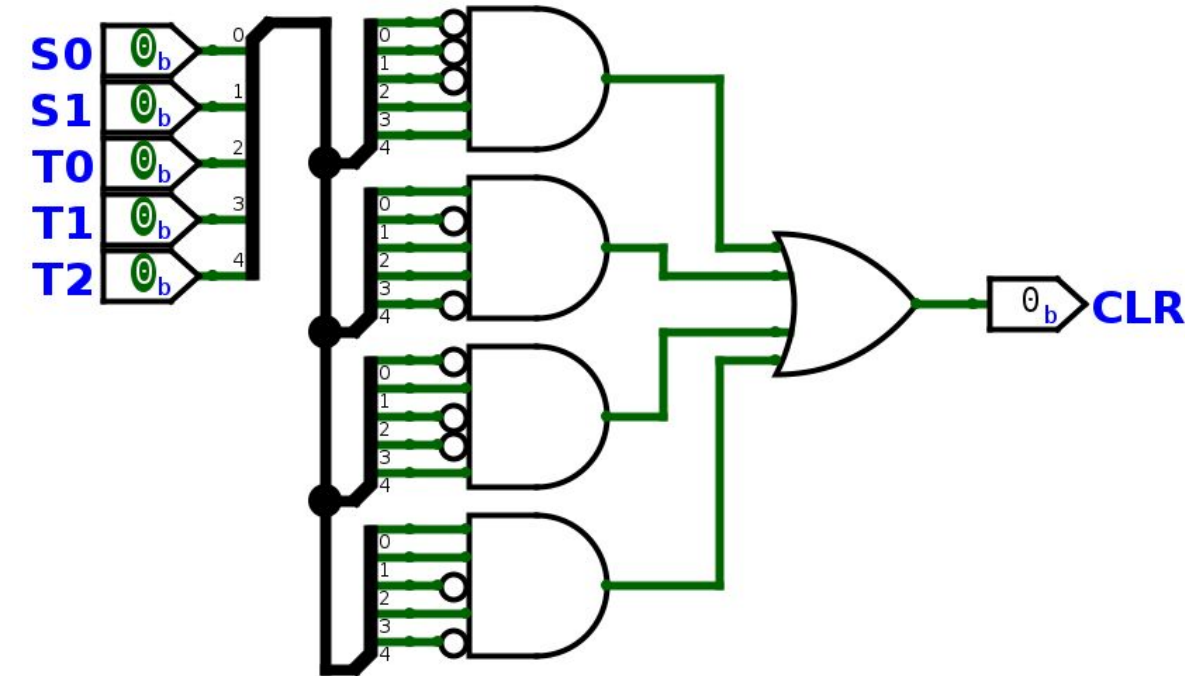


En cualquier otro caso la salida CLR se mantiene en cero. Representamos en la tabla solo los unos.

Entrada T	Estado actual	CLR
110 (6)	00	1
011 (3)	01	1
100 (4)	10	1
010 (2)	11	1

Lógica de reset del contador

Con lógica de dos niveles y suma de productos fácilmente generamos los unos necesarios para resetear el contador en cada transición de estados. Notemos que esta tabla de verdad está incompleta, ya que existen 5 variables de entrada (32 combinaciones) pero solo se representan estas 4. Al usar minitérminos solo representamos los unos, el resto toma cero.

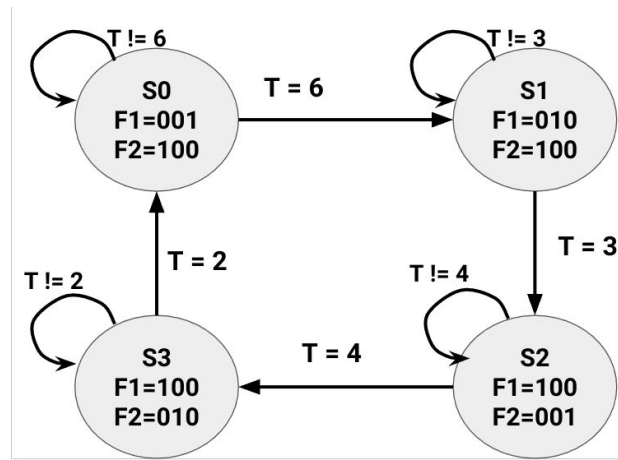


Entrada T	Estado actual	CLR
110 (6)	00	1
011 (3)	01	1
100 (4)	10	1
010 (2)	11	1

Lógica de nuevo estado

Entrada T	Estado actual	Nuevo estado
110 (6)	00	01
011 (3)	01	10
100 (4)	10	11
010 (2)	11	00

Esta tabla de nuevo estado también está incompleta. Existen casos no representados. Ej: Si la entrada es 001 y el estado actual es 01.. no existe valor en esta tabla (debería ser 01).



Lógica de nuevo estado

Entrada T	Estado actual	Nuevo estado
110 (6)	00	01
011 (3)	01	10
100 (4)	10	11
010 (2)	11	00
001(1)	01	01
.	.	.
.	.	.
.	.	.
111	11	XX

Esta tabla de nuevo estado también está incompleta. Existen casos no representados. Ej: Si la entrada es 001 y el estado actual es 01.. no existe valor en esta tabla (debería ser 01). Deberíamos representar las 32 combinaciones de entrada, y en función a eso generar el circuito correspondiente para cada valor de nuevo estado.

Vemos que existen estados que no deberían ocurrir, como llegar al estado 11 y que la entrada sea 111.

Prestemos atención en la entrada 001 para el estado 01. En este caso, estamos en 01 (Amarillo en F1) y debemos esperar hasta que la entrada sea 011 (3). Mientras estemos en un valor menor a 3, debemos **mantener el estado**.

Lógica de nuevo estado

Entrada T	Estado actual	Nuevo estado
110 (6)	00	01
011 (3)	01	10
100 (4)	10	11
010 (2)	11	00
001(1)	01	mantener
.	.	.
.	.	.
.	.	.
111	11	XX

Esta tabla de nuevo estado también está incompleta. Existen casos no representados. Ej: Si la entrada es 001 y el estado actual es 01.. no existe valor en esta tabla (debería ser 01). Deberíamos representar las 32 combinaciones de entrada, y en función a eso generar el circuito correspondiente para cada valor de nuevo estado.

Vemos que existen estados que no deberían ocurrir, como llegar al estado 11 y que la entrada sea 111.

Prestemos atención en la entrada 001 para el estado 01. En este caso, estamos en 01 (Amarillo en F1) y debemos esperar hasta que la entrada sea 011 (3). Mientras estemos en un valor menor a 3, debemos **mantener el estado**.

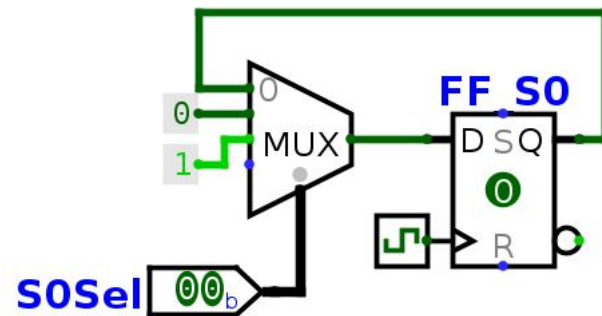
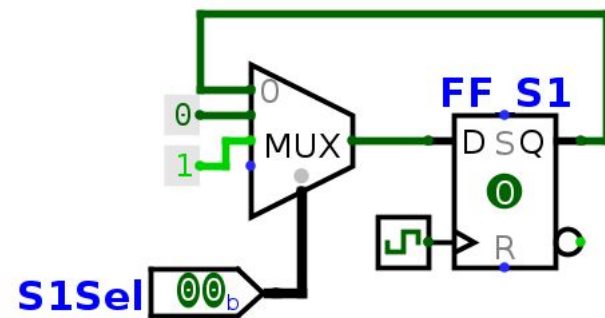
Lógica de nuevo estado

Entrada T	Estado actual	Nuevo estado
110 (6)	00	01
011 (3)	01	10
100 (4)	10	11
010 (2)	11	00
001(1)	01	mantener
.	.	mantener
.	.	mantener
.	.	mantener
111	11	mantener

Vamos entonces a tratar de mantener el estado para todos estos valores que no están definidos explícitamente en la tabla. Realimentamos cada FF mediante un MUX. De esta forma siempre que Sel sea 00 mantiene.

En el caso que Sel sea 01, entonces se fuerza un 0, y si Sel es 10, se fuerza 1.

Pero ahora nuestra tabla de verdad ya no define el nuevo estado con los bits de S1 y S0, sino que con los valores de Sel de cada FF...

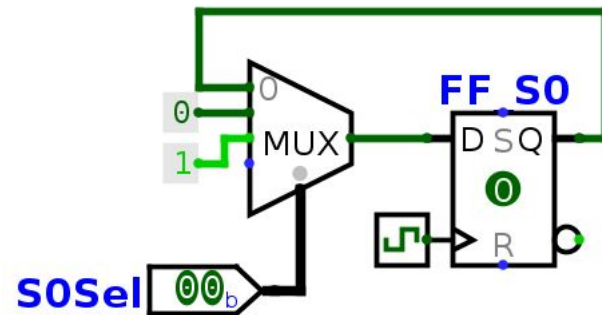
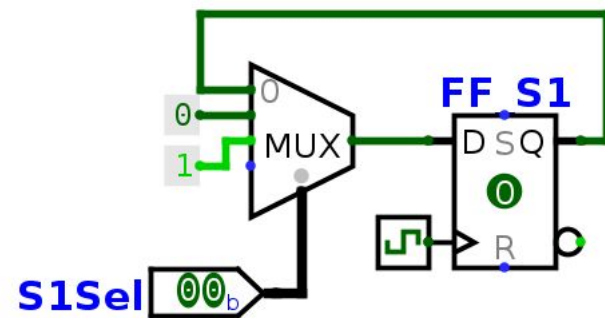


Lógica de nuevo estado

Entrada T	Estado actual	Nuevo estado
110 (6)	00	01
011 (3)	01	10
100 (4)	10	11
010 (2)	11	00
001(1)	01	mantener
.	.	mantener
.	.	mantener
.	.	mantener
111	11	mantener

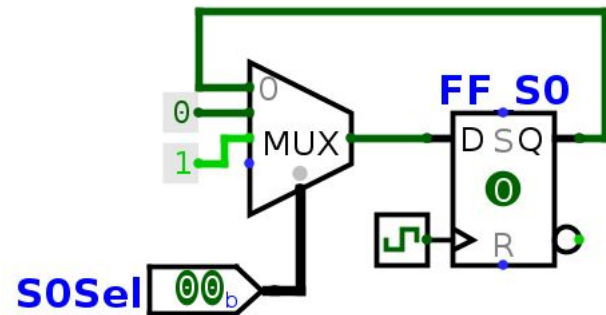
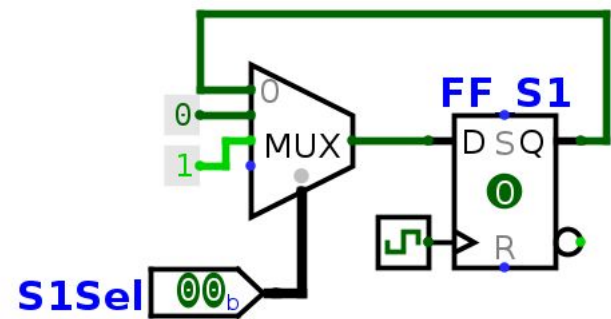
Los 4 valores originales donde se cambiaba el estado ahora van a pasar a ser representados cada uno con 2 bits. Por ende si había un cero, ahora pasa a ser 01, y si había un uno, ahora pasa a ser 10.

En todos los estados donde se mantiene, el valor va a ser 00.



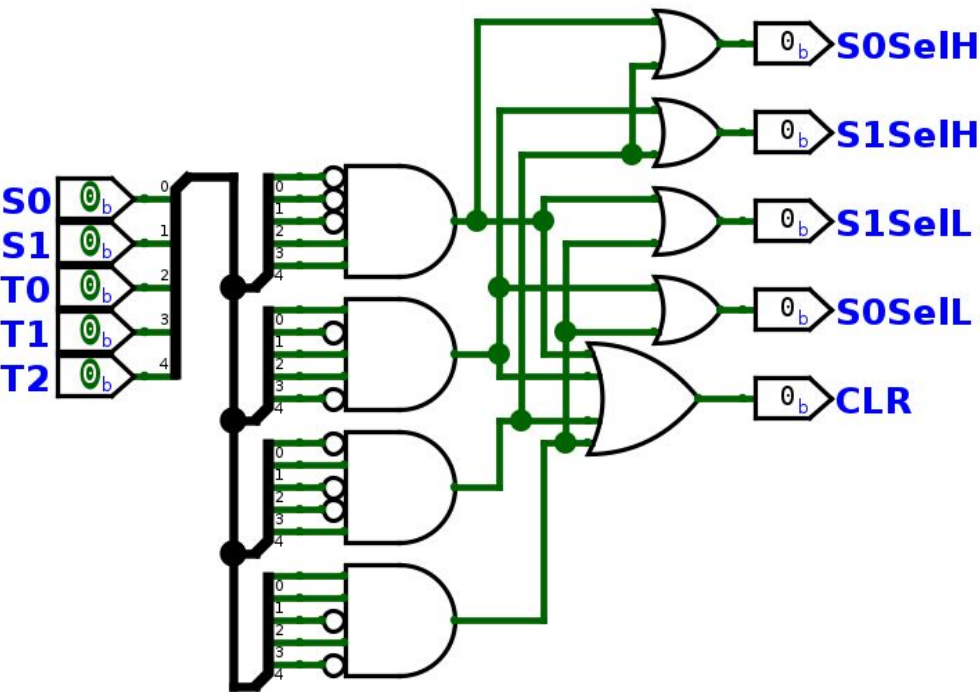
Lógica de nuevo estado

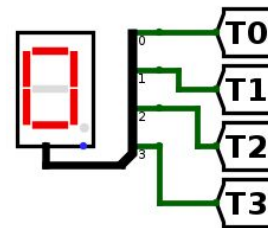
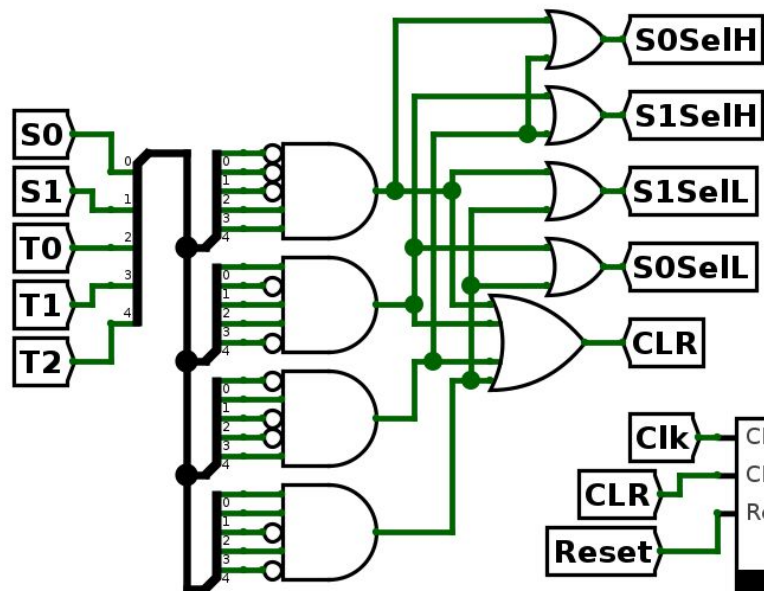
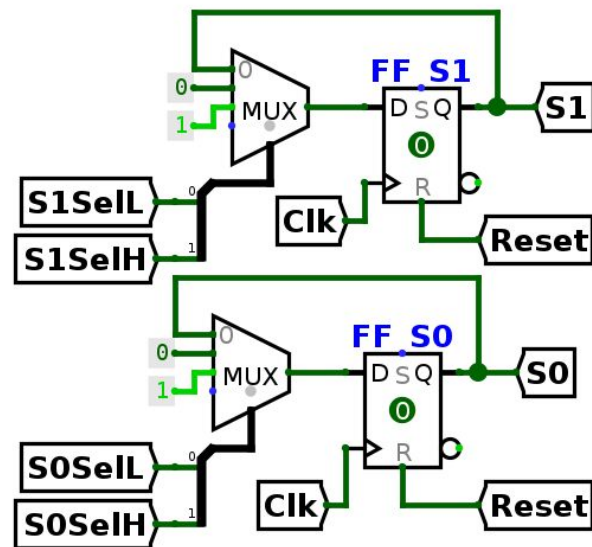
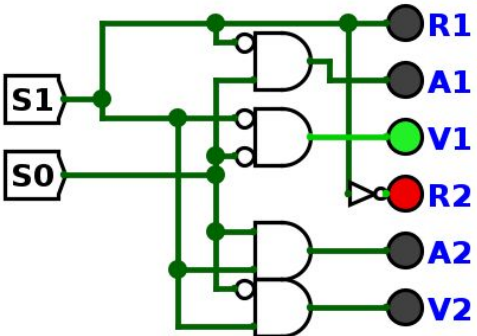
Entrada T	Estado actual	Nuevo Estado	S1sel h l	S0sel h l
110 (6)	00	01	01	10
011 (3)	01	10	10	01
100 (4)	10	11	10	10
010 (2)	11	00	01	01
001(1)	01	mantener	00	00
.	.	mantener	00	00
.	.	mantener	00	00
.	.	mantener	00	00
111	11	mantener	00	00



Lógica de nuevo estado

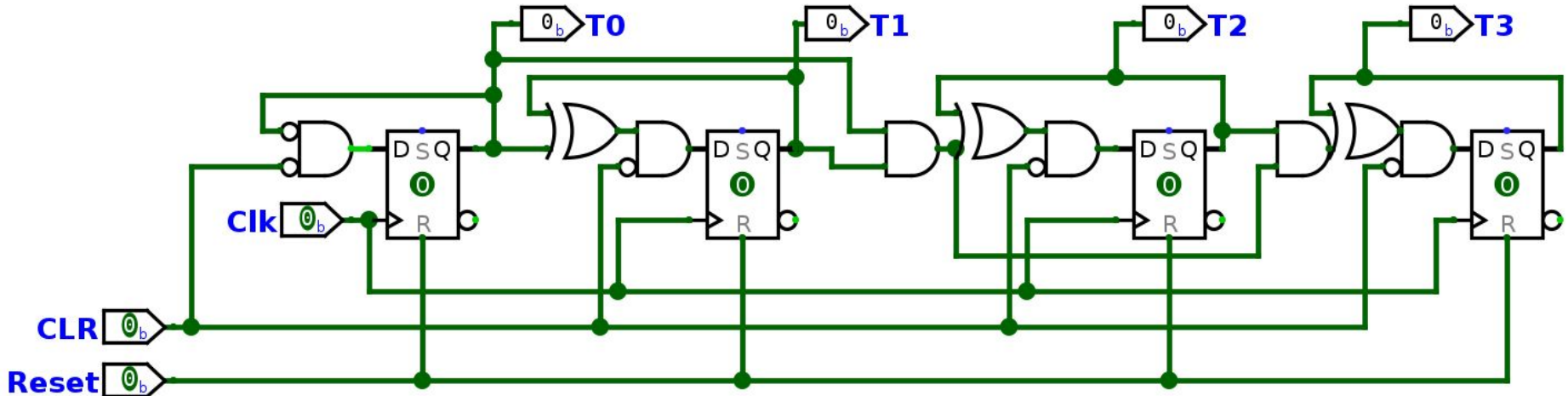
Entrada T	Estado actual	S1sel h l	S0sel h l
110 (6)	00	01	10
011 (3)	01	10	01
100 (4)	10	10	10
010 (2)	11	01	01
001(1)	01	00	00
.	.	00	00
.	.	00	00
.	.	00	00
111	11	00	00





Contador binario síncrono con clear síncrono y reset

Se utiliza como componente un contador binario síncrono (ya que todos los FF tienen el mismo clock), con reset asincrónico (ya que la entrada de Reset en 1 fuerza todo a 0000 sin depender del clock) y con entrada de clear síncrono, ya que cuando llega el siguiente clock, si la entrada de clear está en uno, los FF toman como valor cero.



Ejemplo

Teclado de caja fuerte

Vamos a diseñar un sistema que tiene un teclado numérico con un encoder que genera el valor de la tecla presionada y un pulso de clock cuando se presiona un tecla. Existe una tecla (DA) que inicia el ingreso de clave. El sistema debe reconocer la clave 6502 ingresada luego de DA. En ese caso debe generar un uno en la cerradura de puerta. El mismo se mantiene hasta el próximo DA. En caso de clave inválida, no genera el uno en la cerradura.

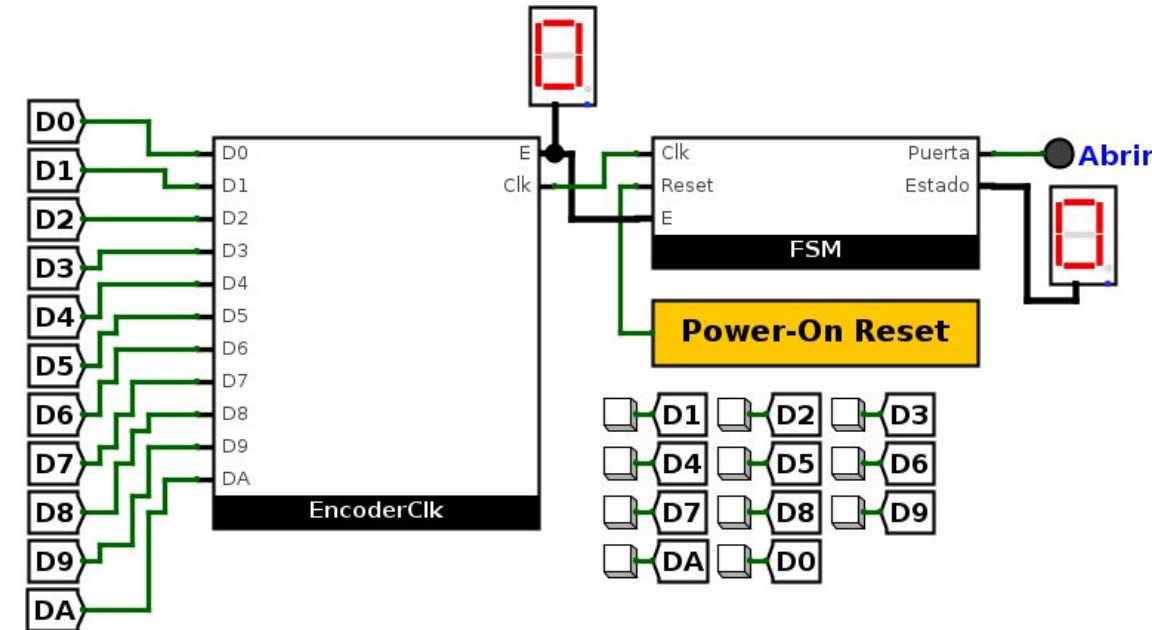


Diagrama de transición de estado

Comenzamos con el diagrama.. El reset nos deja en S0. Luego con A pasamos a S1. De ahí, si ingresa el primer dígito válido (6) pasamos a la rama de arriba. Sino vamos al estado S6 donde esperamos hasta que ingrese otra A.

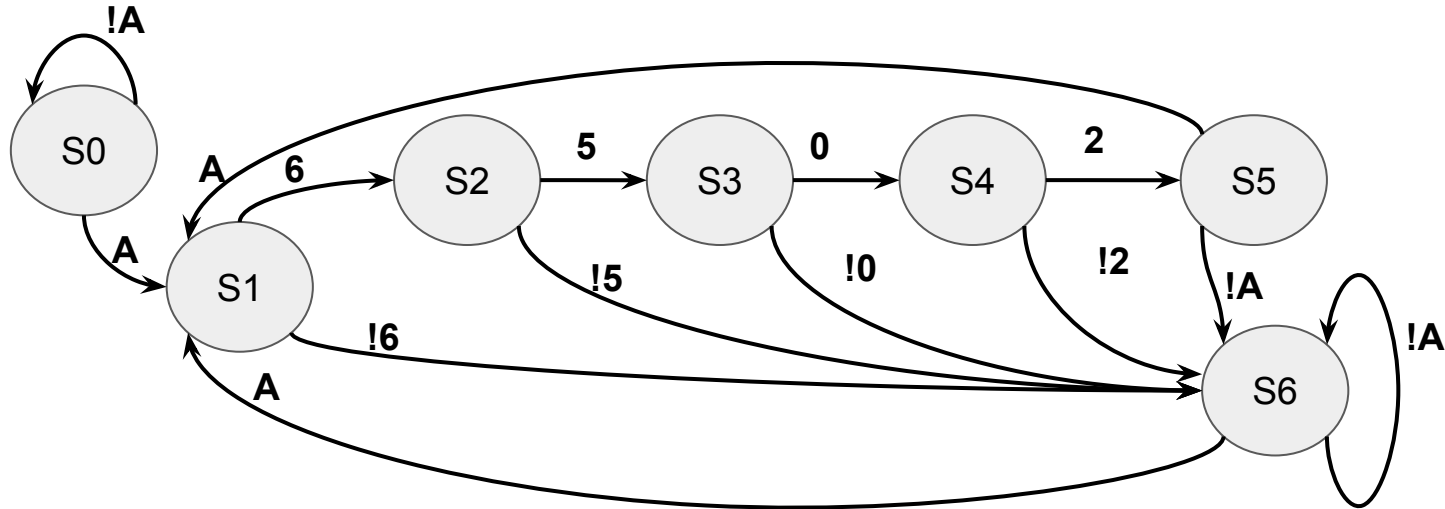
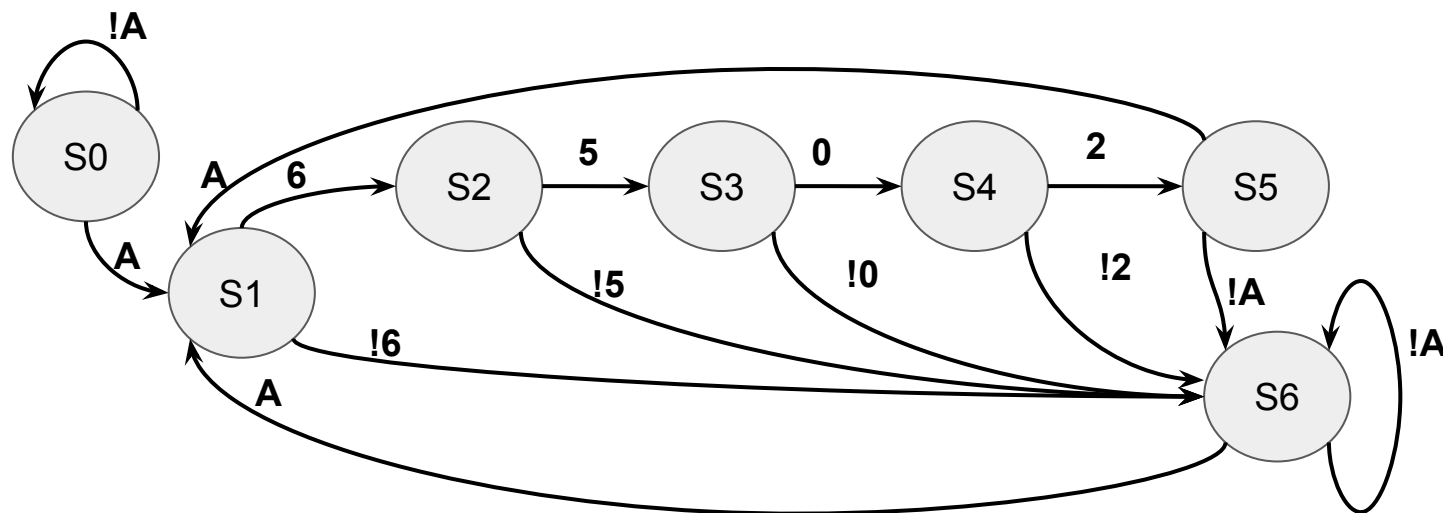


Tabla de verdad

Existen 7 estados (3 bits) y 10 entradas (4 bits) por ende necesitamos una tabla de verdad de 128 combinaciones. Si bien es posible de armar, en la mayoría de los casos la transición es fija. Podemos generar “entradas creativas”...

S	E	S'
S0	0	S0
S0	1	S0
.	.	.
.	.	.
S0	A	S1
.	.	.
.	.	.
.	.	.
S6	9	S6
S6	A	S1



Entradas “virtuales”

Si bien tenemos la entrada E, no nos interesa analizar todos los posibles valores para E. Así que generamos entradas con detectores de ciertas combinaciones que coinciden con los cambios de estado. Ahora podemos cambiar la tabla de verdad con estas nuevas entradas.

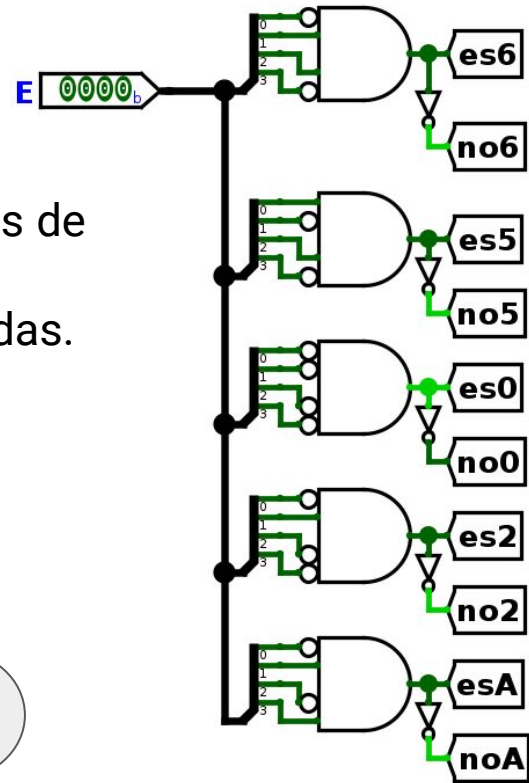
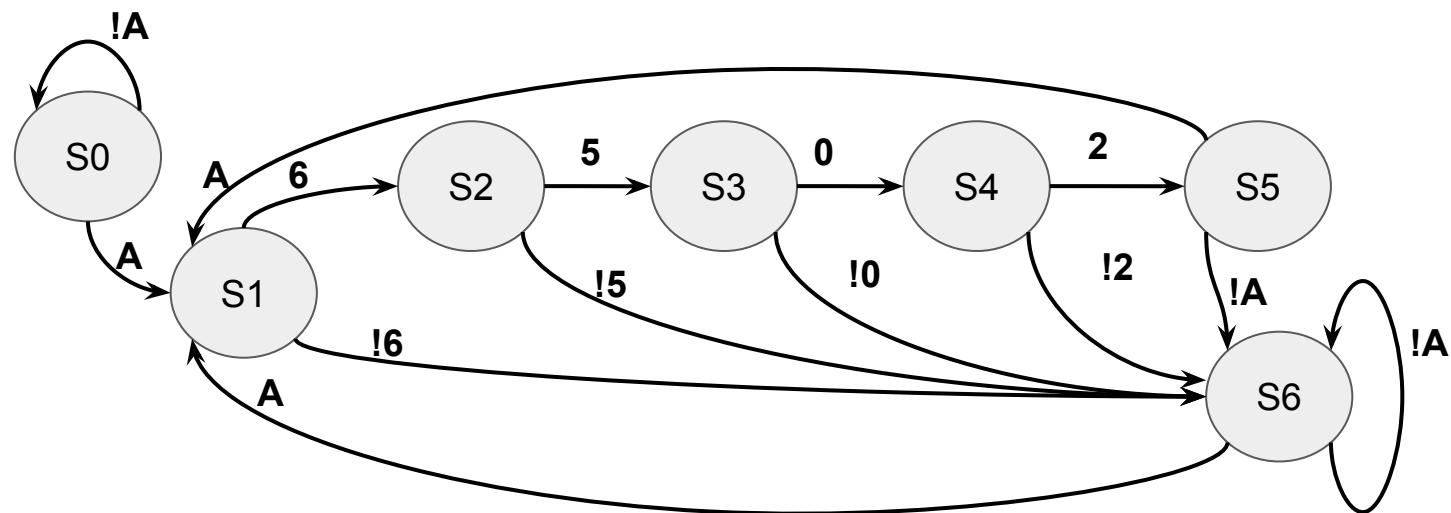


Tabla con entradas virtuales

S	es6	no6	es5	no5	es0	no0	es2	no2	esA	noA	S'
S0	X	X	X	X	X	X	X	X	0	1	S0
S0	X	X	X	X	X	X	X	X	1	0	S1
S1	1	0	X	X	X	X	X	X	X	X	S2
S1	0	1	X	X	X	X	X	X	X	X	S6

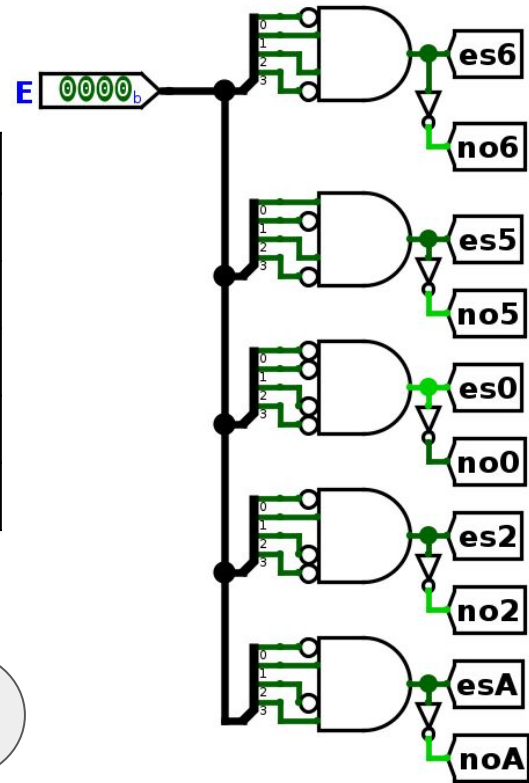
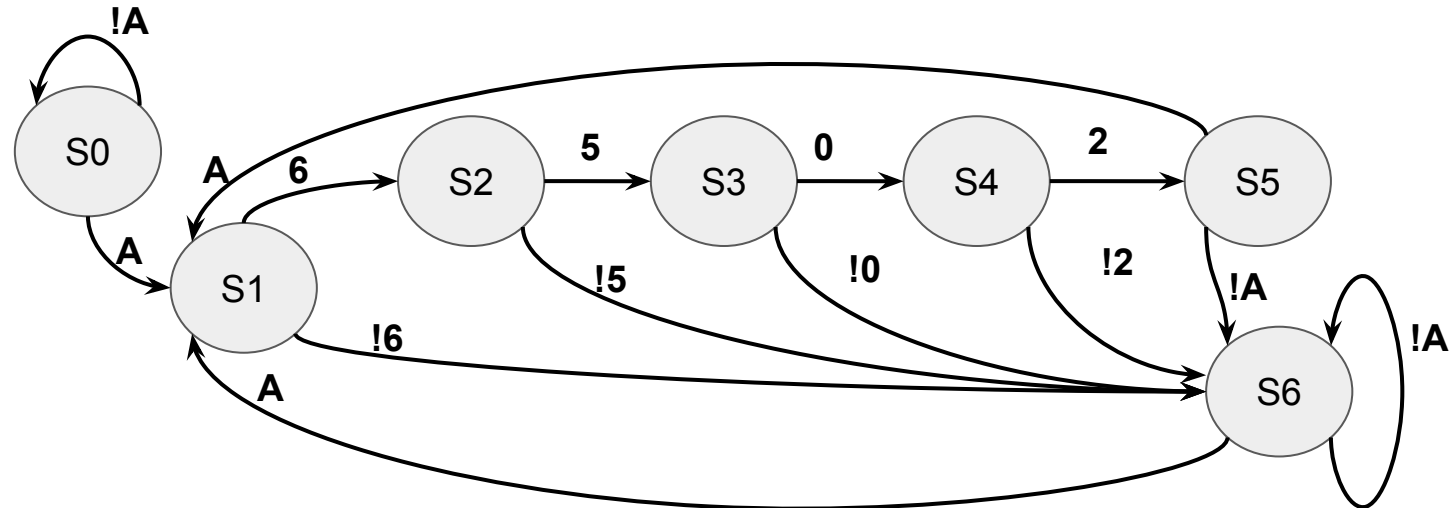
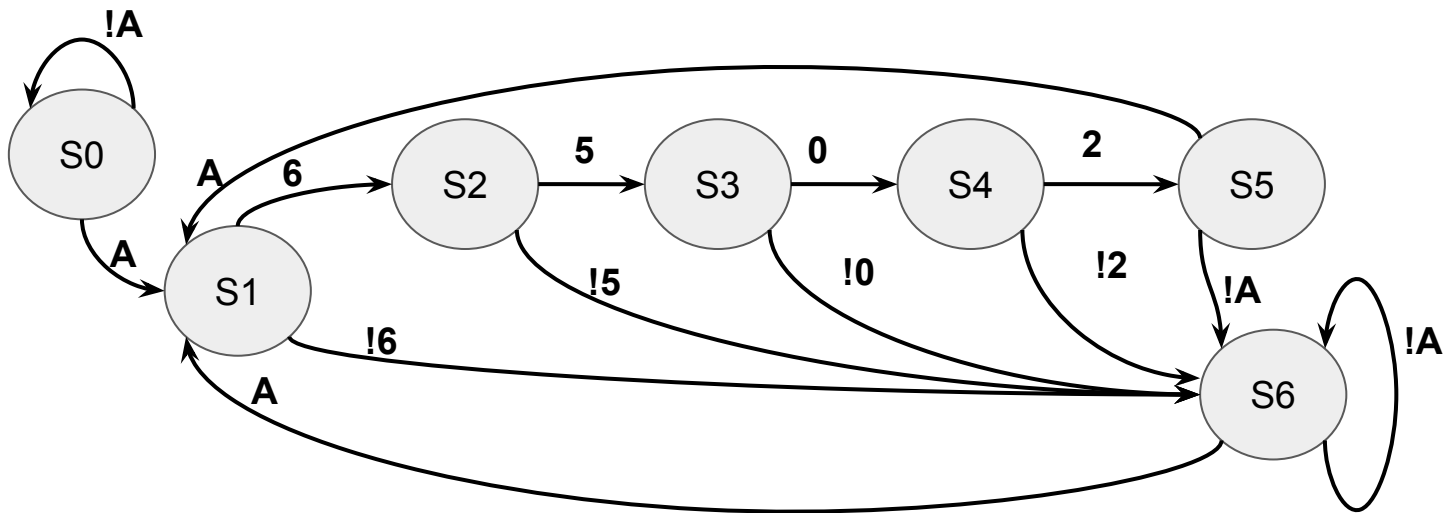


Tabla de verdad resumida

Hacemos una versión resumida entonces. Vemos que por cada flecha del diagrama tenemos una entrada en la tabla.
De esta tabla no podemos directamente escribir la lógica de nuevo estado ya que está incompleta, pero si usamos el método de mantener el estado salvo en las transiciones, entonces solo tenemos que actuar en estos casos en particular.



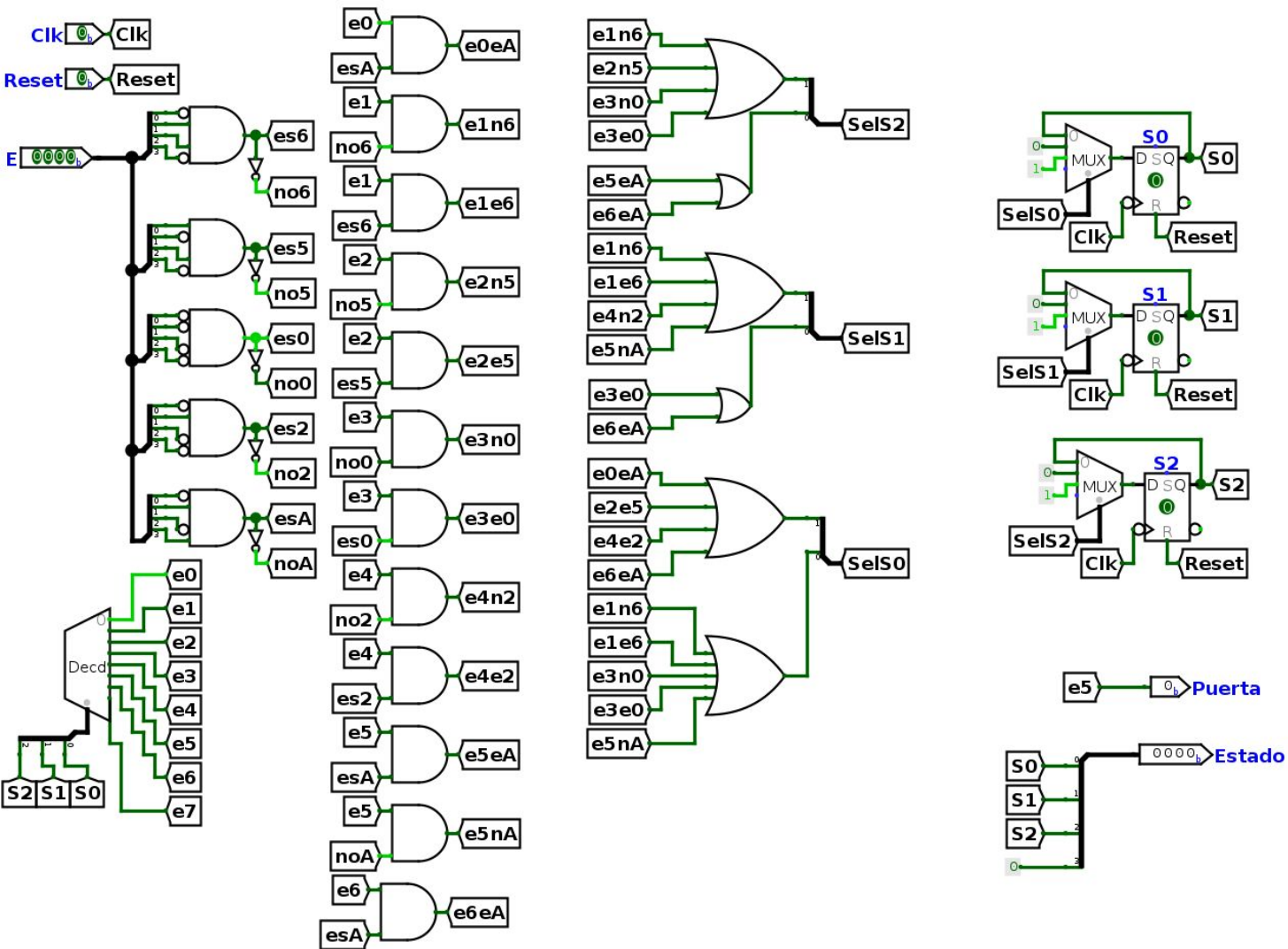
S	E	S'
S0	!A	S0
S0	A	S1
S1	!6	S6
S1	6	S2
S2	!5	S6
S2	5	S3
S3	!0	S6
S3	0	S4
S4	!2	S6
S4	2	S5
S5	A	S1
S5	!A	S6
S6	A	S1
S6	!A	S6

Tabla de verdad resumida

S	S	E	S'	S'	S ₂ hI	S ₁ hI	S ₀ hI
S0	000	!A	S0	000	00	00	00
S0	000	A	S1	001	00	00	10
S1	001	!6	S6	110	10	10	01
S1	001	6	S2	010	00	10	01
S2	010	!5	S6	110	10	00	00
S2	010	5	S3	011	00	00	10
S3	011	!0	S6	110	10	00	01
S3	011	0	S4	100	10	01	01
S4	100	!2	S6	110	00	10	00
S4	100	2	S5	101	00	00	10
S5	101	A	S1	001	01	00	00
S5	101	!A	S6	110	00	10	01
S6	110	A	S1	001	01	01	10
S6	110	!A	S6	110	00	00	00

Tabla de verdad resumida

S	S	E	S'	S'	S ₂ hI	S ₁ hI	S ₀ hI
S0	000	!A	S0	000	00	00	00
S0	000	A	S1	001	00	00	10
S1	001	!6	S6	110	10	10	01
S1	001	6	S2	010	00	10	01
S2	010	!5	S6	110	10	00	00
S2	010	5	S3	011	00	00	10
S3	011	!0	S6	110	10	00	01
S3	011	0	S4	100	10	01	01
S4	100	!2	S6	110	00	10	00
S4	100	2	S5	101	00	00	10
S5	101	A	S1	001	01	00	00
S5	101	!A	S6	110	00	10	01
S6	110	A	S1	001	01	01	10
S6	110	!A	S6	110	00	00	00

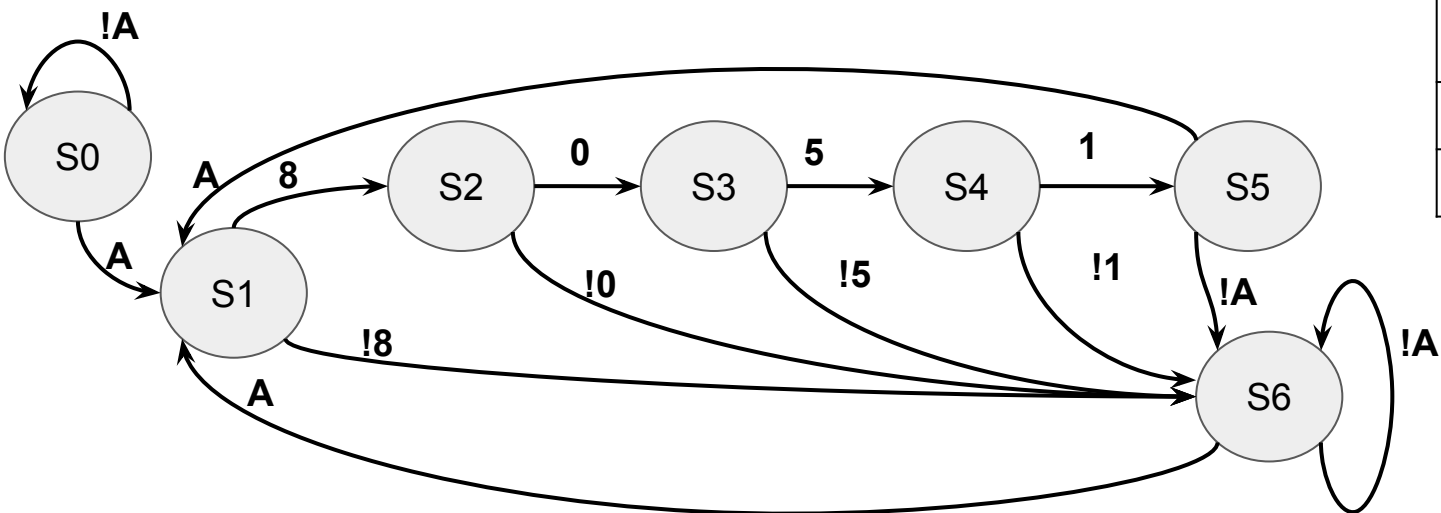


FSM programmable

Cambio de clave

Si ahora tenemos que cambiar la clave (pasa de 6502 a 8051), entonces vemos que tenemos que volver a plantear la tabla de verdad y hacer todo el análisis previo para la implementación. En definitiva por cada tabla de verdad vamos a tener que trabajar para lograr la implementación. Quizás exista una forma más simple...

S	E	S'
S0	0	S0
S0	1	S0
.	.	.
.	.	.
S0	A	S1
.	.	.
.	.	.
.	.	.
S6	9	S6
S6	A	S1



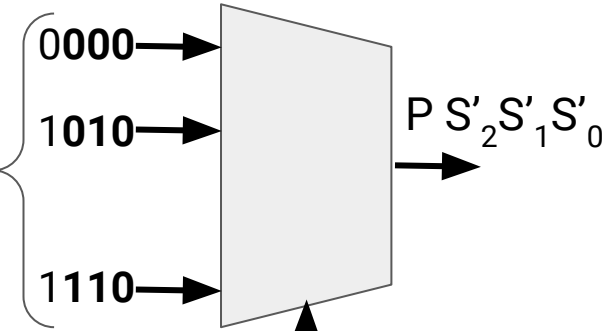
LookUp Table

Pensemos que la función de nuevo estado no es otra cosa más que un circuito combinacional. Sabemos que podemos implementarlos con Multiplexores. En este caso tenemos 1 bit de salida, 3 bits de estado y 4 bits de entrada posible, por ende tenemos 7 bits de selección. Podemos pensar estos 7 bits como una única palabra:

$$S_2 S_1 S_0 E_3 E_2 E_1 E_0$$

De esta forma podríamos definir como si fuese una tabla todas las combinaciones de estado S (cambiando de 000 a 111) para todas las entradas E (cambiando de 0000 a 1111). Los 3 bits menos significativos definen el nuevo estado y el restante controla la salida (Puerta)

Datos de 3 bits que representan el nuevo estado y un cuarto bit para representar la salida



Se selecciona usando parte alta estado actual y parte baja la entrada

Memoria ROM

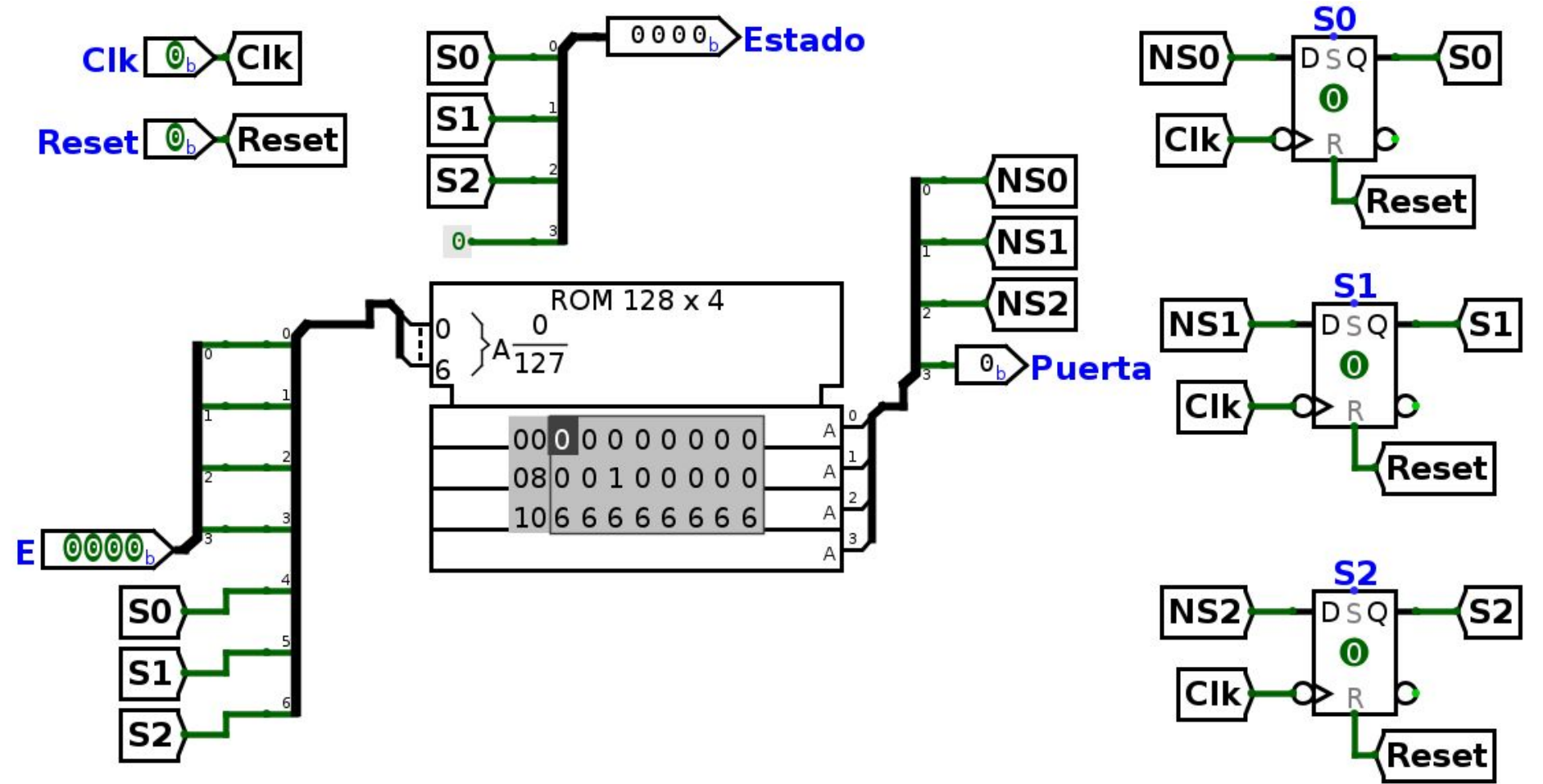
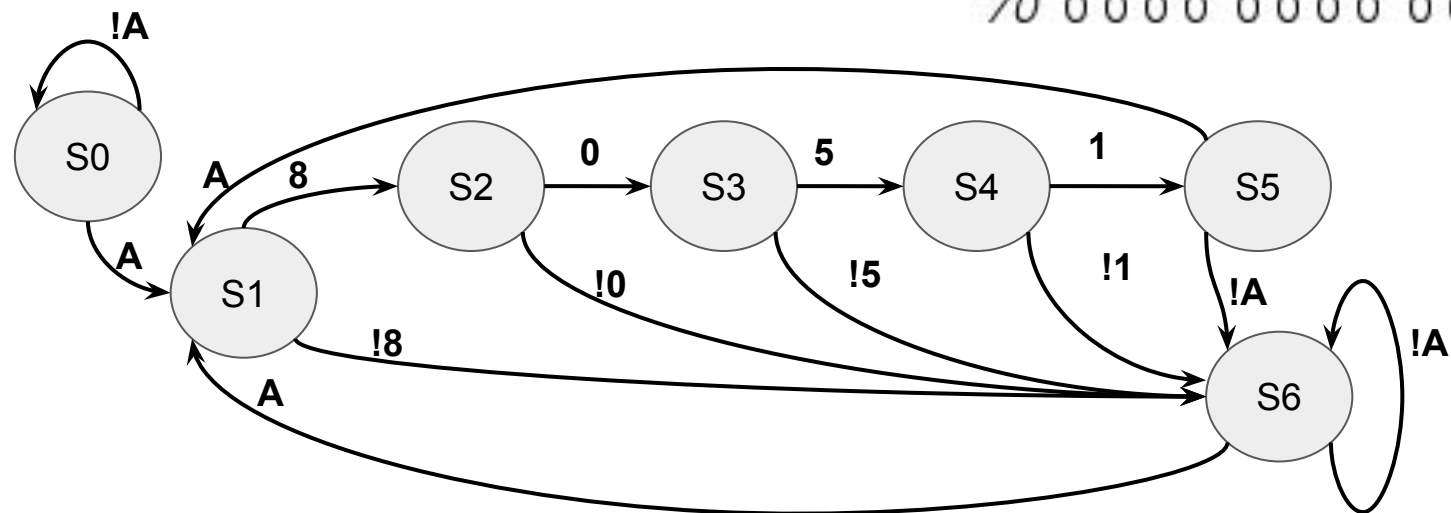


Tabla de verdad programada

Editando el contenido de la ROM indicamos el valor de nuevo estado y la salida puerta. Ej: si estamos en el estado 0, solo una A nos lleva al estado 1, todo otro caso nos deja en cero. Si estamos en el estado 5, el bit más significativo define el valor de puerta (siempre) por ende está en uno para habilitar la puerta.

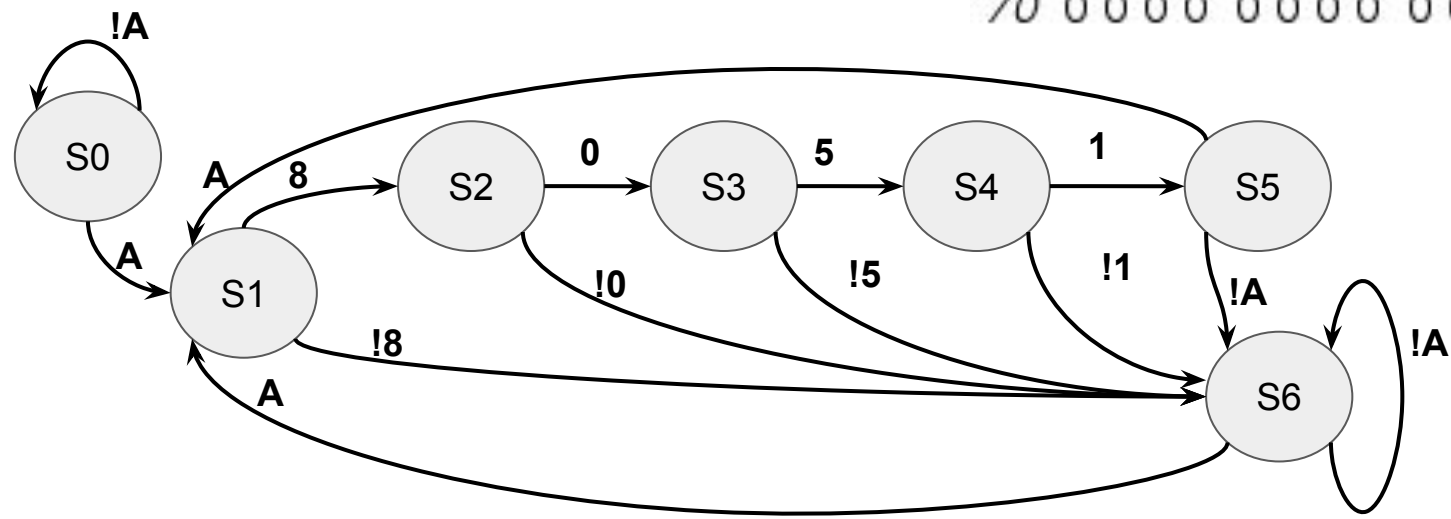
SE	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
10	6	6	6	6	6	6	6	6	2	6	1	0	0	0	0	0
20	3	6	6	6	6	6	6	6	6	6	1	0	0	0	0	0
30	6	6	6	6	6	4	6	6	6	6	1	0	0	0	0	0
40	6	5	6	6	6	6	6	6	6	6	1	0	0	0	0	0
50	e	e	e	e	e	e	e	e	e	e	9	0	0	0	0	0
60	6	6	6	6	6	6	6	6	6	6	1	0	0	0	0	0
70	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0



Conclusiones

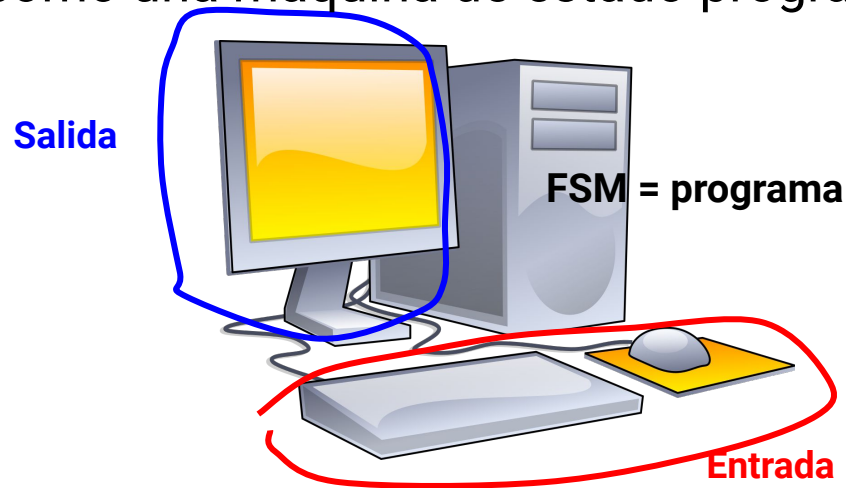
Lo más importante es representar la FSM en el diagrama de transición de estado. Vemos que la implementación física termina siendo una tabla guardada en una memoria.

SE	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
10	6	6	6	6	6	6	6	6	2	6	1	0	0	0	0	0
20	3	6	6	6	6	6	6	6	6	6	1	0	0	0	0	0
30	6	6	6	6	6	4	6	6	6	6	1	0	0	0	0	0
40	6	5	6	6	6	6	6	6	6	6	1	0	0	0	0	0
50	e	e	e	e	e	e	e	e	e	e	9	0	0	0	0	0
60	6	6	6	6	6	6	6	6	6	6	1	0	0	0	0	0
70	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0



Computadora

La teoría de la computación define las características y el alcance de cada modelo. En el caso de FSM, este modelo computacional tiene límites ya que su única memoria se representa en los estados. Si a un estado llegamos por dos caminos posible no podemos recordar cómo llegamos ahí. Existen otros modelos (autómata de pila, máquina de turing) que brindan mayores prestaciones. Sin embargo, podemos pensar una computadora elemental como una máquina de estado programable, con entradas y salidas estándares.



En la siguiente unidad estudiamos cómo implementar máquinas de estado pero usando computadoras mediante un programa. En la siguiente materia vemos como funcionan estas computadoras en detalle.

Maquinas de estado finito (FSM)

Unidad 5.1
FSM en micropython

Versión 1.0.0

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

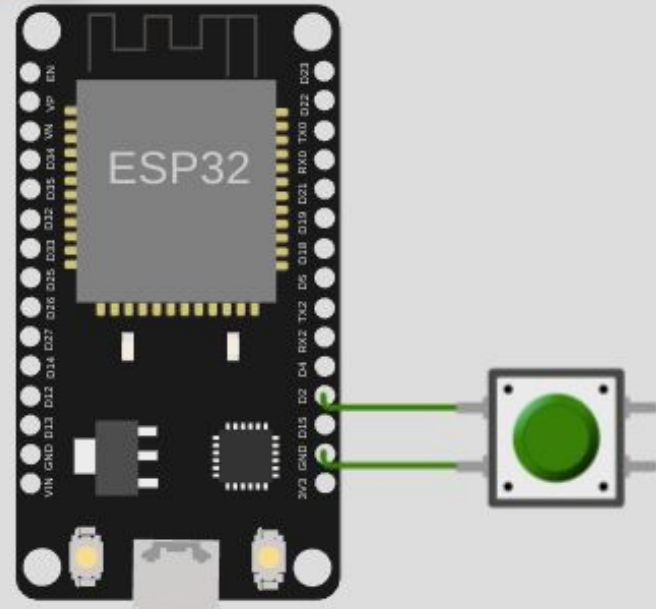
Pulsador

Vamos a describir un problema (bouncing) con un ejemplo. Dado el siguiente programa...

main.py • diagram.json ▾

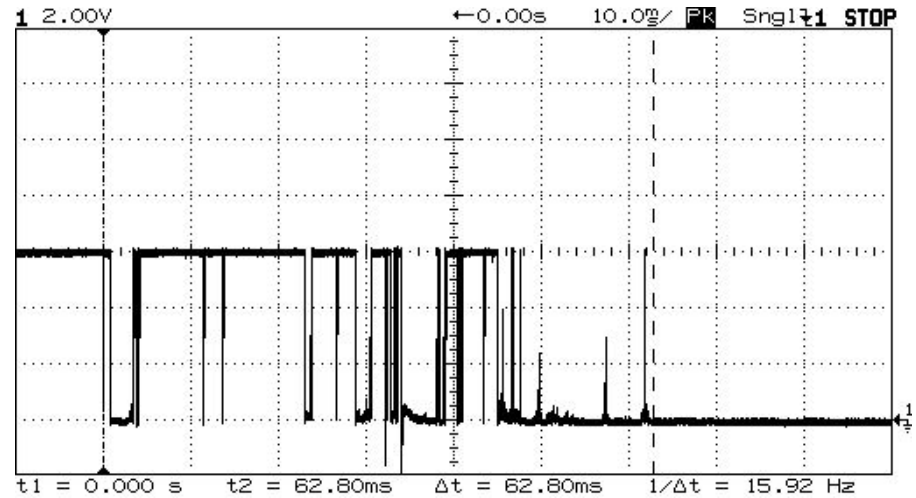
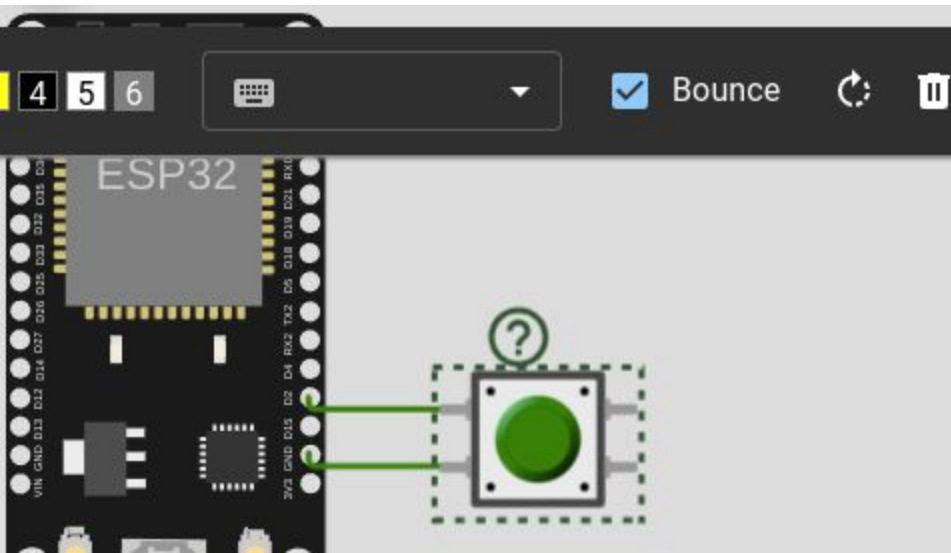
```
1  import machine
2  from machine import Pin
3  pulsador = Pin(2, Pin.IN, Pin.PULL_UP)
4  contador=0
5  while True:
6      while pulsador.value():
7          continue
8      contador = contador + 1
9      print("Contador= "+str(contador))
10     while not pulsador.value():
11         continue
12
```

Simulation



Bouncing

Wokwi simula rebote (bounce) por defecto. Si bien podemos quitar ese tilde y hacer que el pulsador sea perfecto, en la vida real no podemos escapar de esto. Dependiendo de que tan buenos sean los pulsadores el efecto puede ser menor, pero casi seguro no vamos a poder evitarlo mecánicamente en su totalidad.



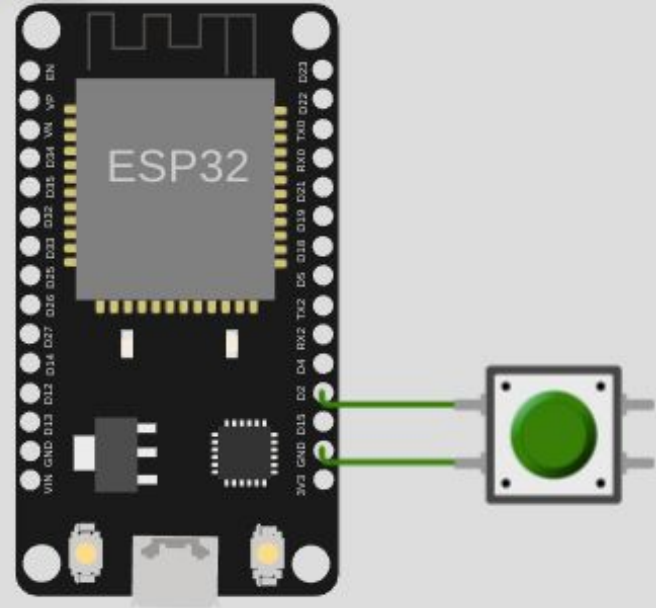
Delay

Podemos hacer una espera de tiempo en el que ignoramos totalmente la entrada. Estaríamos esperando que el pulsador se acomode.

main.py • diagram.json ▾

```
1 import machine
2 from machine import Pin
3 import time
4 pulsador = Pin(2, Pin.IN, Pin.PULL_UP)
5 contador=0
6 while True:
7     while pulsador.value():
8         continue
9     time.sleep_ms(3)
10    contador = contador + 1
11    print("Contador= "+str(contador))
12    while not pulsador.value():
13        continue
14    time.sleep_ms(3)
15
```

Simulation



Suma y resta con dos pulsadores

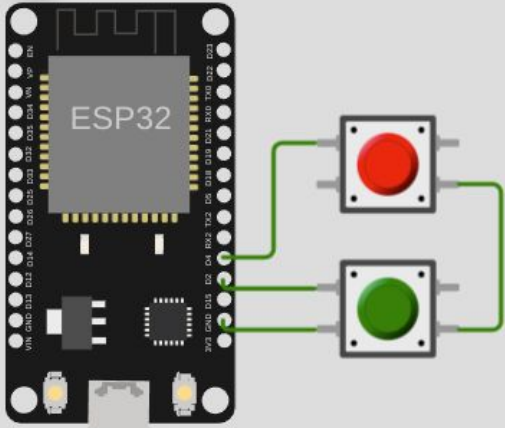
Agregamos otro pulsador, uno suma y el otro resta. Vemos que esta secuencia falla, ya que secuencialmente solo puede revisar un pulsador por vez.

main.py • diagram.json •

```
1 import machine
2 from machine import Pin
3 import time
4 pulsadorRojo = Pin(4,Pin.IN,Pin.PULL_UP)
5 pulsadorVerde = Pin(2,Pin.IN,Pin.PULL_UP)
6 contador=0
7 while True:
8     #Pulsador rojo suma
9     while pulsadorRojo.value():
10         continue
11     time.sleep_ms(3)
12     contador = contador + 1
13     print("Contador= "+str(contador))
14     while not pulsadorRojo.value():
15         continue
16     time.sleep_ms(3)
17     #Pulsador verde resta
18     while pulsadorVerde.value():
19         continue
20     time.sleep_ms(3)
21     contador = contador - 1
22     print("Contador= "+str(contador))
23     while not pulsadorVerde.value():
24         continue
25     time.sleep_ms(3)
26
```

Simulation

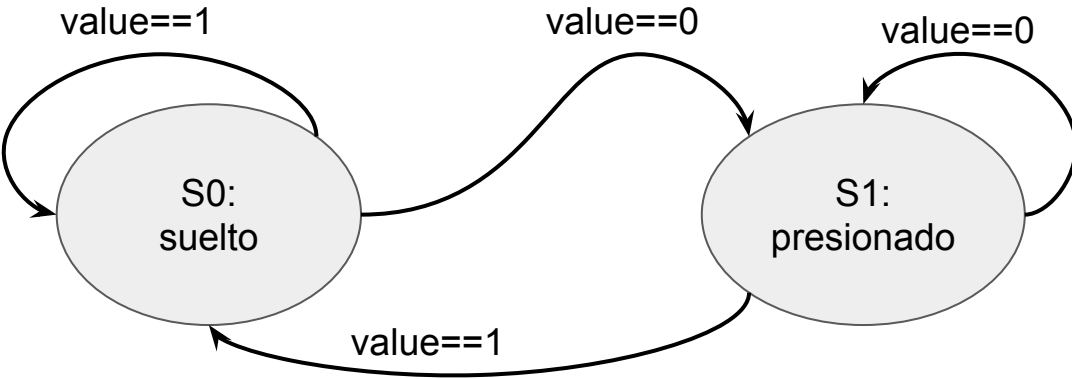
▶ + ⋮



Vamos a manejar esto con una FSM. Cada pulsador tiene su propia FSM y mantiene un estado (presionado, suelto). En la transición de suelto a presionado efectúa una acción sobre el contador.

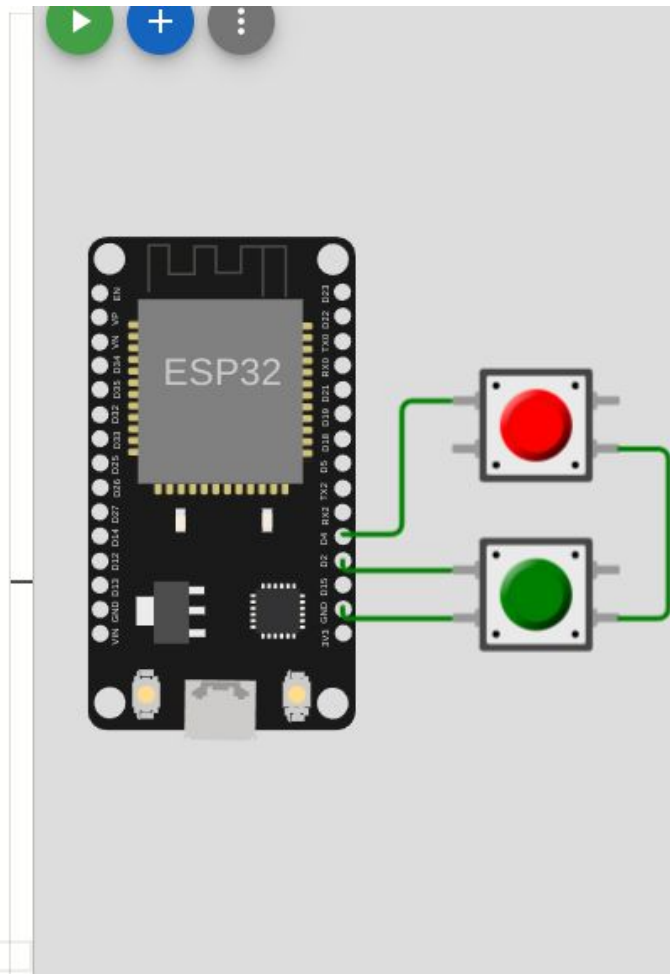
FSM Pulsador

Esta FSM es muy simple. Para evitar el bouncing lo que hacemos es actualizar la FSM cada 3ms.



FSM Pulsador

```
1  import machine
2  from machine import Pin
3  import time
4  pulsadorRojo = Pin(4,Pin.IN,Pin.PULL_UP)
5  pulsadorVerde = Pin(2,Pin.IN,Pin.PULL_UP)
6  contador=0
7  estadoRojo=0 #iniciamos en estado 0
8  estadoVerde=0 #iniciamos en estado 0
9  while True:
10     time.sleep_ms(3) #forzamos esperas de 3ms
11     if not pulsadorRojo.value() and estadoRojo==0:
12         #Si el estado es 0 y el boton se aprieta...
13         #cambiamos de estado y actualizamos el contador
14         estadoRojo=1
15         contador=contador+1
16         print("Contador="+str(contador))
17     if pulsadorRojo.value() and estadoRojo==1:
18         #Si el estado es 1 y el boton se suelta..
19         estadoRojo=0
20     #Ahora el verde
21     if not pulsadorVerde.value() and estadoVerde==0:
22         #Si el estado es 0 y el boton se aprieta...
23         #cambiamos de estado y actualizamos el contador
24         estadoVerde=1
25         contador=contador-1
26         print("Contador="+str(contador))
27     if pulsadorVerde.value() and estadoVerde==1:
28         #Si el estado es 1 y el boton se suelta..
29         estadoVerde=0
30
```



Agregamos un Display 7 segmentos

<https://github.com/mcauser/micropython-tm1637/blob/master/tm1637.py>

main.py

diagram.json

tm1637.py

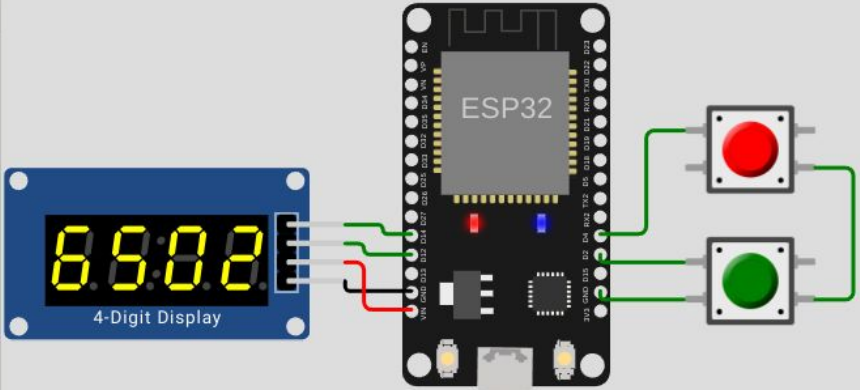
```
1 import machine
2 from machine import Pin
3 import time
4 import tm1637
5 from tm1637 import TM1637
6 display = TM1637(clk=Pin(14), dio=Pin(12))
7 pulsadorRojo = Pin(4,Pin.IN,Pin.PULL_UP)
8 pulsadorVerde = Pin(2,Pin.IN,Pin.PULL_UP)
9 contador=6502
10 display.number(contador)
11 estadoRojo=0 #iniciamos en estado 0
12 estadoVerde=0 #iniciamos en estado 0
13 while True:
14     time.sleep_ms(3) #forzamos esperas de 3ms
15     if not pulsadorRojo.value() and estadoRojo==0:
16         #Si el estado es 0 y el boton se aprieta...
17         #cambiamos de estado y actualizamos el contador
18         estadoRojo=1
19         contador=contador+1
20         display.number(contador)
21     if pulsadorRojo.value() and estadoRojo==1:
22         #Si el estado es 1 y el boton se suelta..
23         estadoRojo=0
24     #Ahora el verde
25     if not pulsadorVerde.value() and estadoVerde==0:
26         #Si el estado es 0 y el boton se aprieta...
27         #cambiamos de estado y actualizamos el contador
28         estadoVerde=1
29         contador=contador-1
30         display.number(contador)
31     if pulsadorVerde.value() and estadoVerde==1:
32         #Si el estado es 1 y el boton se suelta..
33         estadoVerde=0
34
```

Simulation

↺

■

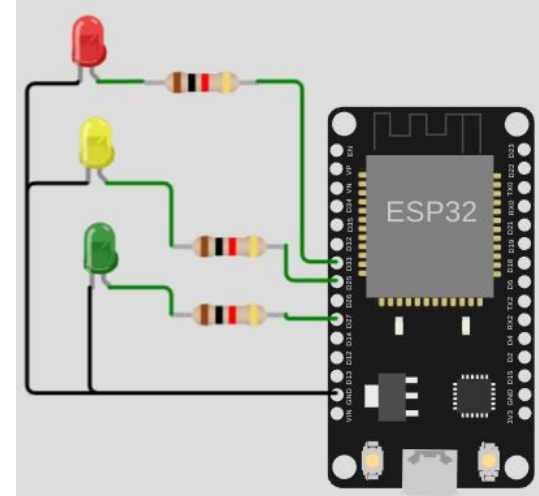
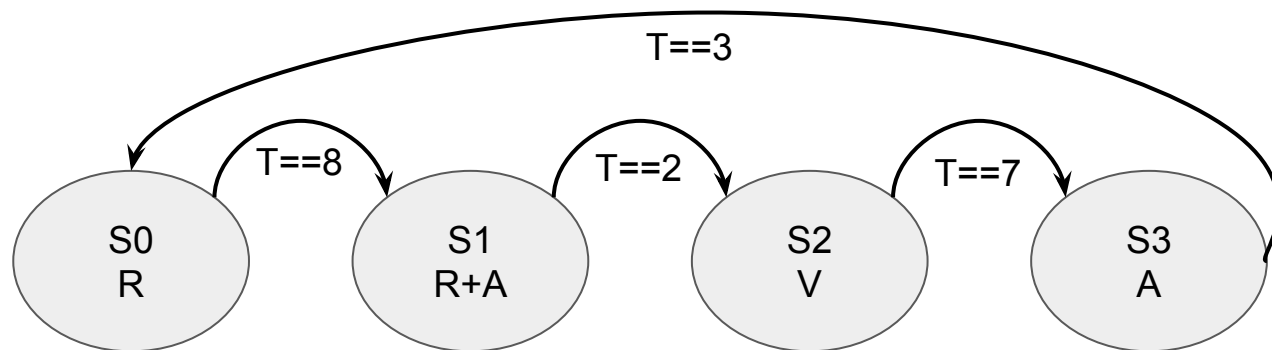
⏸



Semáforo y timers

Semáforo FSM

Implementemos una fase de un semáforo simple.



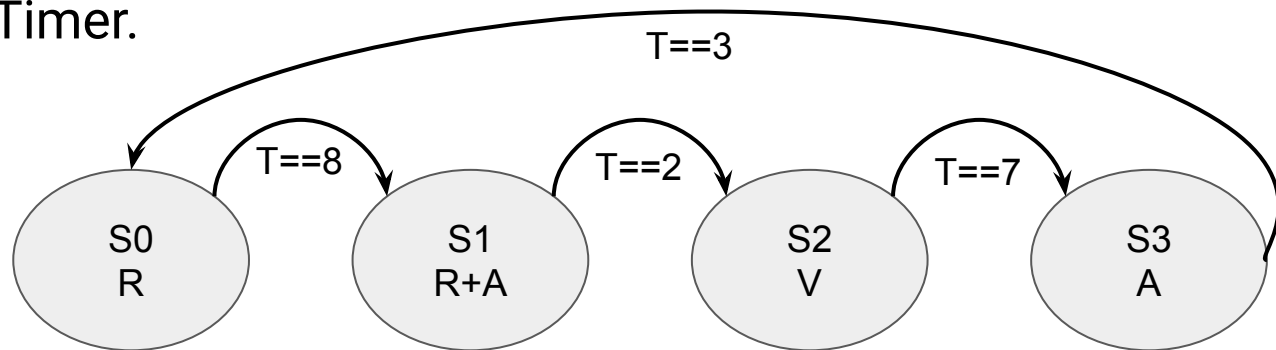
```
1 from machine import Pin, Timer
2 #Definimos variables accesibles globalmente
3 ledVerde = Pin(27, Pin.OUT)
4 ledAmarillo = Pin(25, Pin.OUT)
5 ledRojo = Pin(33, Pin.OUT)
6 contador=0
7 estado=0
```

```
9 def Estado0():
10     ledRojo.value(1)
11     ledAmarillo.value(0)
12     ledVerde.value(0)
13
14 def Estado1():
15     ledRojo.value(1)
16     ledAmarillo.value(1)
17     ledVerde.value(0)
```

```
19 def Estado2():
20     ledRojo.value(0)
21     ledAmarillo.value(0)
22     ledVerde.value(1)
23
24 def Estado3():
25     ledRojo.value(0)
26     ledAmarillo.value(1)
27     ledVerde.value(0)
```

Semáforo FSM

Definimos una función FSM que cambia el estado actual. Se dispara con un Timer.

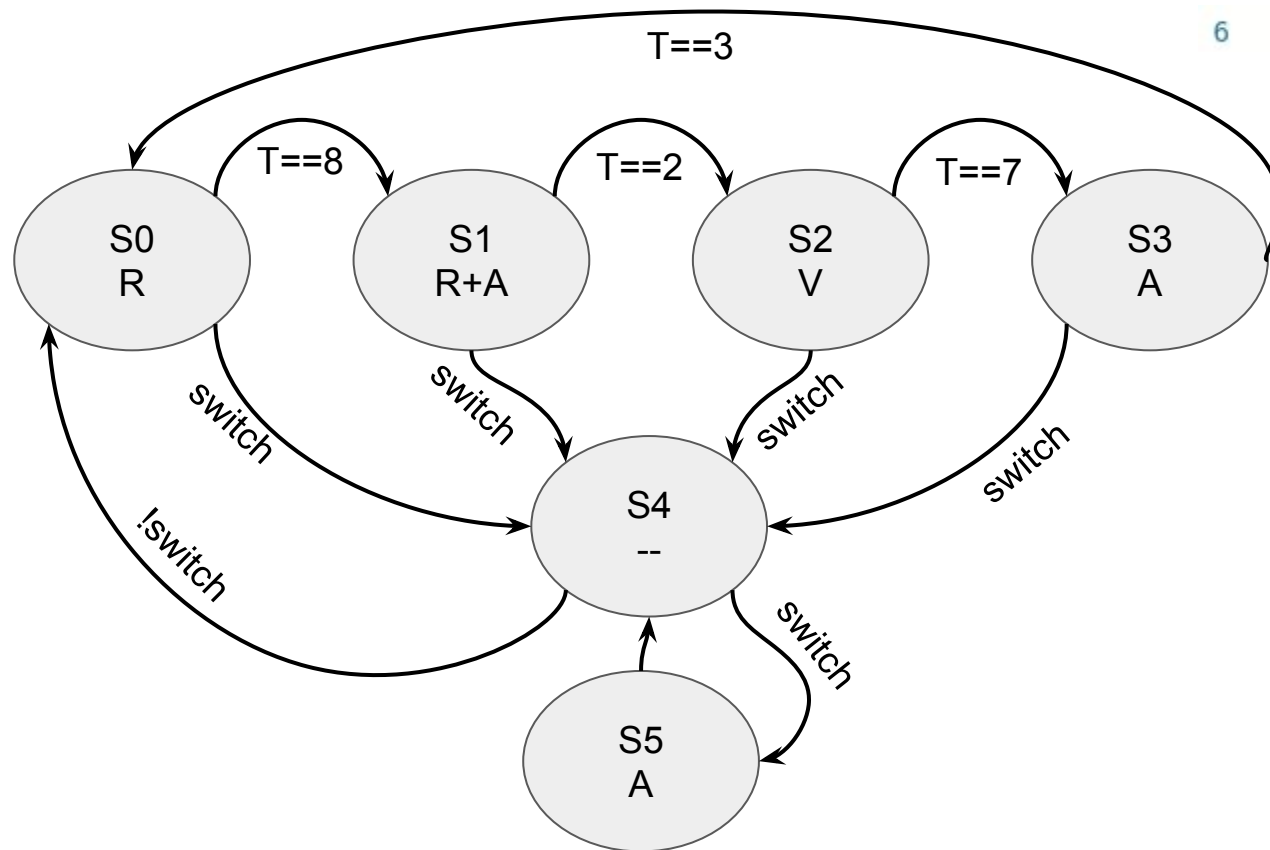


```
50 Estado0() #Estado inicial
51 elTimer = Timer(1)
52 elTimer.init(mode=Timer.PERIODIC, period=1000, callback=FSM)
```

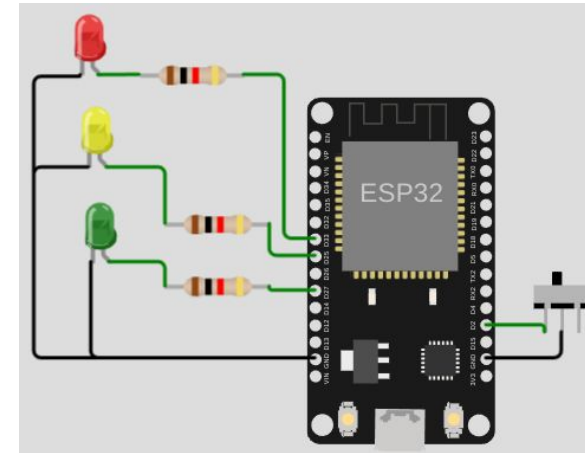
```
29 def FSM(t):
30     global contador
31     global estado
32     contador = contador + 1
33     if estado==0 and contador == 8:
34         estado=1
35         contador=0
36         Estado1()
37     elif estado==1 and contador == 2:
38         estado = 2
39         contador = 0
40         Estado2()
41     elif estado==2 and contador == 7:
42         estado=3
43         contador=0
44         Estado3()
45     elif estado==3 and contador==3:
46         estado = 0
47         contador = 0
48         Estado0()
```

Semáforo FSM con tecla para intermitente

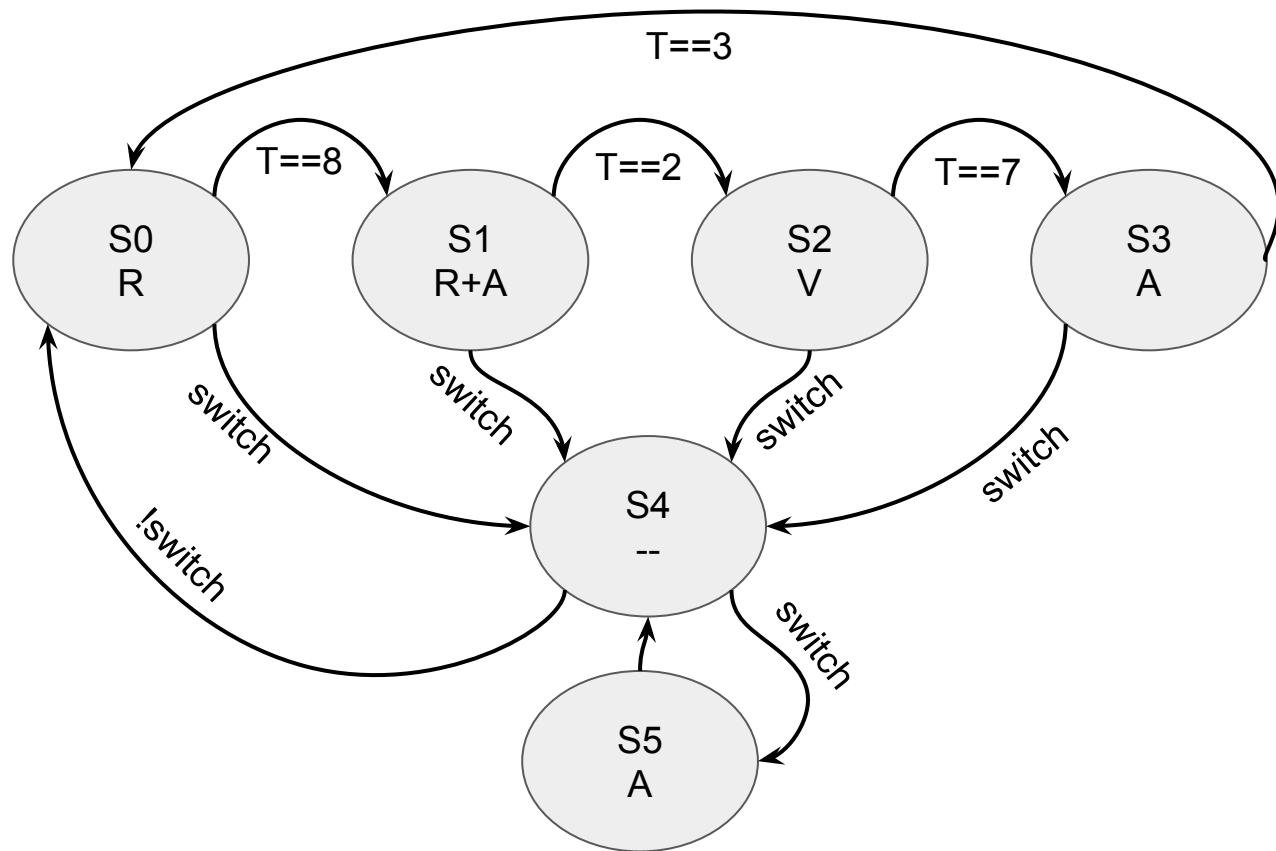
Agregamos una entrada nueva, un switch para forzar intermitente amarillo.



```
6 switch = Pin(2,Pin.IN,Pin.PULL_UP)
```



Semáforo FSM con tecla para intermitente



```
30 def Estado4():
31     ledRojo.value(0)
32     ledAmarillo.value(0)
33     ledVerde.value(0)
34
35 def FSM(t):
36     global contador
37     global estado
38     contador = contador + 1
39     if estado<4 and switch.value():
40         estado=4
41         Estado4()
42     elif estado==4 and switch.value():
43         estado=5
44         Estado3()
45     elif estado==5:
46         estado=4
47         Estado4()
48     elif estado==4 and not switch.value():
49         estado = 0
50         contador = 0
51         Estado0()
52     elif estado==0 and contador == 8:
53         estado=1
54         contador=0
55         Estado1()
56     elif estado==1 and contador == 2:
57         estado = 2
58         contador = 0
59         Estado2()
60     elif estado==2 and contador ==7:
61         estado=3
62         contador=0
63         Estado3()
64     elif estado==3 and contador==3:
65         estado = 0
66         contador = 0
67         Estado0()
```