De la Lógica Digital al diseño de un procesador

Javier Resano, Jose Luis Briz

GAZ: Grupo de Arquitectura de Computadores,

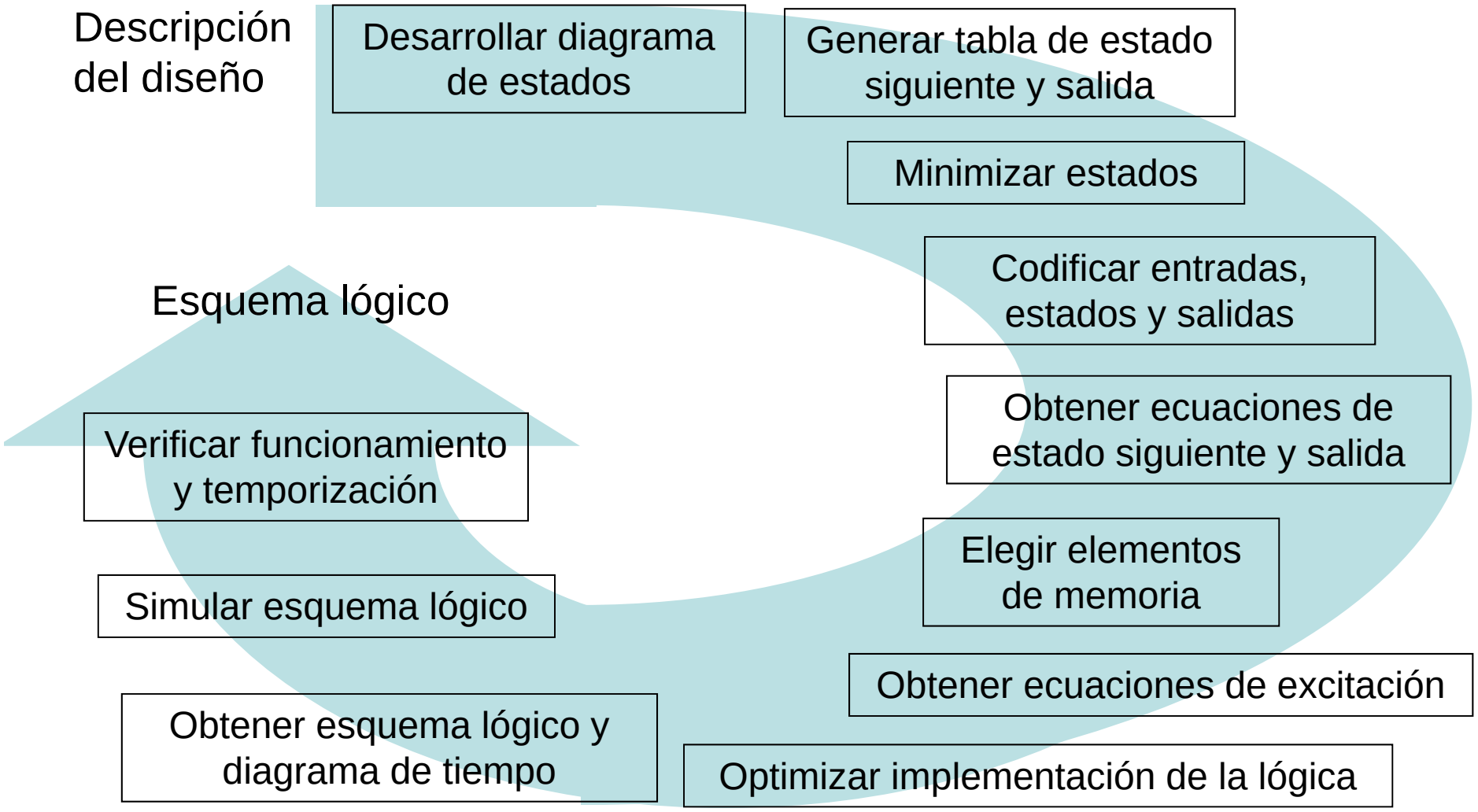
Departamento de Informática e Ingeniería de Sistemas

Universidad Zaragoza

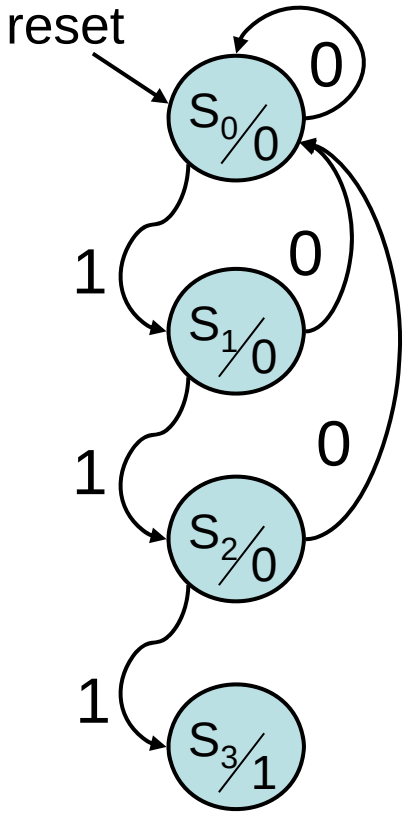
Diseño de un procesador específico

1. **Cómo hacer un diseño digital complejo**
    1. Diseño jerárquico y reutilización
    2. Diseño algorítmico
2. Análisis temporal de un diseño

Motivación: la implementación canónica no es siempre es adecuada



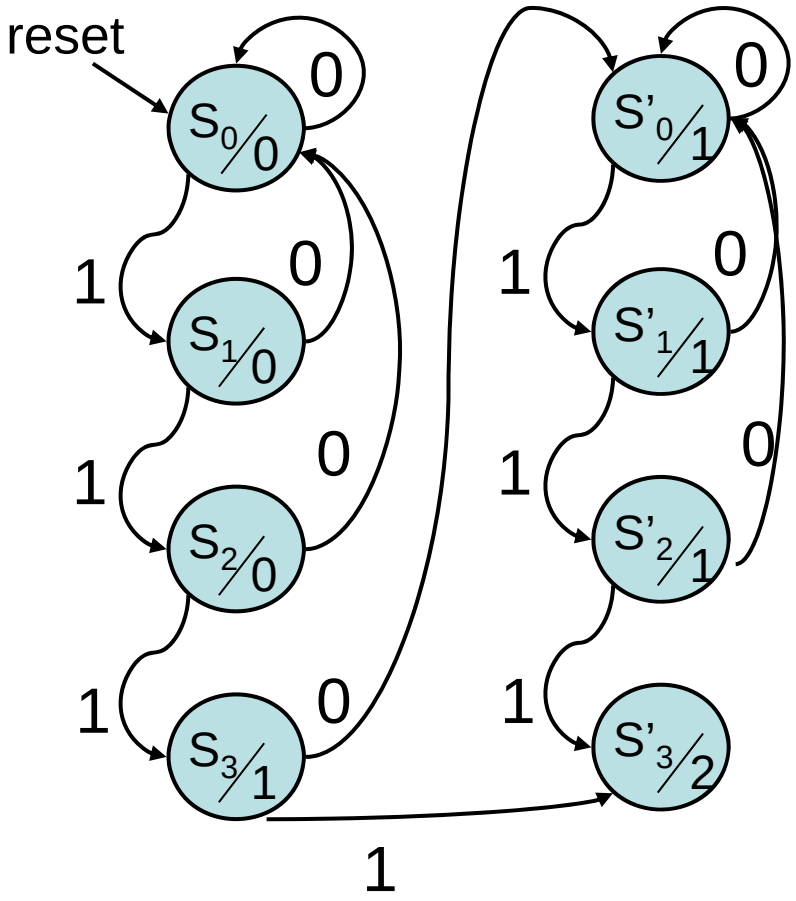
Motivación: contador de la secuencia ”111” con solapamiento



Debe ser capaz de contar hasta 255 secuencias

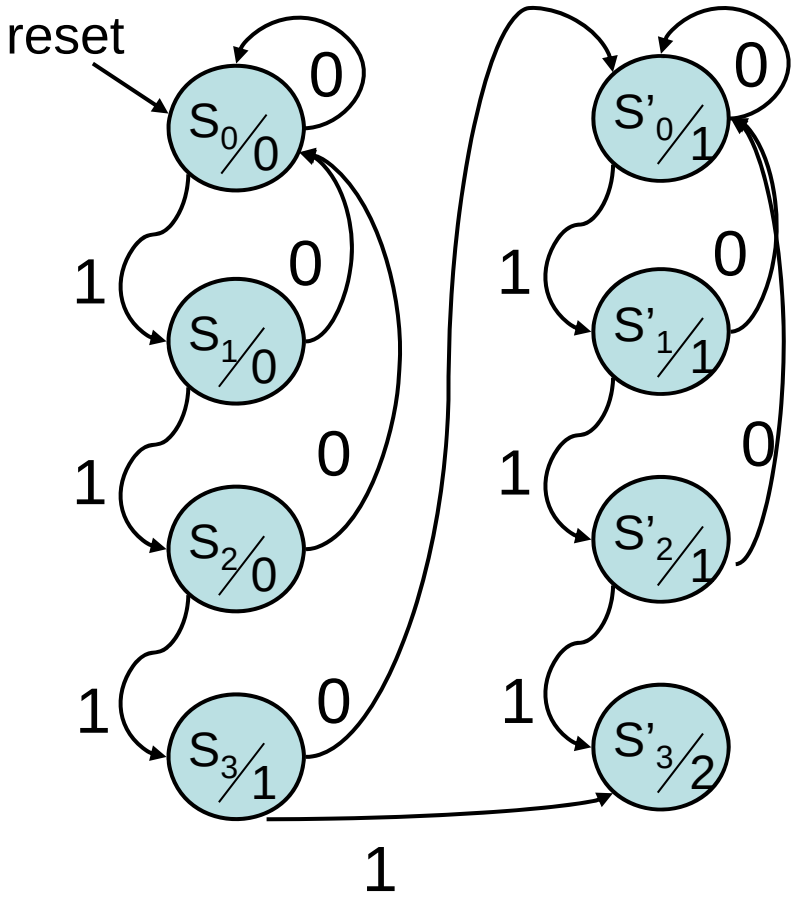
estados que identifican la primera secuencia

Motivación: contador de la secuencia ”111” con solapamiento



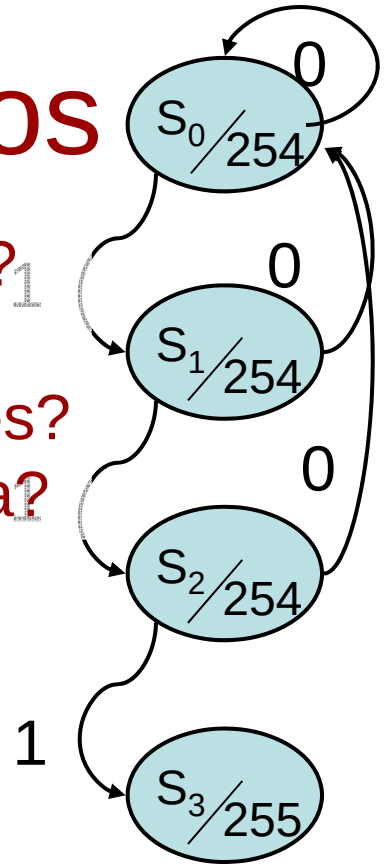
primera y segunda secuencia

Motivación: contador de la secuencia ”111” con solapamiento



1020 estados

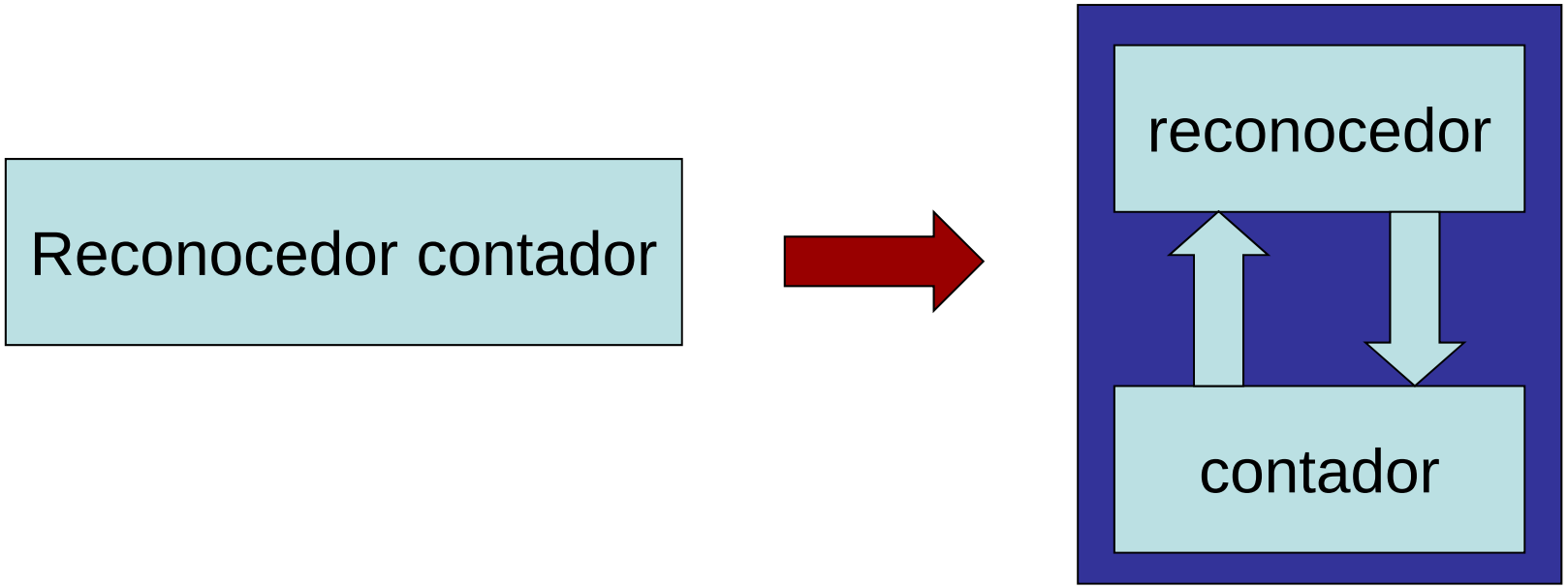
¿Minimizar estados?
¿codificar?
¿Elegir los biestables?
¿Simplificar la lógica?



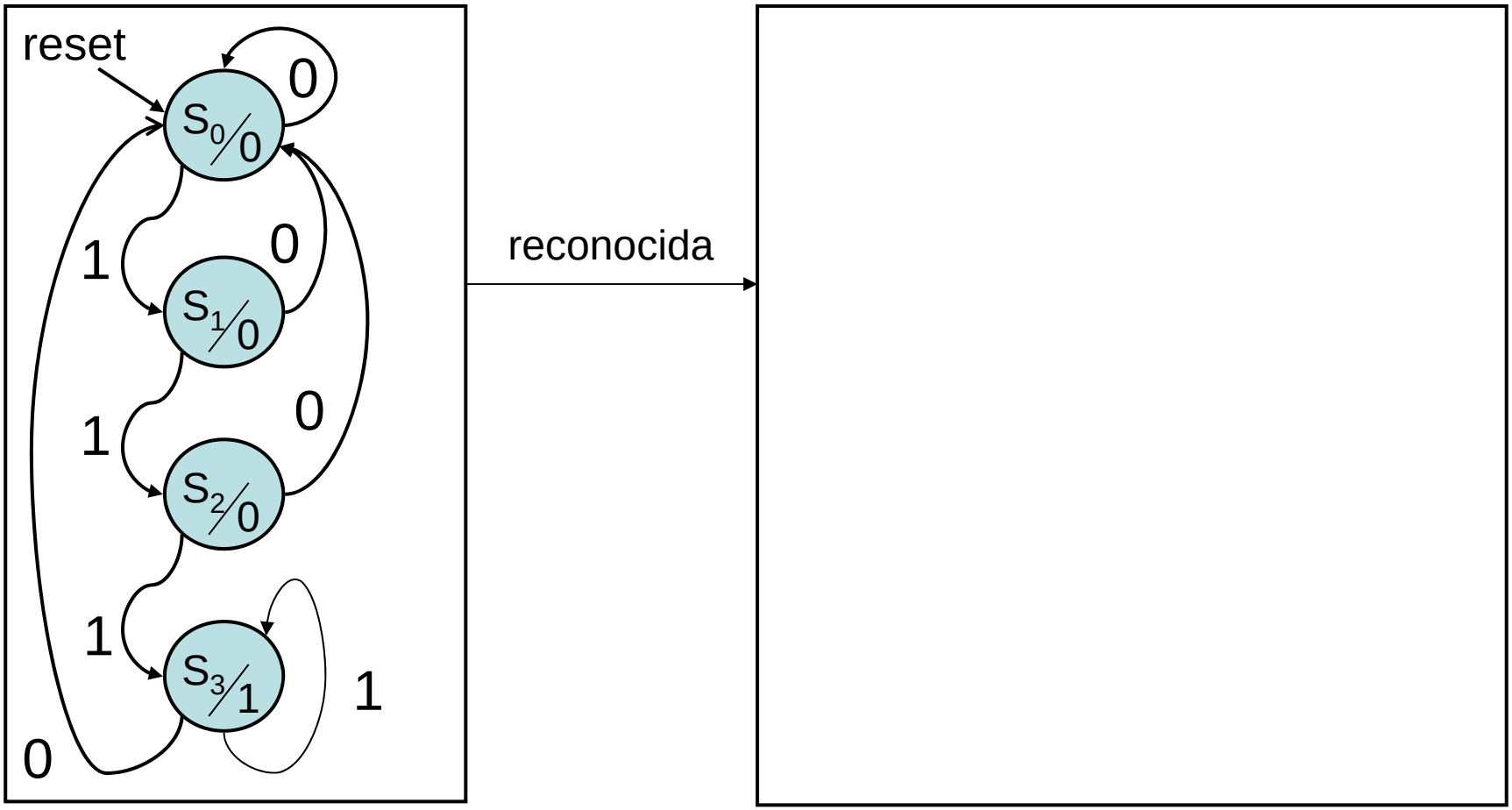
secuencia 255

Motivación: contador de la secuencia ”111” con solapamiento

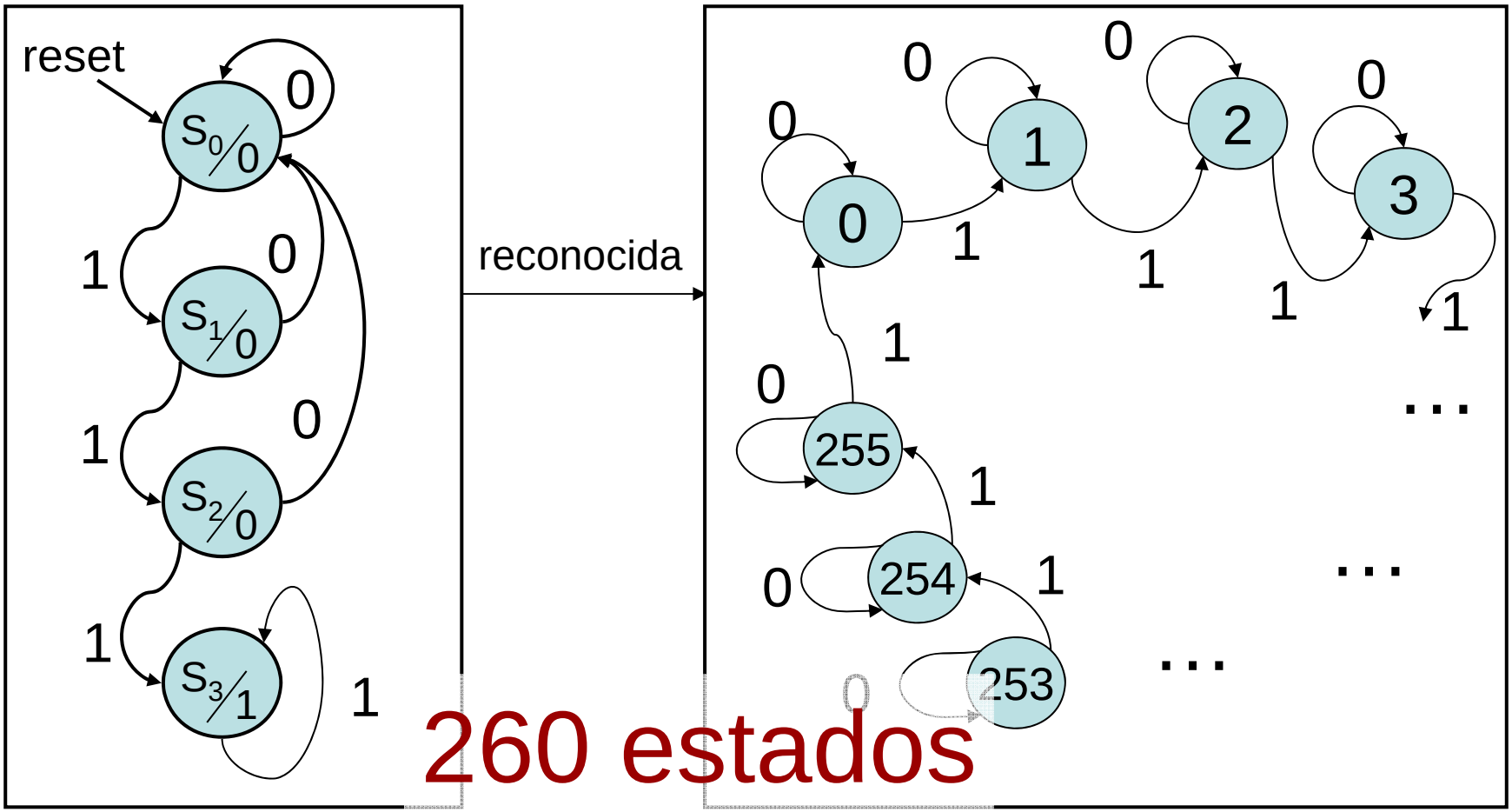
¿Y si dividimos el problema en dos sub-problemas más sencillos?



Motivación: contador de la secuencia ”111” con solapamiento

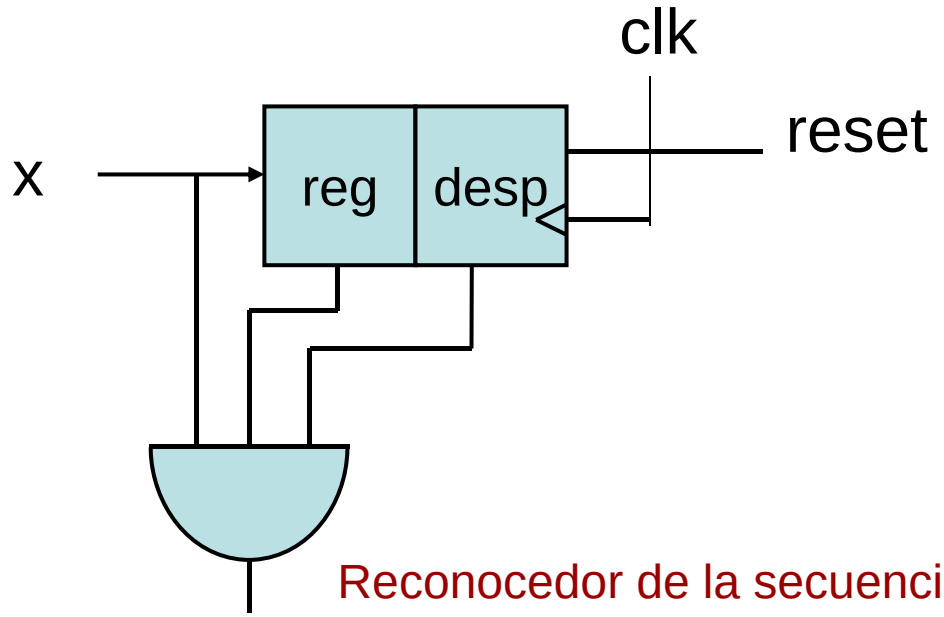


Motivación: contador de la secuencia ”111” con solapamiento



Motivación: contador de la secuencia ”111” con solapamiento

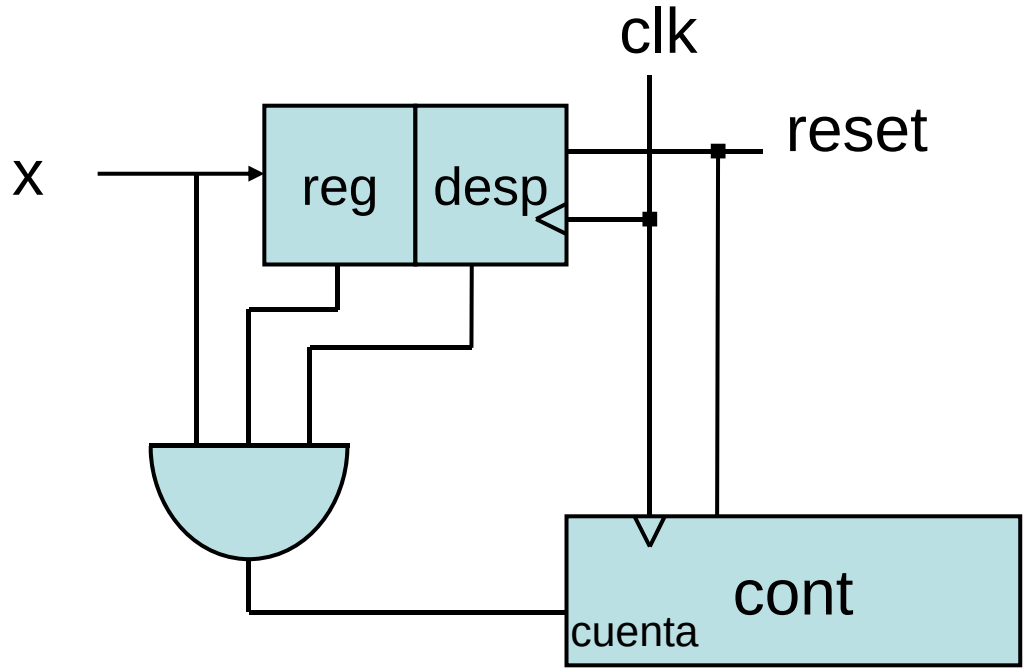
¿Y si utilizamos módulos prediseñados para simplificar el proceso de diseño?



Reconocedor de la secuencia “111”

Motivación: contador de la secuencia ”111” con solapamiento

¿y si utilizamos módulos prediseñados para simplificar el proceso de diseño?



Diseño jerárquico y reutilización

- Realizar diseño complejo
  - dividirlo en sub-problemas más sencillos que seamos capaces de resolver
    - **particionamiento**

- Para simplificar todavía más el diseño en lugar de comenzar el diseño desde cero podemos **reutilizar componentes estándar** :
  - registros
  - contadores
  - memorias RAM
  - ALUs
  - Muxes, Decodificadores, Codificadores…

¿Podemos enfrentarnos al diseño directamente?

- Sí, somos capaces de realizar el diseño
  - Comenzamos a trabajar
- No, el diseño es demasiado complejo
  - Fase de división o particionamiento:
    - Dividimos en subproblemas más sencillos
    - Aplicamos nuestras técnicas a cada subproblema por separado
    - Realizamos el diseño final utilizando los subproblemas
  - Esta fase **simplifica** el problema pero también **limita las posibilidades de optimización**:
    - Las decisiones tomadas por separado para cada subproblema pueden comprometer el agregado final

Reutilización: clave para un diseño rápido y fiable

- Hay módulos que se utilizan muy frecuentemente y cuya implementación:
  - es conocida
  - está optimizada y verificada
  - se incluye en las herramientas de ayuda al diseñador
- Comenzar desde cero no es una estrategia óptima
  - ¿por qué hacer lo que ya está hecho?
- Reutilizar componentes de una librería
  - permite acelerar el proceso de diseño
  - muchos componentes son comunes a todas las librerías
- El diseñador debe conocer estos módulos y saber utilizarlos

Diseño algorítmico: motivación

- Frecuentemente no es práctico representar un sistema utilizando tan sólo tablas y diagramas de estado
- Ejemplo:
  - Entrada: un número de 8 bits.
  - Salida: número de veces que se ha introducido esa misma entrada (módulo 16).
  - Necesitamos un vector de estado de $2^8 * 2^4$ (12 bits)
  - Tabla de transiciones de estado: 32 bits por entrada, 2^{20} filas
- Para describir diseños complejos necesitamos aumentar el nivel de abstracción de la especificación: **utilizamos algoritmos**

Motivación

- ¿Y si representamos el sistema usando un algoritmo?
- Ejemplo:
  - Entrada: un número de 8 bits.
  - Salida: número de veces que se ha introducido esa misma entrada (módulo 16).

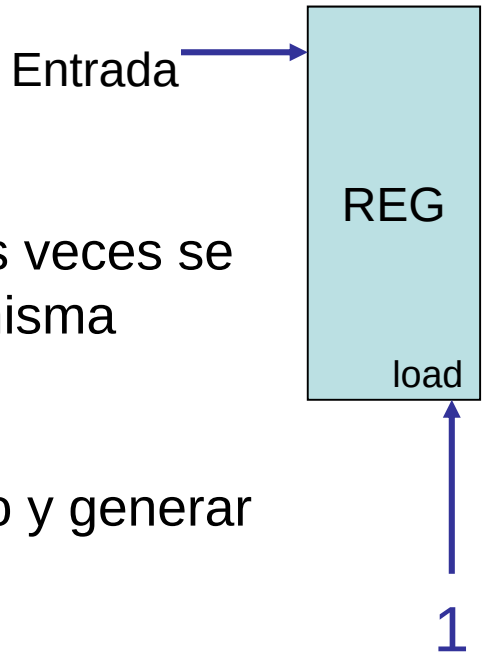
Algoritmo:

- leer la entrada
- comprobar cuántas veces se ha introducido previamente la misma entrada
- sumar 1
- actualizar el estado y generar la salida

¿y cómo hacemos el diseño a partir del algoritmo?

Algoritmo:

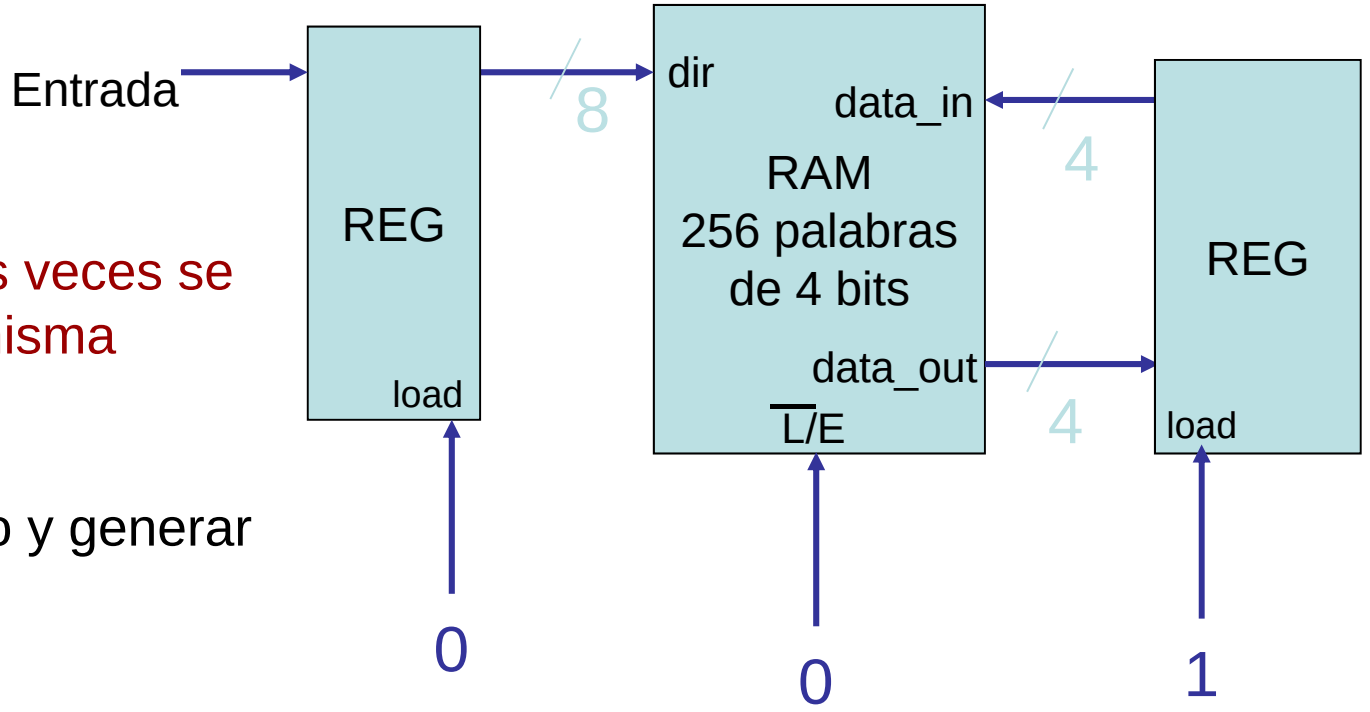
- leer la entrada
- comprobar cuántas veces se ha introducido la misma entrada
- sumar 1
- actualizar el estado y generar la salida



¿y cómo hacemos el diseño a partir del algoritmo?

Algoritmo:

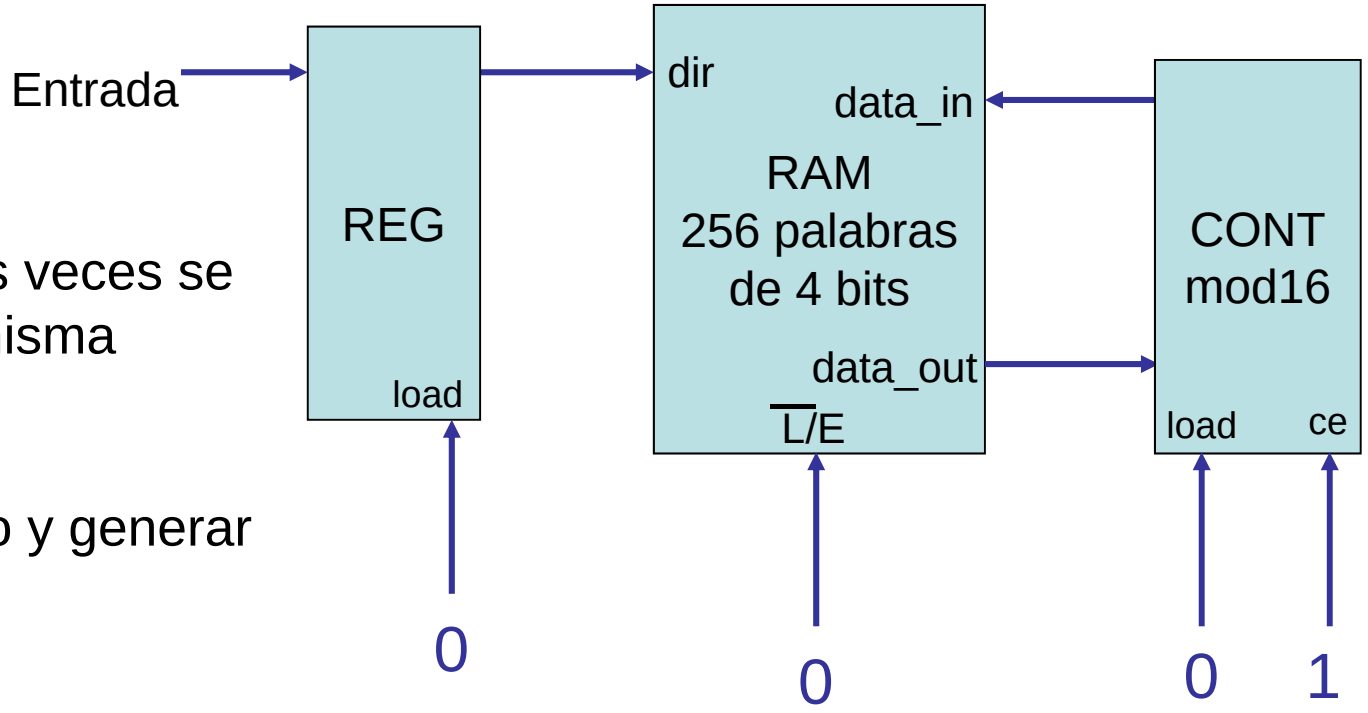
- leer la entrada
- comprobar cuántas veces se ha introducido la misma entrada
- sumar 1
- actualizar el estado y generar la salida



¿y cómo hacemos el diseño a partir del algoritmo?

Algoritmo:

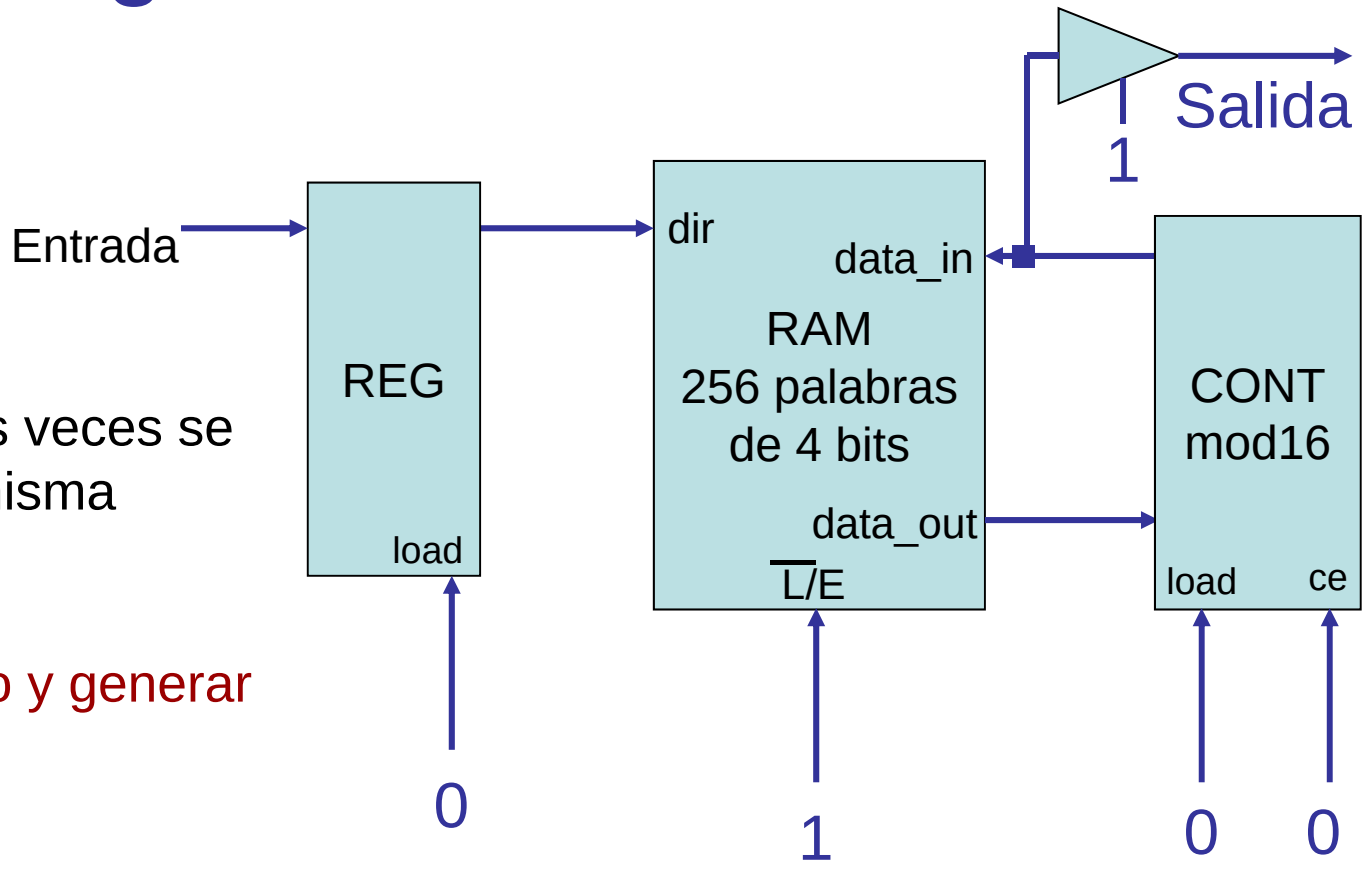
- leer la entrada
- comprobar cuántas veces se ha introducido la misma entrada
- sumar 1
- actualizar el estado y generar la salida



¿y cómo hacemos el diseño a partir del algoritmo?

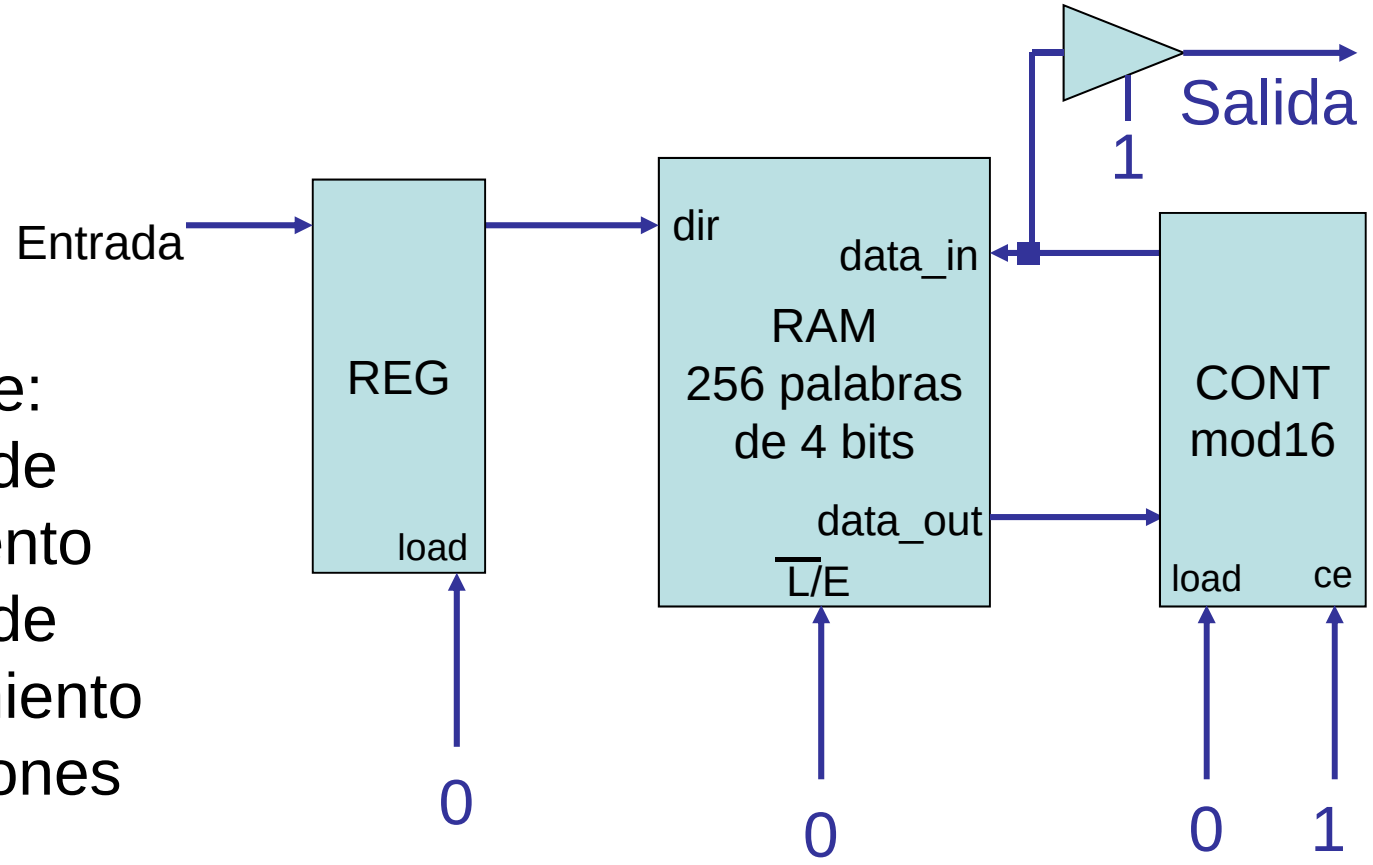
Algoritmo:

- leer la entrada
- comprobar cuántas veces se ha introducido la misma entrada
- sumar 1
- actualizar el estado y generar la salida



El camino de datos es el módulo que procesa los datos de entrada

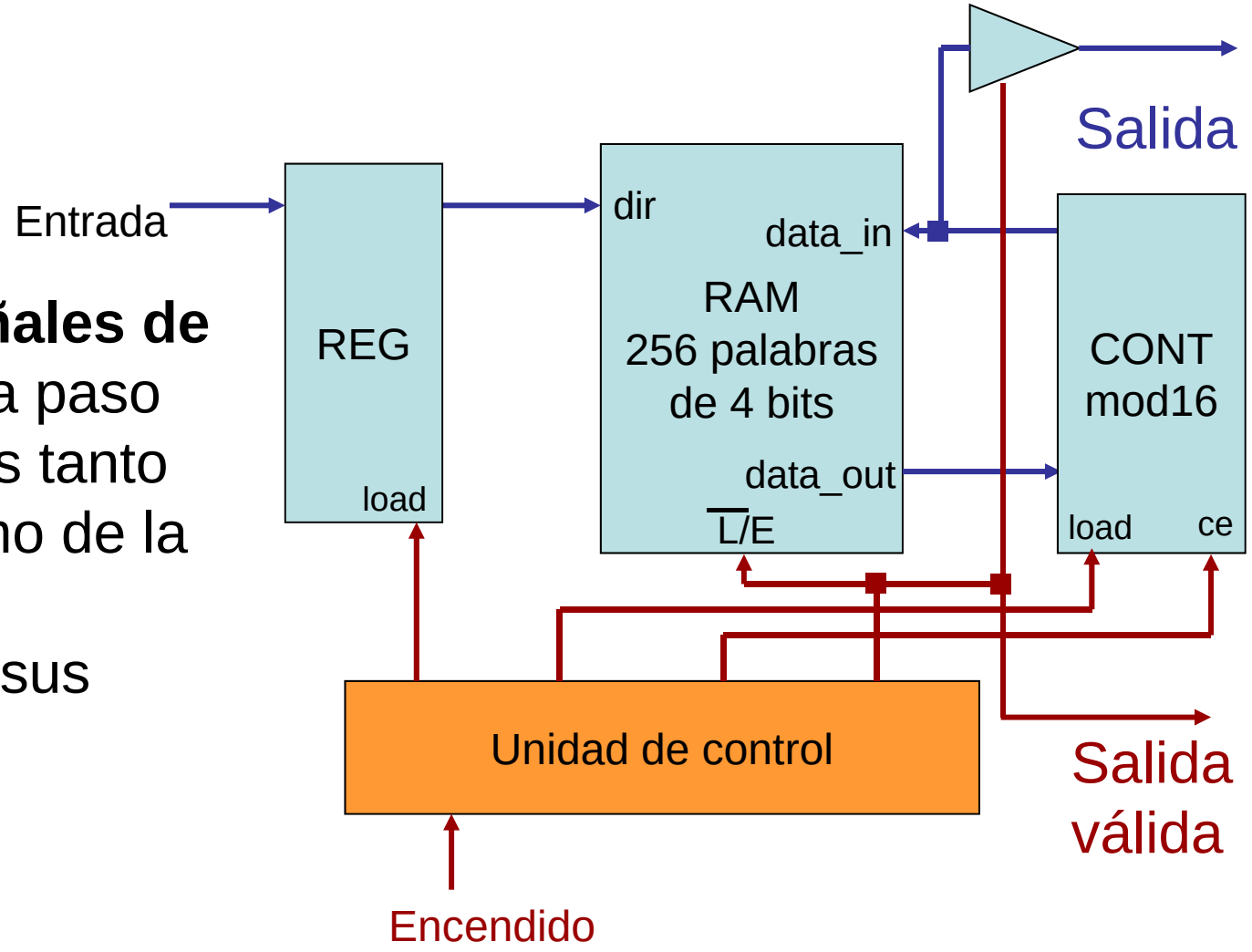
- Se compone de:
  - elementos de procesamiento
  - elementos de almacenamiento
  - interconexiones



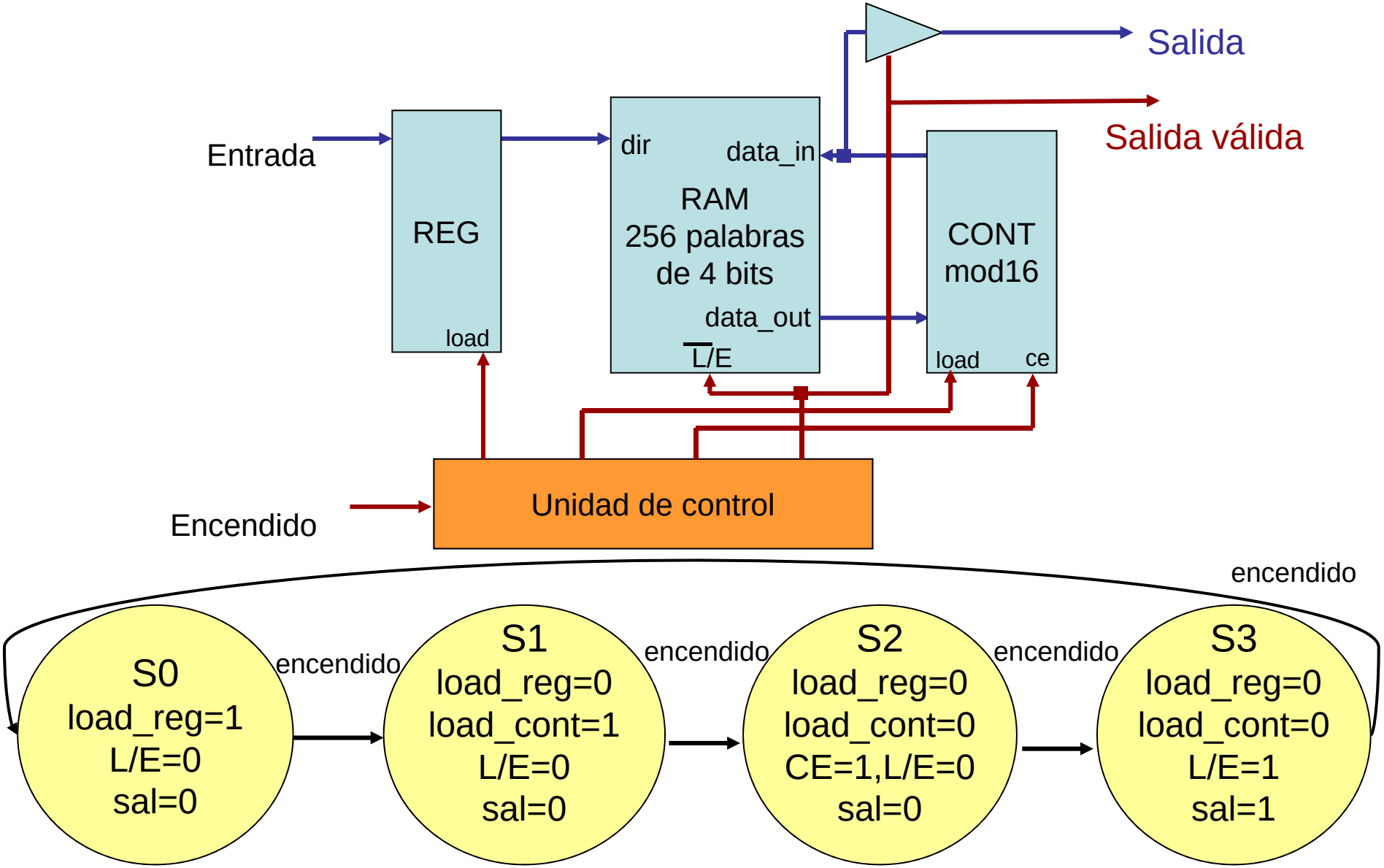
Pero ¿quién asigna el valor correcto a las señales que controlan el camino de datos?

La unidad de control dirige a la ruta de datos

- **Genera las señales de control** de cada paso
- Recibe entradas tanto del exterior como de la ruta de datos
- Puede generar sus propias salidas



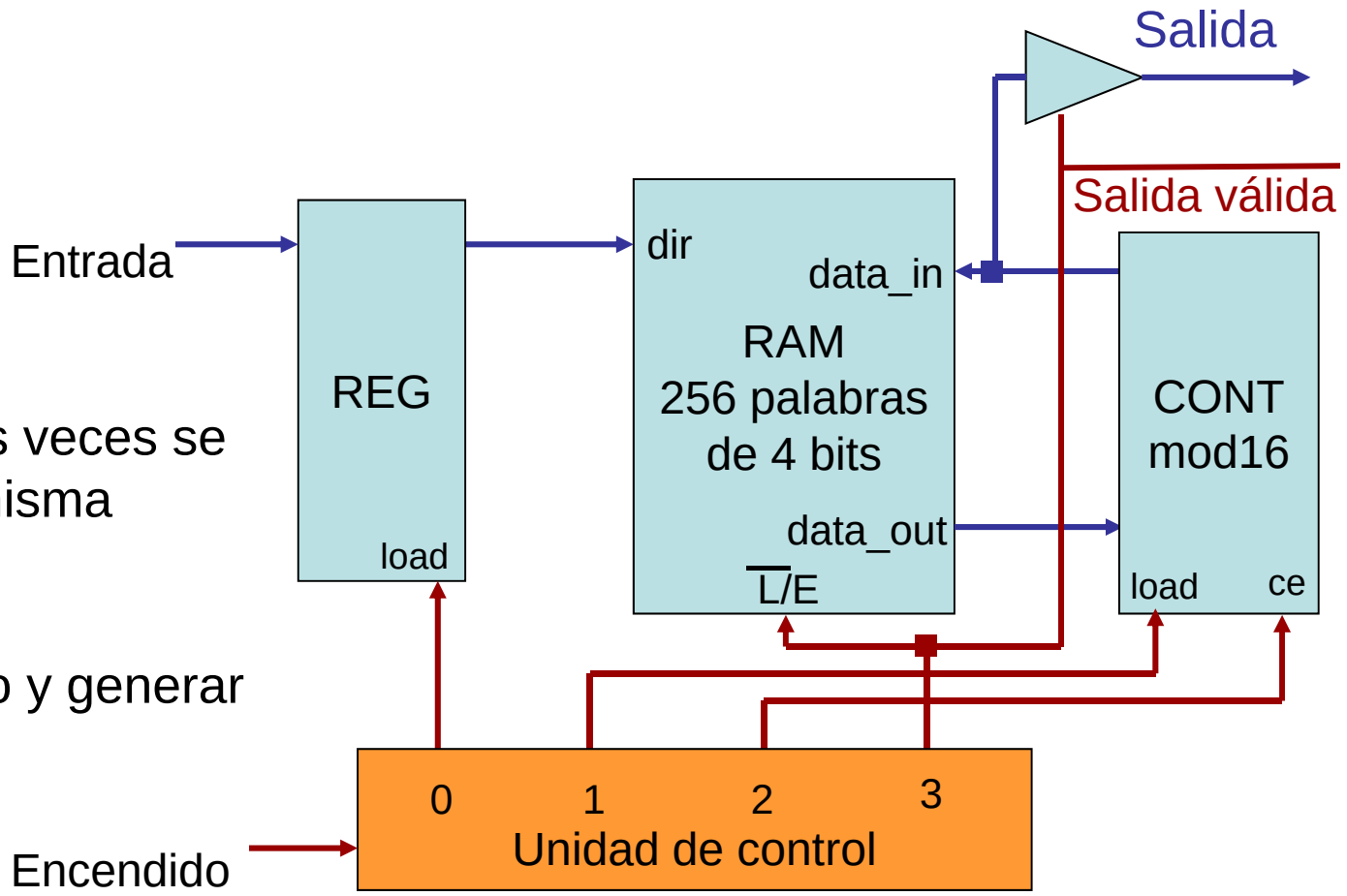
Diseño de la unidad de control



Diseño final

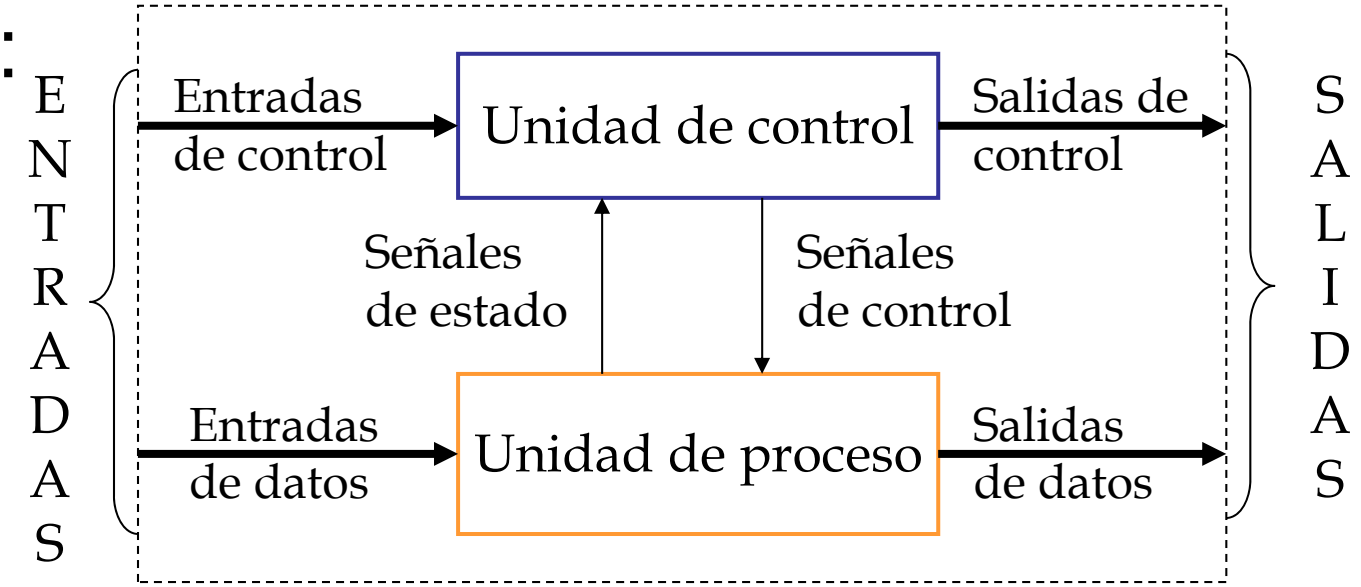
Si encendido =1
- leer la entrada
- comprobar cuántas veces se ha introducido la misma entrada
- sumar 1
- actualizar el estado y generar la salida

Si encendido =0
- esperar



¿Qué son los sistemas algorítmicos?

- Sistemas secuenciales síncronos
- Comportamiento definido IMPLÍCITAMENTE
  - No se especifica el valor de Z sino el modo de calcularlo: el algoritmo.
- Modelo:



Proceso de diseño

1. Definición del sistema
    - Especificar: entradas y salidas, función, bloques disponibles
2. Diseño del algoritmo
    - Obtener conjunto ordenado y finito de operaciones.
    - Modelo de máquina de estados generalizada.
3. Definición de la arquitectura del sistema
    - Determinar entradas, salidas y señales internas
4. Diseño de la unidad de proceso o ruta de datos
5. Diseño de la unidad de control
6. Ensamblado (de U.P y U.C.) y verificación

Conclusiones

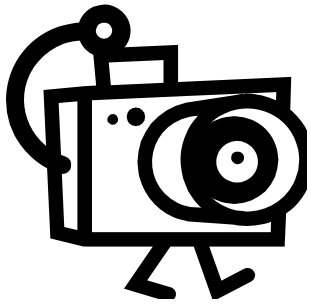
- El flujo de diseño lógico canónico no es adecuado para diseños complejos
- El diseño algorítmico nos permite enfrentarnos a estos problemas **subiendo el nivel de abstracción**
- **dividiremos** los diseños en:
    - el **camino de datos** que se ocupa de procesar los datos
    - la **unidad de control** que genera las señales necesarias para que cada etapa del algoritmo se ejecute correctamente
- Para terminar **ensamblamos** ambas partes y **verificamos** el funcionamiento

Diseño de un procesador específico

1. Alternativas a la implementación canónica
2. **Análisis temporal de un diseño**
   1. Retrasos de propagación de puertas lógicas
   2. Estudio temporal de Sistemas Combinatorios
   3. Estudio Temporal de los Sistemas Secuenciales. Tiempo de Ciclo
   4. Modelo temporal de las memorias

¿Por qué es importante el análisis temporal?

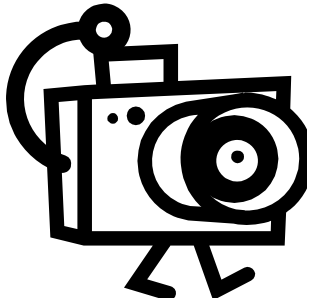
Suponemos que queremos hacer una foto a un ciclista saltando, queremos que salga en el aire



@http://www.sirbikesalot.com/entry.php?fid=71

¿Por qué es importante el análisis temporal?

Si hacemos la foto muy pronto….

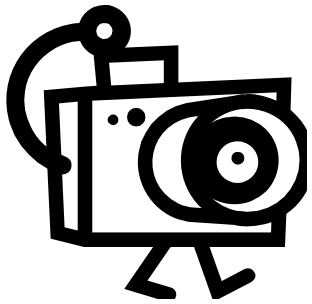


@http://www.sirbikesalot.com/entry.php?fid=71

¿Por qué es importante el análisis temporal?

Si hacemos la foto muy tarde…

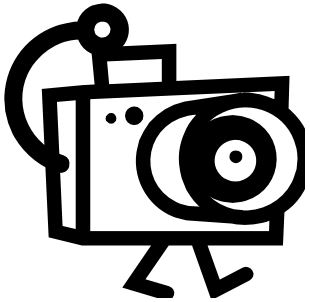
El ciclista estaba en el aire al disparar



@http://www.sirbikesalot.com/entry.php?fid=71

¿Por qué es importante el análisis temporal?

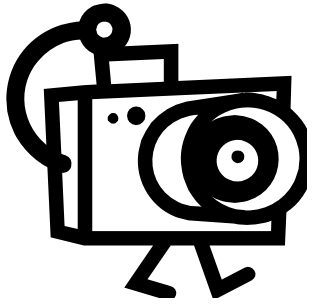
Si la hacemos “muy rápido”



@http://www.sirbikesalot.com/entry.php?fid=71

¿Por qué es importante el análisis temporal?

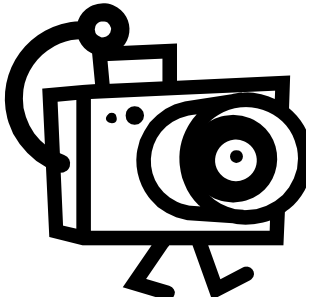
O muy lento…



@http://www.sirbikesalot.com/entry.php?fid=71

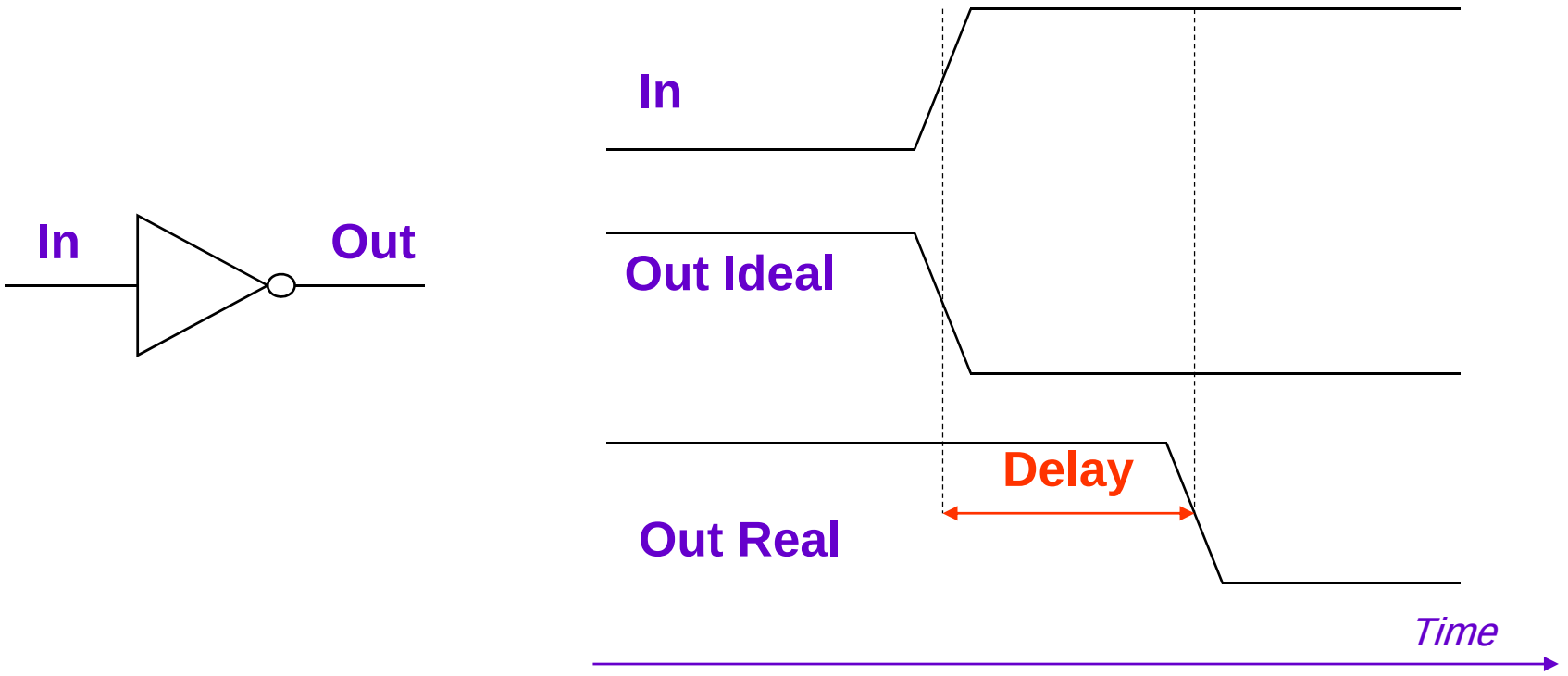
Hay que hacer las fotos en el momento preciso

-Hay que esperar a que se inicie el salto
-Se tiene que disparar cuando todavía esté un cierto tiempo en el aire
-La “velocidad” del disparo no puede ser muy pequeña



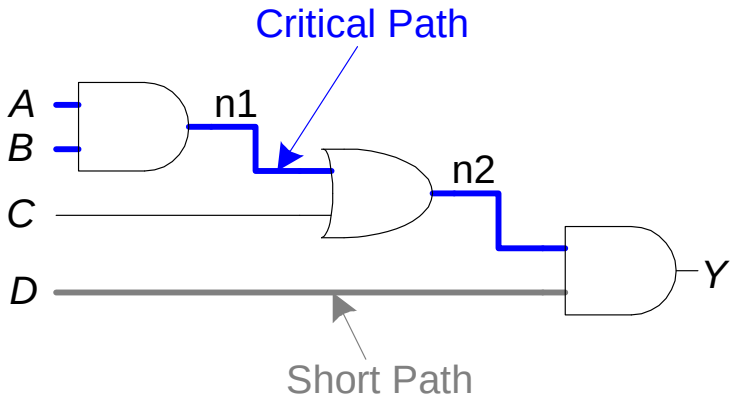
Con los circuitos digitales ocurre algo parecido, **Una de las claves en el diseño de circuitos es el “timing”.**

Los cambios en las entradas tardan un tiempo en propagarse a las salidas

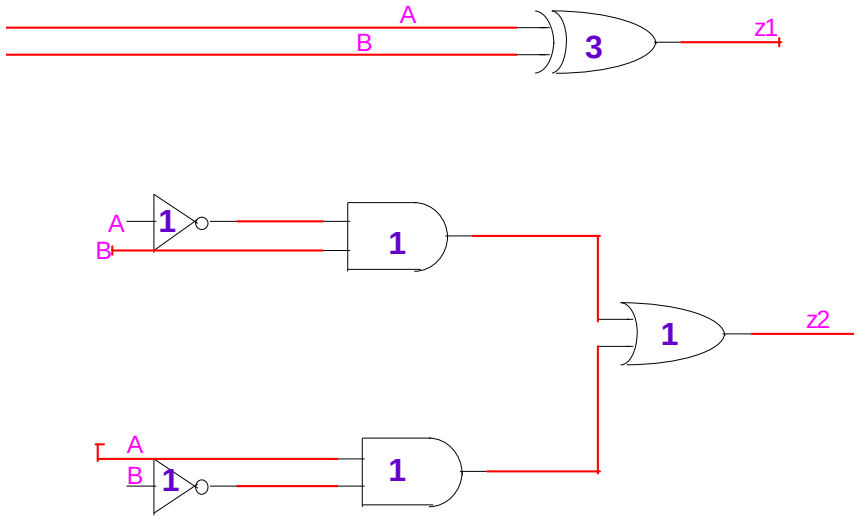


Cada camino puede tener un retardo distinto

- Nos interesa conocer el retardo máximo (o camino crítico) :
  - ¿Cuánto debemos esperar para estar seguros de que leeremos la salida correcta?
- El retardo mínimo puede ser importante
  - nos importa si no queremos que una señal cambie demasiado rápido

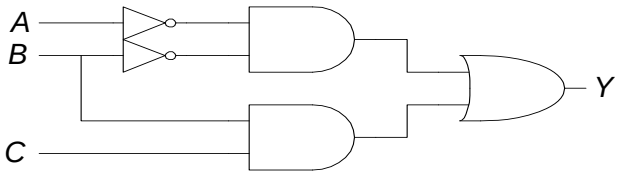


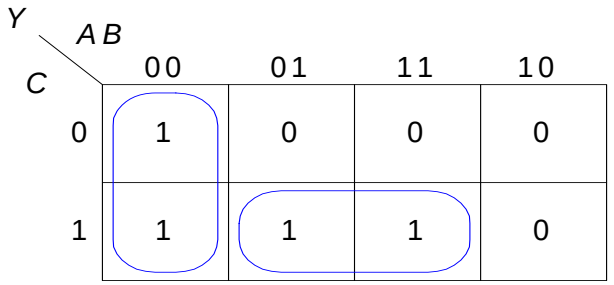
Otro ejemplo



Glitches: salidas temporalmente incorrectas

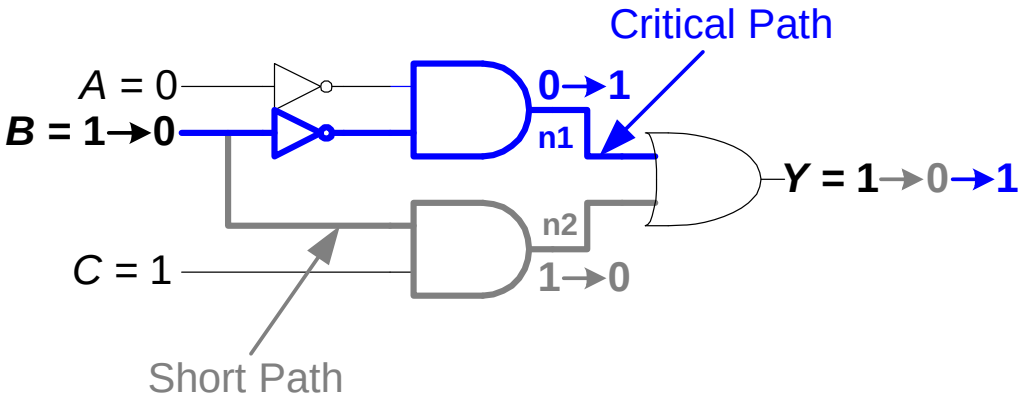
- Cada camino tiene retardos distintos
- Una entrada se puede actualizar antes en un camino que en otro generando brevemente salidas incorrectas
- Por eso debemos conocer el camino crítico





$Y = \overline{A}\,\overline{B} + BC$

Glitch Example (cont.)



Recordatorio: Biestables

- Los elementos de memoria más sencillos son los biestables.
- Se usan de forma independiente o agrupados en registros
- Vistos en Intro. a los Computadores
- Flip-flops, latches,….
- En este tema nos centramos fundamentalmente en su comportamiento temporal.

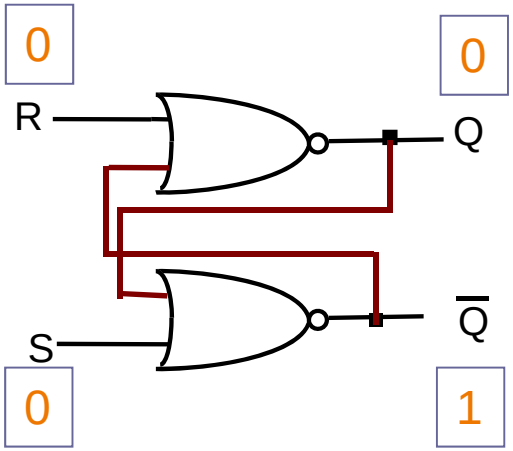
¿Cómo puede un circuito recordar?

- Los circuitos combinacionales carecen de memoria. Siempre dan la misma salida para las mismas entradas
- Un circuito secuencial es aquél capaz de recordar eventos pasados y actuar en consecuencia:
  - **Las salidas** de los circuitos secuenciales dependen de las entradas y **también de su estado**
  - **El estado** de un circuito **depende de** los valores que tomaron las entradas en **el pasado**
  - El estado de un circuito se almacena utilizando **componentes de memoria** dentro del circuito
  - Estos elementos utilizan **lazos de realimentación** para ser capaces de almacenar información

¿Cómo puede un circuito recordar?

R	S	Q
0	0	1
0	1	0
1	0	0
1	1	0

Este circuito no recuerda, tan sólo responde a las entradas



R 0

S 0

Q 0

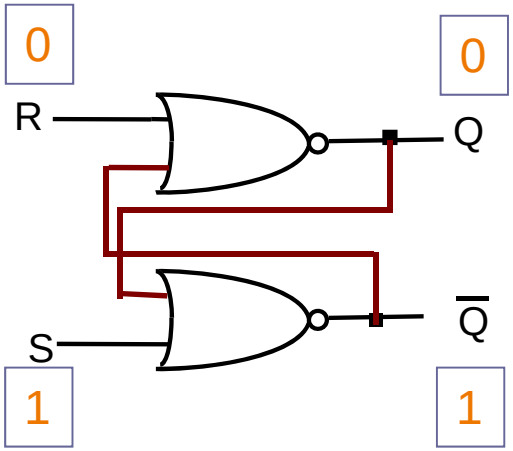
$\overline{Q}$ 1

Añadimos lazos de realimentación

¿Cómo puede un circuito recordar?

R	S	Q
0	0	1
0	1	0
1	0	0
1	1	0

Este circuito no recuerda, tan sólo responde a las entradas

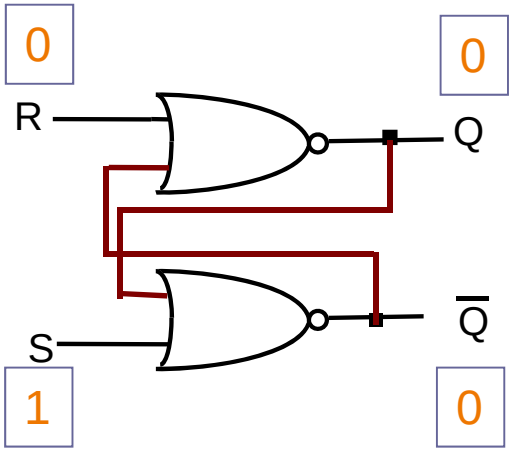


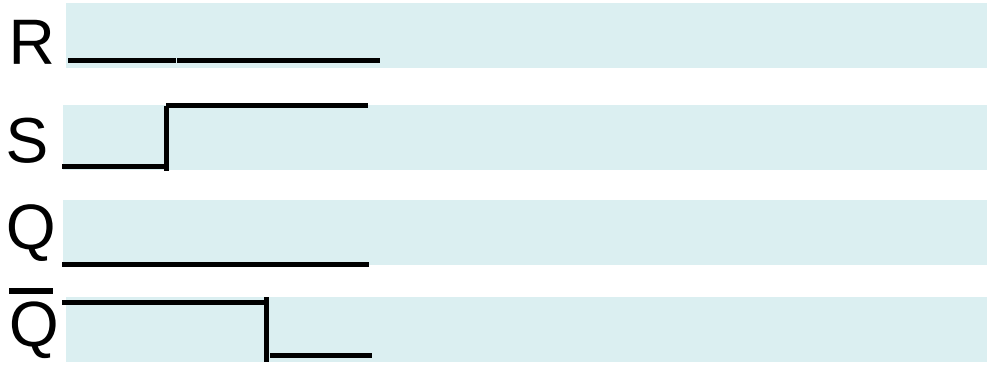
Añadimos lazos de realimentación

¿Cómo puede un circuito recordar?

R	S	Q
0	0	1
0	1	0
1	0	0
1	1	0

Este circuito no recuerda, tan sólo responde a las entradas



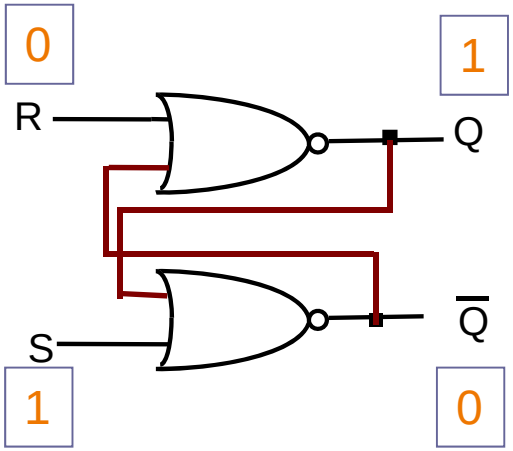


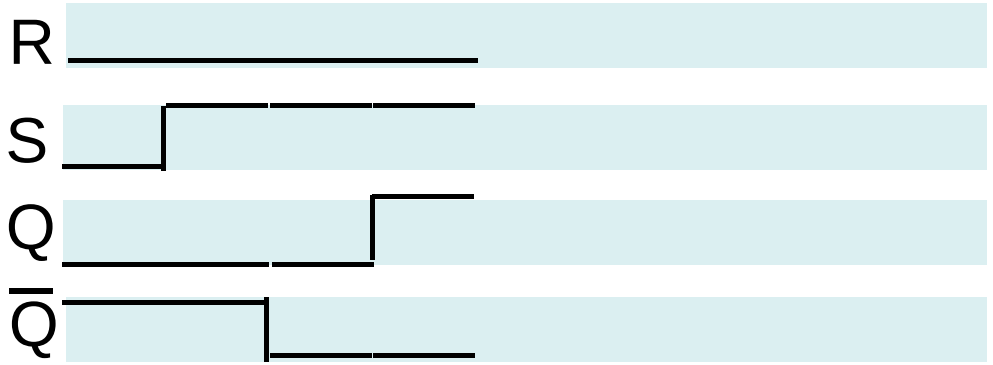
Añadimos lazos de realimentación

¿Cómo puede un circuito recordar?

R	S	Q
0	0	1
0	1	0
1	0	0
1	1	0

Este circuito no recuerda, tan sólo responde a las entradas



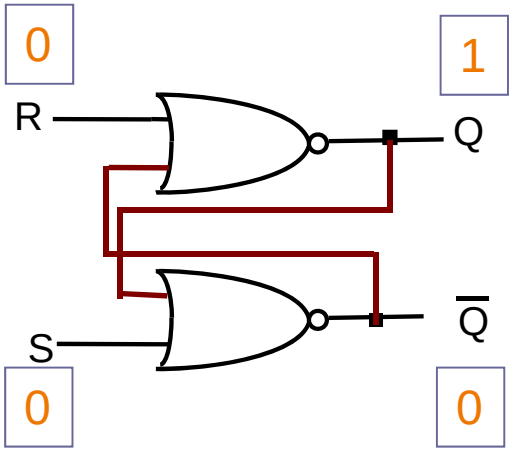


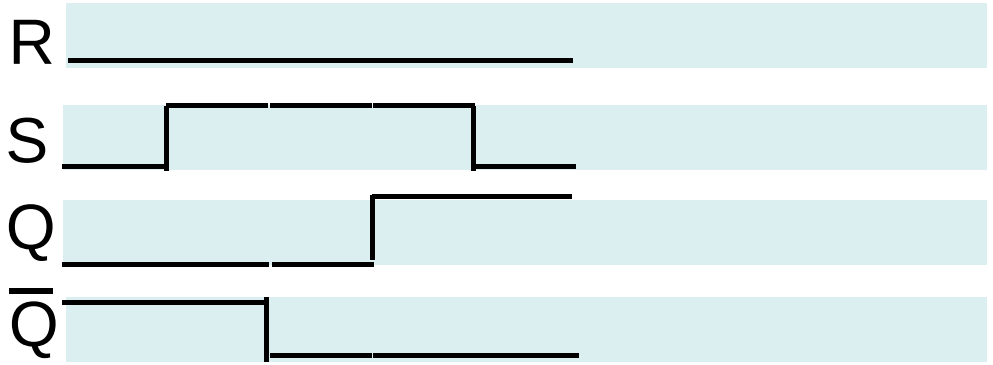
Añadimos lazos de realimentación

¿Cómo puede un circuito recordar?

R	S	Q
0	0	1
0	1	0
1	0	0
1	1	0

Este circuito no recuerda, tan sólo responde a las entradas



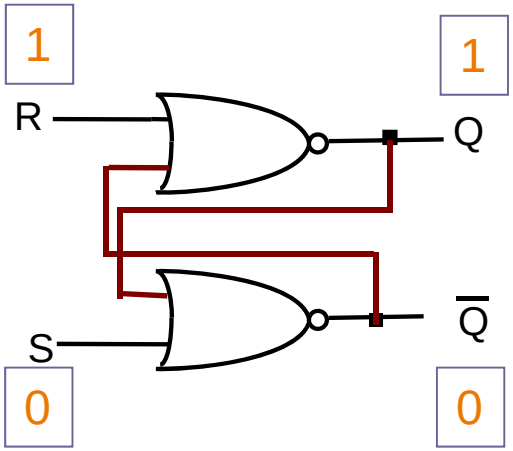


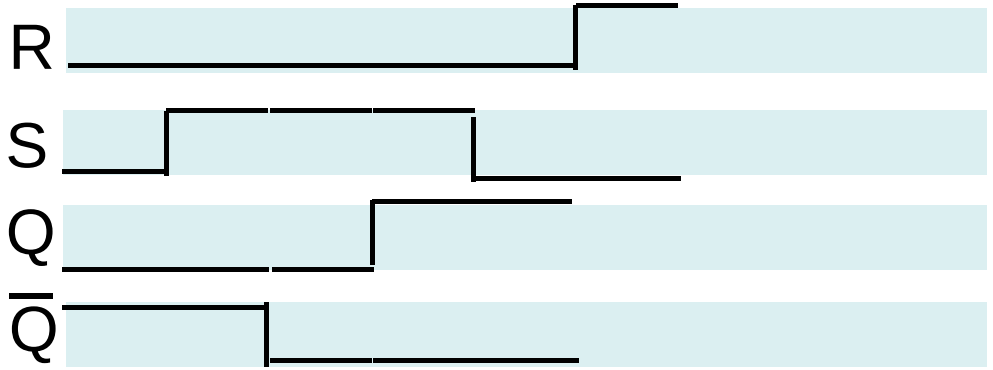
Añadimos lazos de realimentación

¿Cómo puede un circuito recordar?

R	S	Q
0	0	1
0	1	0
1	0	0
1	1	0

Este circuito no recuerda, tan sólo responde a las entradas



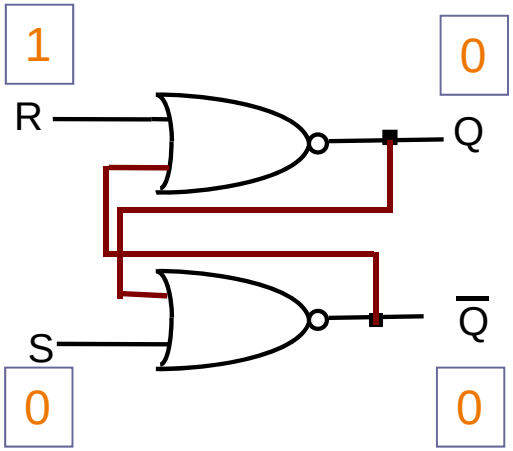


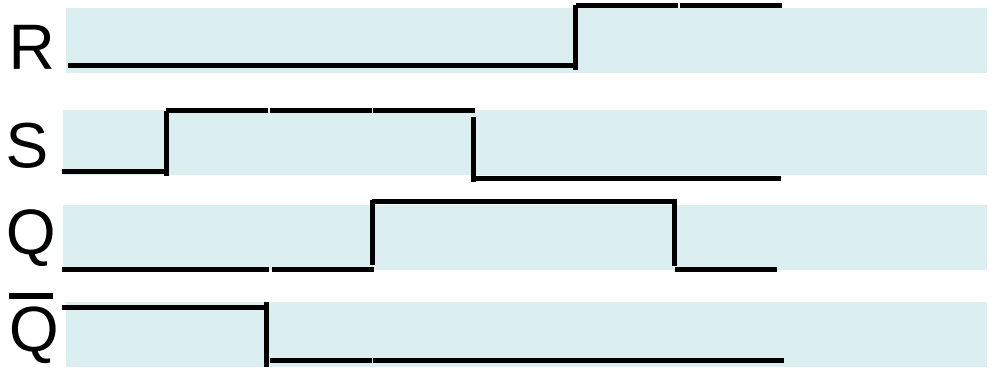
Añadimos lazos de realimentación

¿Cómo puede un circuito recordar?

R	S	Q
0	0	1
0	1	0
1	0	0
1	1	0

Este circuito no recuerda, tan sólo responde a las entradas



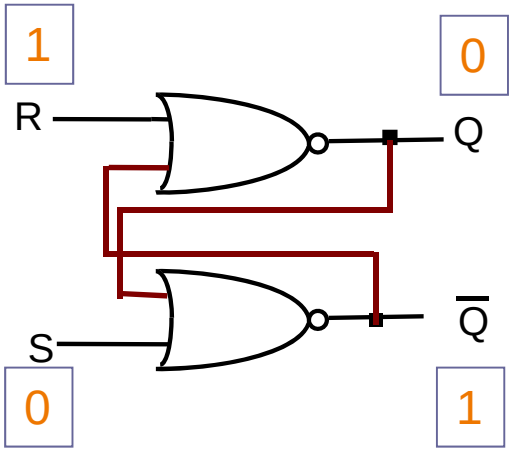


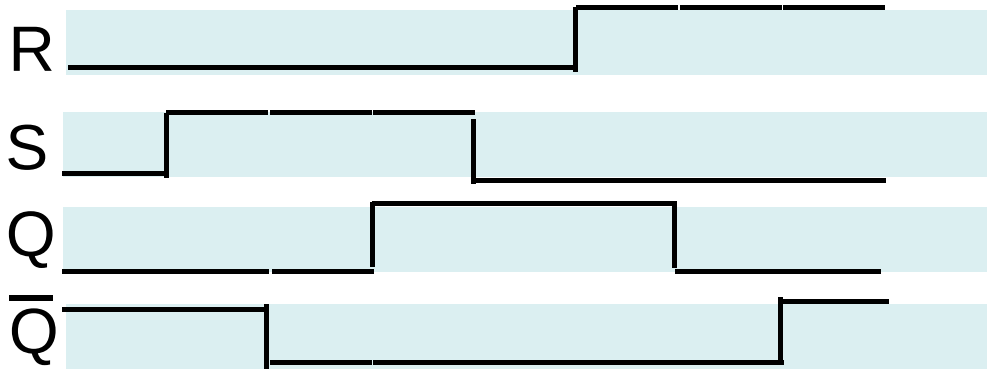
Añadimos lazos de realimentación

¿Cómo puede un circuito recordar?

R	S	Q
0	0	1
0	1	0
1	0	0
1	1	0

Este circuito no recuerda, tan sólo responde a las entradas



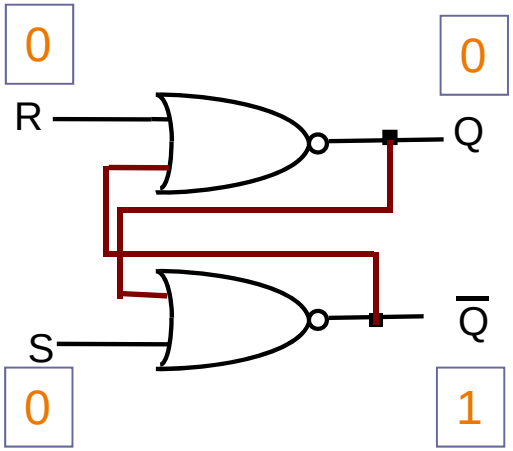


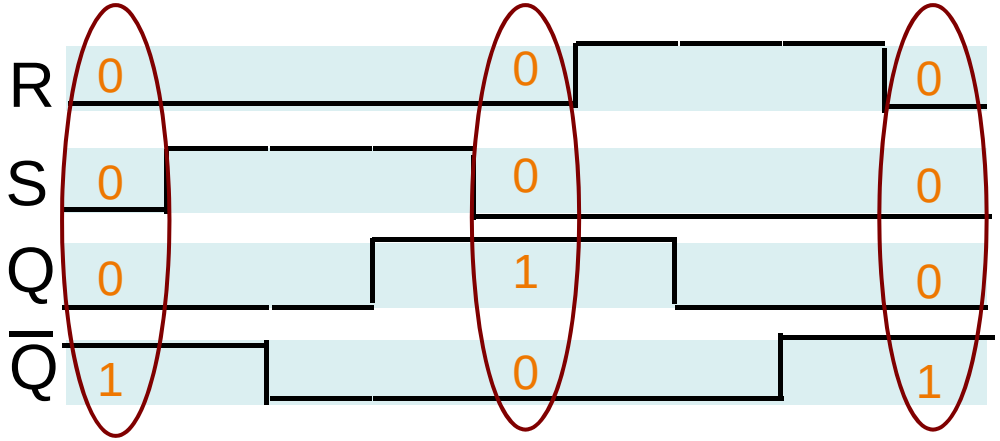
Añadimos lazos de realimentación

¿Cómo puede un circuito recordar?

R	S	Q
0	0	1
0	1	0
1	0	0
1	1	0

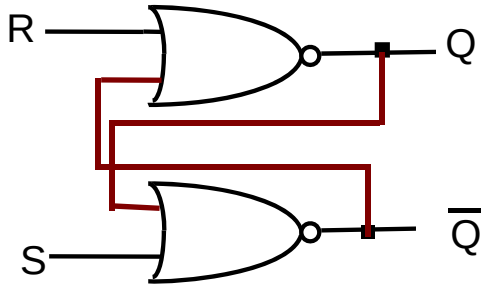
Este circuito no recuerda, tan sólo responde a las entradas



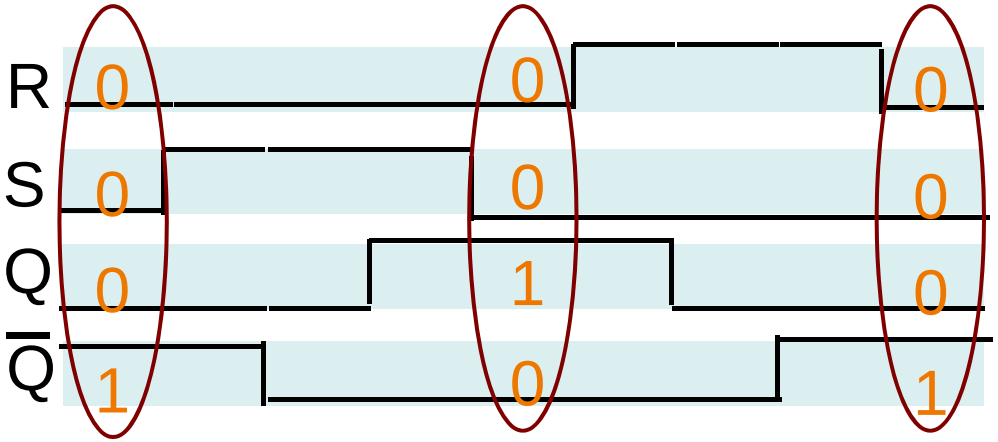


Añadimos lazos de realimentación

Un circuito secuencial pueda tener salidas distintas para las mismas entradas



latch S-R



El comportamiento de un circuito se describe en una tabla con:

- el **estado actual**
- las **entradas**

y a partir de estos datos calcular el:

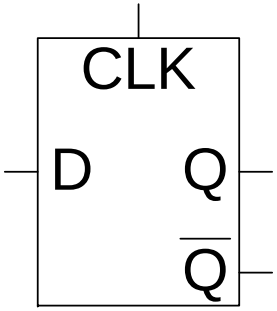
- **estado siguiente**
- **las salidas** (para este ejemplo coinciden)

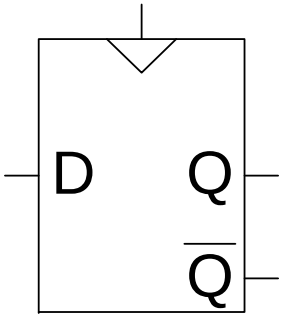
Q	R	S	Qsig
0	**0**	**0**	**0**
0	0	1	1
0	1	0	0
0	1	1	Proh
1	**0**	**0**	**1**
1	0	1	1
1	1	0	0
1	1	1	Proh

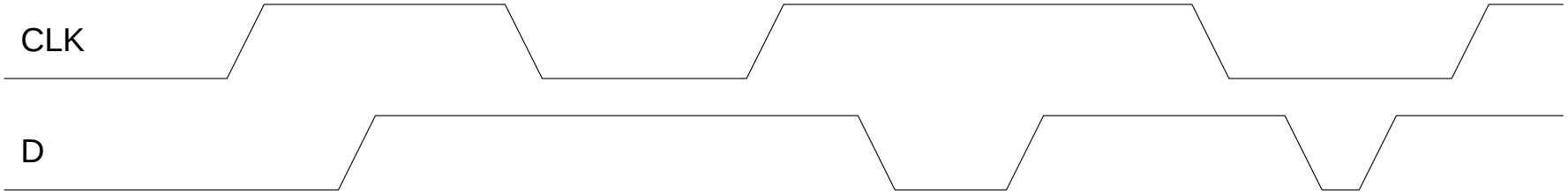
Tipos de biestables: según su comportamiento temporal

- asíncrono (latch) :
  - siempre responde a las entradas
- sensible a nivel (latch síncrono) :
  - responde a las entradas si la señal de capacitación vale 1
- disparado por flanco (flip-flop):
  - sólo lee las entradas cuando detecta un flanco (de subida o bajada) en la señal de capacitación
- maestro-esclavo (flip-flop):
  - Como el anterior pero implementado con un biestable sensible a alta y otro a baja

D Flip-Flop vs. D Latch



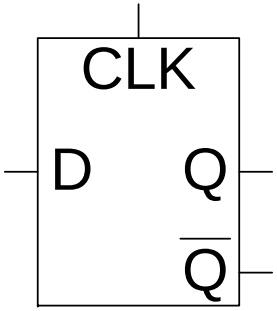


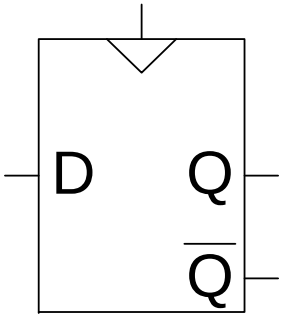


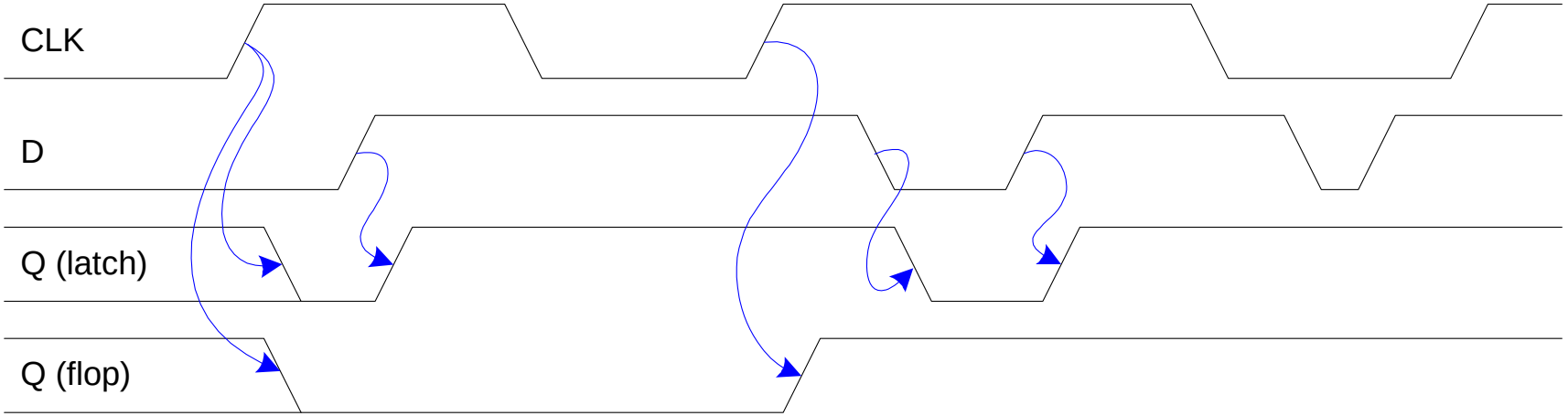
Q (latch) → Se activa por Nivel

Q (flop) → Se activa por Flanco ⇐ En este curso

D Flip-Flop vs. D Latch







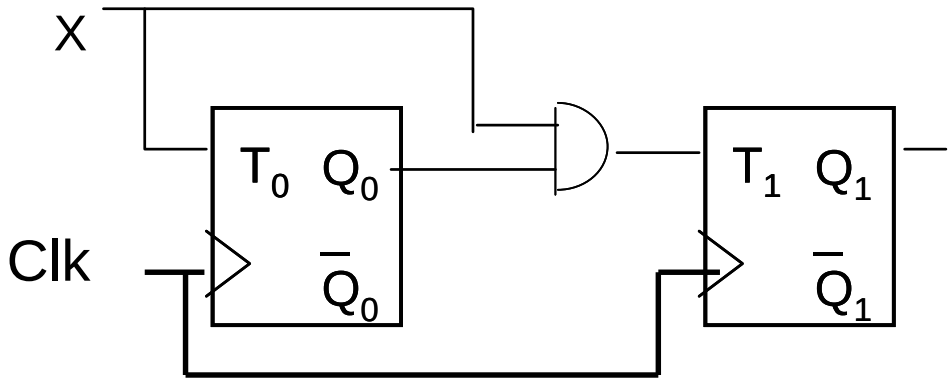
Características temporales de los biestables (y de los registros)

- Retardo de propagación
  - Desde el cambio en la entrada hasta el cambio en la salida
  - Para un biestable hay varios retardos (tantos como distintos cambios en las diferentes entradas)
- Tiempo de set-up (establecimiento)
  - Tiempo mínimo que la entrada debe permanecer estable ANTES del suceso del reloj
- Tiempo de hold (mantenimiento)
  - Tiempo mínimo que la entrada debe permanecer estable DESPUÉS del suceso del reloj

Cálculo del tiempo de ciclo

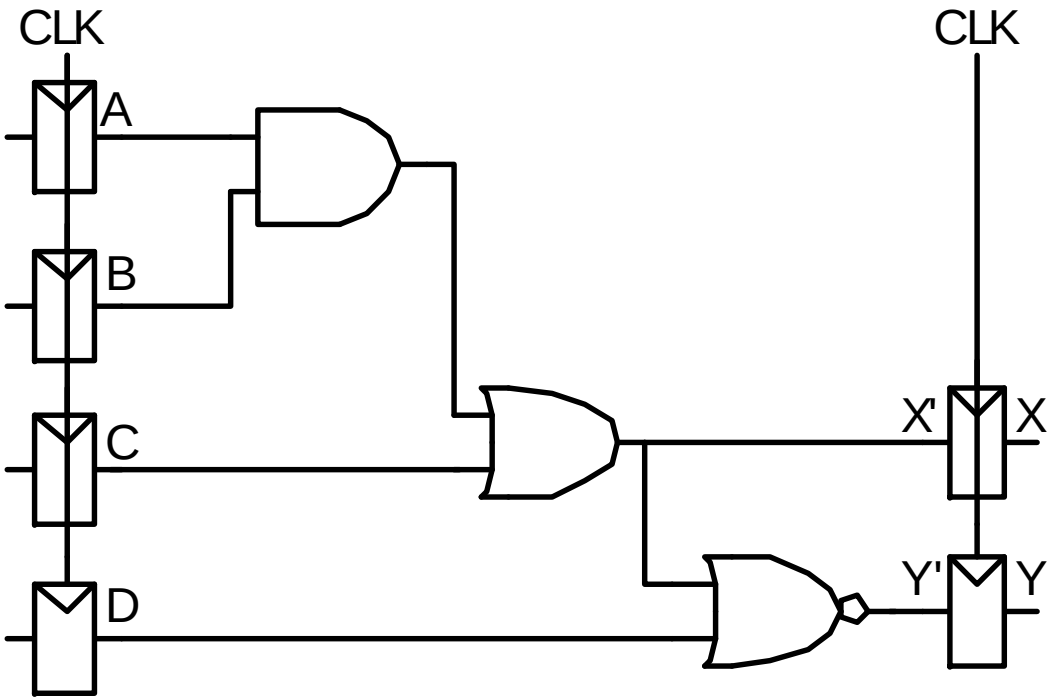
- Calcular el retardo de todos los caminos posibles y quedarnos con el mayor (el camino crítico)
- Caminos:
  - conexiones entre dos biestables,
  - o entre un biestable y una salida
- Considerar el retardo de los biestables y de la lógica combinacional.
- Objetivo:
  - Cuando llegue un nuevo flanco de reloj, todos los cambios generados por el flanco anterior deben ser estables
- Si un camino termina en un biestable ¡hay que incluir el tiempo de set-up!

Cálculo del tiempo de ciclo



Retardo de los biestables (Tdelay): 6ns
Retardo puerta AND: 2 ns
Tset_up: 3ns

Cálculo del tiempo de ciclo



Retardo de los registros(Tdelay): 6ns
Retardo puertas: 2 ns
Tset_up: 3ns

Resumen de los biestables

- Los biestables permiten a los circuitos recordar
- El valor de los biestables en cada instante define el estado del circuito
- En un circuito síncrono los biestables deben actualizarse todos a la vez, y tan sólo una vez por ciclo de reloj
- Por ello es recomendable utilizar biestables disparados por flanco
- Al diseñar un circuito se debe asignar el tiempo de ciclo adecuado para que todos los biestables puedan actualizarse correctamente

Comportamiento temporal de las memorias

Las memorias tienen muchos parámetros temporales y su análisis en detalle es complejo, veamos un modelo muy simplificado

- Modo Lectura
  - Se comporta como un sistema combinacional con tiempo de acceso de lectura
- Modo Escritura
  - Se comporta como un registro: hay que dar tiempo a que se escriban las cosas (~setup)
  - Preparo todas las entradas
  - Espero tiempo de acceso en escritura

Práctica 1

Diseño de un banco de registros

(2 sesiones de laboratorio)

Objetivos

- Aprender a realizar diseños modulares.
  - Vamos a realizar un diseño complejo a partir de módulos estándar: muxes, dec y registros.
- Realizar análisis temporal de un módulo síncrono complejo:
  - tiempo de *setup*
  - retardo de propagación
  - Tiempo de ciclo/frecuencia máxima

Primera sesión: **diseño de un banco de registros**

- Utilizaréis módulos estándar
- **Trabajo previo**:
  - Aprender cómo funciona Logisim:
    - Primeros pasos con Logisim
    - Logisim Beginner’s tutorial
  - Realizad el diseño en papel.
- Resultados:
  - Comprobar que el diseño funciona
  - Entender los retardos de cada operación

Segunda sesión: **Estudio del tiempo de ciclo**

- Integraréis vuestro diseño previo en un procesador sencillo
- Dos versiones:
  - un procesador que realiza todas las operaciones en un ciclo de reloj
  - El mismo procesador pero con algunos registros intermedios
- Debéis calcular el tiempo de ciclo mínimo para que el sistema funcione