Unidad 8. Circuitos secuenciales síncronos

Circuitos Electrónicos Digitales
E.T.S.I. Informática
Universidad de Sevilla

Jorge Juan <jjchico@dte.us.es> 2010-2019

Contenidos

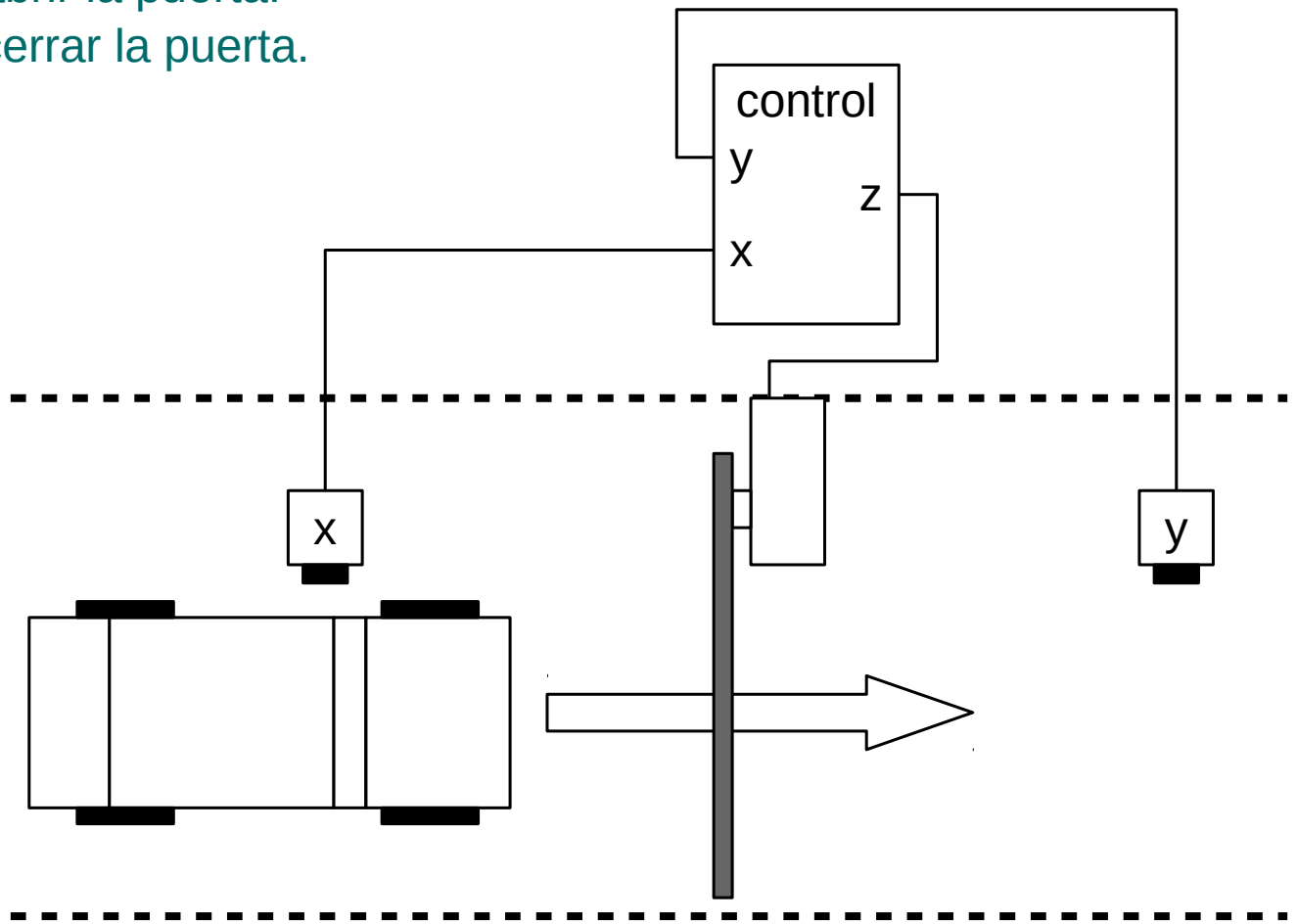
- Introducción
- Biestables (latches)
- Circuitos Secuenciales Síncronos (CSS) y Máquinas de Estados Finitos (MEF)
- Diseño de MEF
- Análisis de CSS

Lecturas y ejercicios recomendados

- Contenidos teóricos:
  - LaMeres: 7.1, 7.2, 7.4, 7.6, 7.7
- Modelado en Verilog
  - LaMeres: 9.1, 9.2, 9.3
  - curso-verilog.v: unidad 6
- Ejercicios recomendados del boletín general (boletín 7):
  - Ej. 15. Detector de paridad por grupos. Debe diseñarse como máquina de Mealy. ¿Por qué?
  - Ej. 17. Complementador a 2 secuencial.
  - Análisis funcional: 5.
  - Análisis temporal sin retrasos: 7, 8.

Introducción. Ejemplo 1

- Diseñe un sistema de control de una puerta de un garaje con dos pulsadores separados por una distancia.
  - x: abrir la puerta.
  - y: cerrar la puerta.



Introducción

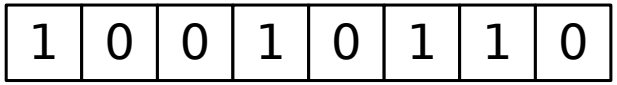
- La mayoría de los problemas prácticos no pueden resolverse usando sólo circuitos combinacionales.
- En muchos casos, la acción a realizar depende de las entradas y del “estado” del sistema: la puerta está abierta, la luz está encendida, etc.
- Para “almacenar” el estado del sistema se necesitan nuevos elementos de circuito: elementos de memoria.
- En esta unidad:
  - Elementos de memoria.
  - Concepto de “estado” y “circuito secuencial síncrono”.
  - Técnicas de diseño y análisis de circuitos secuenciales síncronos.

Contenidos

- Introducción
- **Biestables (latches)**
  - Biestables
  - Biestable SR asíncrono
  - Biestables síncronos
  - Entradas asíncronas
  - Temporización
- Máquinas de estados finitos y circuitos secuenciales síncronos (CSS)
- Diseño de CSS
- Análisis de CSS

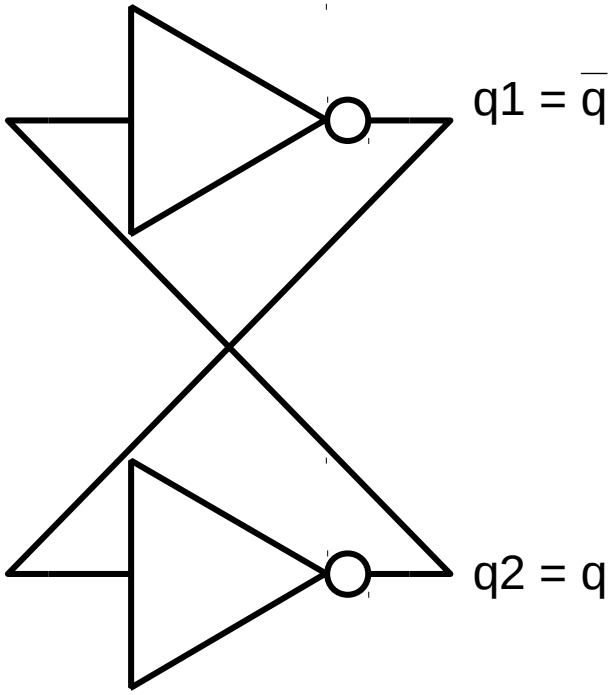
Biestables (*latches*)

- Los circuitos biestables son circuitos que pueden ser permanecer en uno de dos posibles estados estables.
- El estado del biestable se puede conocer por el valor de señales de salida del circuito.
- El estado puede cambiarse actuando sobre entradas de control.
- Los biestables son elementos básicos de memoria: almacenan 1 bit.
- Con n biestables pueden “memorizarse” 2^n estados posibles.



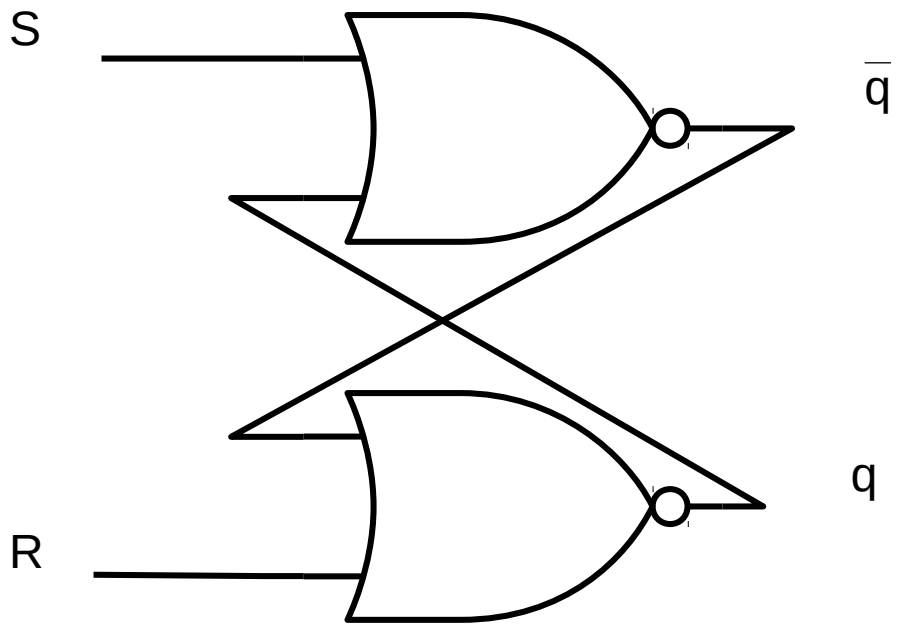
Biestable SR asíncrono

- La capacidad de almacenar información en los biestables se obtiene a menudo usando “realimentación” de las salidas a las entradas del circuito: el valor de salida refuerza el valor de entrada y viceversa.
- Posibles estados estables:
  - q1=0, q2=1
  - q1=1, q2=0
- Notación:
  - q = q2
  - $\overline{q}$ = q1



Biestable SR asíncrono

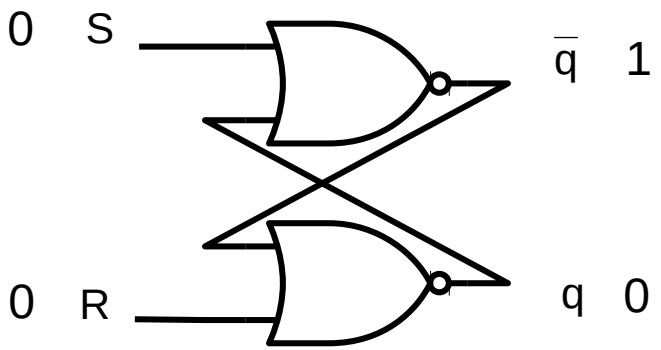
- Además de tener dos estados estables necesitamos una forma simple de forzar cada estado.

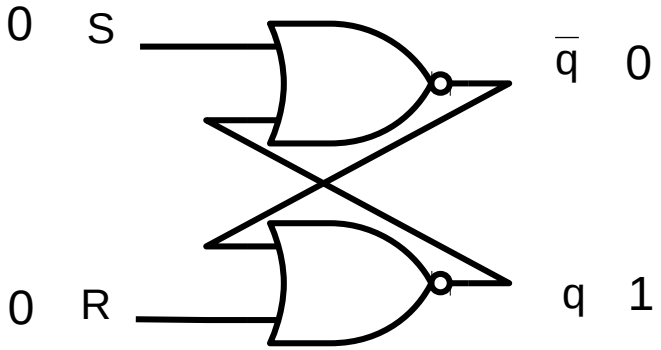


Biestable SR asíncrono

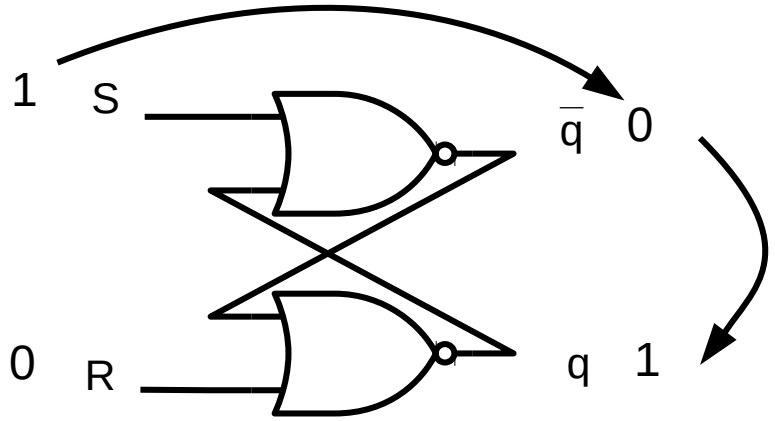
- R=S=0 se conserva el estado

Simulación

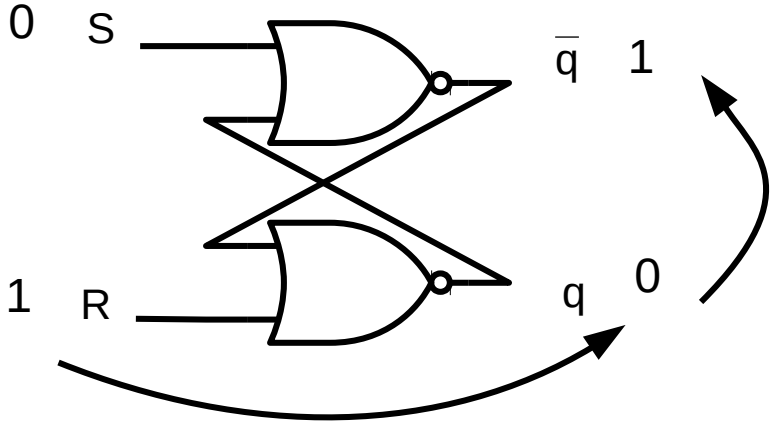




- S=1, R=0 cambio a 1 (*set*)

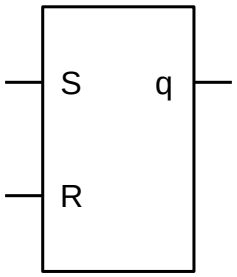


S=0, R=1 cambio a 0 (*reset*)



Biestable SR. Descripciones formales

Símbolo



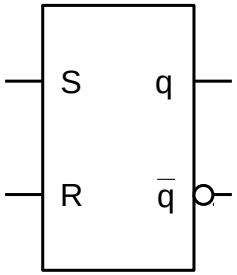


Tabla de estados

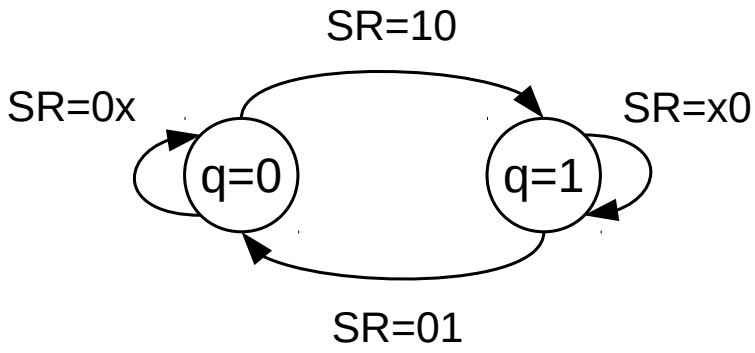
q \ SR	00	01	11	10
0	0	0	-	1
1	1	0	-	1

Q

Tabla de excitación

q → Q	SR
0 → 0	0x
0 → 1	10
1 → 0	01
1 → 1	x0

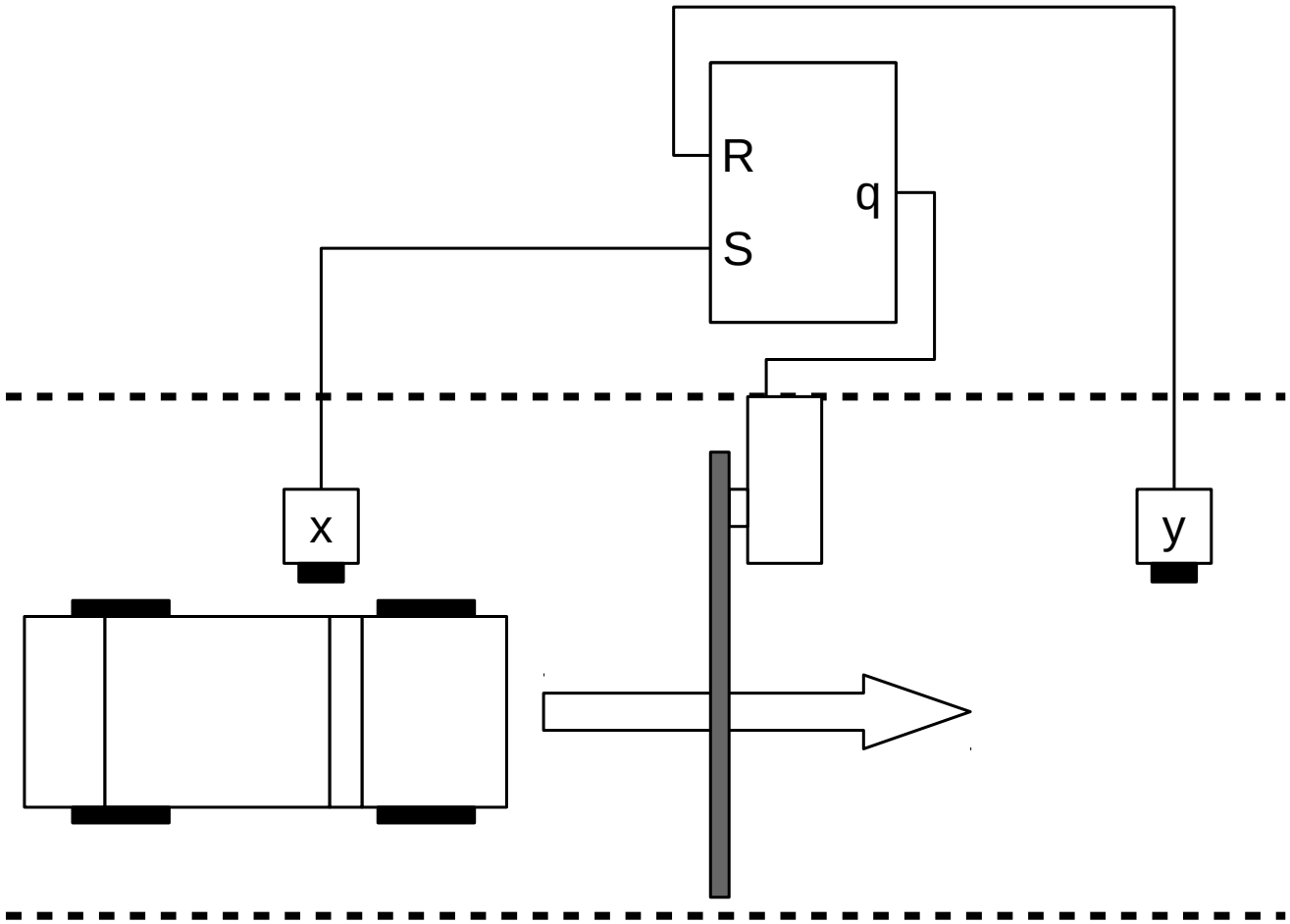
Diagrama de estados



Verilog

```
module sra(
  input wire s,
  input wire r,
  output reg q);

  always @(s, r)
    case ({s, r})
      2'b01: q <= 1'b0;
      2'b10: q <= 1'b1;
      2'b11: q <= 1'bx;
    endcase
endmodule
```

Ejemplo 1: solución

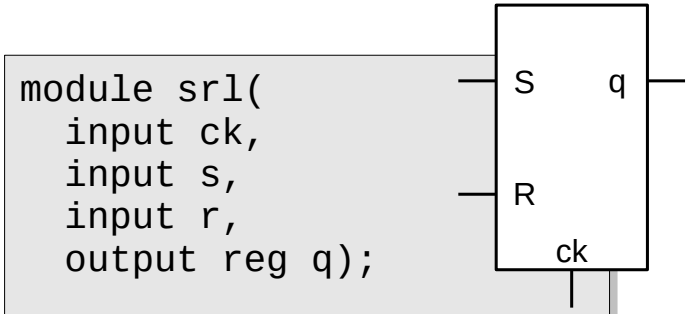


Biestables síncronos

- Los circuitos reales pueden contener miles de millones de biestables. Es muy útil poder controlar cuándo cambian de estado y hacer que lo hagan al mismo tiempo.
- Para ello, los cambios de estado se “sincronizan” con una “señal de reloj” (CK, CLK, MCLK, etc.)
- La señal de reloj suele ser periódica con cambios a una frecuencia prefijada: “frecuencia de reloj”.
- Biestables disparados por nivel (*gated latches*)
  - El cambio de estado sólo es posible cuando la señal de reloj es 1 (nivel alto) o 0 (nivel bajo).
- Biestables disparados por flanco (*flip-flops*)
  - El cambio de estado sólo es posible cuando la señal de reloj cambia de 0 a 1 (flanco de subida) o de 1 a 0 (flanco de bajada).
  - El cambio de estado se determina con precisión.
  - Hace más simple y robusto el diseño de circuitos digitales.

Biestables síncronos

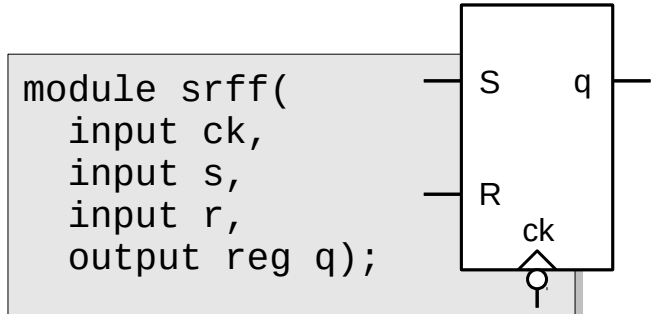
Biestable disparado por nivel




```
module srl(
  input ck,
  input s,
  input r,
  output reg q);

  always @(ck, s, r)
    if (ck == 1)
      case ({s, r})
        2'b01: q <= 1'b0;
        2'b10: q <= 1'b1;
        2'b11: q <= 1'bx;
      endcase
endmodule
```

Cambio de estado cuando ck=1

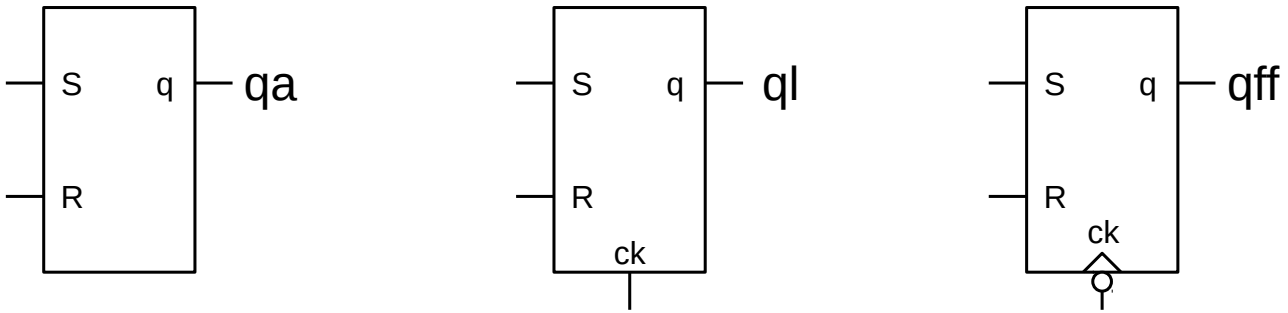
Biestable disparado por flanco

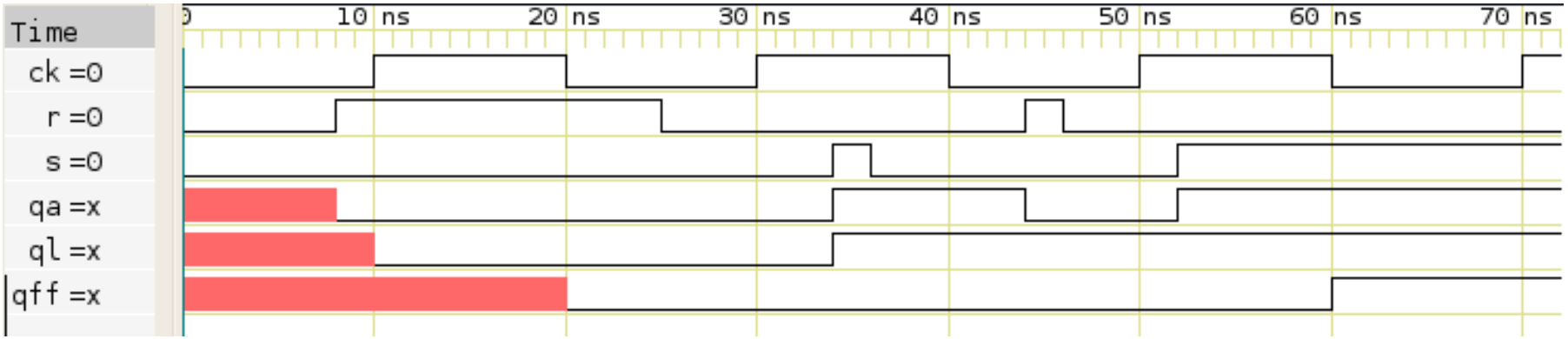



```
module srff(
  input ck,
  input s,
  input r,
  output reg q);

  always @(negedge ck)
    case ({s, r})
      2'b01: q <= 1'b0;
      2'b10: q <= 1'b1;
      2'b11: q <= 1'bx;
    endcase
endmodule
```

Cambio de estado cuando ck cambia de 1 a 0

Biestables síncronos



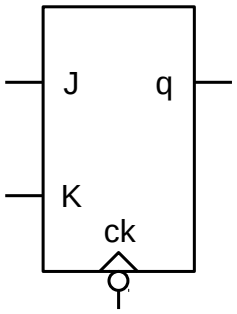


Otros biestables disparados por flanco (flip-flops)

- SR
  - Diseño simple
  - Resultado indeterminado para S=R=1
- JK
  - Similar al SR: J~S, K~R
  - Operación de complementar para J=K=1
- D
  - Única entrada, valor del próximo estado.
  - Fácil de usar e implementar.
- T
  - Única entrada, complementa el estado cuando es 1.
  - Muy útil en algunas aplicaciones especiales: ej. contadores.

Biestable JK

Símbolo



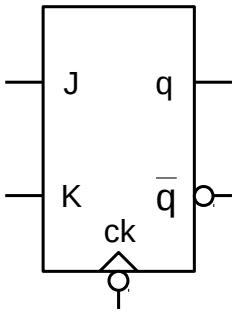


Tabla de estados

q \ JK	00	01	11	10
0	0	0	1	1
1	1	0	0	1

Q

Diagrama de estados

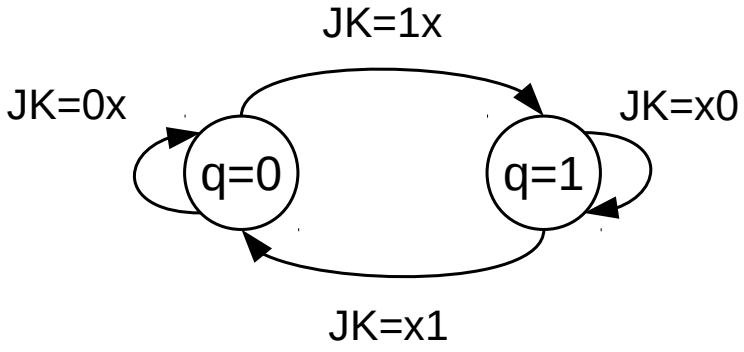


Tabla de excitación

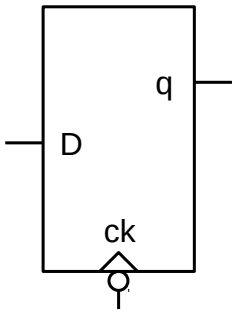
q → Q	JK
0 → 0	0x
0 → 1	1x
1 → 0	x1
1 → 1	x0

Verilog

```
module jkff(
  input ck,
  input j,
  input k,
  output reg q);

  always @(negedge ck)
    case ({j, k})
      2'b01: q <= 1'b0;
      2'b10: q <= 1'b1;
      2'b11: q <= ~q;
    endcase
endmodule
```

Biestable D

Símbolo



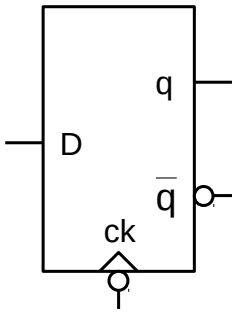


Tabla de estados

q \ D	0	1
0	0	1
1	0	1

Q

Diagrama de estados

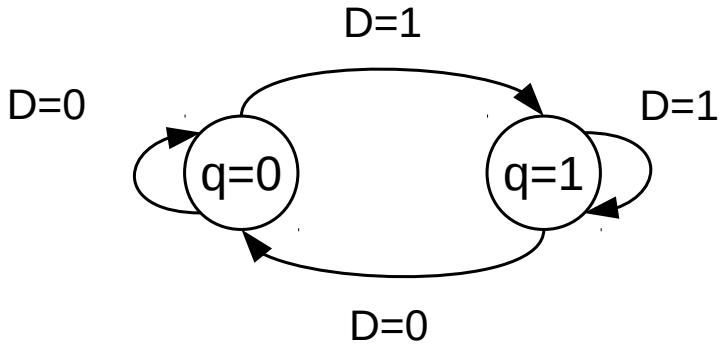


Tabla de excitación

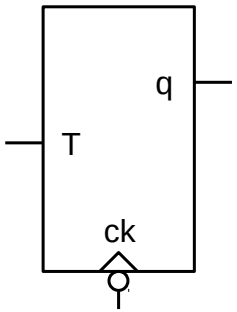
q → Q	D
0 → 0	0
0 → 1	1
1 → 0	0
1 → 1	1

Verilog

```
module dff(
  input ck,
  input d,
  output reg q);

  always @(negedge ck)
    q <= d;

endmodule
```

Biestable T

Símbolo



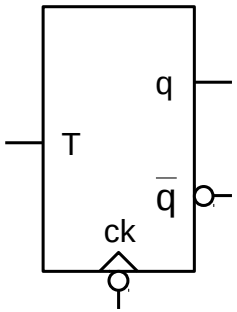


Tabla de estados

T / q	0	1
0	0	1
1	1	0

Q

Diagrama de estados

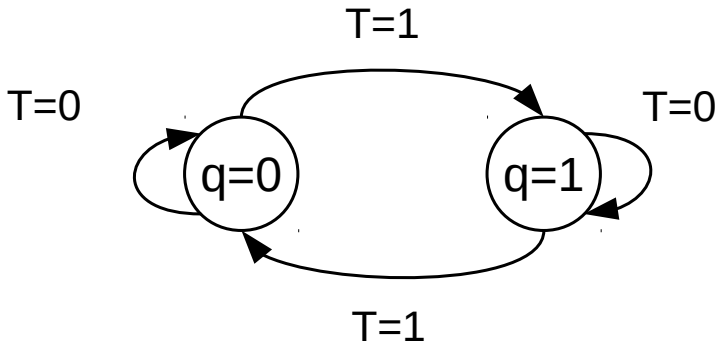


Tabla de excitación

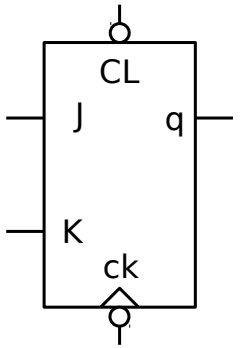
q → Q	T
0 → 0	0
0 → 1	1
1 → 0	1
1 → 1	0

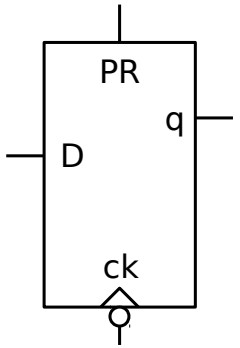
Verilog

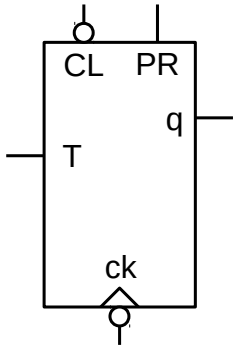
```
module tff(
  input ck,
  input t,
  output reg q);

  always @(negedge ck)
    if (t == 1)
      q <= ~q;

endmodule
```

Entradas asíncronas

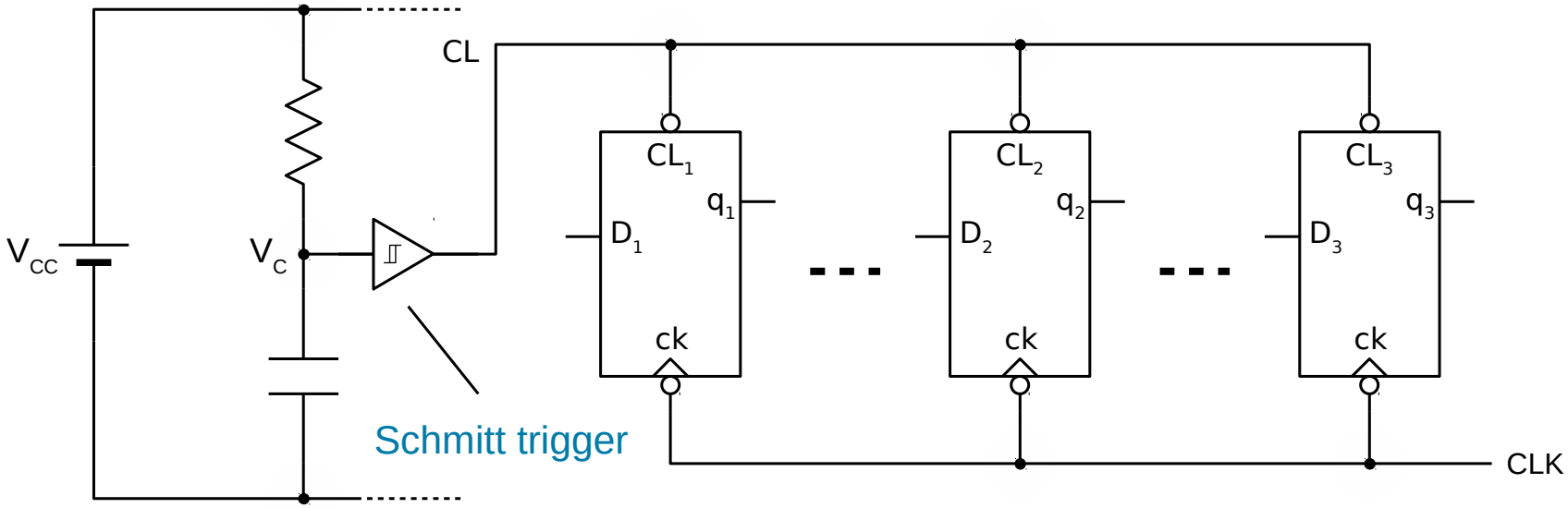
- Forma fácil y directa de forzar un estado
  - CL (clear): puesta a 0
  - PR (preset): puesta a 1
- Efecto inmediato tras su activación:
  - Activa en nivel bajo (0)
  - Activa en nivel alto (1)
- Mayor prioridad que las entradas síncronas
  - J, K, D, T, ...
- Resuelven el problema de la inicialización en sistemas digitales complejos
  - Millones de biestables.
  - Necesidad de comenzar a operar desde un estado conocido.

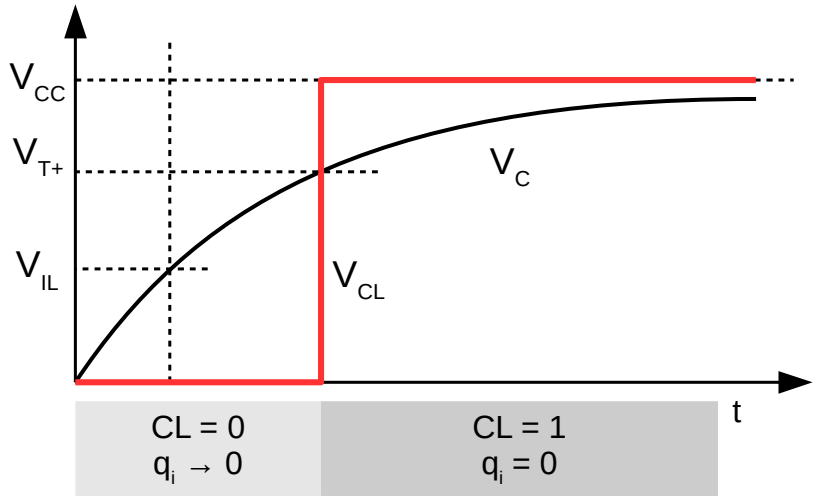




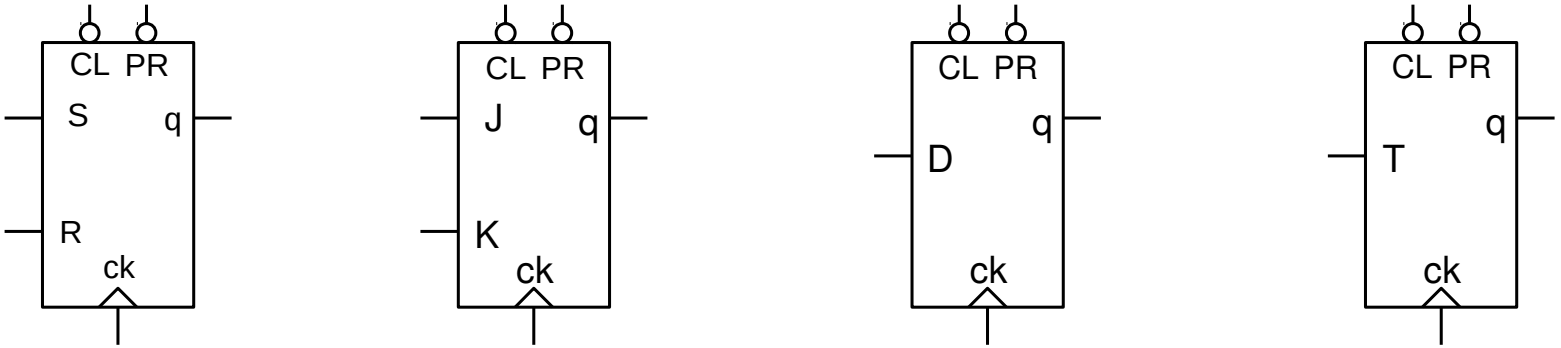


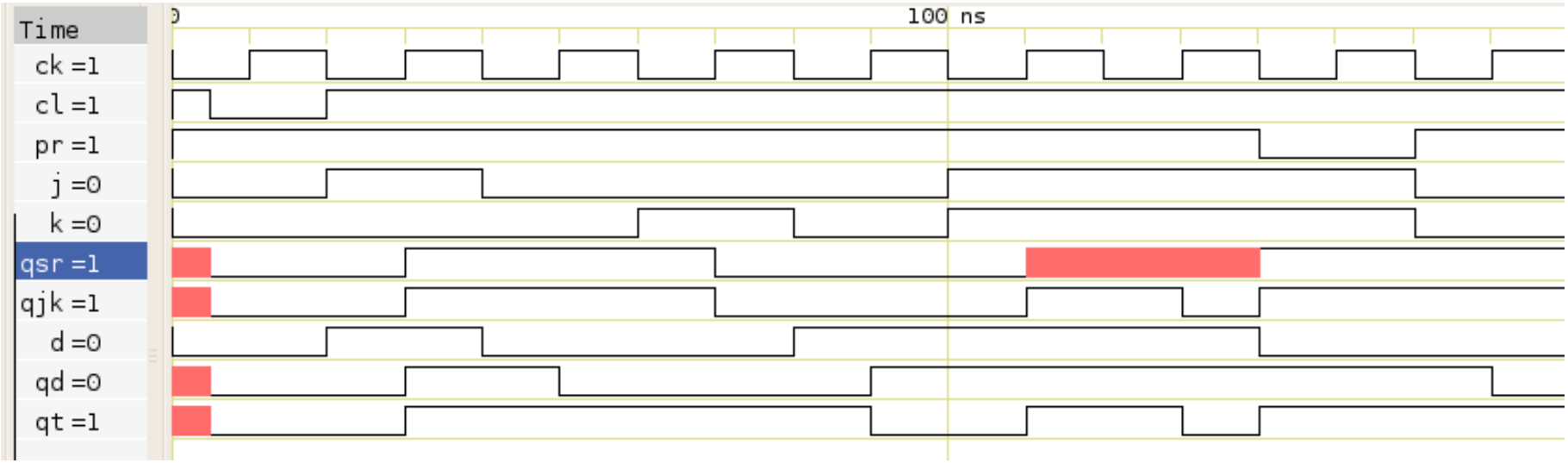
Entradas asíncronas
Circuito de inicialización





Entradas asíncronas

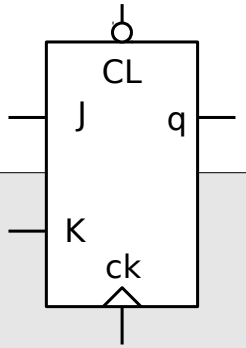




S=J, R=K, T=D

Entradas asíncronas
Ejemplo Verilog

- Biestable JK
  - Disparado por flanco de subida
  - Clear (CL) activo en nivel bajo
- ¿Por qué “negedge cl”?

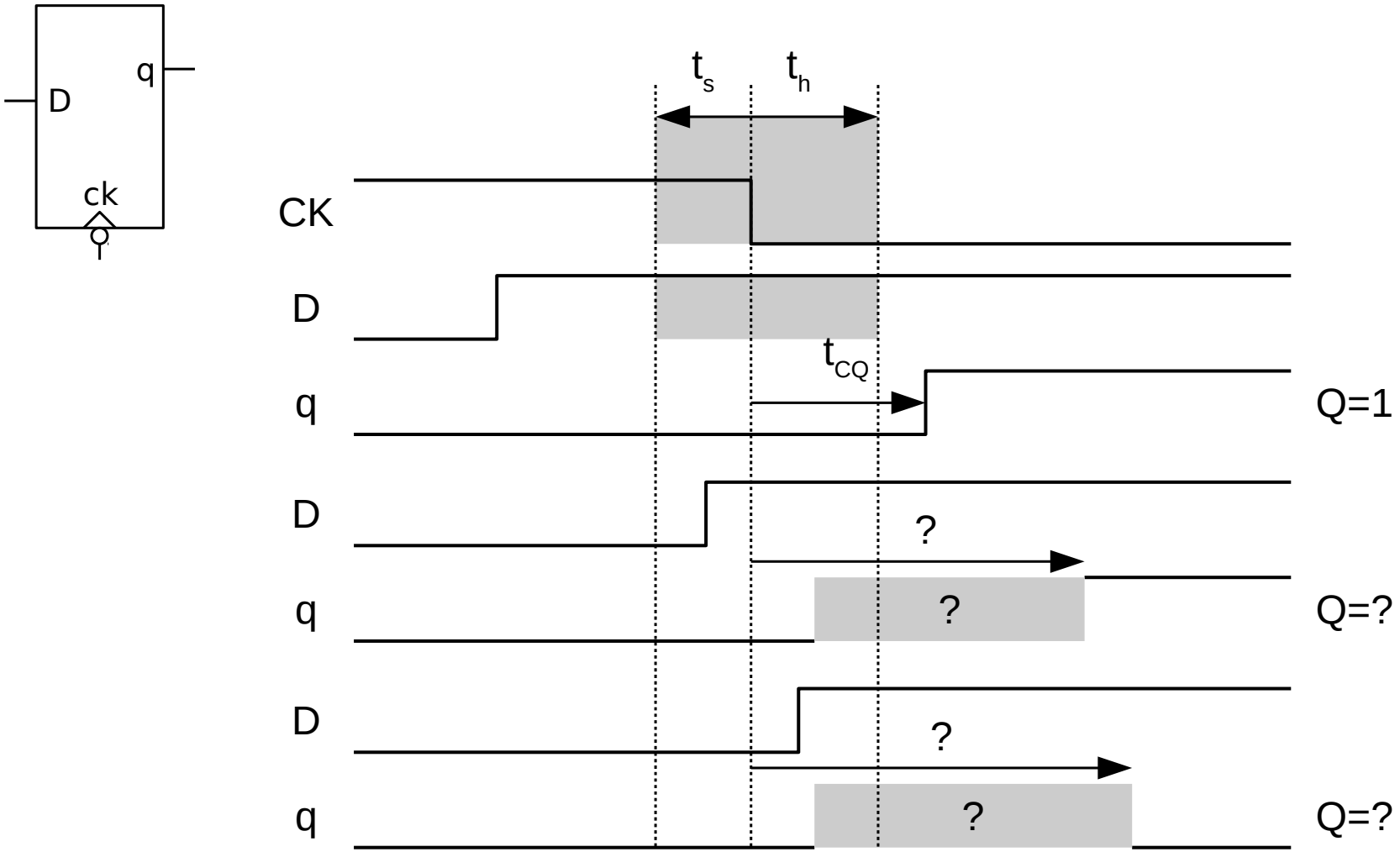



```
module jkff(
  input ck,
  input j,
  input k,
  output reg q);

  always @(posedge ck, negedge cl)
    if (cl == 1'b0)
      q <= 1'b0;
    else
      case ({j, k})
        2'b01: q <= 1'b0;
        2'b10: q <= 1'b1;
        2'b11: q <= ~q;
      endcase
endmodule
```

Restricciones temporales

- Igual que las puertas lógicas, los biestables síncronos presentan un retraso de propagación desde el flanco de reloj a la salida: t_{CQ}
- Las entradas síncronas no deben cambiar en las proximidades del flanco activo del reloj para evitar un cambio de estado no deseado.
  - **Tiempo de Set-up** (t_s)
    - Tiempo antes del flanco durante el que las entradas deben permanecer estables.
  - **Tiempo de Hold time** (t_h)
    - Tiempo después del flanco durante el que las entradas deben permanecer estables.
- Un cambio de las entradas en la zona prohibida hará que la salida:
  - Cambie a un valor no determinado a priori.
  - Tarde un tiempo indeterminado en cambiar: estado **metaestable**.
  - Son situaciones que, en general, se prefiere evitar.

Restricciones temporales. Ejemplo



Contenidos

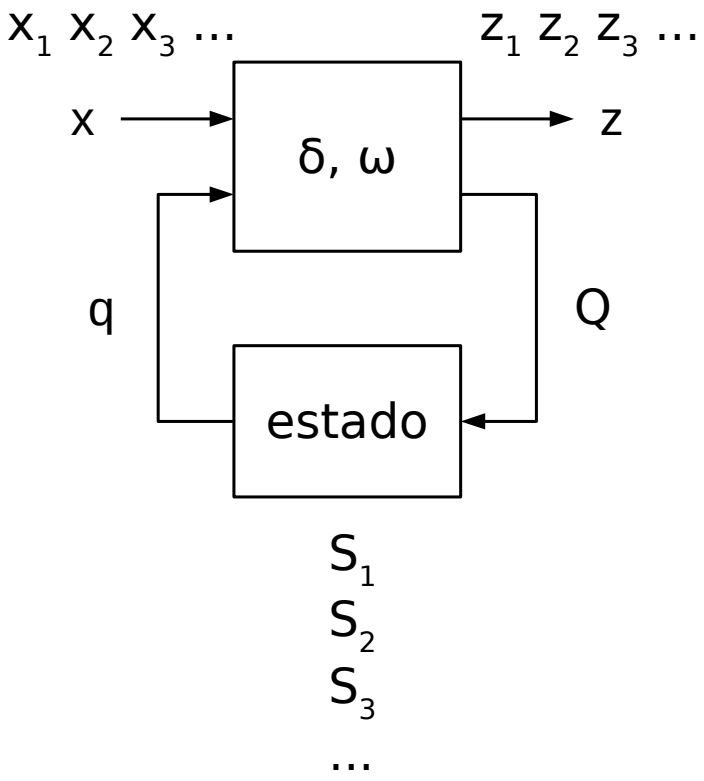
- Introducción
- Biestables (latches)
- **Circuitos Secuenciales Síncronos (CSS) y Máquinas de Estados Finitos (MEF)**
  - Circuitos secuenciales síncronos
  - Máquinas de estados finitos
  - Aplicaciones
  - Representaciones formales
- Diseño de CSS
- Análisis de CSS

Circuitos secuenciales síncronos (CSS)

- Combinando bloques combinacionales y biestables se construyen circuitos secuenciales que operan en función de las entradas y de un estado almacenado.
- En los circuitos secuenciales síncronos, todos los biestables cambian su estado simultáneamente, controlados por una señal de reloj periódica (en el flanco de subida o bajada del reloj).
- Esta restricción (sincronismo con el reloj) permite:
  - Simplificar el proceso de diseño de circuitos digitales secuenciales.
  - Poder establecer procedimientos de diseño sistemáticos: pueden ser programados y ejecutados automáticamente por herramientas software.
  - Hacer que los circuitos sean más robustos frente a variaciones en los componentes y las señales.
- La frecuencia de la señal de reloj se convierte en un factor esencial del rendimiento del circuito:
  - La frecuencia de reloj determina el número de operaciones por segundo que puede hacer el circuito.
  - La frecuencia máxima de reloj viene determinada por el retraso de los componentes y líneas de interconexión en el circuito.

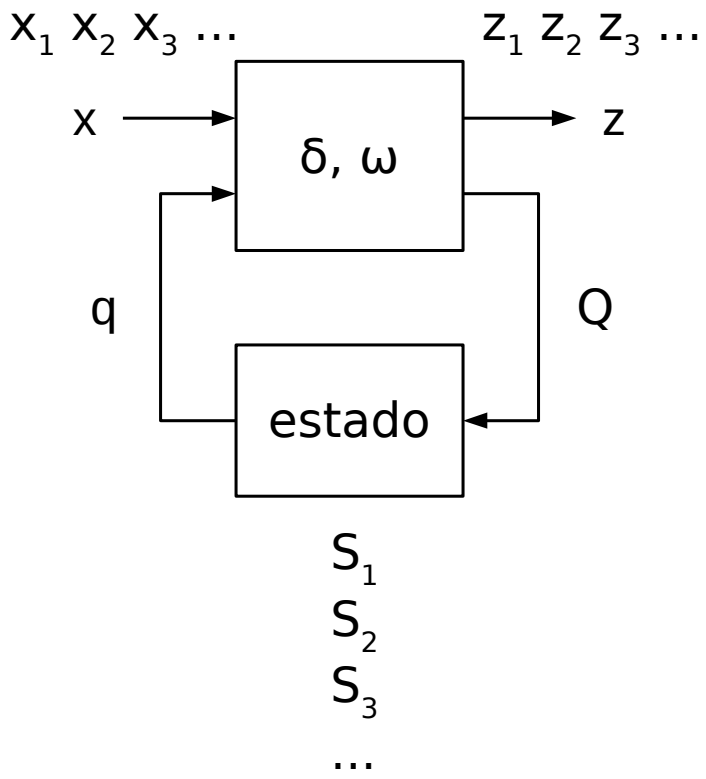
Máquinas de Estados Finitos (MEF)

- Las MEF son una herramienta útil para describir muchos tipos de problemas, incluyendo los CSS.
- Una MEF está formada por:
  - Un conjunto finito de estados (S)
  - Un conjunto de símbolos de entrada (Σ)
  - Un conjunto de símbolos de salida (Γ)
  - Una función de próximo estado (δ)
    - Q = δ(q, x)
  - Una función de salida (ω)
    - Máquina de Mealy: z = ω(q, x)
    - Máquina de Moore: z = ω(q)



Máquinas de Estados Finitos (MEF)
Operación

- Llega un nuevo símbolo a la entrada (x)
- Se calcula un nuevo símbolo de salida
  - $z = \omega(q, x)$
- Se calcula el valor del próximo estado de la máquina
  - $Q = \delta(q, x)$
- Se almacena el nuevo estado
- Se repite la operación

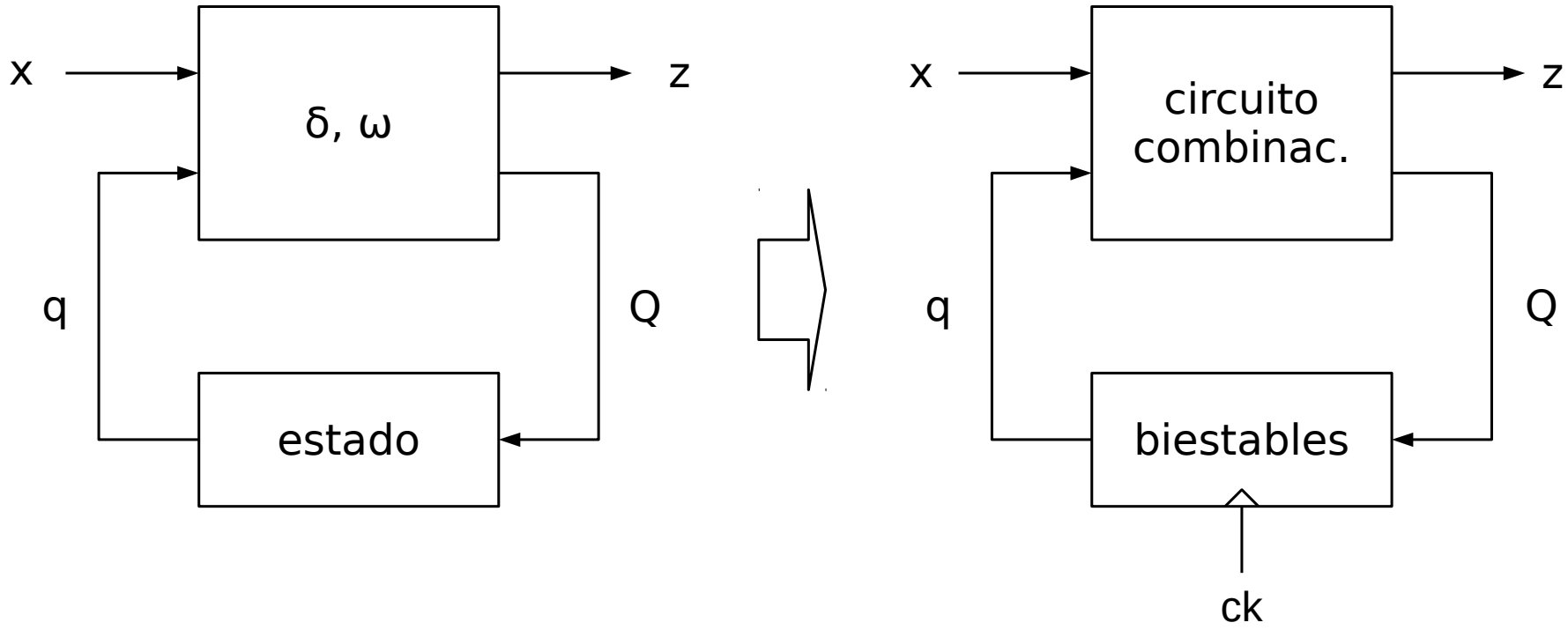


Máquinas de Estados Finitos (MEF) Propiedades

- Comenzando en el mismo estado, las MSF deterministas siempre generan la misma secuencia de salida para la misma secuencia de entrada.
- Dos MSF son equivalentes si generan la misma secuencia de salida para la misma secuencia de entrada.
- Para toda máquina Mealy existe una máquina Moore equivalente, y viceversa.
- Las MSF pueden optimizarse: MSF equivalente con un menor número de estados.
- El estado de la MSF en cada momento depende del estado inicial y de la secuencia de entrada hasta ese momento: el estado actual representa la secuencia de símbolos de entrada pasados (historia de la máquina).
- Una MSF puede no estar completamente especificada: el próximo estado puede no estar definido para un estado actual y entrada dados.

Implementación MEF como CSS

- Equivalencia MEF-CSS
  - Las MEF se pueden implementar como CSS de una forma sistemática.
  - Estado → conjunto de biestables
  - Funciones de salida y próximo estado → circuito combinacional



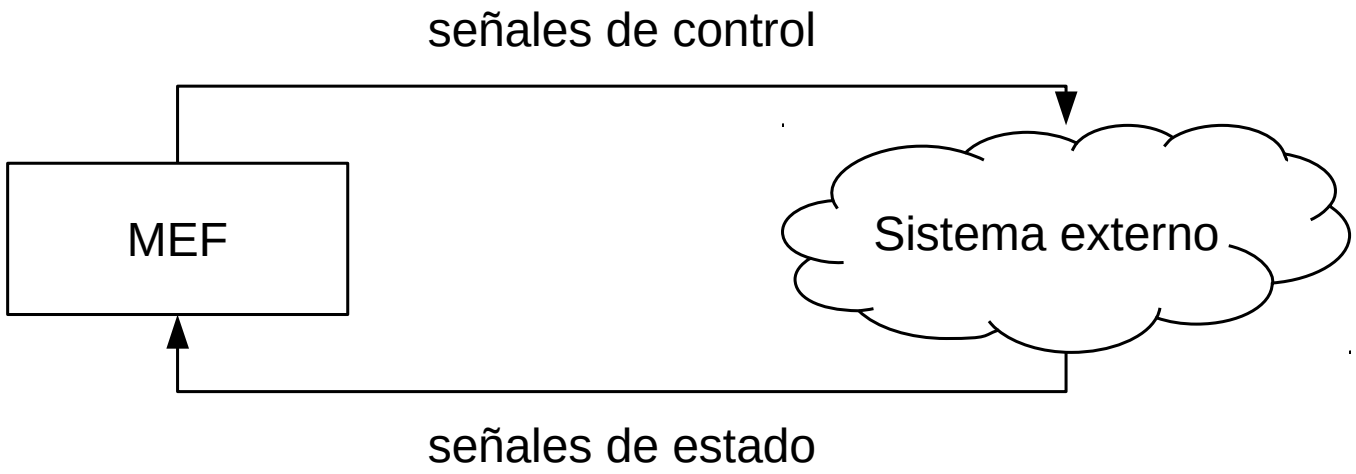
Aplicaciones de los CSS diseñados a partir de MEF

- Detectores de secuencia
  - La salida del circuito se activa o cambia de valor sólo cuando las entradas reciben una determinada secuencia de símbolos.
- Generadores de secuencias
  - La salida genera una secuencia de símbolos fija o variable en función de las entradas.

Muchos problemas prácticos pueden describirse como detectores o generadores de secuencias

Aplicaciones de los CSS diseñados a partir de MEF

- Unidades de control
  - Las entradas reciben información sobre un sistema externo (estado del sistema, sensores mecánicos, de humedad, temperatura, etc.) y las salidas activan actuadores de forma adecuada (operaciones externas, actuadores, calefactores, etc.)
  - La máquina implementa efectivamente el algoritmo de control necesario.
  - La señal de reloj determina con qué frecuencia se evalúa el estado del sistema externo y se ejecuta la acción de control



Aplicaciones de los CSS diseñados a partir de MEF

- Procesado (cálculo) secuencial
  - La salida es el resultado de aplicar algún tipo de operación a los datos de entrada:
    - Cálculo de la paridad
    - Operación aritmética secuencial
    - Codificación/decodificación secuencial, etc.

Contenidos

- Introducción
- Biestables (latches)
- Máquinas de estados finitos y circuitos secuenciales síncronos (CSS)
- **Diseño de MEF**
  - Objetivos del diseño
  - Procedimiento de diseño manual
  - Procedimiento de diseño con herramientas de diseño
- Análisis de CSS

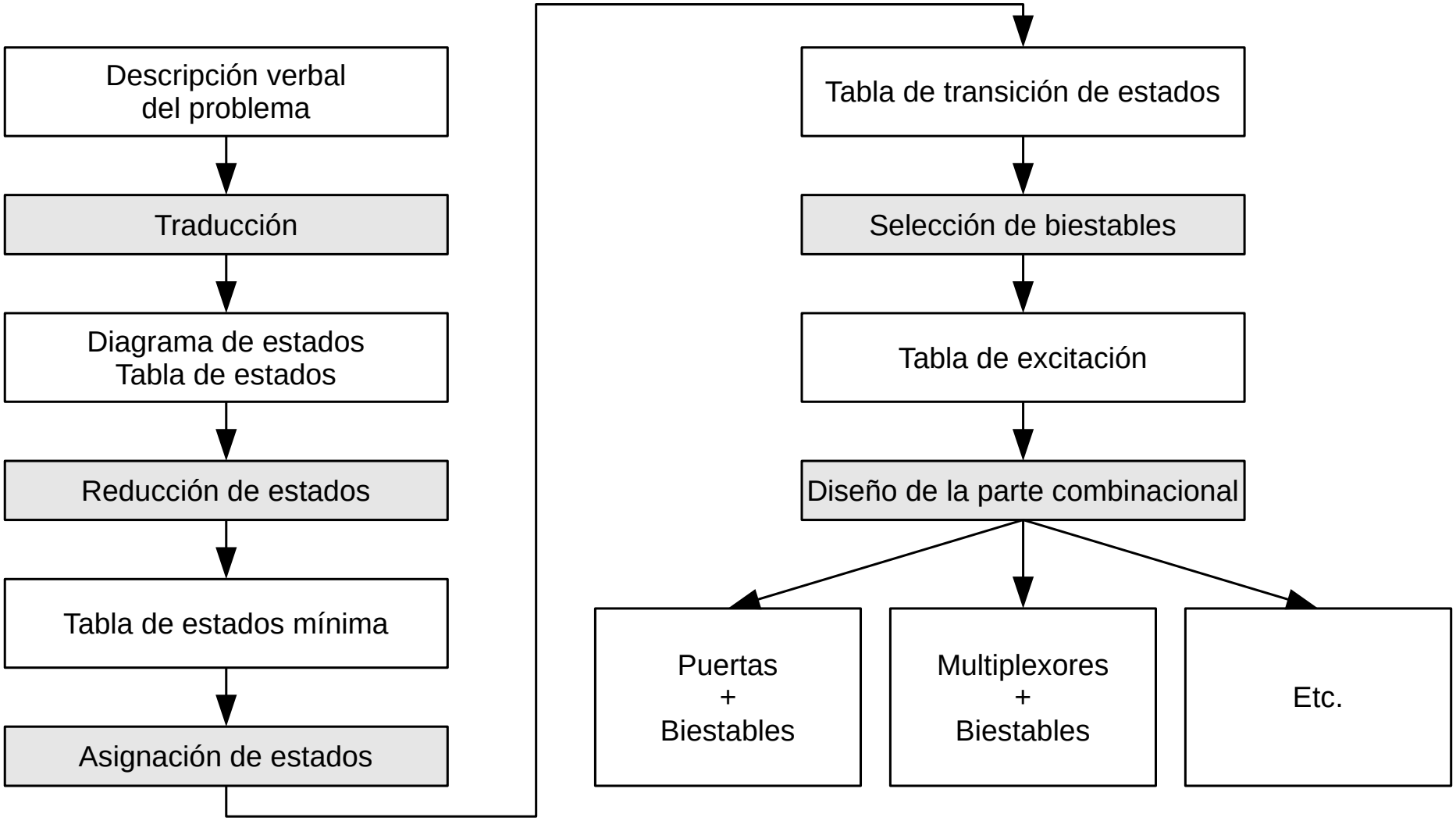
Diseño de CSS a partir de MEF
Objetivo

- Objetivo
  - Definir una MEF que resuelva el problema planteado
  - Implementar la máquina de estados usando un CSS
- Criterios de coste
  - Minimizar el número de elementos de memoria (número de estados)
  - Minimizar número de dispositivos (parte combinacional)
  - Frecuencia de operación (garantizar una frecuencia mínima)
  - Consumo de energía (no superar un consumo máximo)
  - Etc.
- Se debe alcanzar un compromiso entre diferentes factores

Diseño de MEF
Procedimiento

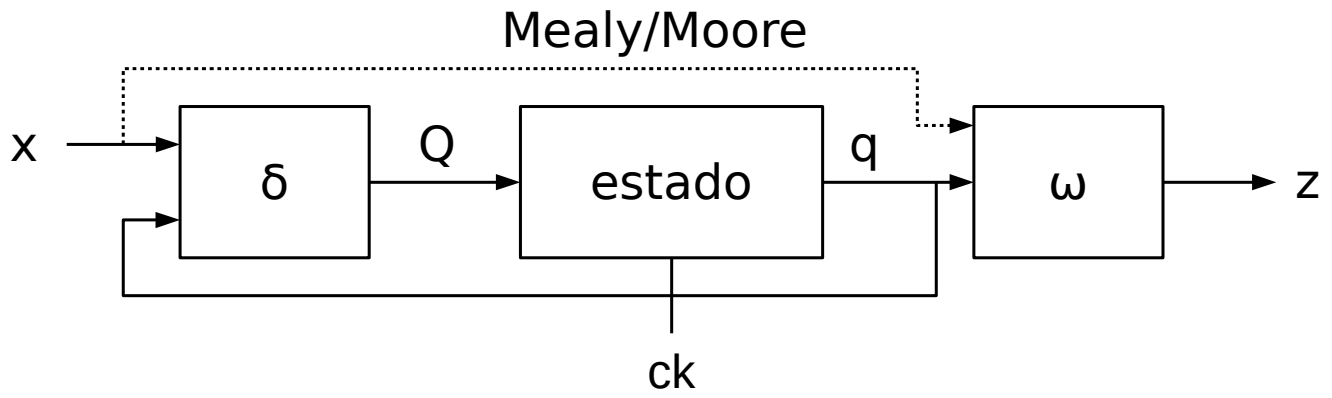
- Procedimiento manual
  - Puede realizarse con papel y lápiz.
  - Comienza con una formalización del problema mediante un diagrama o tabla de estados.
  - La tabla de estados se transforma en diferentes pasos hasta obtener una representación en forma de circuito digital.
- Procedimiento automático mediante herramientas de diseño (CAD)
  - El problema se formaliza mediante una descripción con un lenguaje de descripción de hardware.
  - Se realiza un banco de pruebas y se simula la descripción para asegurar que es correcta.
  - Se usan herramientas de síntesis automática para generar el circuito.

Diseño de CSS
Procedimiento manual



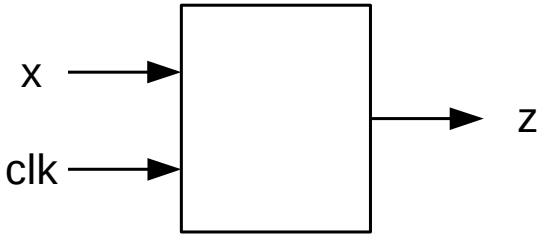
Mealy vs. Moore

- Mealy
  - Salidas diferentes en el mismo estado: posible ahorro de estados.
  - La salida se genera tan pronto está disponible la entrada: menor latencia.
  - Si las entradas no están sincronizadas, la salida tampoco.
- Moore
  - Salidas sincronizadas: menos fallos temporales.
  - Salida sólo depende del estado: funciones de salida más simples. Ej: codificación directa de la salida en el estado.



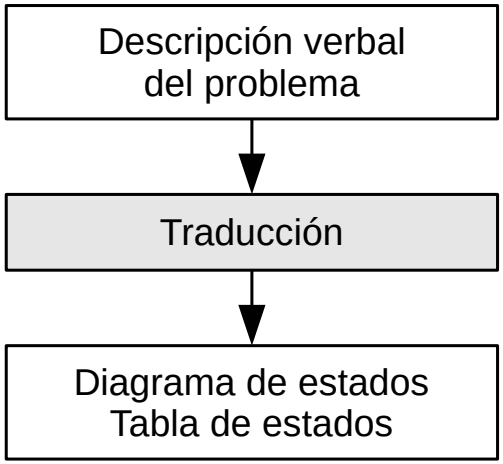
Ejemplo 2: MEF tipo Mealy

- Diseñar un circuito con una entrada x y una salida z que detecte la llegada de la secuencia “1001” en x. Cuando se detecta la secuencia, z permanece a 1 durante un ciclo de reloj. Las secuencias pueden comenzar en cualquier momento y pueden solaparse (el último 1 de una secuencia puede ser el primero de la siguiente).



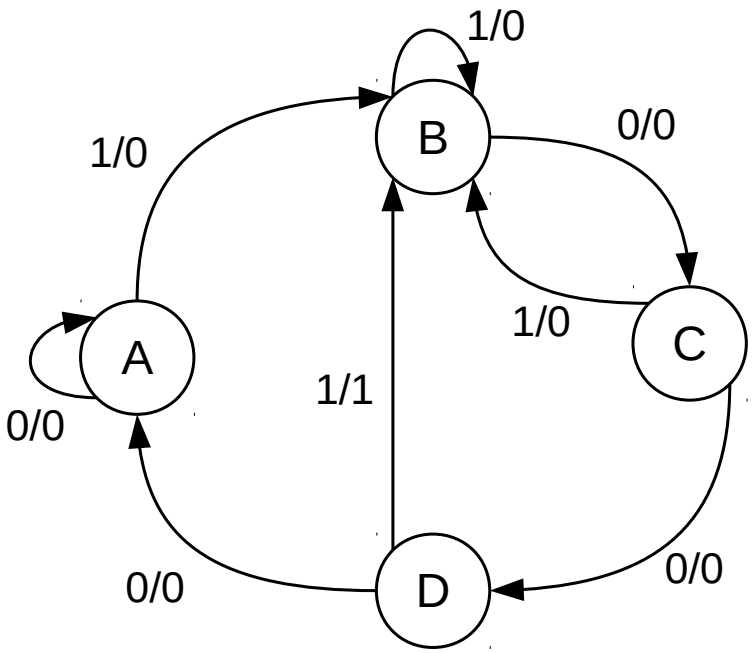

```
x: 00100111000110111001001001010011...
z: 00000100000000000001001001000010...
```

Traducción



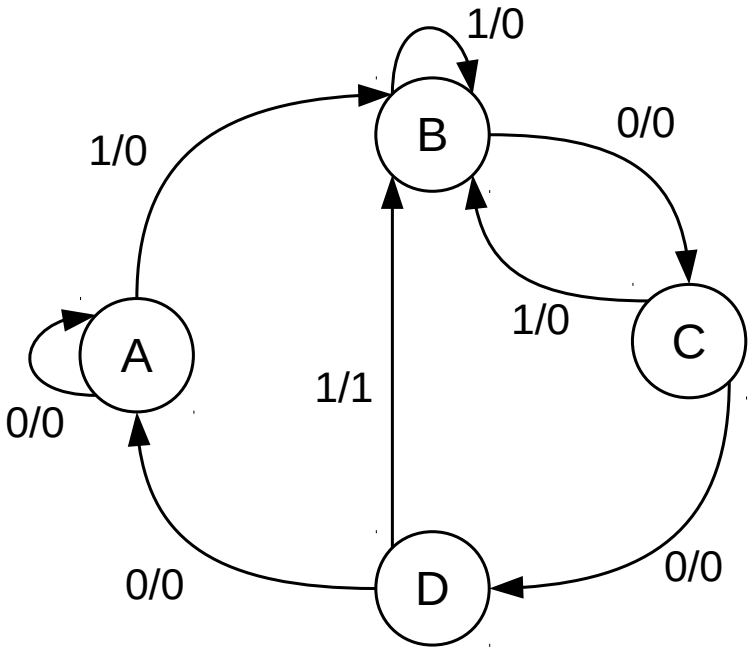
- Parte más importante del proceso de diseño
- No sistemática
- Pistas/consejos:
  - Definir claramente entradas y salidas
  - Seleccionar Mealy o Moore según el problema: sincronización de salidas, etc.
  - Definir secuencias de ejemplo para comprender mejor el problema y detectar posibles casos especiales.
  - Representar el problema mediante un diagrama o tabla de estados.
  - Definir estados de la forma más general posible: estados similares pueden ser el mismo estado.
  - Comprobar el diagrama/table mediante secuencias de ejemplo.

Diagrama de estados. Mealy



- Nodos
  - Representan estados.
  - Nombres intuitivos ¿?
    - {A, B, C, …}
    - {S0, S1, S2, …}
    - {wait, start, receiving, ...}
- Arcos
  - Representan posibles transiciones desde un estado dado (S).
  - Etiqueta x/z:
    - x: valor de entrada que da lugar a la transición desde el estado S.
    - z: valor de salida de la máquina cuando está en el estado S y la entrada es x.

Diagrama de estados. Mealy

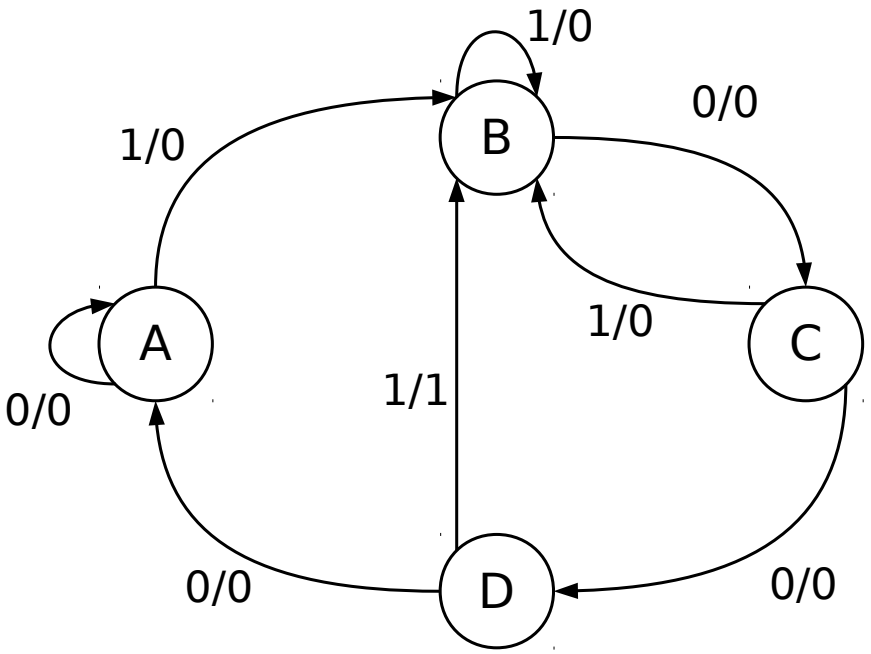


- A: esperando 1er bit de la secuencia ('1')
  - Mientras la entrada sea '0' seguimos en A y mantenemos la salida a '0'.
- B: 1er bit correcto, esperando '0'
  - Si la entrada es '0' pasamos al siguiente estado, si es '1', el segundo bit es incorrecto, pero '1' es el primer bit correcto.
- C: 2º bit correcto, esperando '0'
  - Si la entrada es '0' pasamos al siguiente estado, si es '1' pasamos a B porque hemos recibido el primer bit correcto.
- D: 3er bit correcto, esperando '1'
  - Si la entrada es '0', la secuencia es incorrecta y volvemos al principio (A). Si la entrada es 1 activamos la salida (z=1) y pasamos a B ya que las secuencias pueden solaparse.

Tabla de estados. Mealy

- Representación en forma de tabla de doble entrada con información equivalente al diagrama de estados.
  - Filas: posibles estados.
  - Columnas: posibles valores de entrada.
  - Celdas: próximo estado y valor de salida correspondiente.
- Cada nodo del diagrama de estados y los arcos que salen de él corresponden a una fila en la tabla de estados.
- Pasar desde el diagrama de estados a la tabla de estados y vice-versa es trivial.

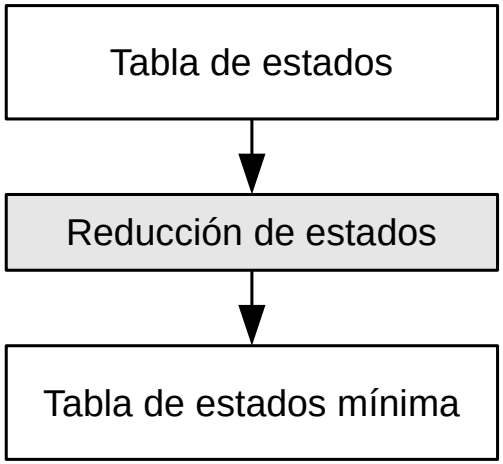
Traducción



S \ x	0	1
A	A,0	B,0
B	C,0	B,0
C	D,0	B,0
D	A,0	B,1

Q,z

Reducción de estados



- Objetivo:
  - Eliminar estados redundantes.
  - Reducir el coste en número de biestables y lógica combinacional.

Estados equivalentes:

Dos estados p y q son equivalentes si cualquier secuencia de entrada aplicada a la máquina comenzando en el estado p produce exactamente la misma secuencia de salida comenzando en q.

Dos estados p y q son equivalentes si y sólo si:

- Todos los próximos de p y q son idénticos o equivalentes para cada valor de entrada.
- Los valores de salida de p y q son iguales para cada valor de entrada.

En una tabla de estados mínima no hay estados equivalentes.

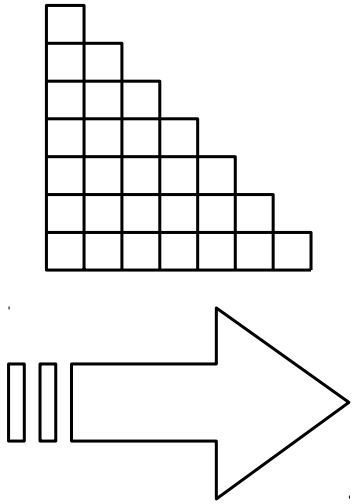
Reducción de estados
Procedimiento

- Comenzando con la tabla de estados, se comparan todas las parejas de estados para identificar posibles estados compatibles.
- La **tabla de estados compatibles** ayuda a identificar estados compatibles y las condiciones de compatibilidad.
- Una vez detectados todos los estados compatibles, se agrupan en clases de equivalencia.
- Se construye una nueva tabla de estados usando las clases de equivalencia obtenidas.

Reducción de estados

S \ x	0	1
A	A,0	B,0
B	C,0	A,0
C	D,0	B,0
D	A,0	B,1

Q,z

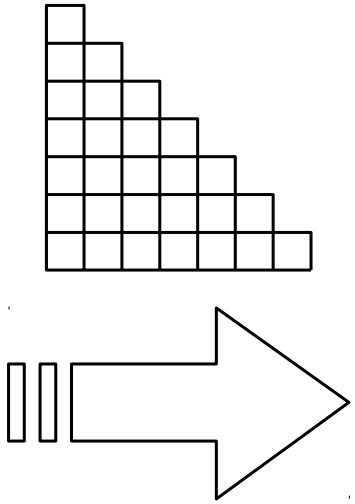


¡Ya es una tabla mínima!

Reducción de estados (otro ejemplo)

S \ x	0	1
A	B,0	C,0
B	D,0	E,0
C	G,0	E,0
D	H,0	F,0
E	G,0	A,0
F	G,1	A,0
G	D,0	C,0
H	H,0	A,0

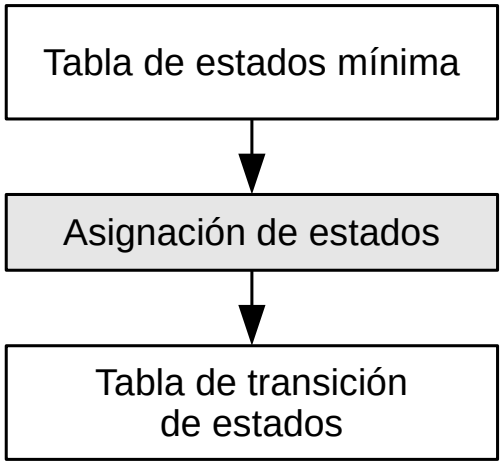
NS, z



S \ x	0	1
a	b,0	a,0
b	d,0	a,0
d	h,0	f,0
f	b,0	a,0
h	h,0	a,0

NS, z

Asignación de estados



- Objetivo:
  - Asignar valores binarios a los estados (codificación de estados).
  - Permite almacenar el estado en biestables.
- Selección de la codificación
  - Afecta al resultado final: número de dispositivos, tamaño del circuito, retraso/velocidad de operación, consumo de energía, etc.
- Opciones típicas
  - Algoritmos de asignación complejos: optimizan el resultado final.
  - Asignación arbitraria o aleatoria: circuitos simples o cuando el coste no es importante.
  - Un biestable por estado (codificación one-hot): cuando no es importante minimizar el uso de biestables. Uso típico en FPGA.
  - Código Gray: secuencia de estados consecutivos, reduce transiciones y consumo.

Asignación de estados (gray)

Tabla de estados

S \ x	0	1
A	A,0	B,0
B	C,0	B,0
C	D,0	B,0
D	A,0	B,1

NS,z

Asignación de estados

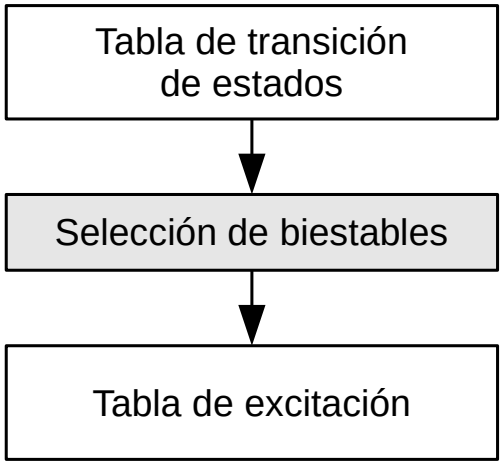
S	q_1q_0
A	00
B	01
C	11
D	10

Tabla de transición de estados

q_1q_2 \ x	0	1
00	00,0	01,0
01	11,0	01,0
11	10,0	01,0
10	00,0	01,1

Q_1Q_2,z

Selección de biestables



- Objetivo
  - Seleccionar el tipo de biestables que almacenarán el estado codificado.
- Opciones:
  - JK: reduce el coste de la parte combinacional pero necesita dos entradas de control.
  - SR: más simple que el JK internamente, pero menos flexible.
  - D: Simplifica el diseño y reduce el número de conexiones (una entrada de control).
  - T: útil en aplicaciones con cambios de estado frecuentes.
- Restricciones
  - Sujeto a disponibilidad de biestables.
  - Componentes discretos: JK, por flexibilidad y simplificación de la parte combinacional.
  - Diseño integrado (ej. FPGA): D, elemento básico disponible.

Selección de biestables
Ejemplo: JK

Tabla de transición de estados

q_1q_2 \ x	0	1
00	00,0	01,0
01	11,0	01,0
11	10,0	01,0
10	00,0	01,1

Q_1Q_2,z

Tabla excitación JK

q → Q	JK
0 → 0	0x
0 → 1	1x
1 → 0	x1
1 → 1	x0

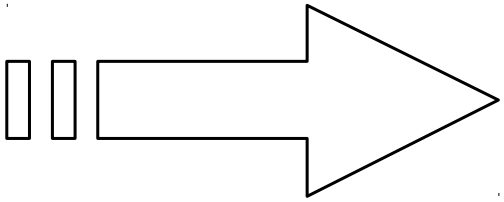


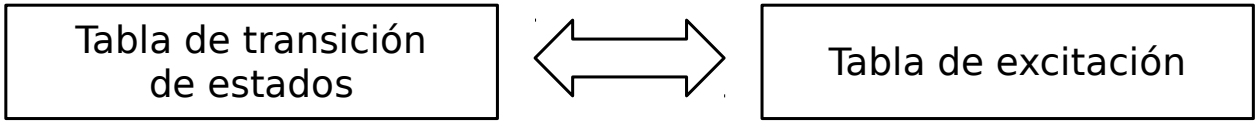
Tabla de excitación

q_1q_2 \ x	0	1
00	0x,0x,0	0x,1x,0
01	1x,x0,0	0x,x0,0
11	x0,x1,0	x1,x0,0
10	x1,0x,0	x1,1x,1

J_1K_1,J_2K_2,z

Selección de biestables
Ejemplo: D

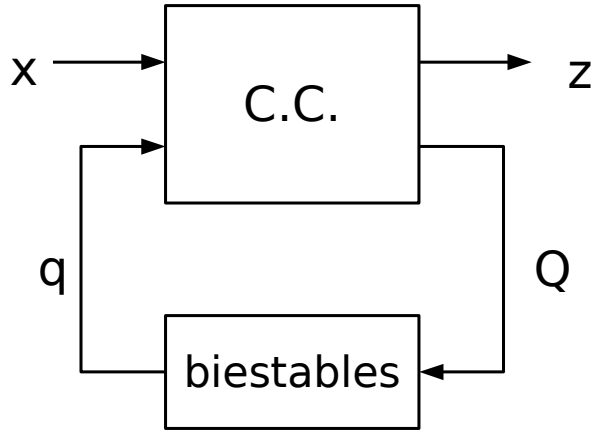
- Biestable D:
  - El próximo estado es igual a D
  - La excitación para alcanzar un próximo estado Q es: D = Q

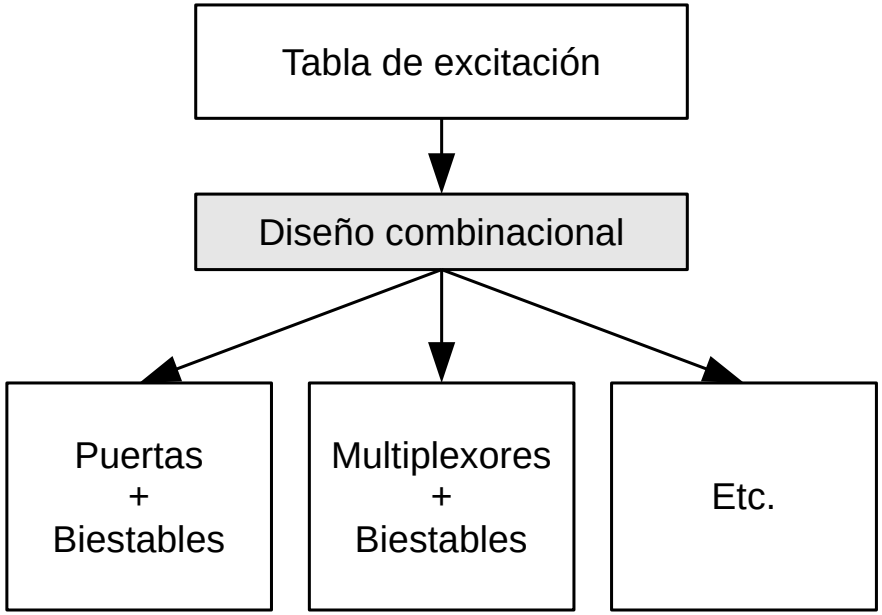


q_1q_2 \ x	0	1
00	00,0	01,0
01	11,0	01,0
11	10,0	01,0
10	00,0	01,1

Q_1Q_2,z
D_1D_2,z

Diseño de la parte combinacional





- La tabla de excitación especifica completamente la parte combinacional.
- La implementación de la parte combinacional puede usar cualquiera de las técnicas disponibles:
  - Diseño en dos niveles con puertas lógicas (K-mapa, etc.)
  - Diseño con subsistemas: multiplexores, decodificadores, etc.
  - Otras.

Diseño de la parte combinacional
Ejemplo: diseño con puertas lógicas

q_1q_2 \ x	0	1
00	0x,0x,0	0x,1x,0
01	1x,x0,0	0x,x0,0
11	x0,x1,0	x1,x0,0
10	x1,0x,0	x1,1x,1

J_1K_1,J_2K_2,z

q_1q_2 \ x	0	1
00	0	0
01	1	0
11	x	x
10	x	x

J_1

q_1q_2 \ x	0	1
00	x	x
01	x	x
11	0	1
10	1	1

K_1

q_1q_2 \ x	0	1
00	0	0
01	0	0
11	0	0
10	0	1

z

q_1q_2 \ x	0	1
00	0	1
01	x	x
11	x	x
10	0	1

J_2

q_1q_2 \ x	0	1
00	x	x
01	0	0
11	1	0
10	x	x

K_2

$J_1 = \bar{x}q_2$

$K_1 = x+\bar{q}_2$

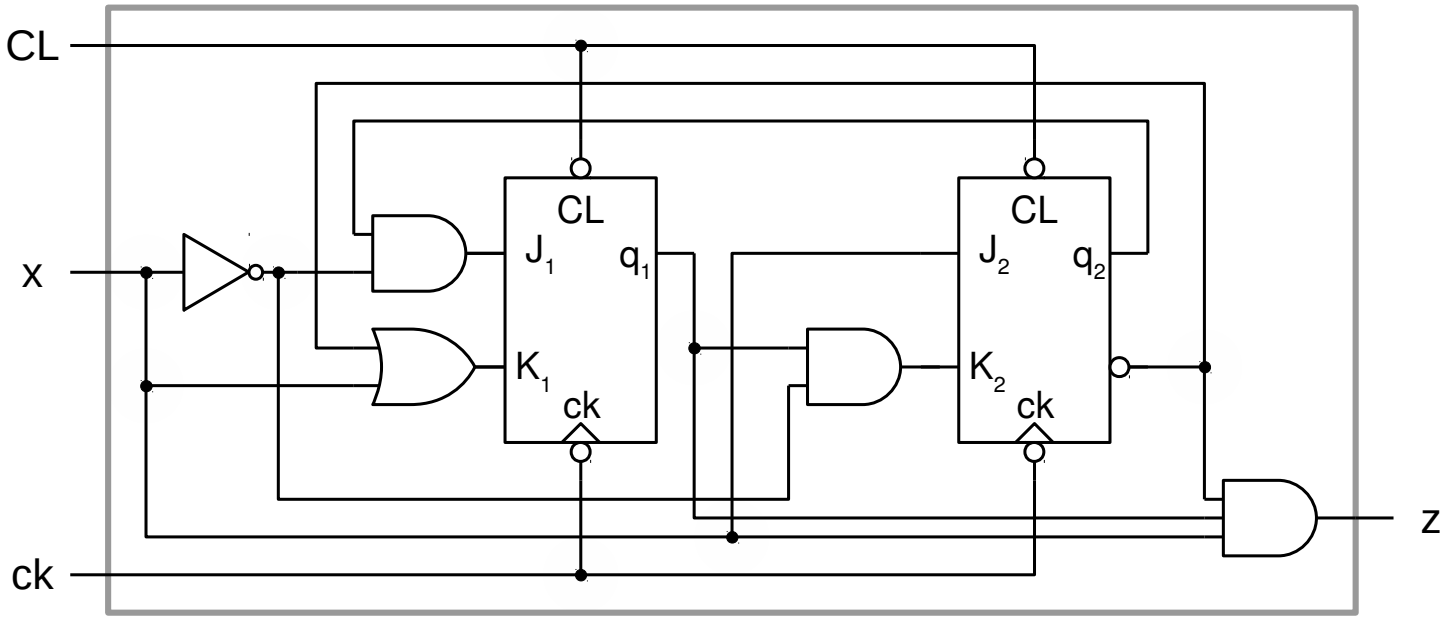
$J_2 = x$

$K_2 = \bar{x}q_1$

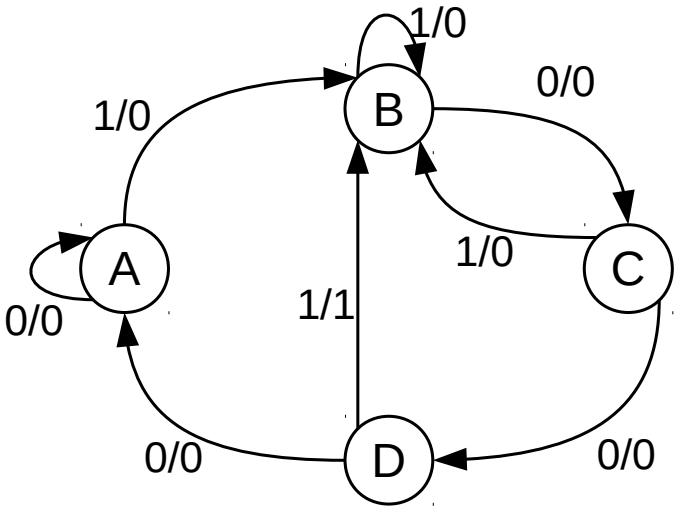
$z = xq_1\bar{q}_2$

Implementación del circuito
Ejemplo: diseño con puertas lógicas

$J_1 = \bar{x}q_2$
$K_1 = x+\bar{q}_2$
$J_2 = x$
$K_2 = \bar{x}q_1$
$z = xq_1\bar{q}_2$



Diseño de CSS
Resumen del ejemplo



S \ x	0	1
A	A,0	B,0
B	C,0	B,0
C	D,0	B,0
D	A,0	B,1

Q,z

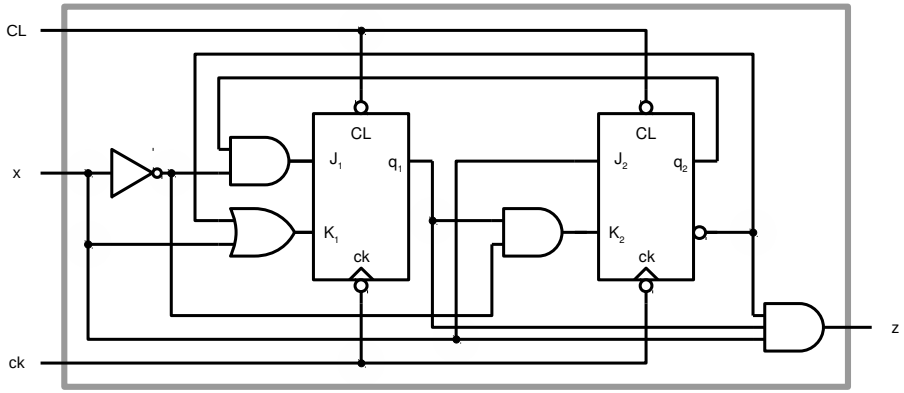
q_1q_2 \ x	0	1
00	00,0	01,0
01	11,0	01,0
11	10,0	01,0
10	00,0	01,1

Q,z

q_1q_2 \ x	0	1
00	0x,0x,0	0x,1x,0
01	1x,x0,0	0x,0x,0
11	x0,x1,0	x1,x0,0
10	x1,0x,0	x1,1x,1

J_1K_1,J_2K_2,z

$$J_1 = \bar{x}q_2$$
$$K_1 = x+\bar{q}_2$$
$$J_2 = x$$
$$K_2 = \bar{x}q_1$$
$$z = xq_1\bar{q}_2$$

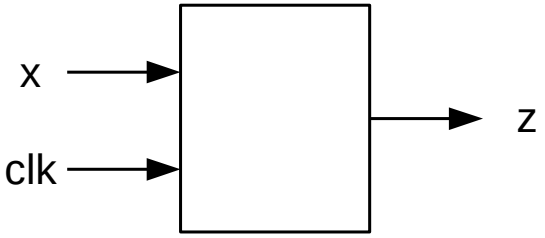


Ejercicios

- Ejercicio 1: Implementa la parte combinacional con multiplexores.
- Ejercicio 2: Implementa el circuito empleando biestables D y...
  - puerta lógicas
  - multiplexores

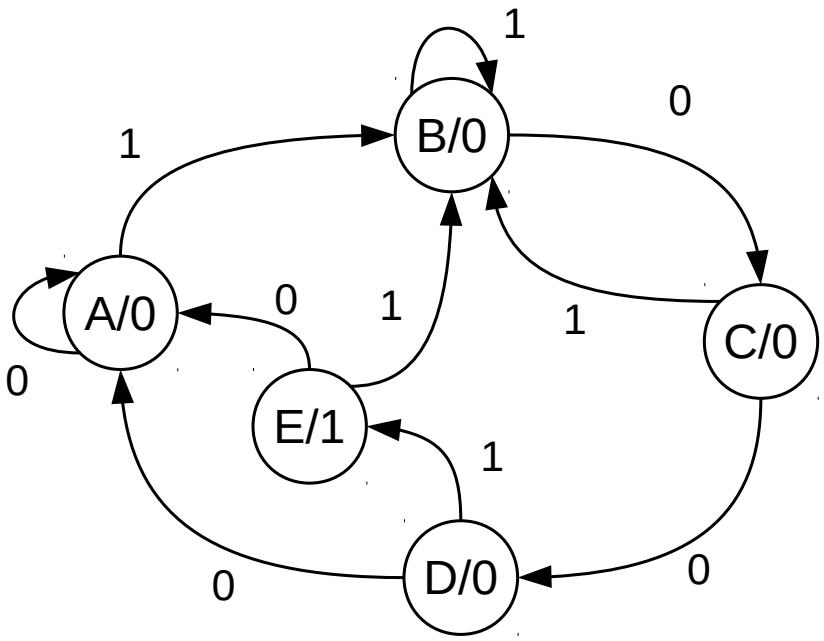
Ejemplo 3: MEF tipo Moore

- Diseñar un circuito con una entrada x y una salida z que detecte la llegada de la secuencia “1001” en x. Cuando se detecta la secuencia, z permanece a 1 durante un ciclo de reloj. Las secuencias pueden comenzar en cualquier momento y pueden solaparse (el último 1 de una secuencia puede ser el primero de la siguiente).



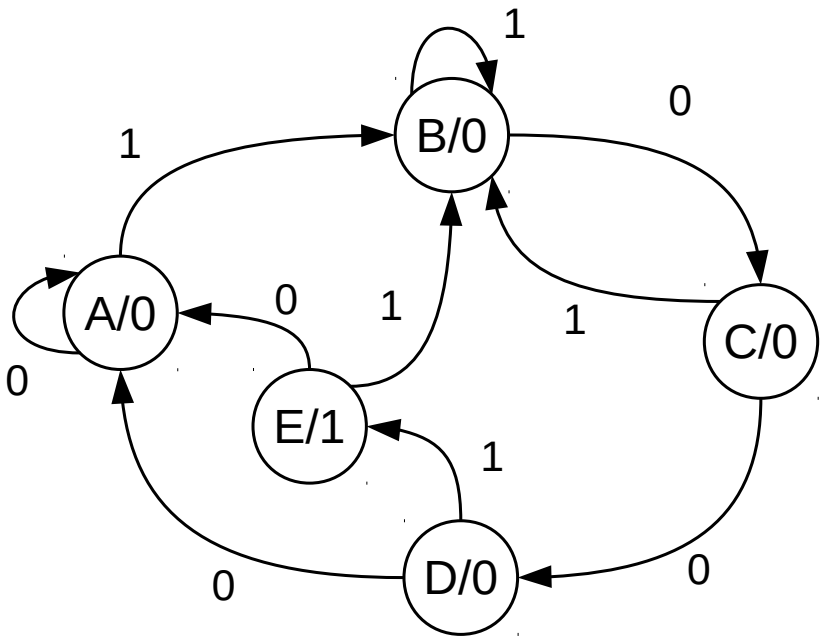

```
x: 00100111000110111001001001010011 0...
z: 00000010000000000000100100100001 0...
```

Diagrama de estados. Moore



- Nodos
  - Representan estados.
  - Nombres intuitivos ¿?
    - {A, B, C, …}
    - {S0, S1, S2, …}
    - {wait, start, receiving, ...}
  - Cada nodo incluye el valor de salida correspondiente a ese estado (Moore)
- Arcos
  - Representan posibles transiciones desde un estado dado (S).
  - Etiqueta: valor de entrada que da lugar a la transición desde el estado S.

Diagrama de estados. Moore



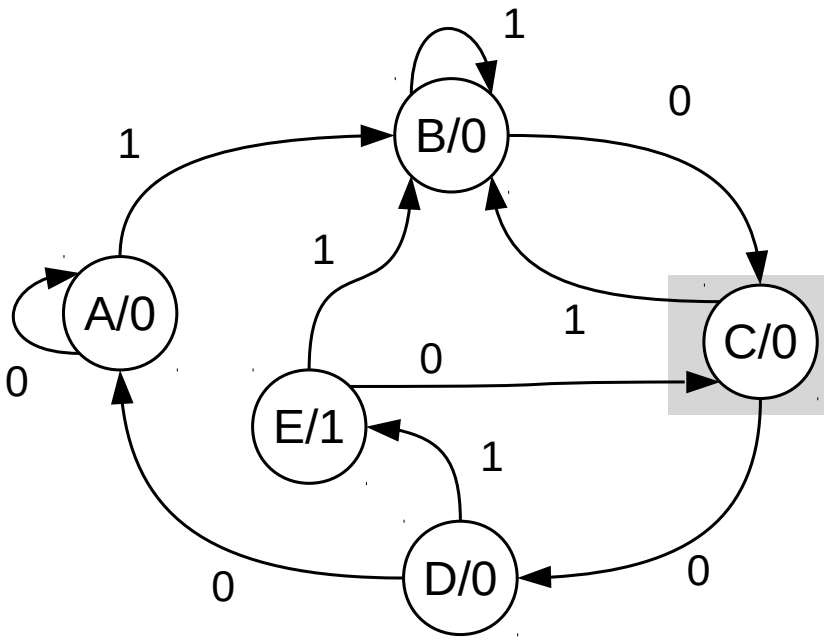
- A: esperando 1er bit de la secuencia ('1')
- B: 1er bit correcto, esperando '0'
- C: 2º bit correcto, esperando '0'
- D: 3er bit correcto, esperando '1'
  - Si la entrada es '0', la secuencia es incorrecta y volvemos al principio (A)
  - Si la entrada es '1' tenemos que activar la salida, pero no podemos pasar directamente a B (salida '0'), tenemos que introducir un nuevo estado E para activar la salida.

¿Hay algún error?

Tabla de estados. Moore

- Representación en forma de tabla de doble entrada con información equivalente al diagrama de estados.
  - Filas: posibles estados.
  - Columnas: posibles valores de entrada.
  - Celdas: próximo estado y valor de salida correspondiente.
    - Opcionalmente: valor de salida en columna aparte, ya que la salida sólo depende del estado, no del valor de entrada.
- Cada nodo del diagrama de estados y los arcos que salen de él corresponden a una fila en la tabla de estados.
- Pasar desde el diagrama de estados a la tabla de estados y vice-versa es trivial.

Tabla de estados. Moore



S \ x	0	1	z
A	A	B	0
B	C	B	0
C	D	B	0
D	A	E	0
E	C	B	1

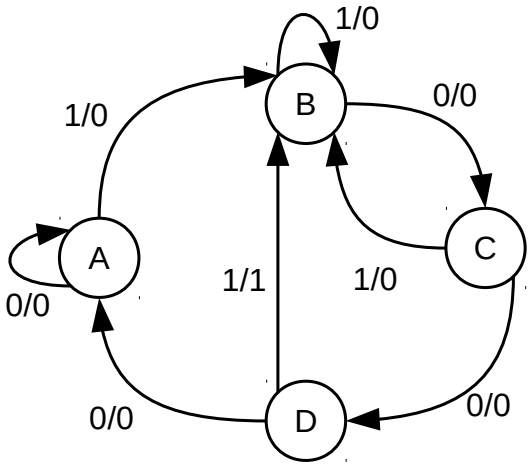
NS

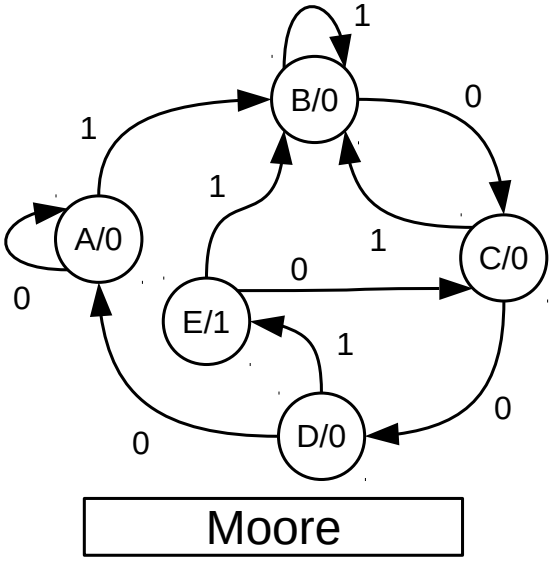
Ejercicios

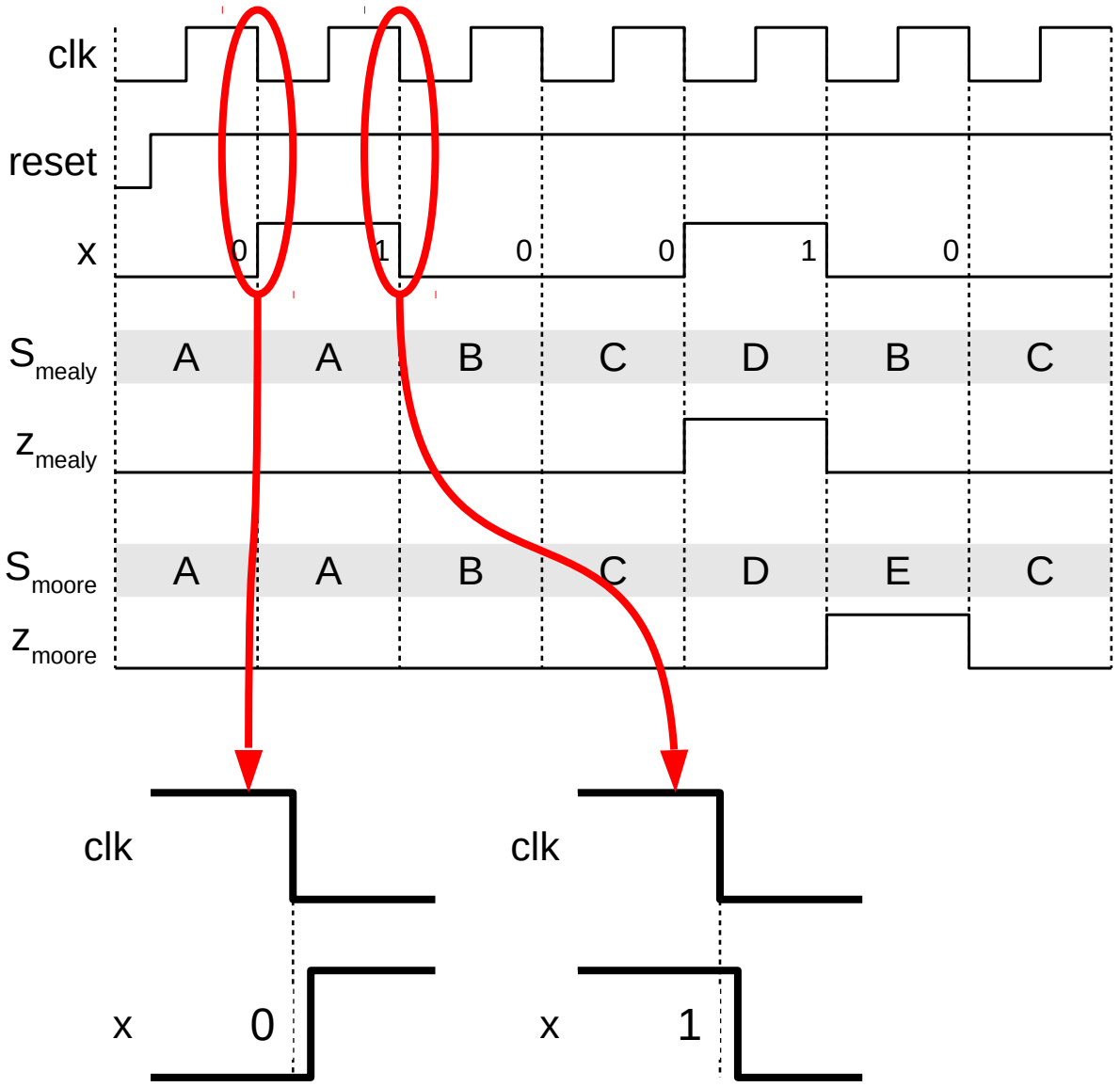
- Ejercicio 3: completa el diseño de la MEF del ejemplo 3 con biestables JK y puerta lógicas.
- Ejercicio 4: Completa el diseño de la MEF del ejemplo 3 con biestables D y multiplexores.

- ¿Cuántos biestables necesitamos?
- ¿Qué ocurre con los estados de los biestables no asignados?
- ¿Podría la máquina quedarse bloqueada en un estado no asignado?
- ¿Se puede resolver el posible bloqueo?

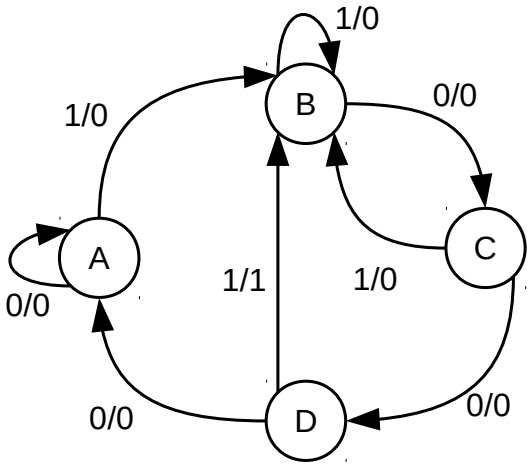
Mealy vs Moore
Entradas sincronizadas



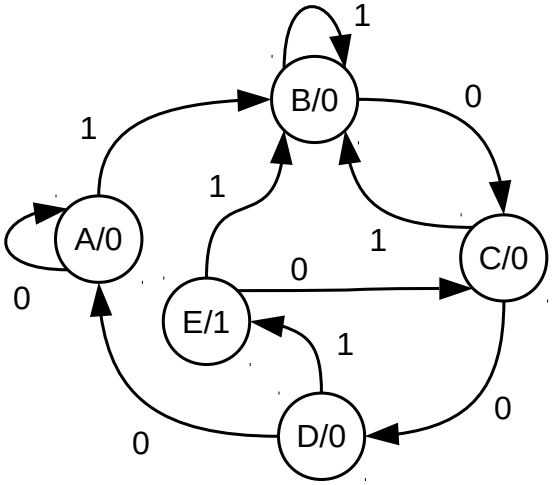




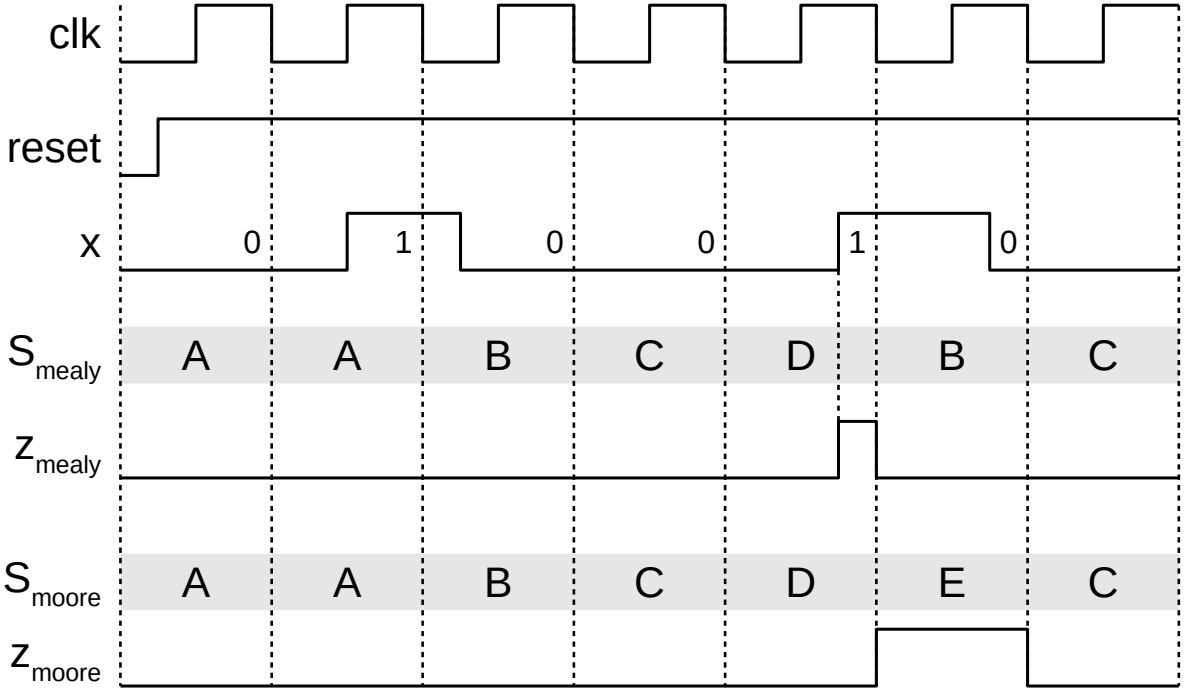
Mealy vs Moore
Entradas no sincronizadas



Mealy

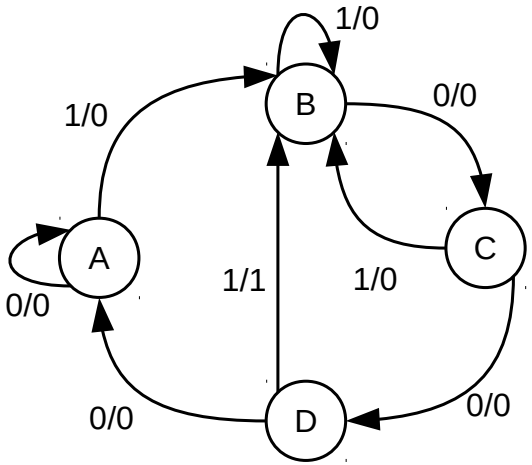


Moore

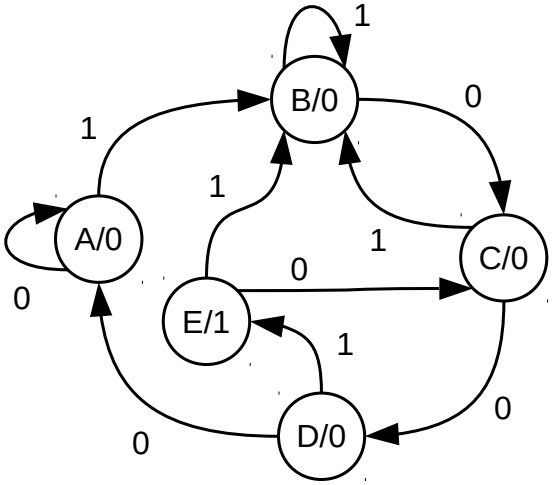


Z_{mealy} se activa durante un tiempo inferior a un ciclo de reloj

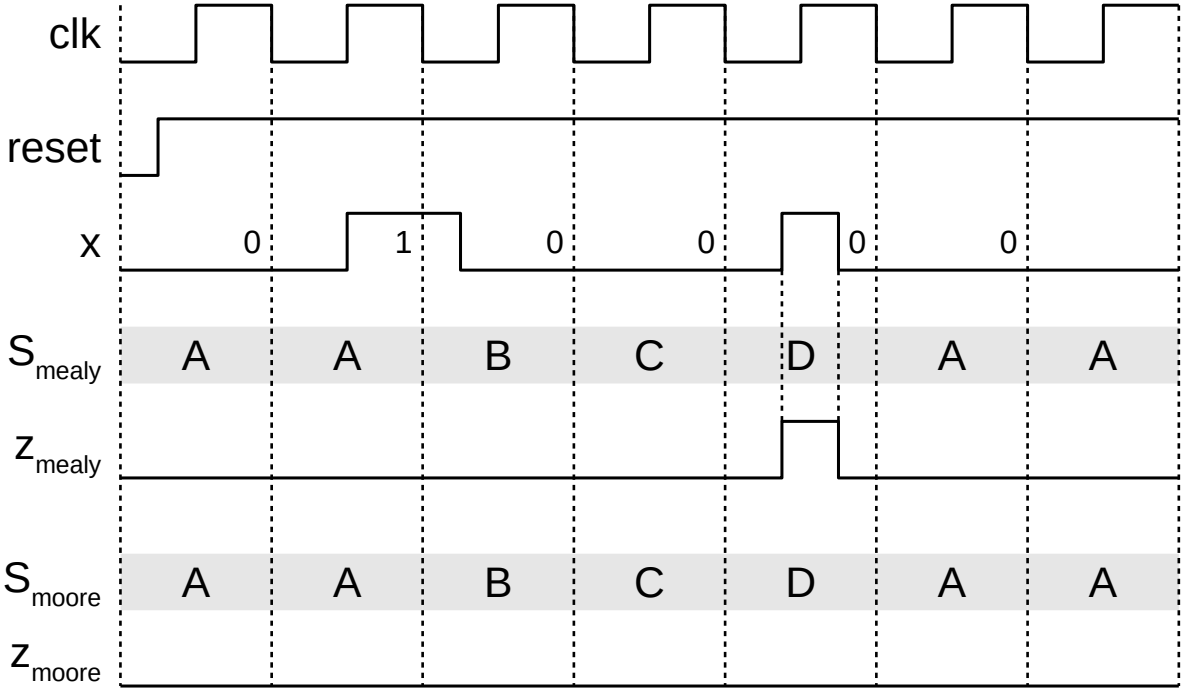
Mealy vs Moore
Entradas no sincronizadas



Mealy



Moore

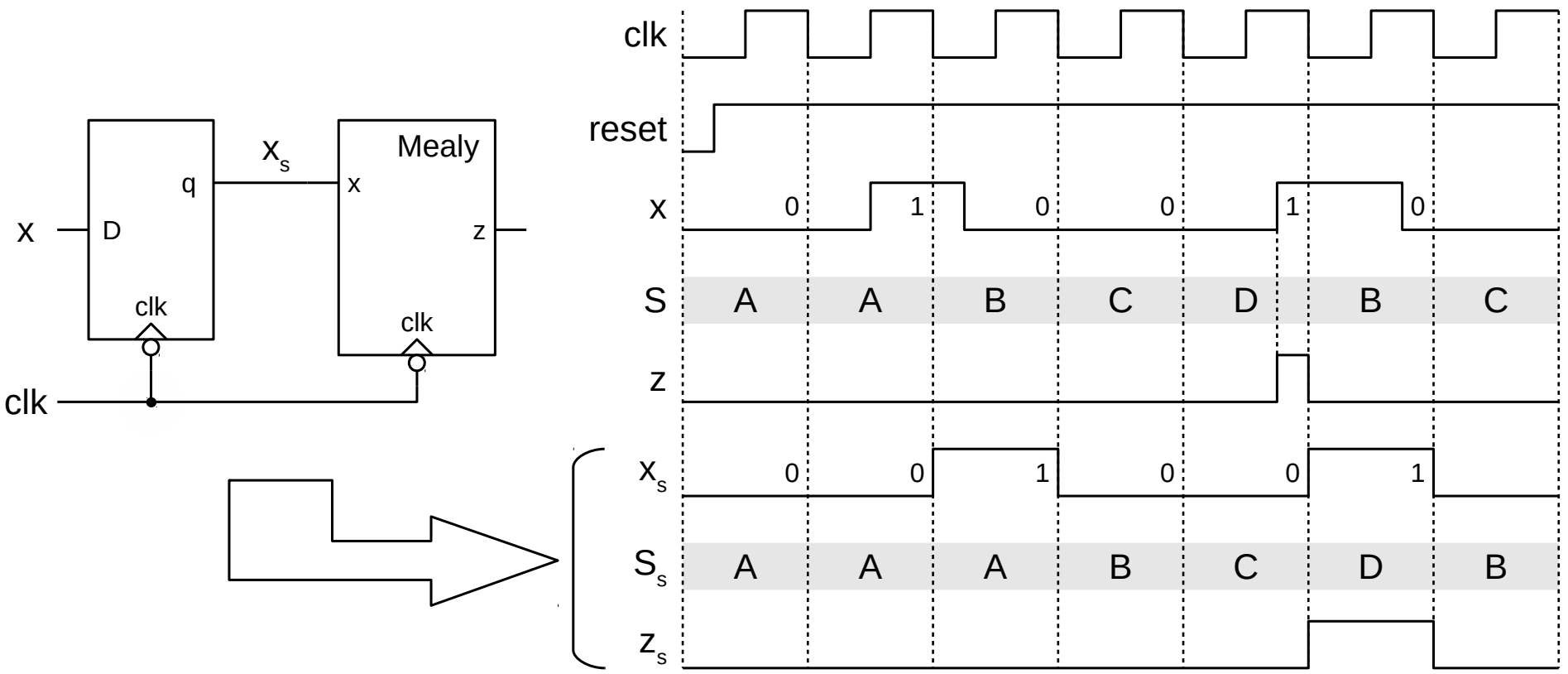


¡Z_{mealy} se activa a pesar de no detectar la secuencia correcta!

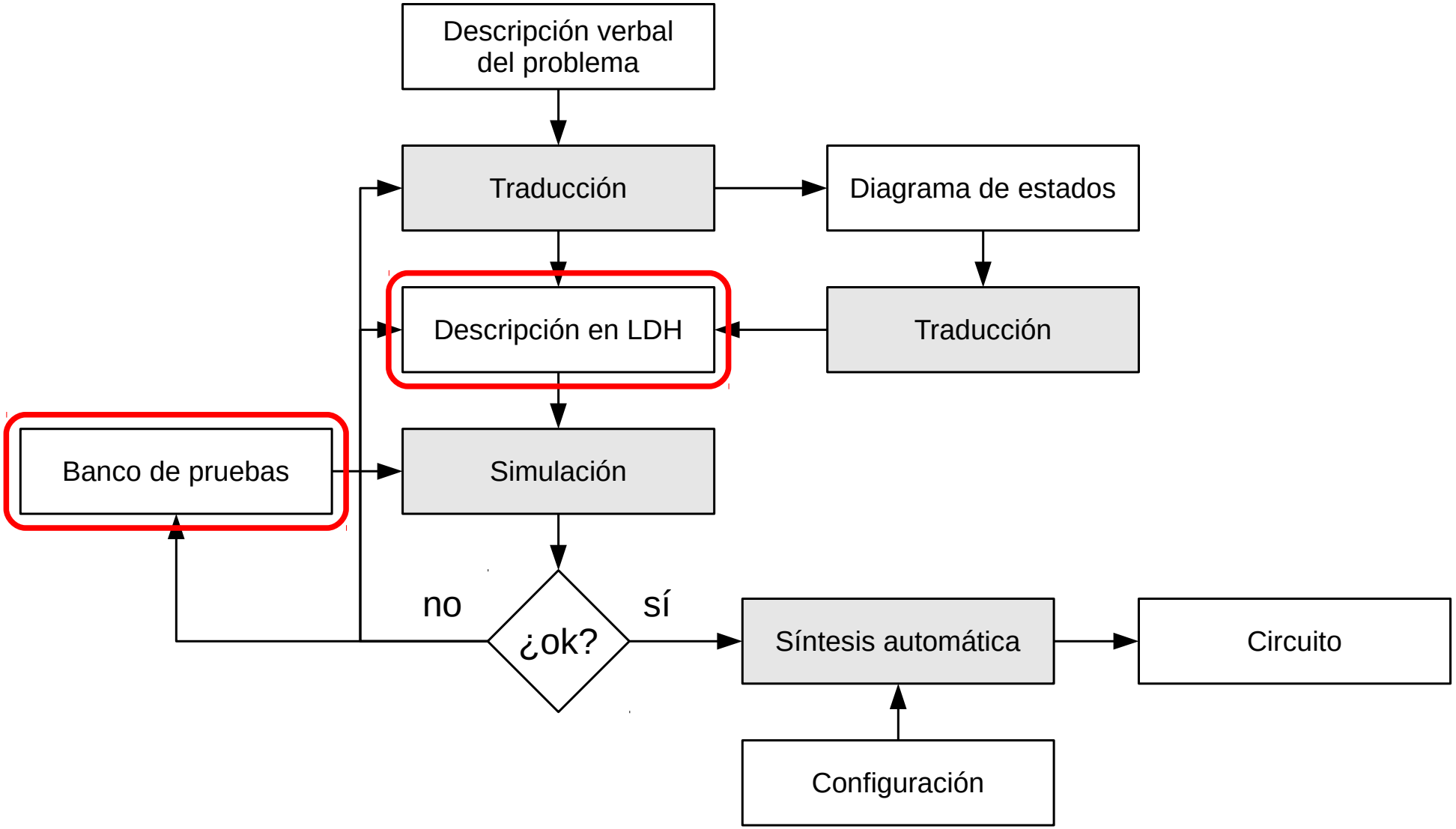
Sincronización de las señales de entrada

- Las señales exteriores pueden cambiar en cualquier momento: no sincronizadas con la señal de reloj.
- Las entradas no sincronizadas pueden provocar comportamientos “extraños” en las salidas de las máquinas de Mealy.
  - Mejor usar máquinas de Mealy sólo con entradas sincronizadas
- Los cambios en las señales de entrada no sincronizadas pueden violar los tiempos de *setup* y *hold* de los biestables produciendo:
  - Retraso excesivo en el cambio del estado
  - Cambios a un nuevo estado erróneo
  - Cambios inesperados a estados de bloqueo
- Las entrada no sincronizadas pueden sincronizarse empleando biestables D.
  - Ventaja: reducir problemas por entradas no sincronizadas.
  - Inconveniente: el valor de la entrada se lee con un ciclo de retraso.

Sincronización de las señales de entrada



Procedimiento de diseño empleando herramientas CAD

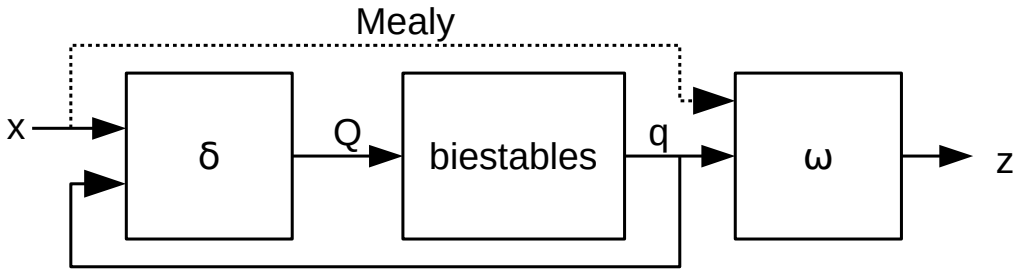


Descripción de MSF en Verilog

- Codificación de estados
  - Se emplean parámetros para asociar un nombre a cada estado codificado.
  - La herramienta de síntesis puede optimizar el código asignado.
- Variable de estado
  - Almacena el valor del estado actual.
- Señal de próximo estado
  - Se le asigna el valor del nuevo estado a almacenar.

```
// Codificación de estados

parameter [1:0]
    A = 2'b00,
    B = 2'b01,
    C = 2'b11,
    D = 2'b10;

// Estado (q): dos biestables

reg [1:0] state;

// Próximo estado (Q): dos bits

reg [1:0] next_state;
```

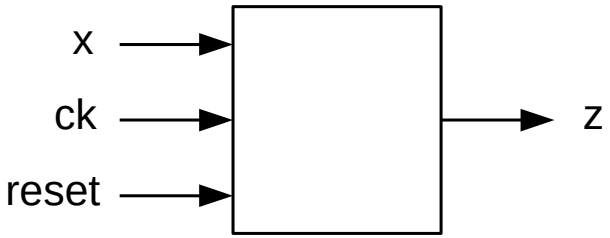
Descripción de MSF en Verilog



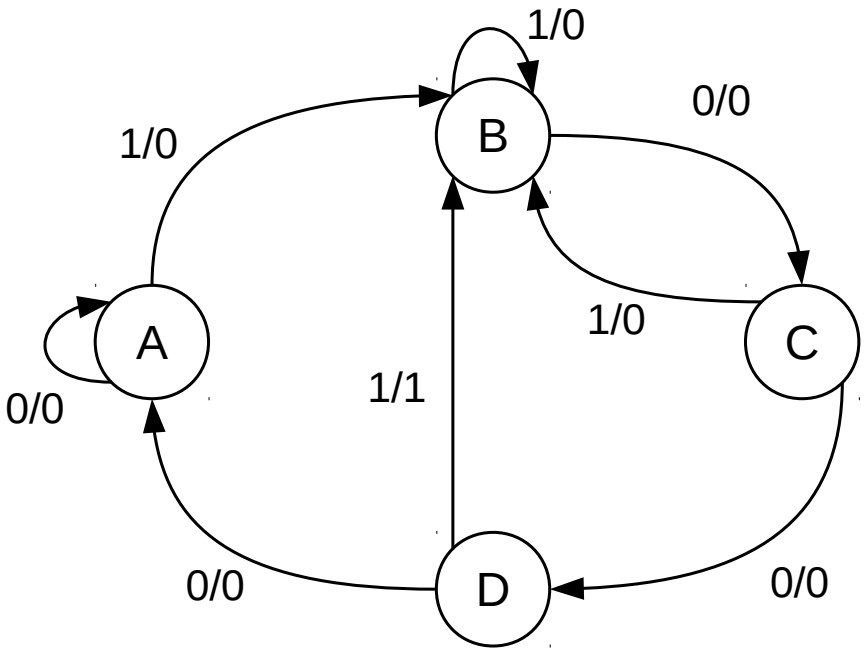
- Tres procesos
  - Cambio de estado. Representa el bloque de biestables.
  - Cálculo del próximo estado. Representa las funciones de próximo estado (δ)
  - Cálculo de la salida. Representa las funciones de salida (ω).
- Sólo el proceso de cambio de estado es secuencial (incluye elementos de memoria)

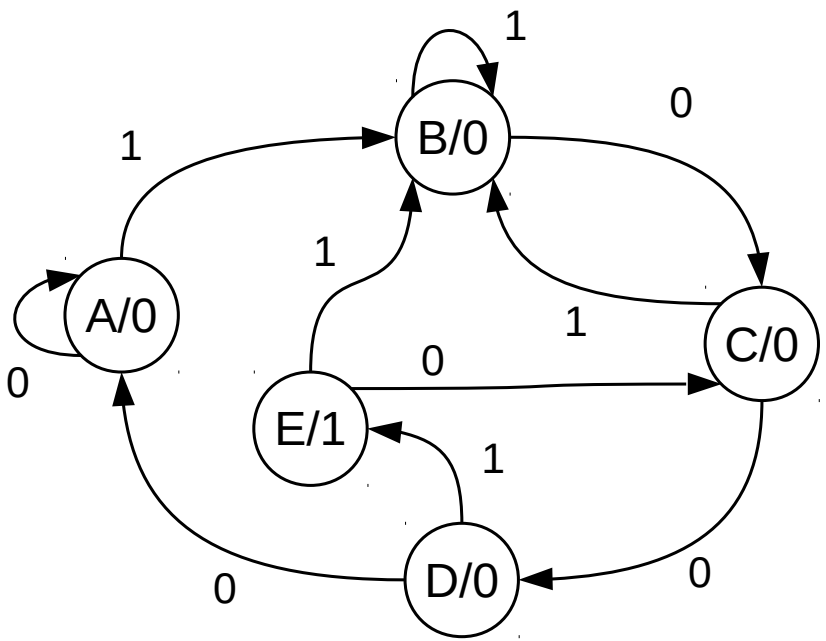
```
// Proceso de cambio de estado

always @(posedge ck, posedge reset)
    if (reset)
        state <= A;
    else
        state <= next_state;

// Proceso de cálculo de próximo estado

always @* begin
    case (state)
    A:
        next_state = ...;
    B:
        next_state = ...;
    ...
    endcase
end

// Proceso de cálculo de la salida

always @* begin
    z = ...;
end
```

Descripción de MSF en Verilog
Ejemplo



Ver ejemplos detectores
de secuencia en curso-verilog.v





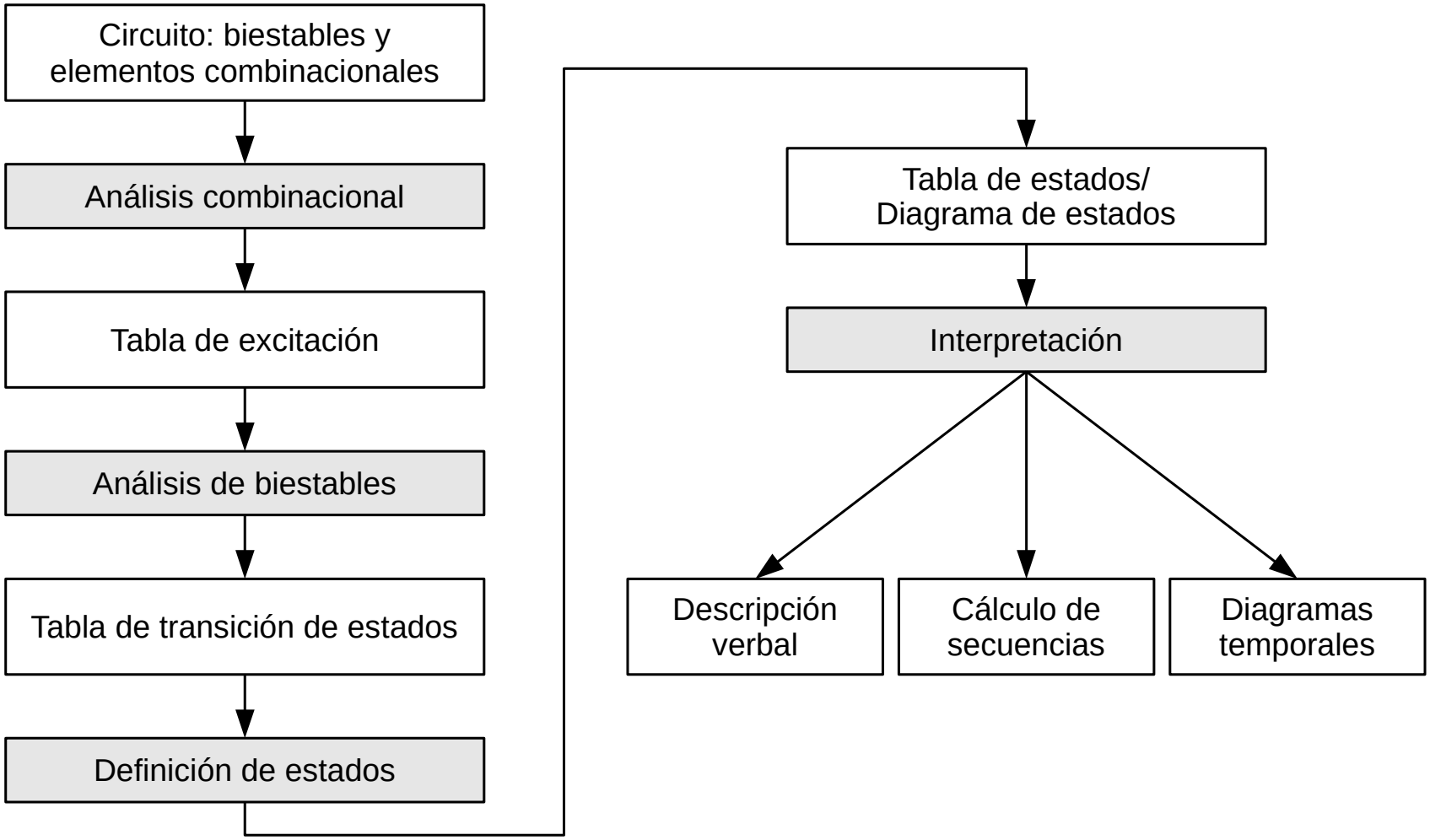
Contenidos

- Introducción
- Biestables (latches)
- Circuitos Secuenciales Síncronos (CSS) y Máquinas de Estados Finitos (MEF)
- Diseño de MEF
- **Análisis de CSS**
  - Análisis funcional de MEF
  - Análisis temporal de CSS

Análisis funcional de MEF

- Proceso opuesto al de síntesis
- Objetivo:
  - A partir del circuito construido, describir la operación general del circuito y su utilidad.
- Procedimiento
  - Obtención de comportamiento síncrono mediante la representación con diagrama o tabla de estados. Este proceso es sistemático y consiste en realizar los pasos de la síntesis en orden inverso.
  - Descripción del comportamiento asíncrono: inicialización, etc.
  - Descripción de la operación: determinar la utilidad y comportamiento del circuito. No sistemático. Depende de la experiencia del diseñador y de posible información adicional:
    - ¿En qué tipo de sistema se encuentra el circuito?
    - ¿Se conoce el tipo de aplicación? Alarma, control, etc.

Análisis funcional de MEF



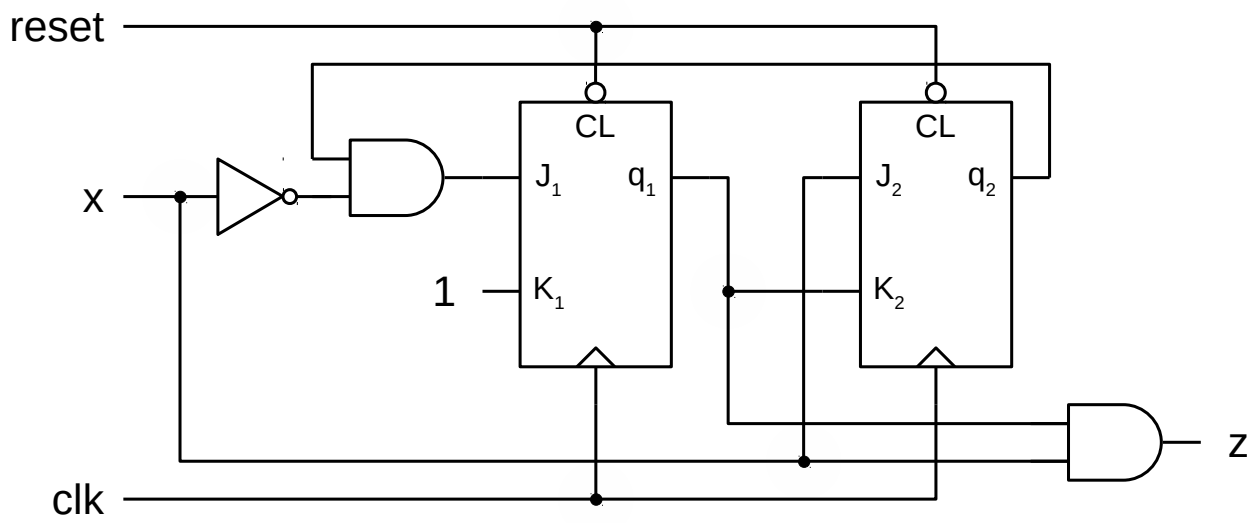
Análisis funcional
Ejemplo 5

Se sabe que el circuito de la figura activa la salida z cuando se recibe una secuencia de 3 bits determinada por la entrada x.

Analice el circuito y determine:

1) Cuál es la secuencia de activación de la salida.
2) El valor activo de la salida.
3) Si puede haber solapamiento entre secuencias de activación consecutivas.
4) La secuencia de estados y de valores de la salida para la secuencia de entrada:

x: 00011001010111100100

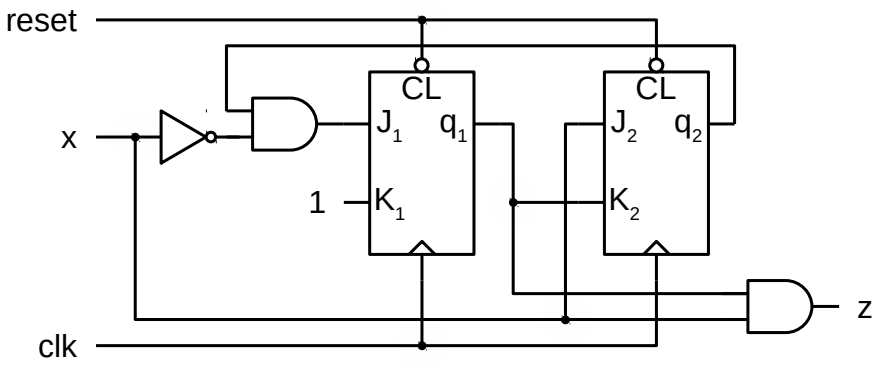


Análisis temporal

- Objetivo: dado un CSS y un conjunto de señales de entrada, obtener las señales de salida correspondientes y su evolución en el tiempo.
- Consideraciones
  - Es posible analizar circuitos con biestables que no son CSS.
  - Si el circuito es un CSS, la secuencia de estados/salida extraída del análisis temporal debe corresponder con la secuencia de estados/salida de la MSF que implementa.
- Similar al análisis temporal de circuitos combinacionales, añadiendo biestables.
  - Parte combinacional: idéntica.
  - Biestables
    - Comportamiento síncrono: aplicar el próximo estado en cada flanco activo del reloj, usando la tabla de estados del biestable.
    - Comportamiento asíncrono: aplicar próximo estado según señales asíncronas.
  - Retrasos: los biestables tienen retrasos desde el reloj a la salida (síncrono) y desde las entradas asíncronas a la salida (asíncrono): t_{ck-q}, t_{CL-q}, etc.

Análisis temporal
Ejemplo 6

- Obtenga la secuencia de salida correspondiente a la secuencia de entrada en la diapositiva siguiente. Distinga dos casos:
  - a) Sin considerar retrasos
  - b) Considerando retrasos, todos iguales ($t_p = t_{ck-q} = t_{CL-q} = \Delta$)
- Verifique si la salida se activa tras la secuencia esperada, según análisis funcional anterior.



$x_n = \overline{x}$

$J_1 = x_n\, q_2$

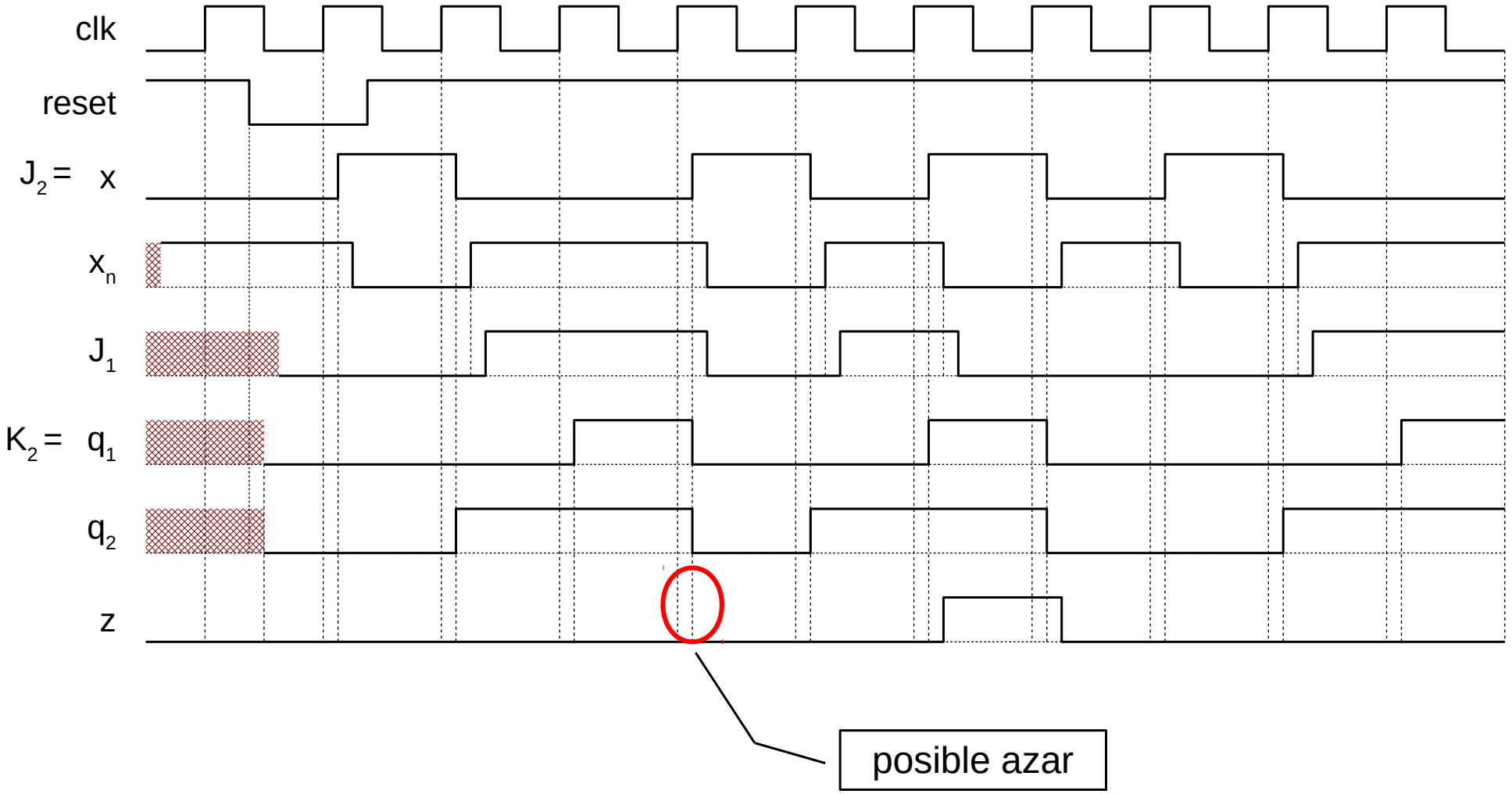
$K_1 = 1$

$J_2 = x$

$K_2 = q_1$

$z = x\, q_1$

Análisis temporal
Ejemplo 6

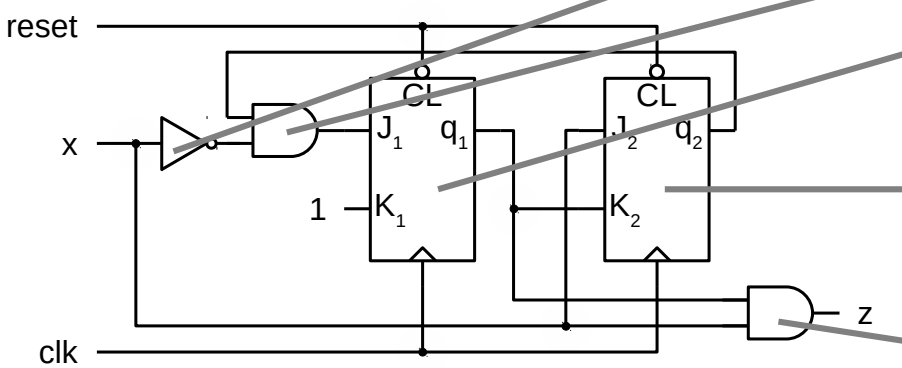


Análisis temporal
Ejemplo 6

- Notas:
  - La entrada x está sincronizada con el reloj.
  - El valor inicial de las señales es desconocido.
- Procedimiento
  - Dibujamos $x_n = \bar{x}$ a partir de x.
  - Aplicamos “reset” y dibujamos q1 y q2 hasta el tercer flanco activo.
  - Hacemos repetidamente:
    - Dibujamos J1 hasta donde conocemos el valor de q2 ($J_1 = x_n\ q_2$)
    - Dibujamos q1 hasta donde conocemos el valor de J1
    - Dibujamos q2 hasta donde conocemos el valor de q1 ($K_2 = q_1$)
  - Dibujamos z a partir de x y q_1.

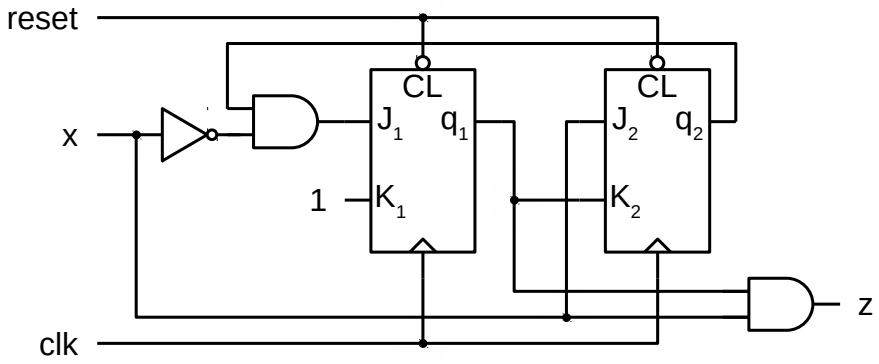
Análisis temporal con Verilog

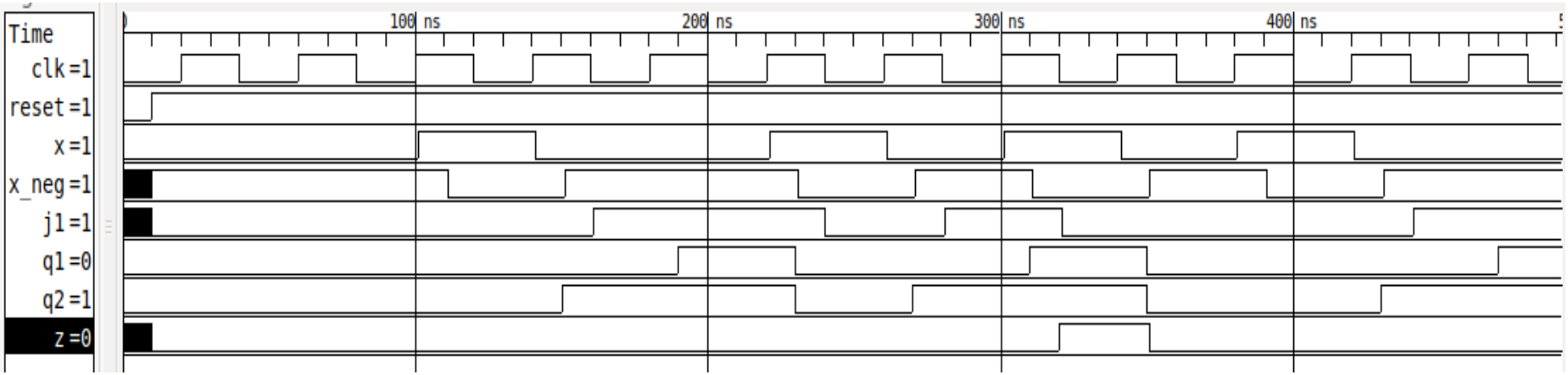
- Descripción estructural incluyendo retrasos.
- Banco de pruebas con la señal de entrada.
- Dejamos que el simulador haga el trabajo :)



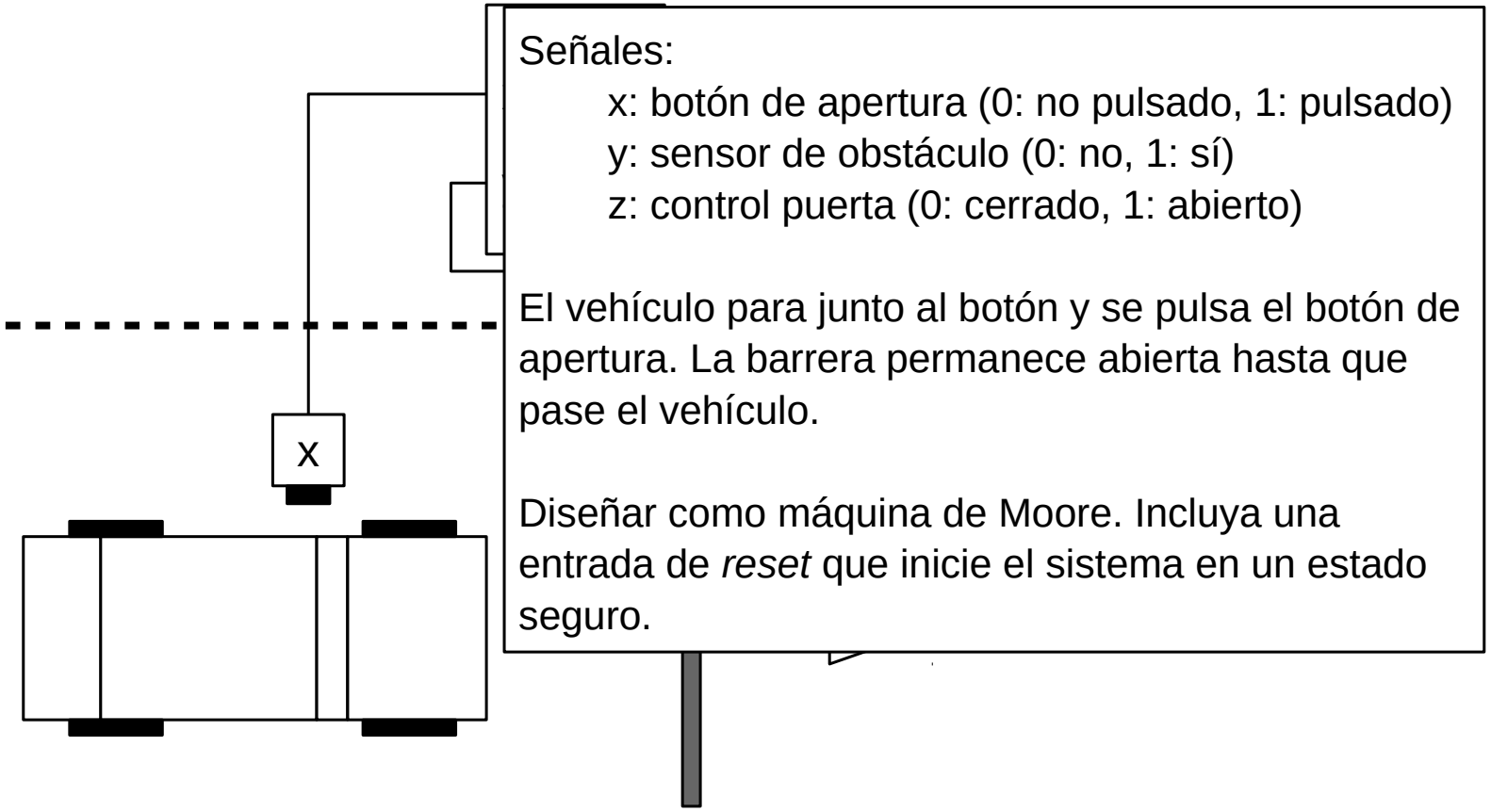

```
module sequence #(
    parameter delay = 10
    )(
    input wire clk,
    input wire reset,
    input wire x,
    output wire z
    );

    // internal wires
    wire x_neg, j1, q1, q2;

    not #delay not1 (x_neg, x);

    and #delay and1 (j1, x_neg, q2);

    jkff #(.delay(delay)) jkff1
        (.clk(clk), .cl(reset),
         .j(j1), .k(1'b1), .q(q1));

    jkff #(.delay(delay)) jkff2
        (.clk(clk), .cl(reset),
         .j(x), .k(q1), .q(q2));

    and #delay and2 (z, x, q1);
endmodule
```

Análisis temporal con Verilog





Ejercicio 5: barrera con sensor de obstáculos



Señales:

- x: botón de apertura (0: no pulsado, 1: pulsado)
- y: sensor de obstáculo (0: no, 1: sí)
- z: control puerta (0: cerrado, 1: abierto)

El vehículo para junto al botón y se pulsa el botón de apertura. La barrera permanece abierta hasta que pase el vehículo.

Diseñar como máquina de Moore. Incluya una entrada de *reset* que inicie el sistema en un estado seguro.

Ejemplo 6. Sumador secuencial

- Diseña un circuito secuencial con dos entradas (a, b) por la que llegan bit a bit y sincronizadas con una señal de reloj las cifras de dos números enteros A y B, comenzando por los bits menos significativos, y que genera en su salida z, bit a bit, las cifras de la suma A+B.
- Entradas:
  - clk: señal de reloj activa en flanco de subida.
  - reset: puesta a cero asíncrona.
  - a, b: entradas de datos.
- Salida:
  - z: resultado de la suma.

Ejercicio 7. Multiplicador secuencial

- Diseña un circuito secuencial con una entradas x por la que llegan bit a bit y sincronizadas con una señal de reloj las cifras de un número enteros X, comenzando por los bits menos significativos, y que genera en su salida z, bit a bit, las cifras del producto 3*X.
- Entradas:
  - clk: señal de reloj activa en flanco de subida.
  - reset: puesta a cero asíncrona.
  - x: entradas de datos.
- Salida:
  - z: resultado del producto.