

---

# Tema 8. Subsistemas secuenciales

## Circuitos Electrónicos Digitales

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

Usted es libre de copiar, distribuir y comunicar públicamente la obra y de hacer obras derivadas siempre que se cite la fuente y se respeten las condiciones de la licencia Attribution-Share alike de Creative Commons.

Puede consultar el texto completo de la licencia en:

<http://creativecommons.org/licenses/by-sa/3.0/es>

<http://creativecommons.org/licenses/by-sa/3.0> (original en inglés)



05/12/2022 20:54:06

# Contenidos

---

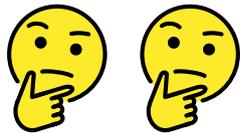
- Introducción
- Registros
- Contadores
- Diseño con subsistemas secuenciales

# Leyenda de ejercicios

---



Ejercicio básico/fácil



Ejercicio para pensar un poco



Ejercicio para pensar mucho

# Introducción

---

- Subsistema secuencial
  - Componente secuencial con una funcionalidad lo bastante general para encontrar aplicación en una diversidad de problemas de diseño de circuitos secuenciales.
  - Se usan para almacenar información codificada en tuplas de bits (palabras) y realizar operaciones sobre dicha información.
  - Como mínimo, un subsistema secuencial debe tener un biestable por cada bit de la palabra que almacena.
- Tipos básicos de subsistemas secuenciales
  - Registros: Se caracterizan porque en ellos pueden escribirse palabras que pueden ser leídas posteriormente. Algunos permiten realizar operaciones sobre dichas palabras.
  - Contadores: Permiten incrementar/decrementar el dato que almacenan.

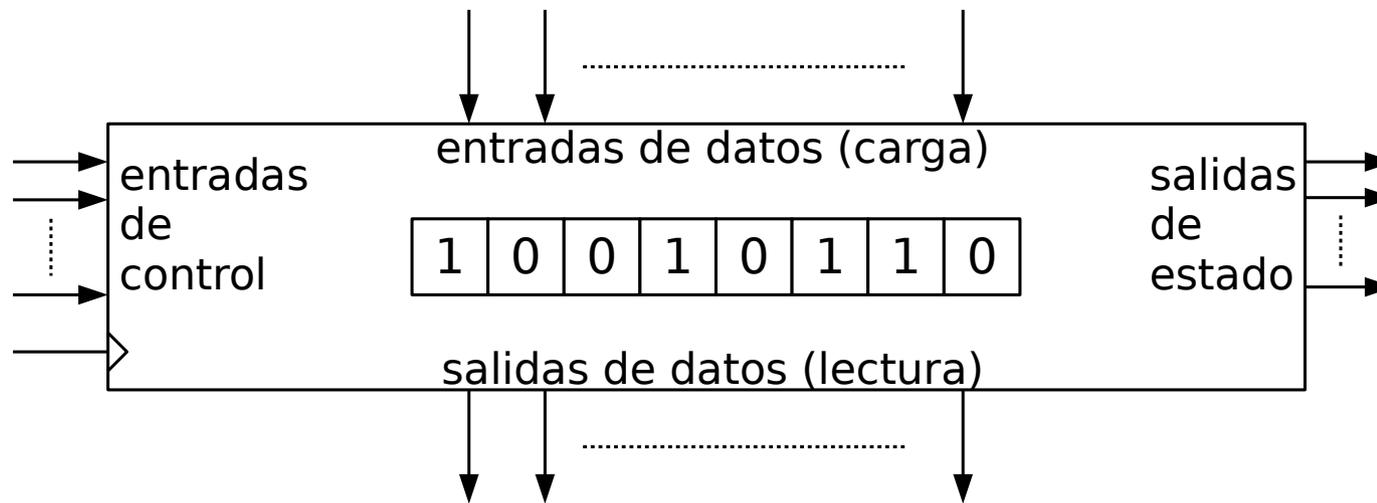
# Introducción

## entradas y salidas típicas

---

- Entradas de control
  - Las entradas de control determinan la operación a realizar.
  - La operación es *asíncrona* si se realiza tan pronto como se ordena.
  - La operación es *síncrona* si sólo puede realizarse en el flanco o nivel activo de la señal de reloj.
  - Ejemplos: puesta a cero (clear -CLEAR-), habilitación (enable -EN-), carga de un dato (load -LD-).
- Entradas de datos
  - Proporcionan el dato a cargar en el subsistema
- Salidas de datos
  - Permiten obtener (observar) el dato almacenado
- Salidas de estado
  - Indican información sobre el contenido del subsistema: si el dato almacenado es cero, fin de estado de cuenta, etc.

# Introducción representación esquemática



# Introducción

---

- Descripción de la funcionalidad
  - Un subsistema secuencial puede describirse como una máquina de estados finitos, pero no suele ser apropiado: pueden tener gran número de estados.
  - Su funcionalidad suele definirse sobre el valor que la palabra almacenada codifica y no en base a cada bit por separado.

# Registros

---

- Introducción
- Registros
  - Clasificación
    - paralelo/paralelo
    - de desplazamiento
    - universal
- Contadores
- Diseño con subsistemas secuenciales

# Registros

---

1	0	0	1	0	1	1	0
---	---	---	---	---	---	---	---

- Almacén de datos de  $n$  bits ( $n$  biestables)
  - Habitualmente nos referimos al contenido por el dato que la palabra almacenada representa y no por los bits individuales.
- Operaciones básicas:
  - Escritura (carga): modificación del dato almacenado indicando de forma explícita el nuevo dato.
  - Lectura: consulta del contenido del registro.

# Registros. Clasificación

---

- Escritura en paralelo
  - Una línea de entrada de datos para cada bit ( $X_{n-1}..X_0$ ).
  - Todos los bits se cargan a la vez en el mismo ciclo de reloj con los valores de esas líneas.
  - Notación:  $REG \leftarrow X$
  - La parte derecha de la asignación es una expresión que puede incluir nombres de registros. En tal caso, dichos nombres representan el contenido de los registros en el ciclo de reloj en el que se realiza la operación de escritura.

# Registros. Clasificación

---

- Escritura en serie con desplazamiento a la derecha
  - Una línea de entrada de datos ( $X_R$ ).
  - El bit más significativo se escribe con la entrada de datos.
  - Si un bit no es el más significativo, se escribe con el contenido del siguiente más significativo.
  - Notación:  $REG \leftarrow SHR(REG, X_R)$ 
    - Si  $A$  es una palabra de  $n > 1$  bits  $SHR(A, B) = \{B, A_{n-1..1}\}$

# Registros. Clasificación

---

- Escritura en serie con desplazamiento a la izquierda
  - Una línea de entrada de datos ( $X_L$ ).
  - El bit menos significativo se escribe con la entrada de datos.
  - Si un bit no es el menos significativo, se escribe con el contenido del siguiente menos significativo.
  - Notación:  $REG \leftarrow SHL(REG, X_L)$ 
    - Si  $A$  es una palabra de  $n > 1$  bits  $SHL(A, B) = \{A_{n-2..0}, B\}$

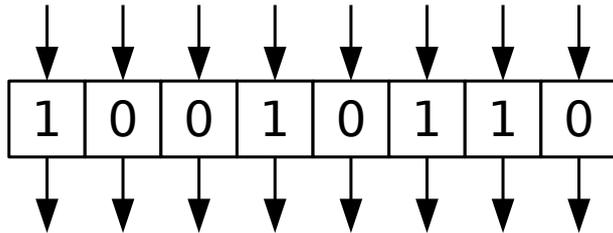
# Registros. Clasificación

---

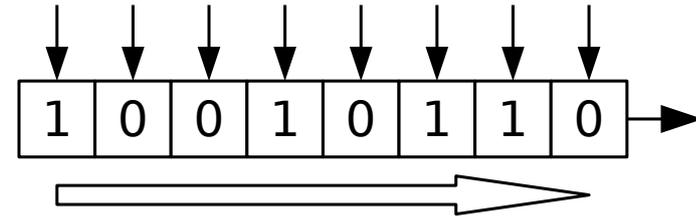
- Lectura en paralelo
  - Hay una línea de salida para cada bit ( $Z_{n-1} \dots Z_0$ ).
  - Todos los bits pueden ser leídos a la vez.
- Lectura serie
  - Solo se aplica en registros con escritura en serie.
  - Si el registro puede realizar escritura serie con desplazamiento a la derecha, hay una línea de salida que muestra el estado del bit menos significativo.
  - Si el registro puede realizar escritura serie con desplazamiento a la izquierda, hay una línea de salida que muestra el estado del bit más significativo.

# Registros. Clasificación

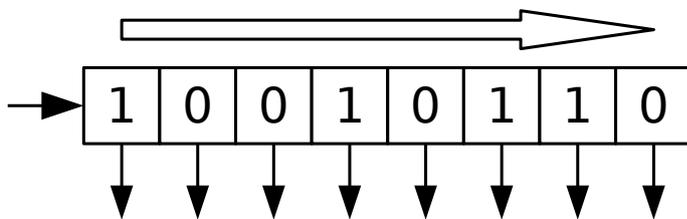
paralelo/paralelo



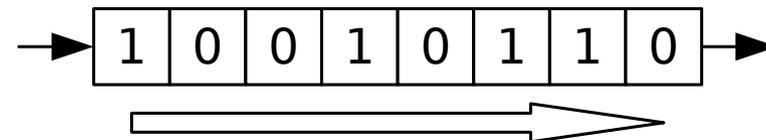
paralelo/serie



serie/paralelo



serie/serie

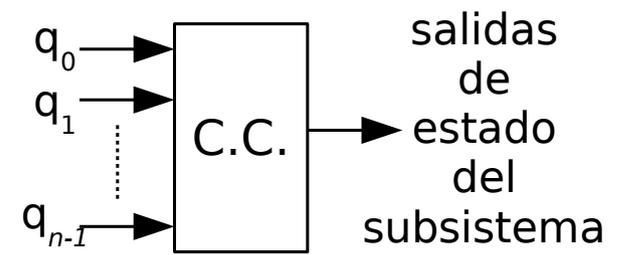
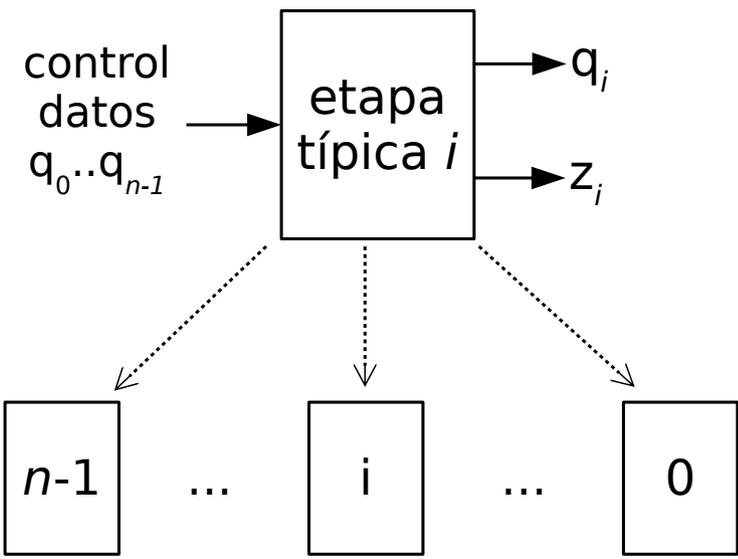
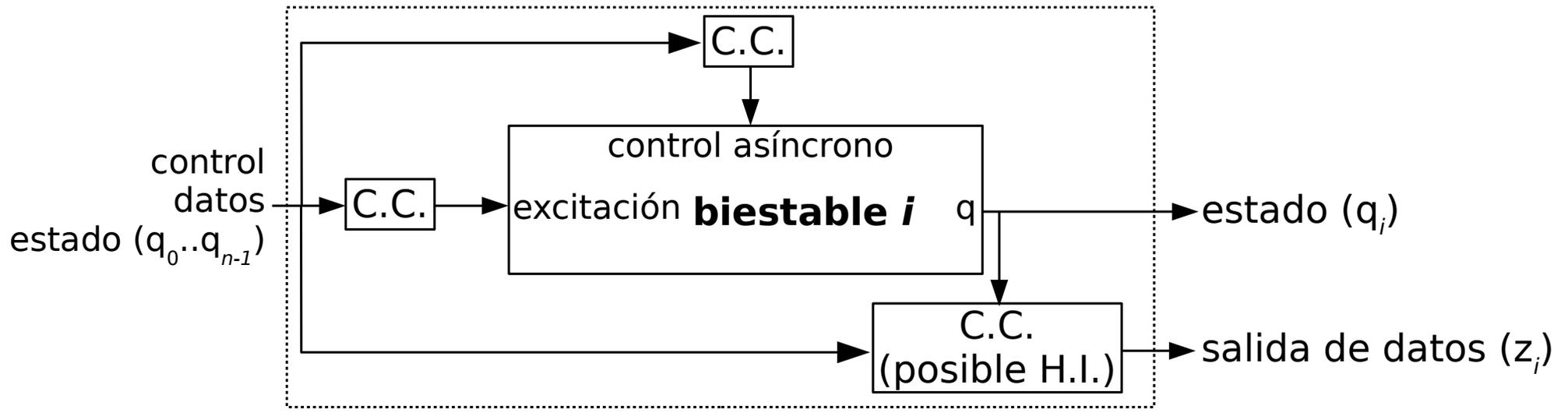


# Diseño

---

- Diseño de subsistemas secuenciales
  - Debido al gran número de estados no es factible diseñarlo con el procedimiento utilizado para máquinas de estados finitos genéricas.
  - Para implementarlo es más práctico y eficiente un enfoque modular:
    - Se diseña una etapa genérica asociada a un solo bit, esto es, un sub-subsistema con un sólo biestable que almacena uno de los bits.
    - Se replica la etapa para los  $n$  bits del subsistema.
    - Se consideran los casos especiales en los bits extremos y las señales globales.
    - La complejidad del diseño no depende del número de bits.

# Diseño modular: etapa típica



# Registro entrada paralelo/salida paralelo

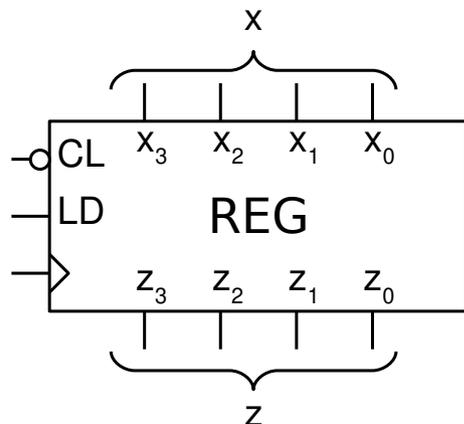


Tabla de operación

CL LD	Operación	Tipo
0 x	REG $\leftarrow$ 0	asíncrona
1 1	REG $\leftarrow$ X	síncrona
1 0	REG $\leftarrow$ REG	síncrona

Salida: z=REG

## Código Verilog

```
module reg(  
    input ck,  
    input cl,  
    input ld,  
    input [3:0] x,  
    output [3:0] z  
);  
  
    reg [3:0] q;  
  
    always @(posedge ck, negedge cl)  
        if (cl == 0)  
            q <= 0;  
        else if (ld == 1)  
            q <= x;  
  
    assign z = q;  
  
endmodule
```

# Registro entrada paralelo/salida paralelo

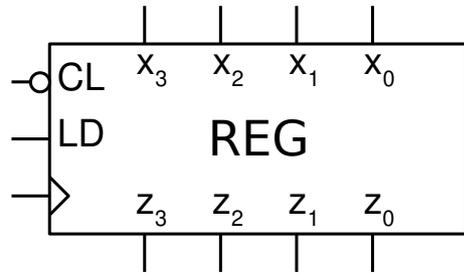


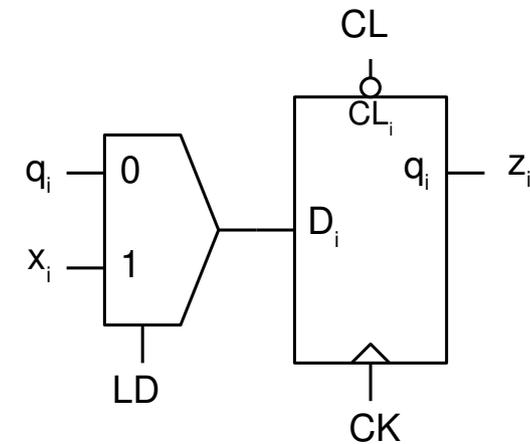
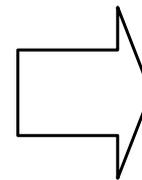
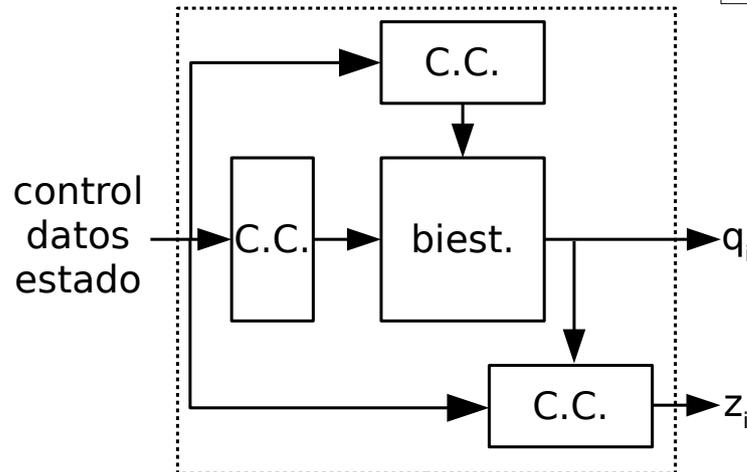
Tabla de operación asíncrona

CL	Operación	Et. típica	Entr. asínc.
0	$REG \leftarrow 0$	$Q_i = 0$	$CL_i = 0$
1	$REG \leftarrow REG$	$Q_i = q_i$	$CL_i = 1$

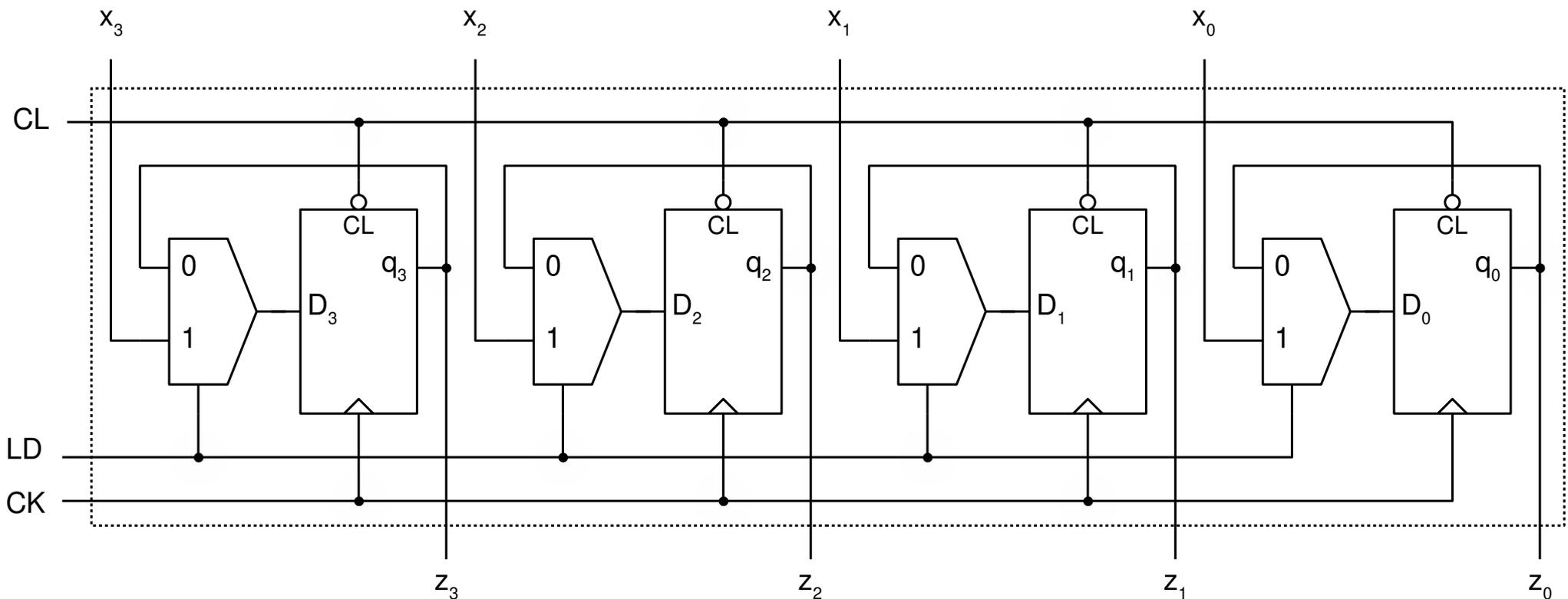
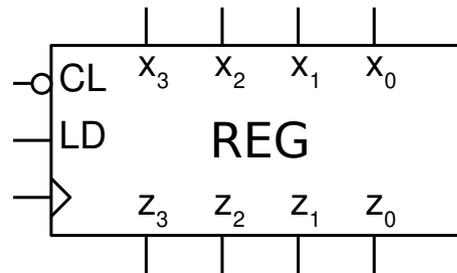
Tabla de operación síncrona

LD	Operación	Et. típica	Ecs. excit.
1	$REG \leftarrow X$	$Q_i = x_i$	$D_i = x_i$
0	$REG \leftarrow REG$	$Q_i = q_i$	$D_i = q_i$

Salida:  $z_i = q_i$



# Registro entrada paralelo/salida paralelo



# Ejercicio . Alternativas de diseño



- El método más directo para diseñar registros es con:
  - Biestables D: la entrada D es el próximo estado deseado.
  - Multiplexores: implementación directa de la tabla de operaciones.
- Un registro puede diseñarse con cualquier tipo de biestable y con cualquier tipo de técnica para la parte combinacional.
- Ejercicio:
  - a) Diseñar un registro paralelo/paralelo usando biestables D y puertas lógicas.
  - b) Diseñar un registro paralelo/paralelo usando biestables JK y multiplexores.
  - b) Diseñar un registro paralelo/paralelo usando biestables JK y puertas lógicas.

# Registro de desplazamiento

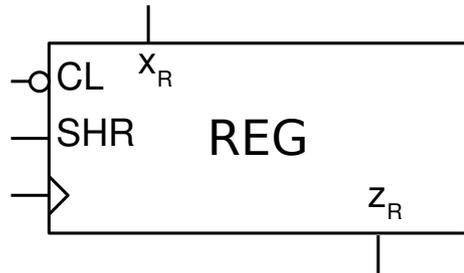


Tabla de operación

CL SHR	Operación	Tipo
0 x	REG $\leftarrow$ 0	asíncrona
1 1	REG $\leftarrow$ SHR(REG, X <sub>R</sub> )	síncrona
1 0	REG $\leftarrow$ REG	síncrona

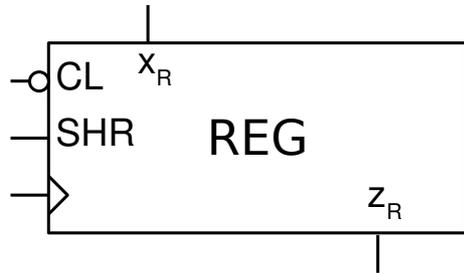
Salida:  $z_R = \text{REG}_0$

## Código Verilog

```
module reg_shr(  
    input ck,  
    input cl,  
    input shr,  
    input xr,  
    output zr  
);  
  
    reg [3:0] q;  
  
    always @(posedge ck, negedge cl)  
        if (cl == 0)  
            q <= 0;  
        else if (shr == 1)  
            q <= {xr, q[3:1]};  
  
    assign zr = q[0];  
  
endmodule
```

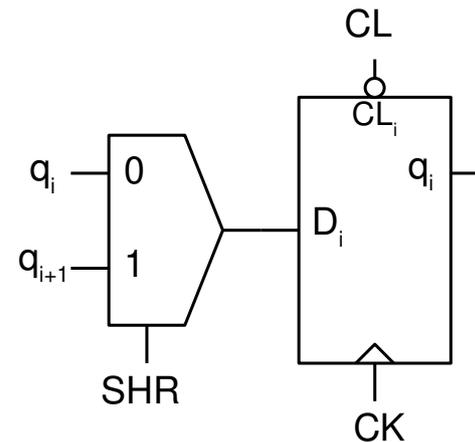
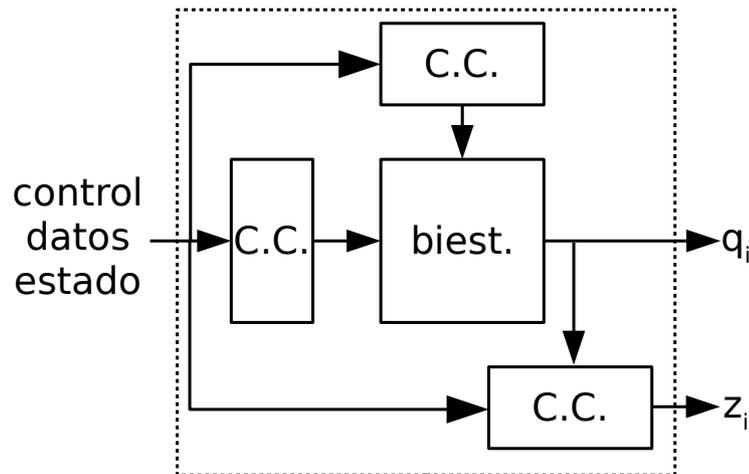
# Registro de desplazamiento

Tabla de operación síncrona

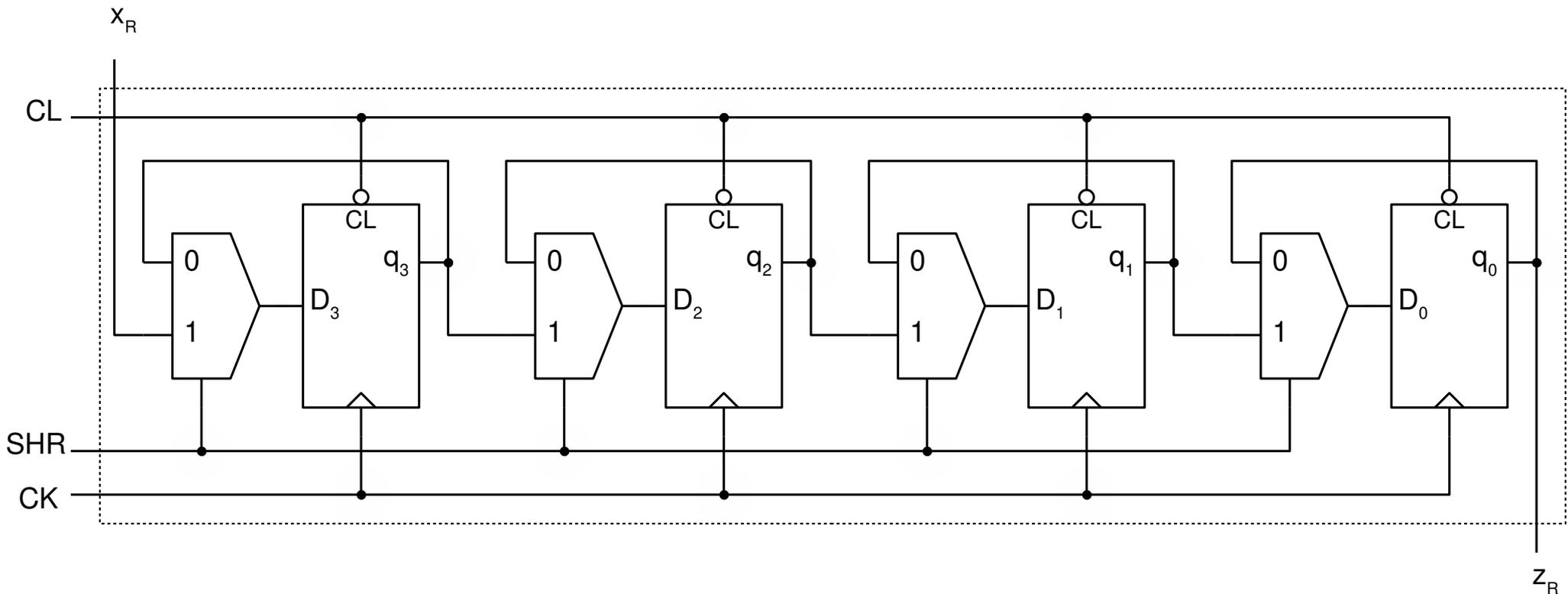
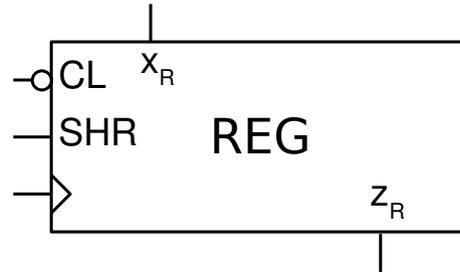


SHR	Operación	Et. típica	Et. 3	Ecs. exc.	EE et. 3
1	$REG \leftarrow SHR(REG, X_R)$	$Q_i = q_{i+1}$	$Q_3 = x_R$	$D_i = q_{i+1}$	$D_3 = x_R$
0	$REG \leftarrow REG$	$Q_i = q_i$	$Q_3 = q_3$	$D_i = q_i$	$D_3 = q_3$

Salida:  $z_R = q_0$



# Registro de desplazamiento



# Registro universal

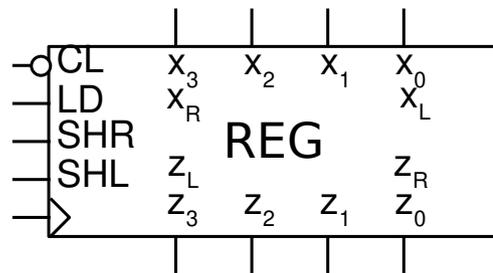


Tabla de operación

CL	LD	SHR	SHL	Operación	Tipo
0	x	x	x	$REG \leftarrow 0$	asínc.
1	1	x	x	$REG \leftarrow X$	sínc.
1	0	1	x	$REG \leftarrow SHR(REG, X_R)$	sínc.
1	0	0	1	$REG \leftarrow SHL(REG, X_L)$	sínc.
1	0	0	0	$REG \leftarrow REG$	sínc.

## Código Verilog

```

module ureg(
    input ck,
    input cl,
    input ld,
    input shr,
    input shl,
    input [3:0] x,
    output [3:0] z
);

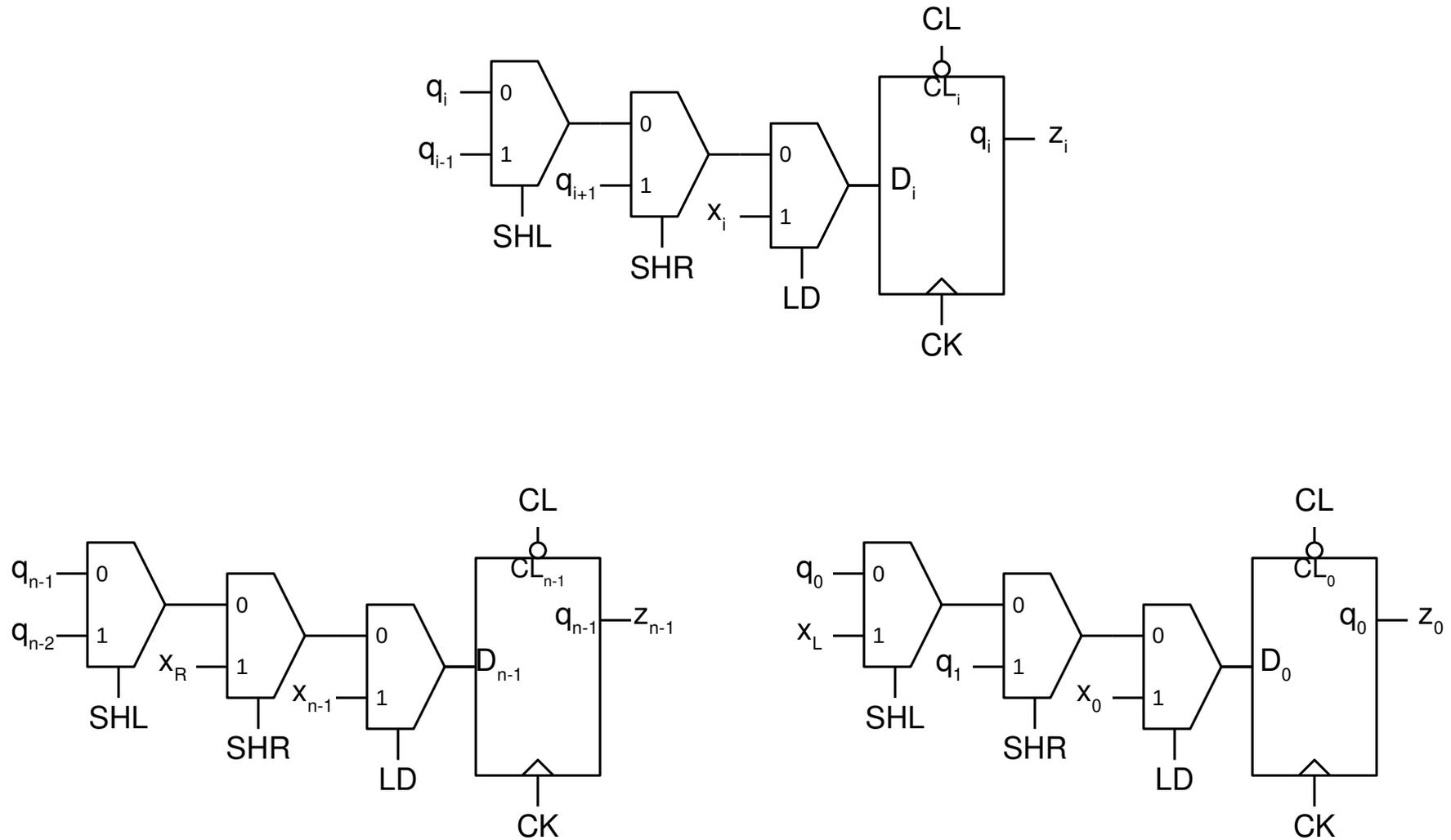
    reg [3:0] q;

    always @(posedge ck, negedge cl)
        if (cl == 0)
            q <= 0;
        else if (ld == 1)
            q <= x;
        else if (shr == 1)
            q <= {x[3], q[3:1]};
        else if (shl == 1)
            q <= {q[2:0], x[0]};

    assign z = q;

endmodule
    
```

# Registro universal



# Contadores

---

- Introducción
- Registros
- Contadores
  - Contador binario ascendente módulo  $2^n$
  - Límite de estados de cuenta
  - Contador descendente
  - Contador reversible
  - Contadores no binarios
- Diseño con subsistemas secuenciales

# Contadores

---

- Son dispositivos que cuentan los flancos de la señal de reloj
  - Al número de estados de cuenta se llama *módulo*.
  - A menudo, después del último estado de cuenta se pasa al primero (la cuenta suele ser cíclica).
- Diseño
  - La implementación es más sencilla con biestables JK o T
- Operaciones típicas
  - Cuenta ascendente
  - Cuenta descendente
  - Reinicio de cuenta. En los contadores ascendentes suele ser una puesta a cero (clear).
  - Carga de estado de cuenta
- Salidas típicas
  - Estado de cuenta
  - Fin de cuenta

# Contador binario módulo $2^n$

- El contador binario módulo  $2^n$  cuenta los flancos activos en base 2

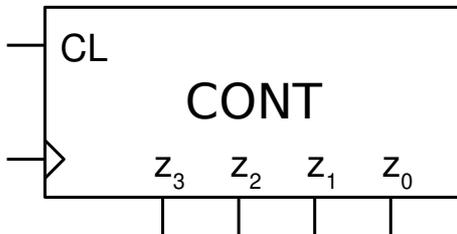


Tabla de operación

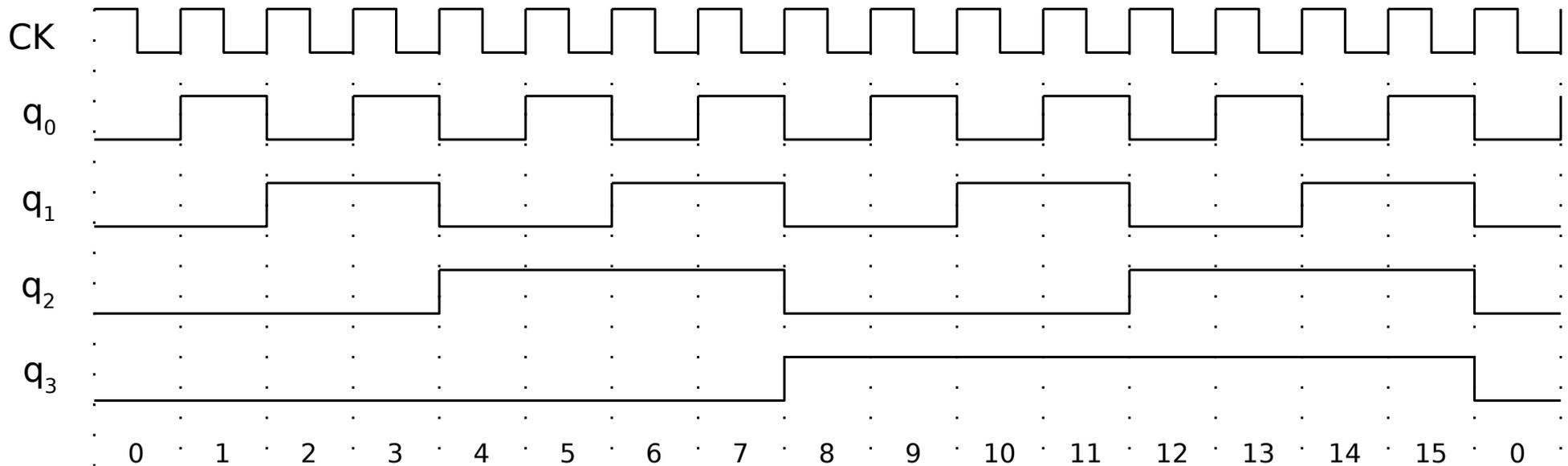
CL	Operación	Tipo
1	$\text{CONT} \leftarrow 0$	asínc.
0	$\text{CONT} \leftarrow \text{CONT} + 1 \mid_{\text{mod } 16}$	sínc.

Salida:  $z = \text{CONT}$

## Código Verilog

```
module count_mod16(  
    input ck,  
    input cl,  
    output [3:0] z  
);  
  
    reg [3:0] q;  
  
    always @(posedge ck, posedge cl)  
        if (cl == 1)  
            q <= 0;  
        else  
            q <= q + 1;  
  
    assign z = q;  
  
endmodule
```

# Contador binario módulo $2^n$ en operación de cuenta ascendente



- Cuenta ascendente

- $Q_0 = \bar{q}_0$
- $Q_i = \bar{q}_i$  si  $q_j = 1, \forall j < i$
- Si no,  $Q_i = q_i$

- $T_0 = 1$

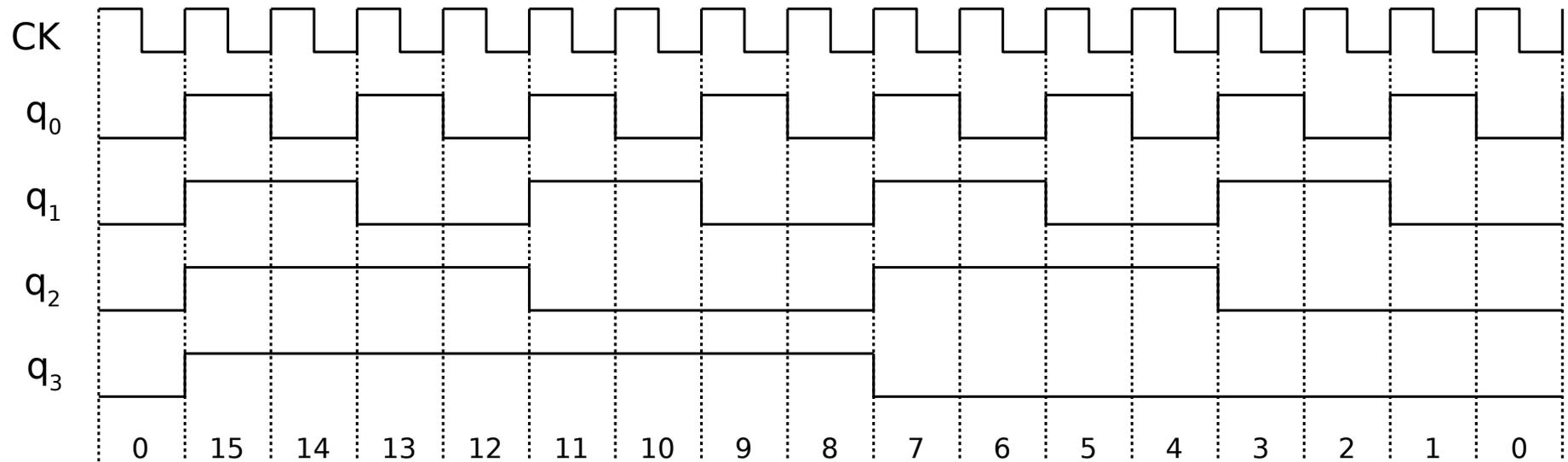
- $J_0 = K_0 = 1$

Para  $i > 0$ :

- $T_i = q_{i-1} q_{i-2} \dots q_0$

- $J_i = K_i = q_{i-1} q_{i-2} \dots q_0$

# Contador binario módulo $2^n$ en operación de cuenta descendente



- Cuenta descendente

- $Q_0 = \bar{q}_0$
- $Q_i = \bar{q}_i$  si  $q_j = 0, \forall j < i$
- Si no,  $Q_i = q_i$

- $T_0 = 1$

- $J_0 = K_0 = 1$

Para  $i > 0$ :

- $T_i = \text{NOT}(q_{i-1} + q_{i-2} + \dots + q_0) = \bar{q}_{i-1} \bar{q}_{i-2} \dots \bar{q}_0$

- $J_i = K_i = \bar{q}_{i-1} \bar{q}_{i-2} \dots \bar{q}_0$

# Contador binario ascendente módulo $2^n$

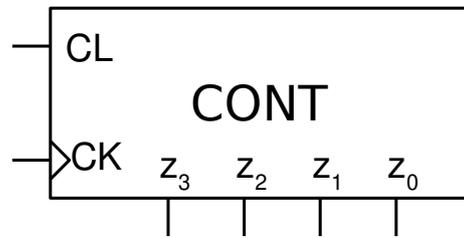
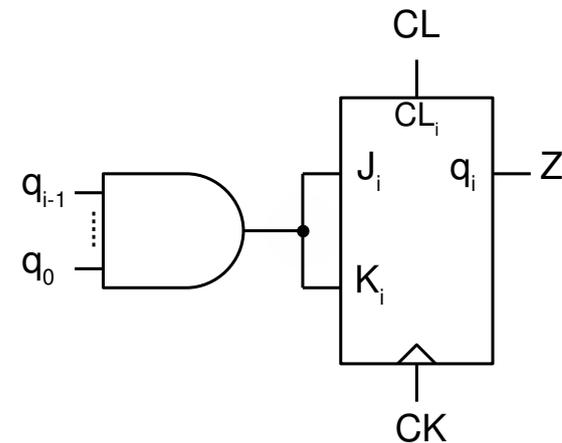
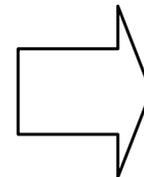
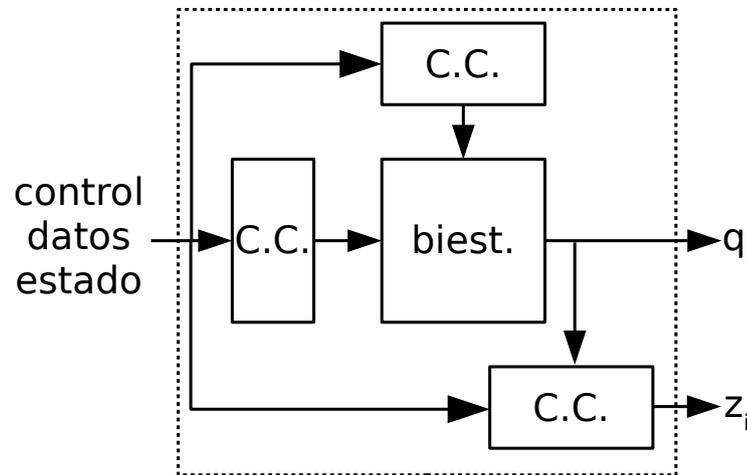
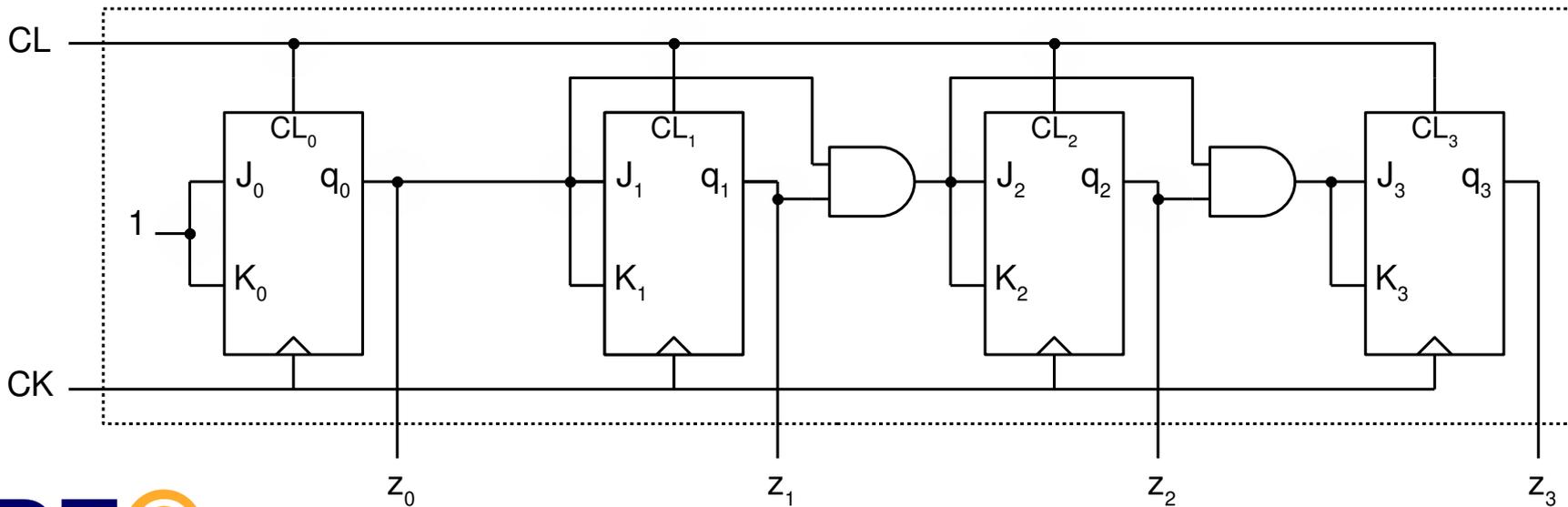
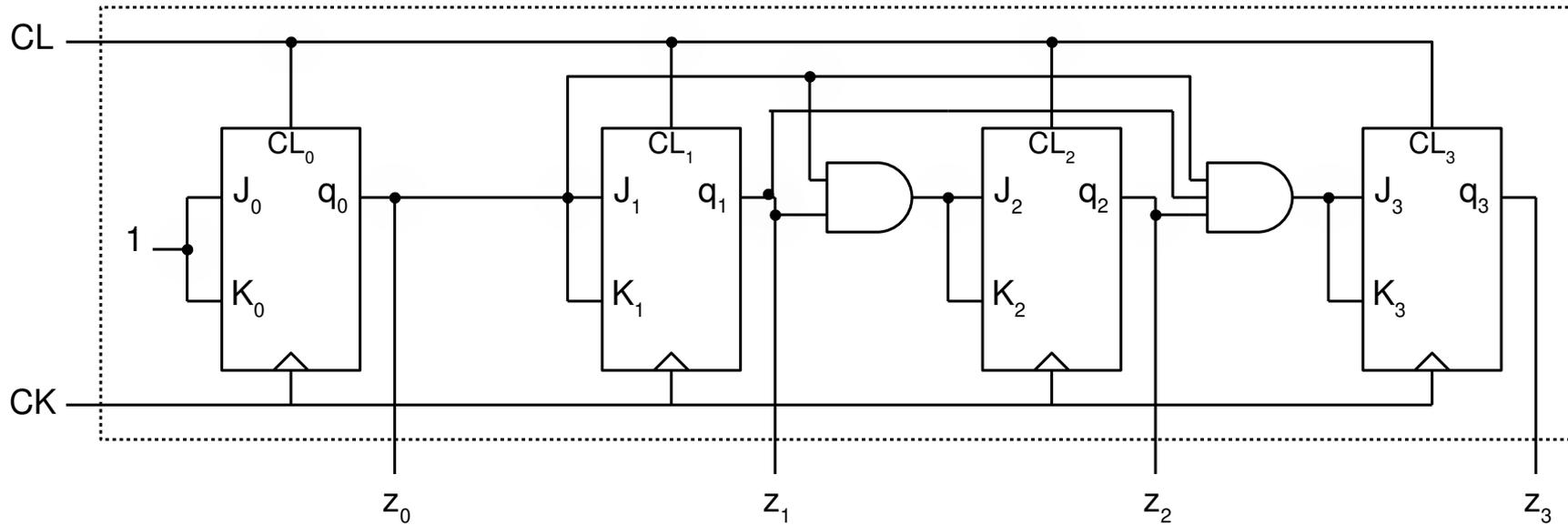


Tabla de operación síncrona

Operación	Et. típica	Et. 0
$CONT \leftarrow CONT + 1 \mid_{\text{mod } 16}$	$J_i = K_i = q_{i-1} \dots q_0$	$J_0 = K_0 = 1$



# Contador binario ascendente módulo 2<sup>n</sup>. Circuito



# Ejercicio . Alternativas de diseño



- El biestable JK suele ser el más adecuado para diseñar contadores porque:
  - Es fácil implementar las operaciones de cuenta:
    - Ascendente:  $J_i = K_i = \overline{q_0} \dots \overline{q_{i-1}}$
    - Descendente:  $J_i = K_i = \overline{q_0} \dots \overline{q_{i-1}}$
  - Facilita implementar otras operaciones, como puesta a cero síncrona
- Como con los registros, un contador puede diseñarse con cualquier tipo de biestable.
- Ejercicio:
  - a) Diseñar un contador ascendente módulo  $2^n$  con biestables T.
  - b) Diseñar un contador ascendente módulo  $2^n$  con biestables D.

# Contador binario módulo $2^n$ con puesta a cero síncrona y habilitación

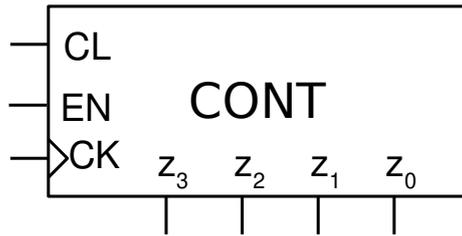


Tabla de operación

CL EN	Operación	Tipo
1 x	$\text{CONT} \leftarrow 0$	sínc.
0 1	$\text{CONT} \leftarrow \text{CONT} + 1 \mid_{\text{mod } 16}$	sínc.
0 0	$\text{CONT} \leftarrow \text{CONT}$	sínc.

Salida:  $z = \text{CONT}$

## Código Verilog

```
module count_mod16(
    input ck,
    input cl,
    input en,
    output [3:0] z
);

    reg [3:0] q;

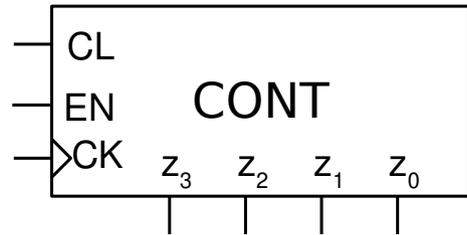
    always @(posedge ck)
        if (cl == 1)
            q <= 0;
        else if (en == 1)
            q <= q + 1;

    assign z = q;

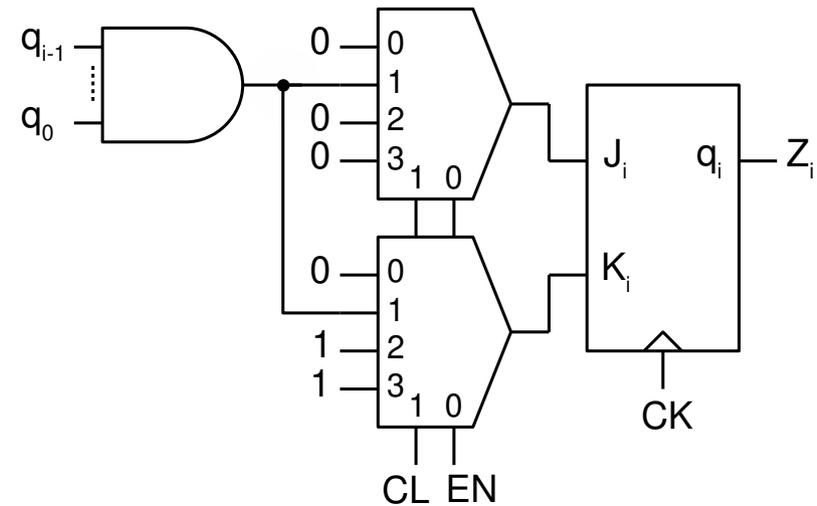
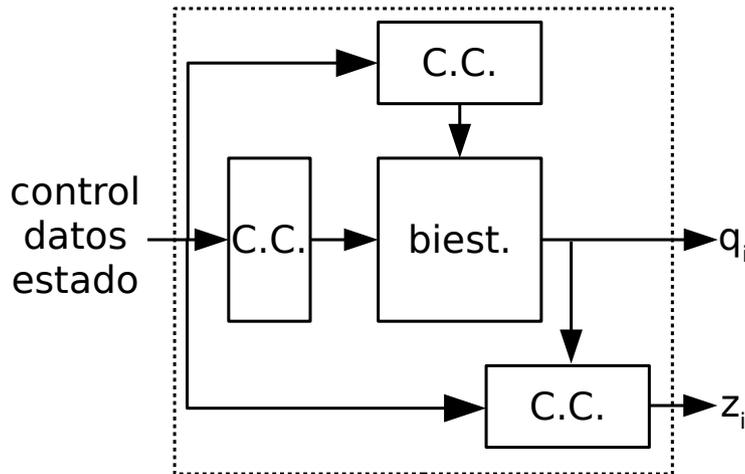
endmodule
```

# Contador binario módulo $2^n$ con puesta a cero síncrona y habilitación

Tabla de operación síncrona

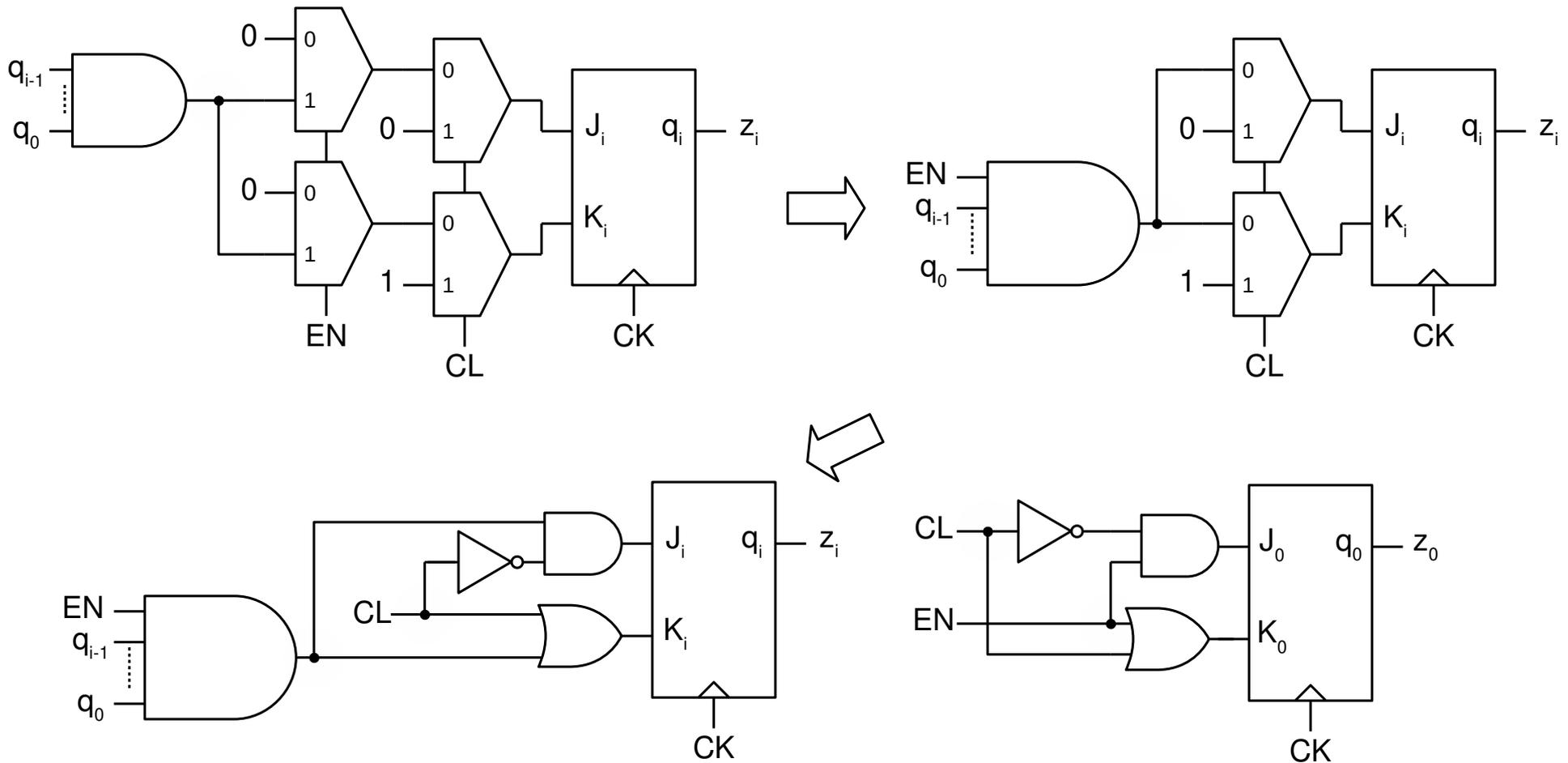


CL EN	Operación	Et. típica	Et. 0
1 x	$\text{CONT} \leftarrow 0$	$J_i=0, K_i=1$	$J_0=0, K_0=1$
0 1	$\text{CONT} \leftarrow \text{CONT} + 1 \mid_{\text{mod } 16}$	$J_i=K_i=q_{i-1} \dots q_0$	$J_0=K_0=1$
0 0	$\text{CONT} \leftarrow \text{CONT}$	$J_i=K_i=0$	$J_0=K_0=0$



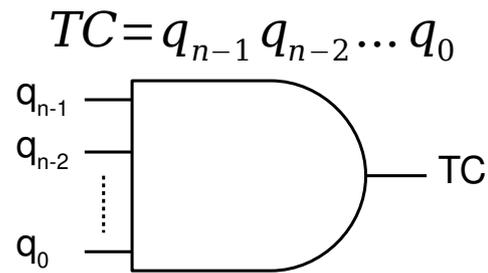
# Contador binario módulo $2^n$ con puesta a cero síncrona y habilitación

Implementaciones alternativas



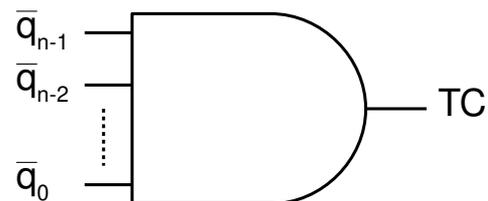
# Salida de fin de cuenta

- La salida F.C. (Terminal Count, Ripple Carry Output) se activa únicamente en el último estado de cuenta.
- Cuenta ascendente en contadores binarios
  - Normalmente  $TC = 1$  si y solo si  $q_{(2)} = 2^n - 1$

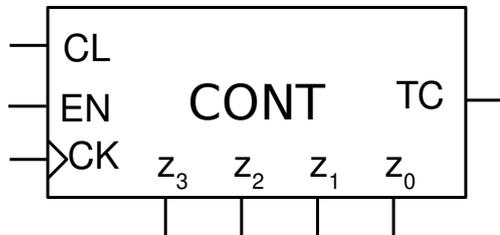


- Cuenta descendente en contadores binarios
  - Normalmente  $TC = 1$  si y solo si  $q_{(2)} = 0$

$$TC = \bar{q}_{n-1} \bar{q}_{n-2} \dots \bar{q}_0 = \overline{q_{n-1} + q_{n-2} + \dots + q_0}$$



# Salida de fin de cuenta



## Código Verilog

```
module count_mod16(  
    input ck,  
    input cl,  
    input en,  
    output [3:0] z,  
    output tc  
);  
  
    reg [3:0] q;  
  
    always @(posedge ck)  
        if (cl == 1)  
            q <= 0;  
        else if (en == 1)  
            q <= q + 1;  
  
    assign z = q;  
    assign tc = &q;  
  
endmodule
```

# Unión de contadores

---

- Se pueden combinar contadores para obtener un nuevo contador con mayor número de estados de cuenta que los originales.
- Combinando dos contadores módulo  $k$  y  $l$  se puede conseguir un nuevo contador módulo  $k \cdot l$ .
- Ejemplo: se puede obtener un contador módulo 256 (8 bits) a partir de dos contadores módulo 16 (4 bits).
- La combinación de contadores se puede hacer de forma simple si los contadores poseen entradas de control y estado adecuadas. Ej:
  - Habilitación
  - Fin de cuenta



# Ejercicio

## Combinación de contadores (1)

---

- Queremos:
  - Contador ascendente módulo 16 (4 bits) con CL (clear) síncrono
- Tenemos:
  - Contador ascendente módulo 4 (2 bits) con CL (clear) síncrono, EN (enable) y TC (fin de cuenta)
- Análisis del problema:
  - Necesitamos dos contadores módulo 4: C0 y C1. Si llamamos a sus salidas de estado Z0 y Z1 respectivamente y Z a la salida del contador módulo 16:
    - $Z[3:2] = Z1[1:0]$
    - $Z[1:0] = Z0[1:0]$
  - Será totalmente síncrono:
    - $CK0 = CK1 = CK$
  - Los dos contadores se ponen a cero a la vez:
    - $CL0 = CL1 = CL$
  - C1 se incrementa sólo cuando  $C0 = 3$ :
    - $EN1 = TC0$

# Ejercicio

## Combinación de contadores (2)



- Queremos:
  - Contador binario ascendente módulo 256 (8 bits) con CL (clear) síncrono, EN (enable) y TC (fin de cuenta)
- Tenemos:
  - Contadores binarios ascendentes módulo 16 (4 bits) con CL (clear) síncrono, EN (enable) y TC (fin de cuenta)
- Análisis del problema:
  - Necesitamos dos contadores MOD16: C0 y C1
    - $z[7:4] = z1[3:0]$ ,  $z[3:0] = z0[3:0]$
  - Será totalmente síncrono:
    - $CK0 = CK1 = CK$
  - Los dos contadores se ponen a cero a la vez:
    - $CL0 = CL1 = CL$
  - C0 se incrementa sólo cuando  $EN=1$ 
    - $EN0 = EN$
  - C1 se incrementa sólo cuando  $EN=1$  y  $C0 = 15$ :
    - $EN1 = EN \cdot TC0$
  - TC se activa cuando  $C=255$ , esto es,  $C0=15$  y  $C1=15$ :
    - $TC = TC0 \cdot TC1$

# Contador binario módulo $2^n$ con habilitación y carga

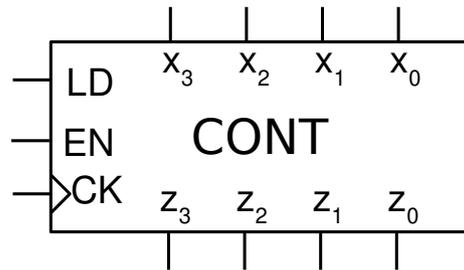


Tabla de operación

LD EN	Operación	Tipo
1 x	$\text{CONT} \leftarrow X$	sínc.
0 1	$\text{CONT} \leftarrow \text{CONT} + 1 \mid_{\text{mod } 16}$	sínc.
0 0	$\text{CONT} \leftarrow \text{CONT}$	sínc.

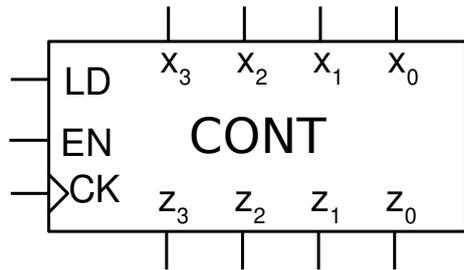
Salida:  $z = \text{CONT}$

## Código Verilog

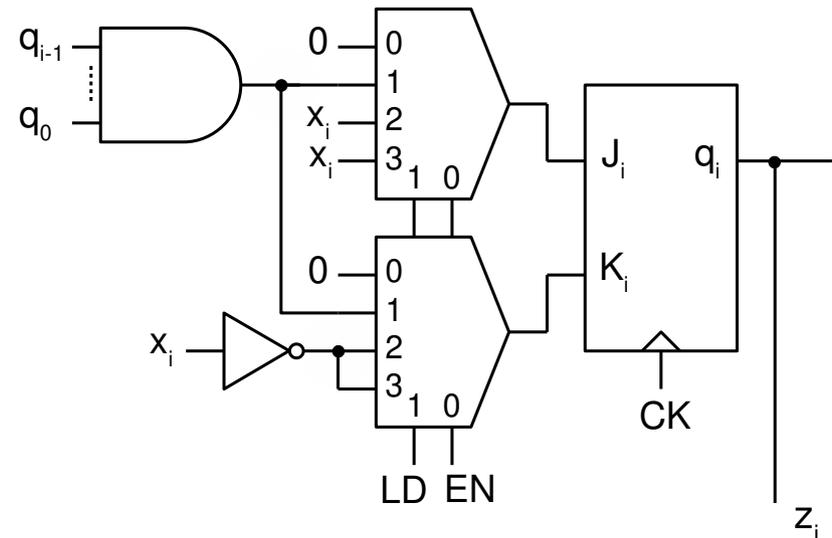
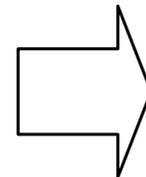
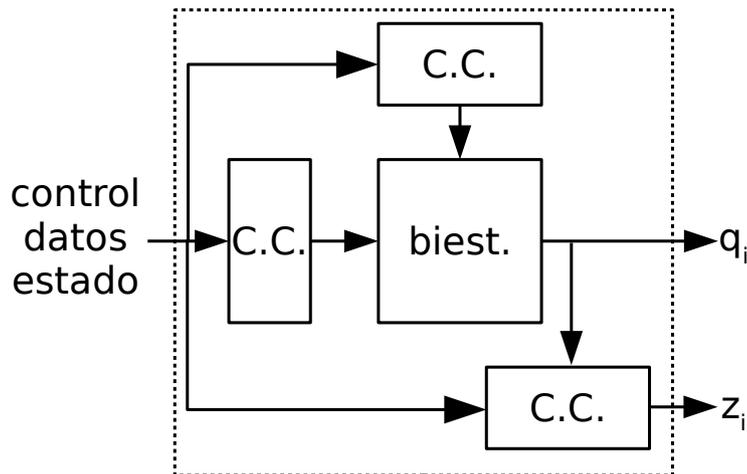
```
module count_mod16(  
    input ck,  
    input ld,  
    input en,  
    input [3:0] x,  
    output [3:0] z  
);  
  
    reg [3:0] q;  
  
    always @(posedge ck)  
        if (ld == 1)  
            q <= x;  
        else if (en == 1)  
            q <= q + 1;  
  
    assign z = q;  
  
endmodule
```

# Contador binario módulo $2^n$ con habilitación y carga

Tabla de operación síncrona



LD EN	Operación	Et. típica	Et. 0
1 x	$\text{CONT} \leftarrow X$	$J_i = x_i, K_i = \bar{x}_i$	$J_0 = x_0, K_0 = \bar{x}_0$
0 1	$\text{CONT} \leftarrow \text{CONT} + 1 \mid_{\text{mod } 16}$	$J_i = K_i = q_{i-1} \dots q_0$	$J_0 = K_0 = 1$
0 0	$\text{CONT} \leftarrow \text{CONT}$	$J_i = K_i = 0$	$J_0 = K_0 = 0$



# Contador de módulo arbitrario

---

- Los contadores de un módulo  $k$  que no es potencia de 2 se obtienen normalmente limitando los estados de cuenta de un contador módulo  $2^n > k$ .
- Casos
  - Límite superior:  $0 \dots \max$ ,  $\max < 2^n - 1$
  - Límite inferior:  $\min \dots 2^n - 1$ ,  $\min > 0$
  - Límites inferior y superior:  $\max \dots \min$ ,  $\min > 0$ ,  $\max < 2^n - 1$
- Estrategia
  - Se detecta la llegada al último estado de cuenta y se activa una operación para volver a cero (CL o LD), o para volver a  $k$  (LD).
  - Si CL/LD son asíncronos, se ha de detectar el estado de cuenta posterior a último para activar la operación deseada, esto implica la aparición transitoria de dicho estado.
  - Esto último requiere de un análisis de tiempo detallado.

# Ejemplo: contador BCD

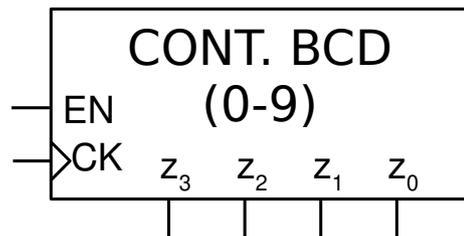


Tabla de operación

EN	Operación	Tipo
1	$\text{CONT} \leftarrow \text{CONT} + 1 \mid_{\text{mod } 10}$	sínc.
0	$\text{CONT} \leftarrow \text{CONT}$	sínc.

Salida:  $z = \text{CONT}$

## Código Verilog

```
module count_mod10(
    input ck,
    input cl,
    input en,
    output [3:0] z,
);

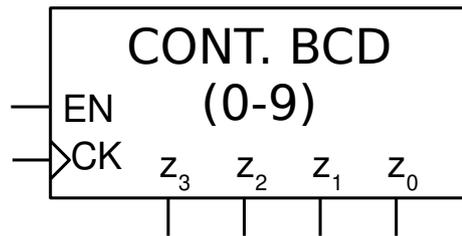
    reg [3:0] q;

    always @(posedge ck)
        if (en == 1)
            if (q == 9)
                q <= 0;
            else
                q <= q + 1;

    assign z = q;

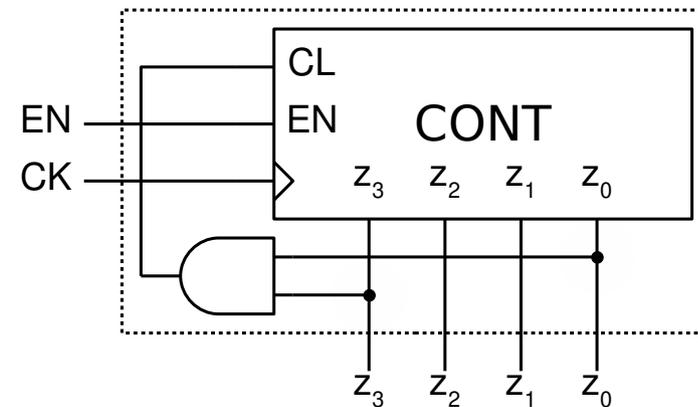
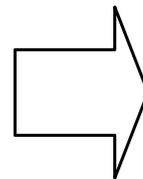
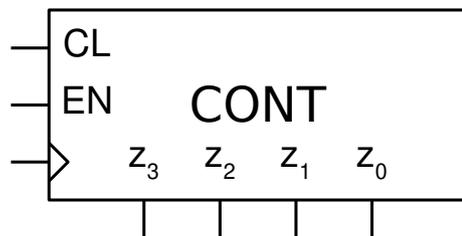
endmodule
```

# Ejemplo: contador BCD



$z_3z_2$ \ $z_1z_0$	00	01	11	10
00	0	0	-	0
01	0	0	-	1
11	0	0	-	-
10	0	0	-	-

CL =  $z_3 z_0$



# Ejercicio. Contador de 3 a 12

---



- Diseñar un contador cíclico de 3 a 12 con las funciones:
  - Inicialización a 3 (INIT)
  - Habilitación (EN)
  - Salida de fin de cuenta (C)
- Basándose en un contador binario con las funciones:
  - Puesta a cero (CL)
  - Habilitación (EN)
  - Carga de dato (LD).

# Contador binario descendente mód. $2^n$ con puesta a 0, habilit. y fin de cuenta

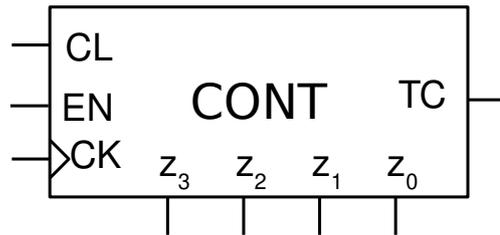


Tabla de operación

CL EN	Operación	Tipo
1 x	$\text{CONT} \leftarrow 0$	sínc.
0 1	$\text{CONT} \leftarrow \text{CONT} - 1 \mid_{\text{mod } 16}$	sínc.
0 0	$\text{CONT} \leftarrow \text{CONT}$	sínc.

TC = 1 sii [CONT] = 0000  
Z = CONT

## Código Verilog

```
module count_mod16(
    input ck,
    input cl,
    input en,
    output [3:0] z,
    output tc
);

    reg [3:0] q;

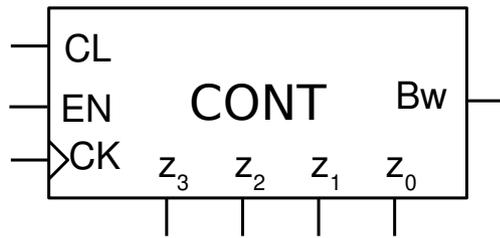
    always @(posedge ck)
        if (cl == 1)
            q <= 0;
        else if (en == 1)
            q <= q - 1;

    assign z = q;
    assign tc = ~(|q);

endmodule
```

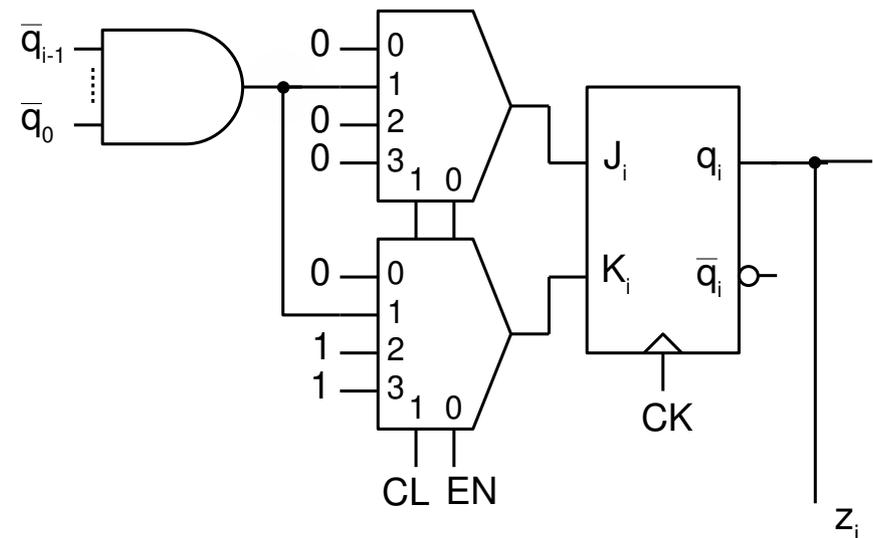
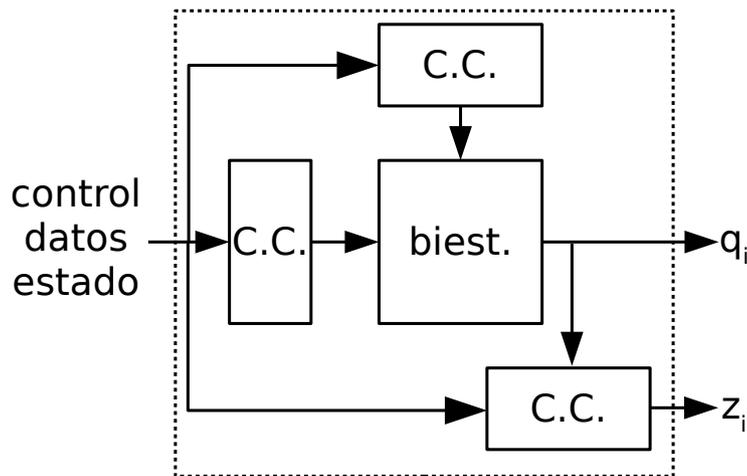
# Contador binario descendente mód. $2^n$ con puesta a 0, habilit. y fin de cuenta

Tabla de operación síncrona



CL EN	Operación	Et. típica	Et. 0
1 x	$\text{CONT} \leftarrow 0$	$J_i=0, K_i=1$	$J_0=0, K_0=1$
0 1	$\text{CONT} \leftarrow \text{CONT}-1 \pmod{16}$	$J_i=K_i=\bar{q}_{i-1} \dots \bar{q}_0$	$J_0=K_0=1$
0 0	$\text{CONT} \leftarrow \text{CONT}$	$J_i=K_i=0$	$J_0=K_0=0$

TC = 1 sii [CONT] = 0000



# Contador reversible

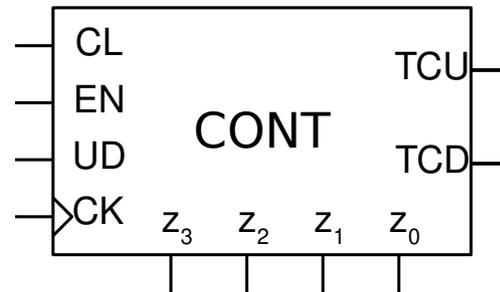


Tabla de operación

CL EN UD	Operación	Tipo
1 x x	$\text{CONT} \leftarrow 0$	asínc.
0 0 x	$\text{CONT} \leftarrow \text{CONT}$	sínc.
0 1 0	$\text{CONT} \leftarrow \text{CONT} + 1 \Big _{\text{mod } 16}$	sínc.
0 1 1	$\text{CONT} \leftarrow \text{CONT} - 1 \Big _{\text{mod } 16}$	sínc.

TCU = 1 sii [CONT] = 1111

TCD = 1 sii [CONT] = 0000

Z = CONT

# Contador reversible

## Código Verilog

```
module rev_counter1(
    input ck, cl, en, ud,
    output [3:0] z, TCU, TCD
);

reg [3:0] q;

always @(posedge ck, posedge cl)
begin
    if (cl == 1)
        q <= 0;
    else if (en == 1)
        if (ud == 0)
            q <= q + 1;
        else
            q <= q - 1;
end

assign z = q;
assign TCU = &q;
assign TCD = ~(|q);

endmodule
```

## Código Verilog

```
module rev_counter2(
    input ck, cl, en, ud,
    output [3:0] z, TCU, TCU
);

reg [3:0] q;

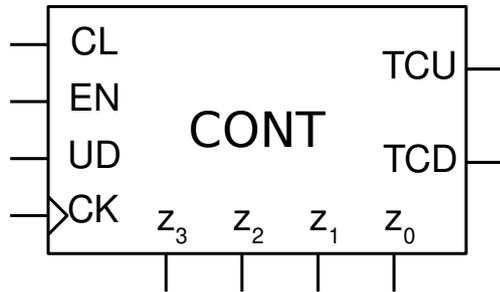
always @(posedge ck, posedge cl)
begin
    if (cl)
        q <= 0;
    else if (en)
        if (!ud)
            q <= q + 1;
        else
            q <= q - 1;
end

assign z = q;
assign TCU = &q;
assign TCU = ~(|q);

endmodule
```

# Contador reversible

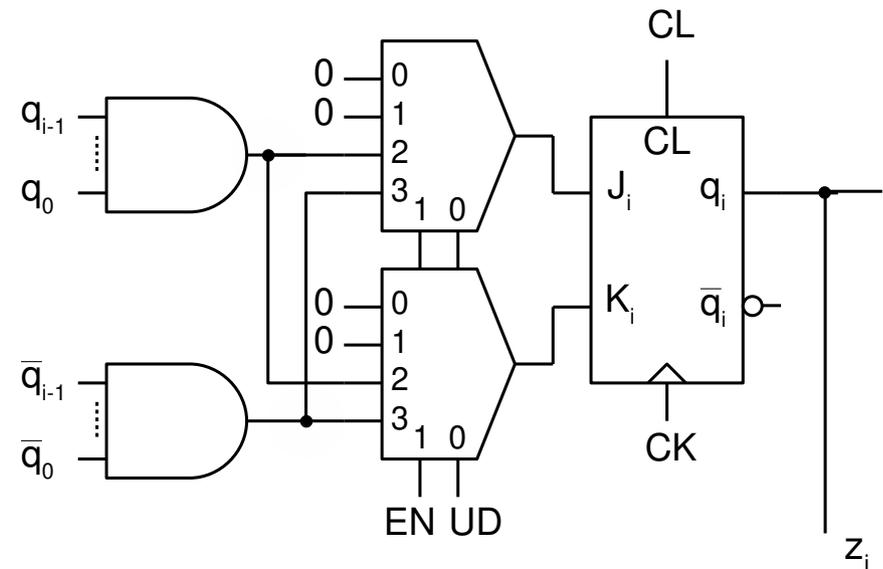
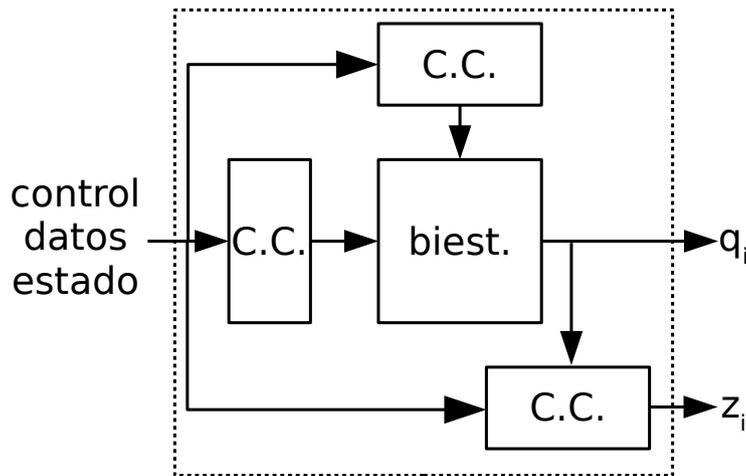
Tabla de operación síncrona



EN UD	Operación	Et. típica	Et. 0
0 x	$\text{CONT} \leftarrow \text{CONT}$	$J_i = K_i = 0$	$J_0 = K_0 = 0$
1 0	$\text{CONT} \leftarrow \text{CONT} + 1 \pmod{16}$	$J_i = K_i = q_{i-1} \dots q_0$	$J_0 = K_0 = 1$
1 1	$\text{CONT} \leftarrow \text{CONT} - 1 \pmod{16}$	$J_i = K_i = \bar{q}_{i-1} \dots \bar{q}_0$	$J_0 = K_0 = 1$

TCU = 1 sii [CONT] = 1111

TCD = 1 sii [CONT] = 0000





# Ejercicio . Contador reversible

- Diseñar un contador reversible con las siguientes funciones

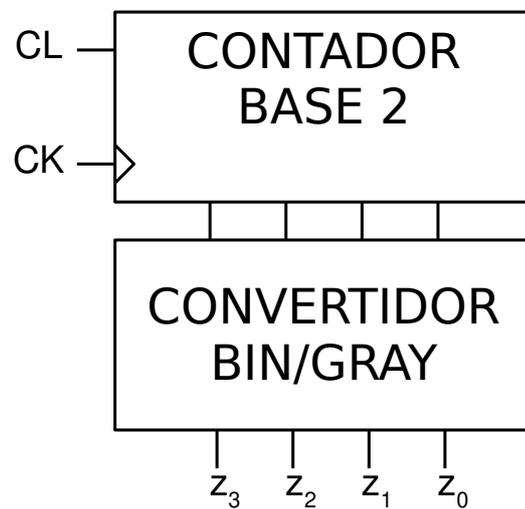
CL, UP, DW	Operación	Tipo
1xx	$\text{CONT} \leftarrow 0$	sínc.
01x	$\text{CONT} \leftarrow (\text{CONT}+1) \bmod 16$	sínc.
001	$\text{CONT} \leftarrow (\text{CONT}-1) \bmod 16$	sínc.
000	$\text{CONT} \leftarrow \text{CONT}$	sínc.

# Contadores no binarios

---

- Secuencia no binaria (ej. Gray)
- Implementación:
  - A partir de contador binario y convertidor de código
  - Diseño nativo
    - Contadores de desplazamiento / contador en anillo

# Contador Gray con convertidor de código



$z_3$	$z_2$	$z_1$	$z_0$
0	0	0	0
0	0	0	1
0	0	1	1
0	0	1	0
0	1	1	0
0	1	1	1
0	1	0	1
0	1	0	0
1	1	0	0
1	1	0	1
1	1	1	1
1	1	1	0
1	0	1	0
1	0	1	1
1	0	0	1
1	0	0	0

## Código Verilog

```
module graycounter_mod16(
    input wire ck,
    input wire cl,
    output [3:0] z
);

    reg [3:0] q;

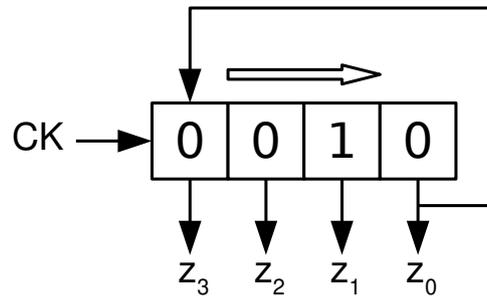
    always @(posedge ck)
        if (cl == 1)
            q <= 0;
        else
            q <= q + 1;

    assign z = q ^ (q >> 1);

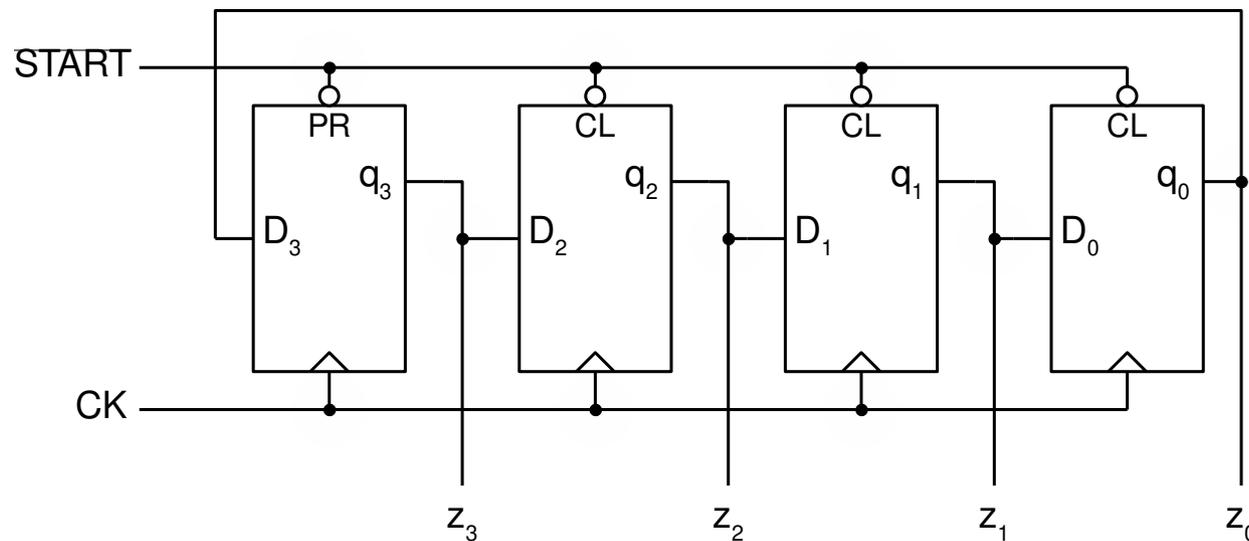
endmodule
```

# Contador en anillo

- Cuenta en código one hot o en código one cold.



$z_3$	$z_2$	$z_1$	$z_0$
1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1
1	0	0	0
...	...	...	...



# Ejercicio



- Diseña un contador en anillo de 8 bits con desplazamiento a la izquierda con las siguientes entradas activas en nivel alto:
  - START: regresa al valor inicial 00000001
  - EN: habilitación
- Emplea un registro universal como el visto en el tema.

START,EN	Operación	Tipo
1x	$\text{CONT} \leftarrow 1$	sínc.
01	$\text{CONT} \leftarrow \text{SHL}(\text{CONT}, \text{CONT}_7)$	sínc.
00	$\text{CONT} \leftarrow \text{CONT}$	sínc.

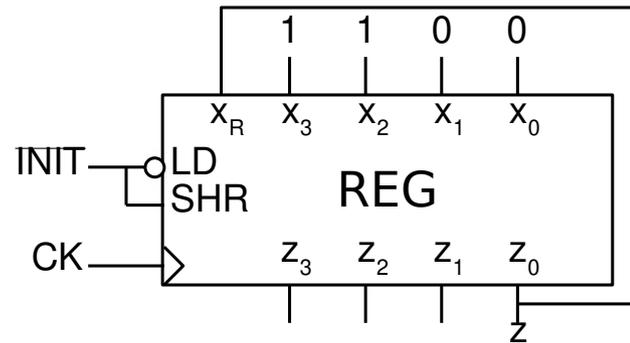
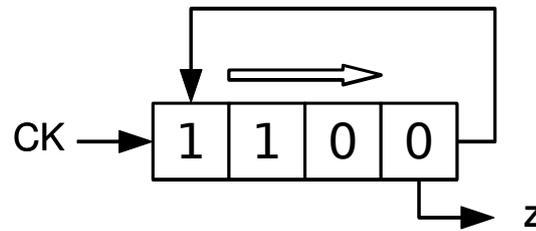
# Diseño con subsistemas secuenciales

---

- Introducción
- Registros
- Contadores
- Diseño con subsistemas secuenciales
  - Detectores y generadores de secuencia
  - Ejemplos

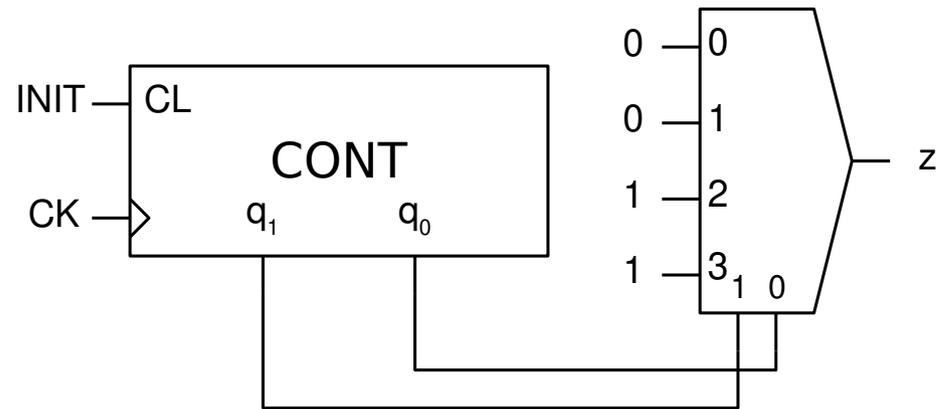
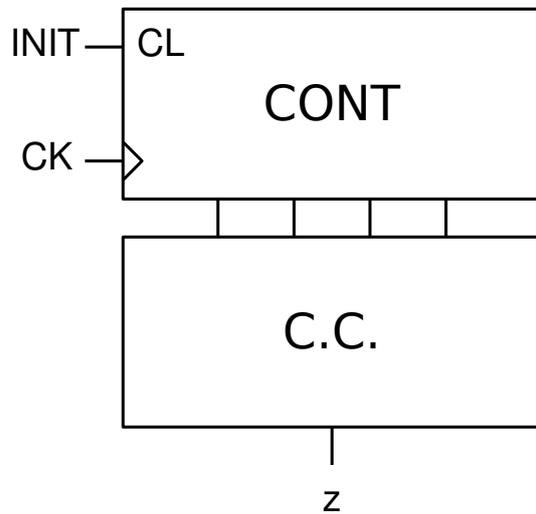
# Generador de secuencia

- Generador de la secuencia 0011 con registro de desplazamiento



# Generador de secuencia

- Generador de la secuencia 0011 con contador y CC



# Detector de secuencia

