# Tema 8. Subsistemas secuenciales

## Circuitos Electrónicos Digitales

Jorge Juan <jjchico@dte.us.es> 

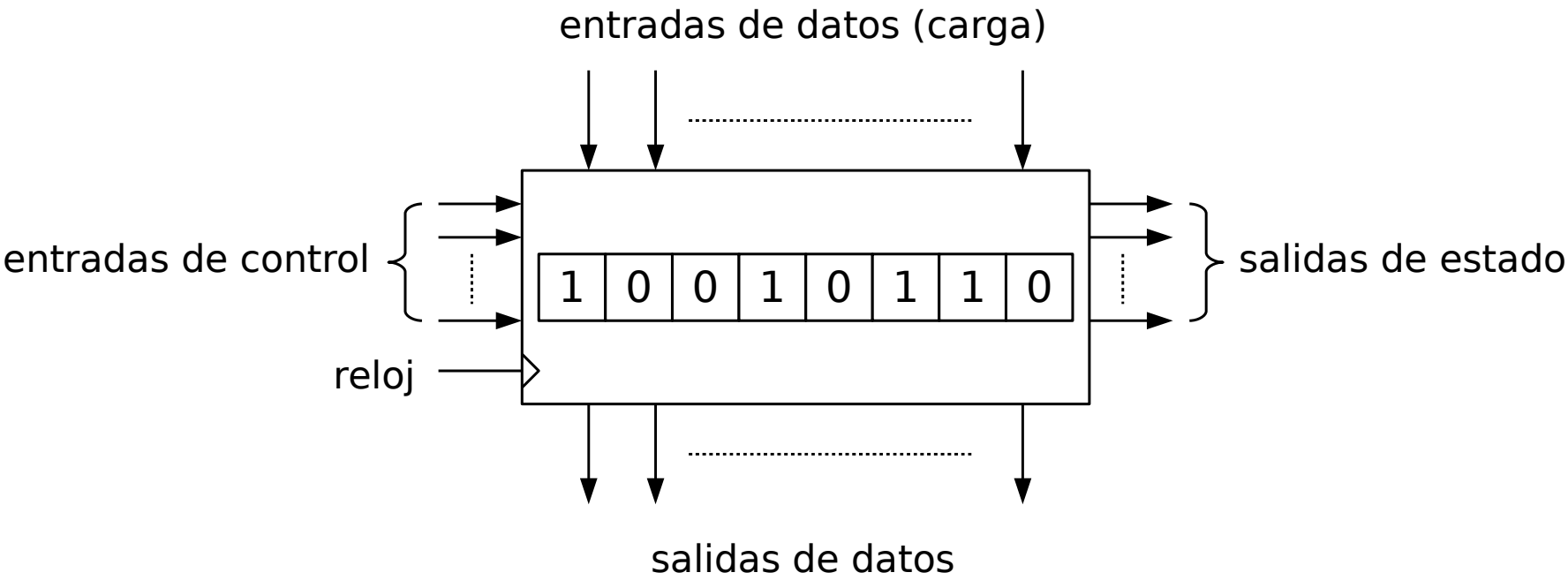
# Contenidos

- Introducción
- Registros
- Contadores
- Diseño con subsistemas secuenciales

# Introducción

- Subsistema secuencial
  - Circuito formado por *n* biestables (*n* bits) que operan de forma conjunta para la realización de una tarea o tareas determinadas.
  - Su operación se interpreta en base al dato de *n* bits que almacenan y no en base a cada bit por separado.
  - Su funcionalidad es lo bastante general para encontrar aplicación en una diversidad de problemas de diseño de circuitos secuenciales.
- Tipos básicos de subsistemas secuenciales
  - Registros: almacenan un dato para su uso posterior
  - Contadores: proporcionan una secuencia de números consecutivos (cuenta)

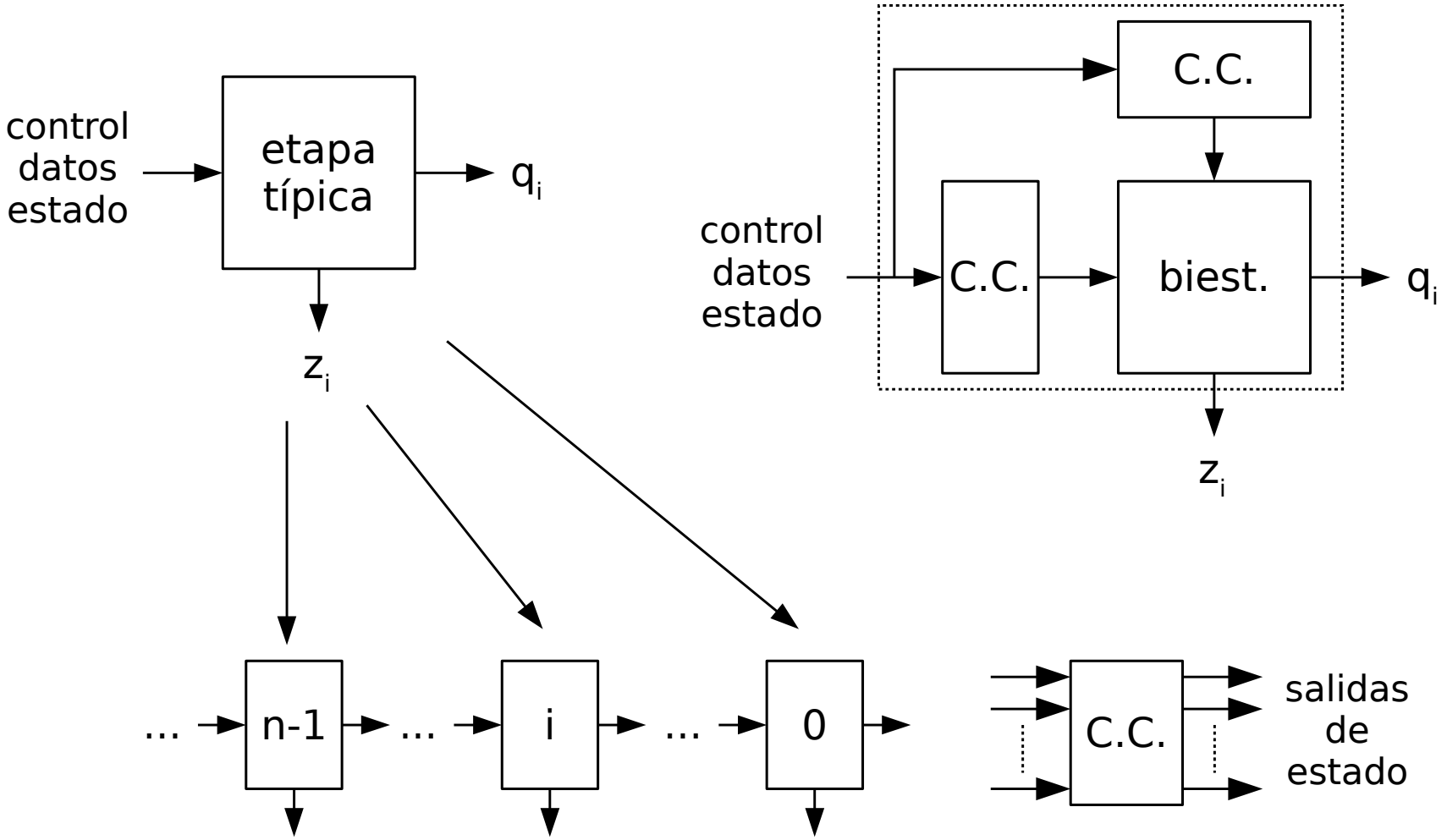
# Introducción



# Introducción

- Señales de control
  - Las señales de control determinan la operación a realizar.
  - Pueden ser *asíncronas*: alteran el dato de forma inmediata tras la activación de la señal de control
  - Pueden ser *síncronas*: alteran el dato en el flanco activo de la señal de reloj si la señal de control está activada
  - Señales de control genéricas: puesta a cero (clear -CL-), inhibición (INH), 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 es cero, fin de estado de cuenta, etc.

# Diseño modular



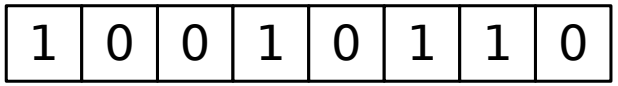
# Introducción

- Estructura modular
  - Todas las etapas del subsistema operan de forma similar
  - La función no depende del número de bits, salvo por el rango de valores del dato que pueden manejar.
- Diseño de subsistemas secuenciales
  - Un subsistema secuencial puede describirse como una máquina de estados finitos, pero no suele ser apropiado: pueden tener gran número de estados, pero todos ellos similares.
  - Diseño más práctico y eficiente con un enfoque modular:
    - Se diseña una etapa genérica asociada a un solo bit.
    - Se replica la etapa para *n* bits
    - Se consideran los casos especiales en los bits extremos y las señales globales.
    - La complejidad del diseño no depende del número de bits.

# Registros

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

# Registros



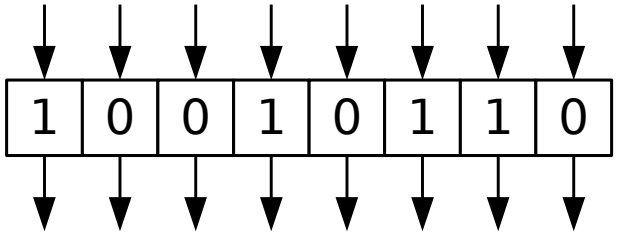

- Almacén de datos de *n* bits (*n* biestables)
  - Habitualmente nos referimos al contenido por el dato que representa y no por los bits individuales.
- Operaciones básicas:
  - Escritura (carga): modificación del dato almacenado.
  - Lectura: acceso al contenido del registro.

# Registros. Clasificación

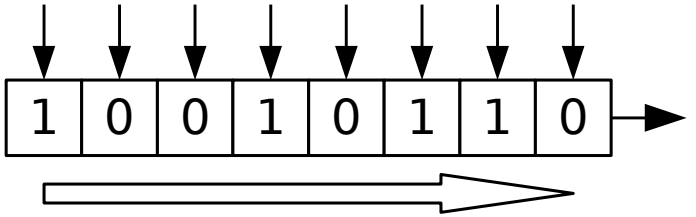
- Entrada en paralelo
  - Todos los bits pueden cargarse a la vez (en el mismo ciclo de reloj).
  - Una línea de entrada para cada bit.
- Entrada serie
  - Se carga un bit en cada ciclo de reloj.
  - Una única línea de entrada para todos los bits.
- Salida en paralelo
  - Todos los bits pueden ser leídos a la vez.
  - Una línea de salida para cada bit.
- Salida serie
  - Sólo puede leerse un bit en cada ciclo de reloj.
  - Una única línea de salida para cada bit.

# Registros. Clasificación

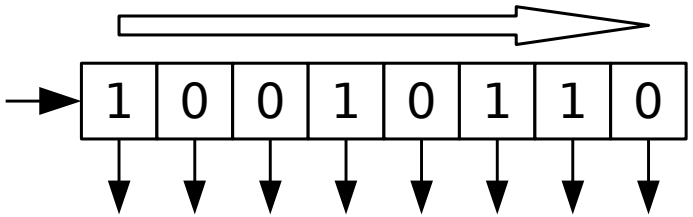
paralelo/paralelo



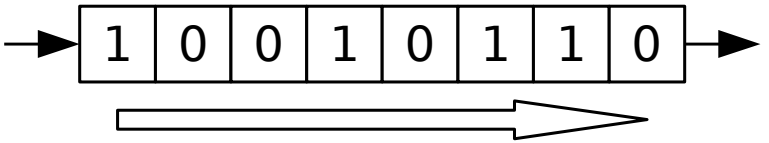

paralelo/serie




serie/paralelo




serie/serie



# Registro entrada paralelo/salida paralelo

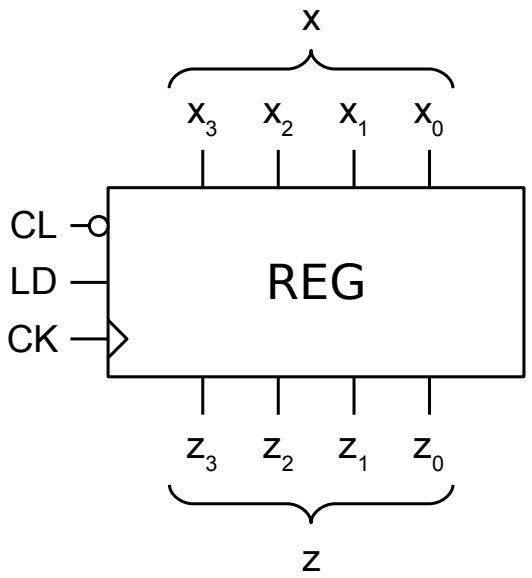



Tabla de operación

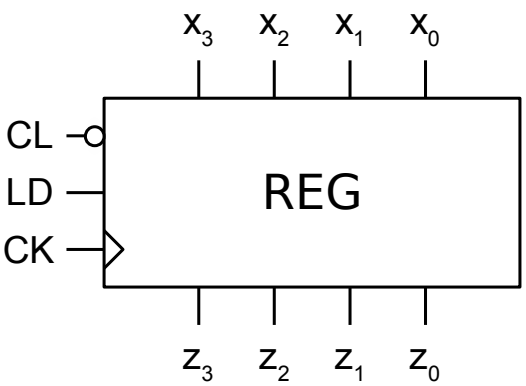
| CL LD | Operación | Tipo |
|---|---|---|
| 0 x | REG ← 0 | asíncrona |
| 1 1 | REG ← X | síncrona |
| 1 0 | REG ← REG | síncrona |

Salida: $z_i = q_i$

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



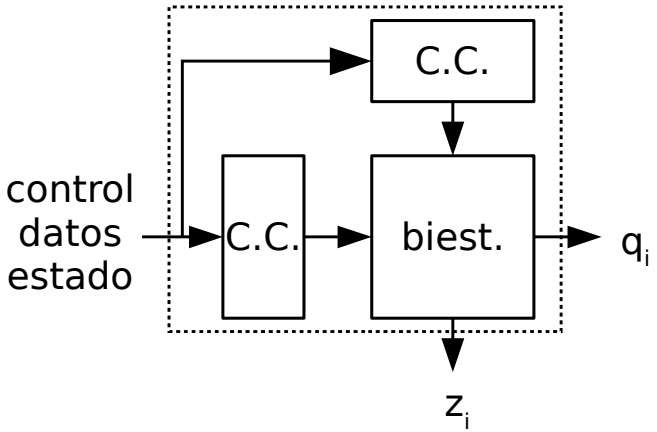

## Tabla de operación asíncrona

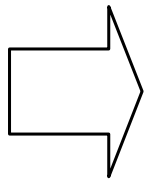
| CL | Operación | Et. típica | Entr. asínc. |
|---|---|---|---|
| 0 | REG ← 0 | $Q_i = 0$ | $CL_i = 0$ |
| 1 | REG ← REG | $Q_i = q_i$ | $CL_i = 1$ |

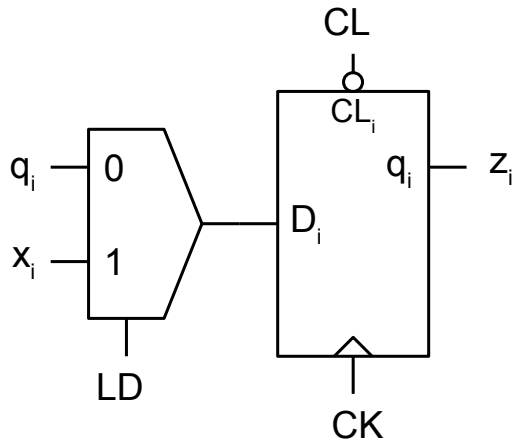
## Tabla de operación síncrona

| LD | Operación | Et. típica | Ecs. excit. |
|---|---|---|---|
| 1 | REG ← X | $Q_i = x_i$ | $D_i = x_i$ |
| 0 | REG ← REG | $Q_i = q_i$ | $D_i = q_i$ |

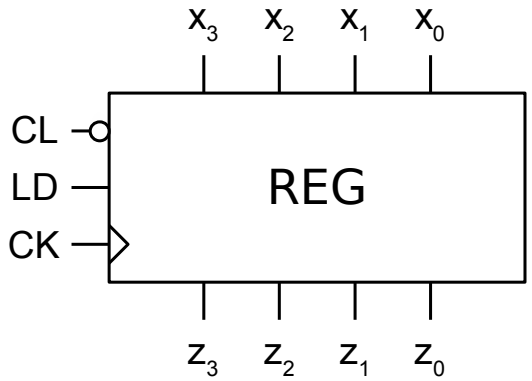
Salida: $z_i = q_i$

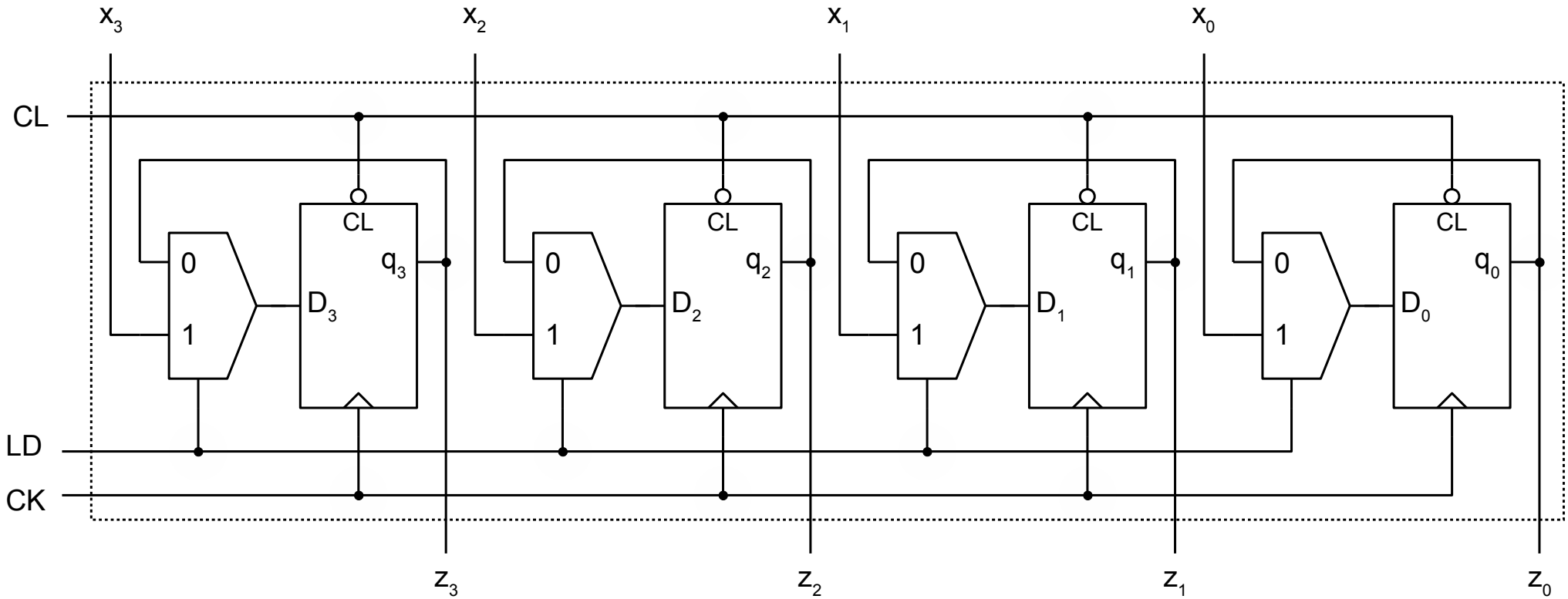




# Registro entrada paralelo/salida paralelo



# Registro de desplazamiento

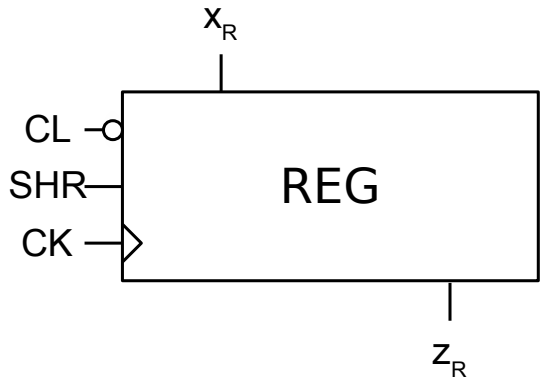



Tabla de operación

| CL SHR | Operación | Tipo |
|---|---|---|
| 0 x | REG ← 0 | asíncrona |
| 1 1 | REG ← SHR(REG,$X_R$) | síncrona |
| 1 0 | REG ← REG | síncrona |

Salida: $z_R = q_0$

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

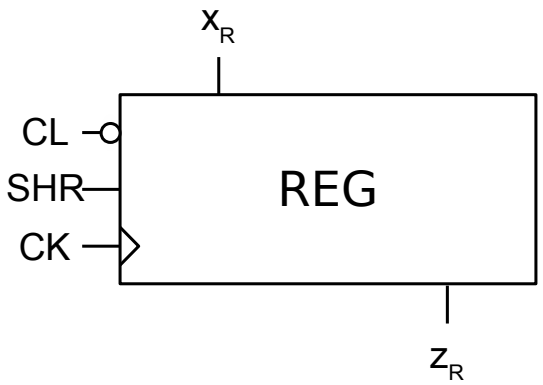
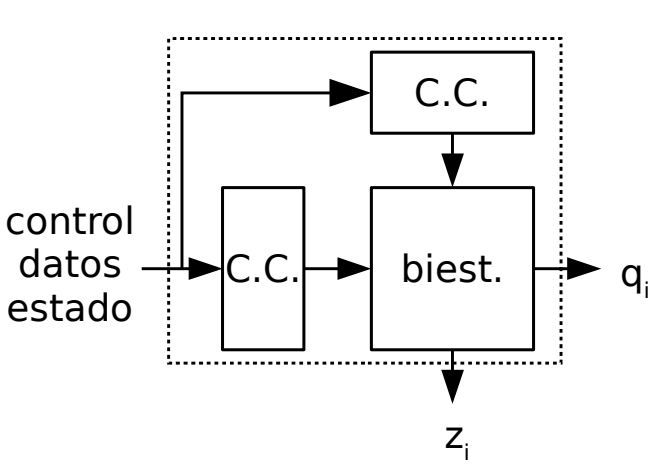
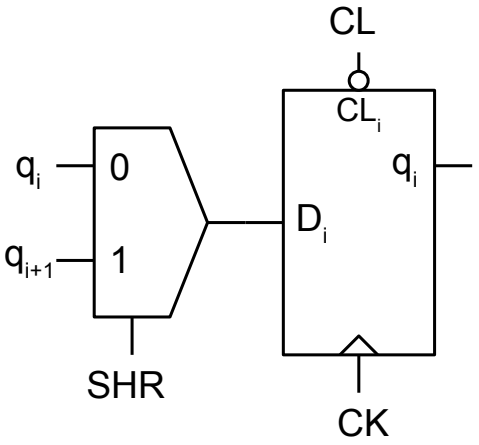



Tabla de operación síncrona

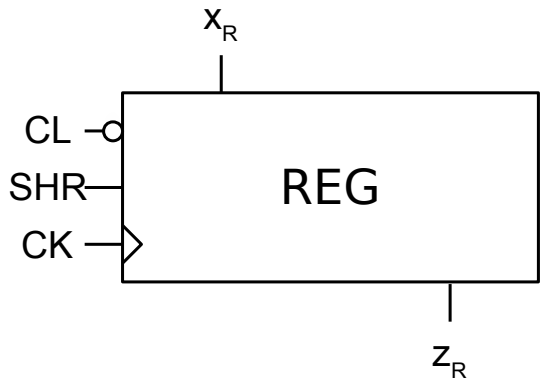
| SHR | Operación | Et. típica | Et. 3 | Ecs. exc. | EE et. 3 |
|---|---|---|---|---|---|
| 1 | REG ← SHR(REG,$X_R$) | $Q_i = q_{i+1}$ | $Q_3 = x_R$ | $D_i = q_{i+1}$ | $D_3 = x_R$ |
| 0 | REG ← REG | $Q_i = q_i$ | $Q_3 = q_3$ | $D_i = q_i$ | $D_3 = q_3$ |

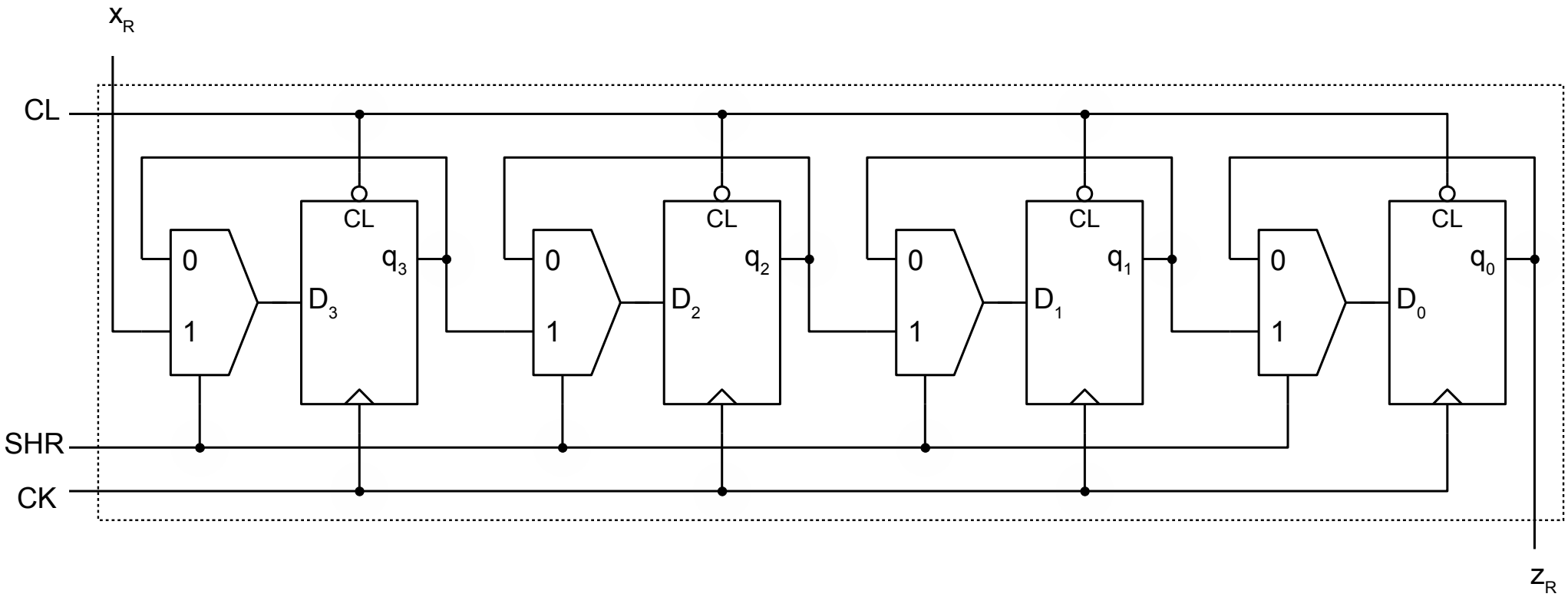
Salida: $z_R = q_0$



# Registro de desplazamiento



# Registro universal

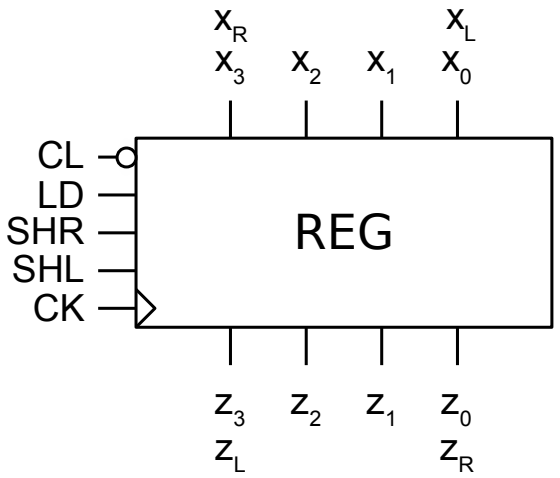



Tabla de operación

| CL LD SHR SHL | Operación | Tipo |
|---|---|---|
| 0 x x x | REG ← 0 | asínc. |
| 1 1 x x | REG ← X | sínc. |
| 1 0 1 x | REG ← SHR(REG,$X_R$) | sínc. |
| 1 0 0 1 | REG ← SHL(REG,$X_L$) | sínc. |
| 1 0 0 0 | REG ← 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
```

# 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
  - El número de estados de cuenta del contador se llama módulo: después del último estado de cuenta se pasa al primero (cuenta cíclica)
- Diseño
  - La implementación es más sencilla con biestables JK o T (simplifican la operación de cuenta)
- Operaciones típicas
  - Cuenta ascendente
  - Cuenta descendente
  - 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 de Ck en base 2

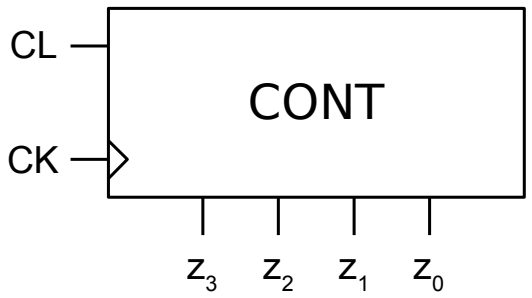



Tabla de operación

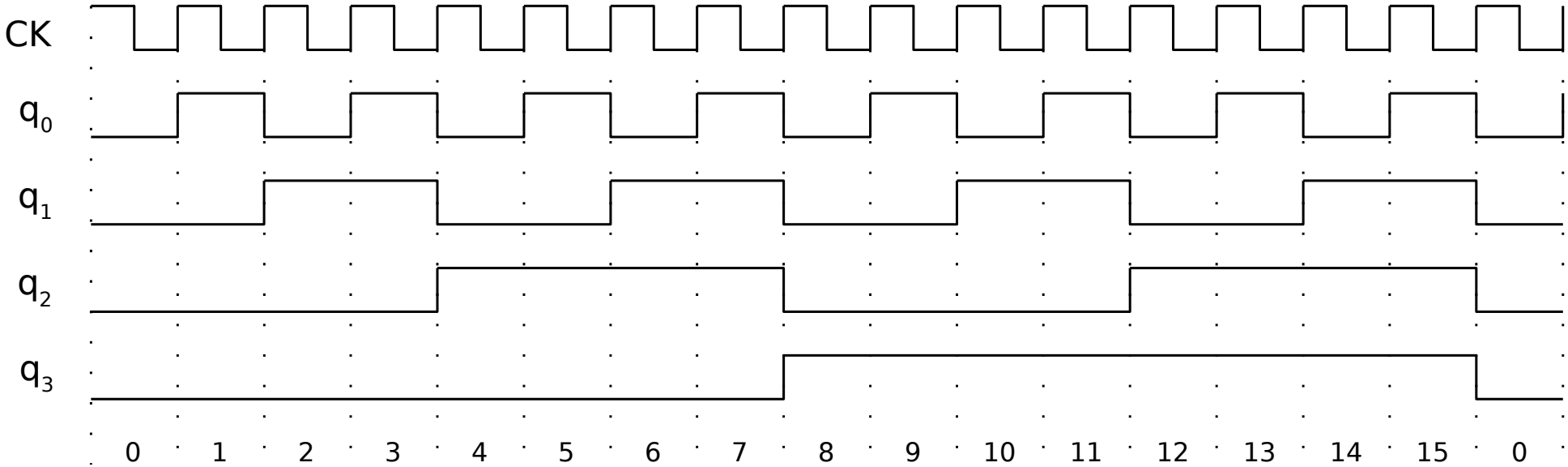
| CL | Operación | Tipo |
|---|---|---|
| 1 | CONT ← 0 | asínc. |
| 0 | CONT ← CONT+1$\|_{\text{mod 16}}$ | sínc. |

Salida: $z_i=q_i$

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 $2^n$
# Operación de cuenta




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

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

# Contador binario módulo $2^n$

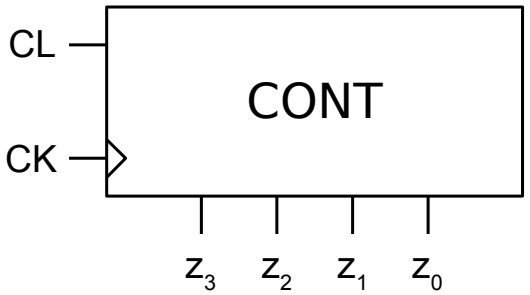
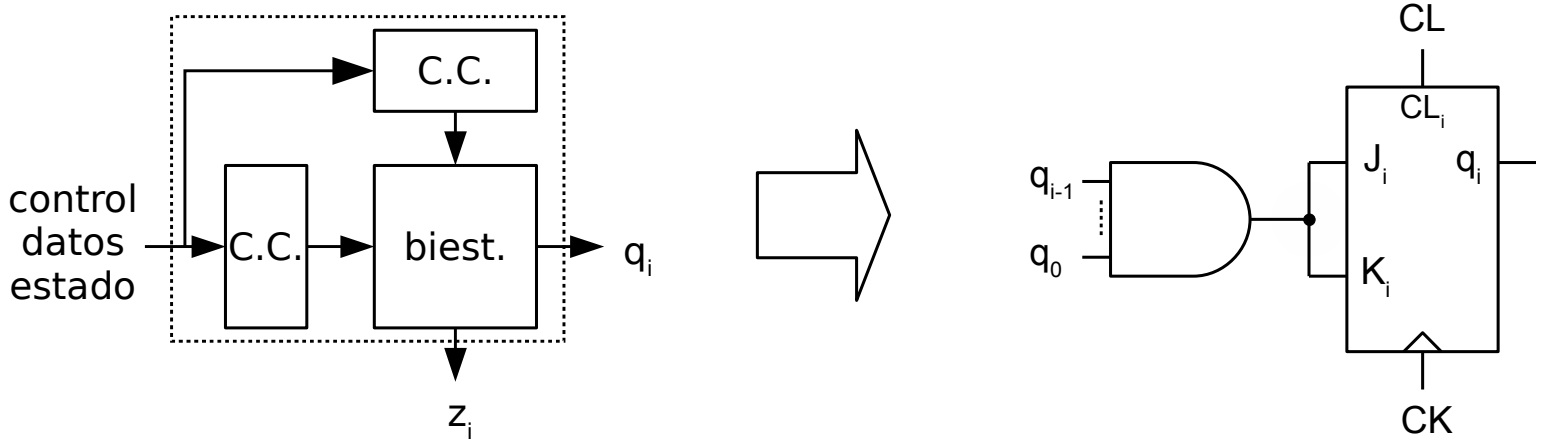
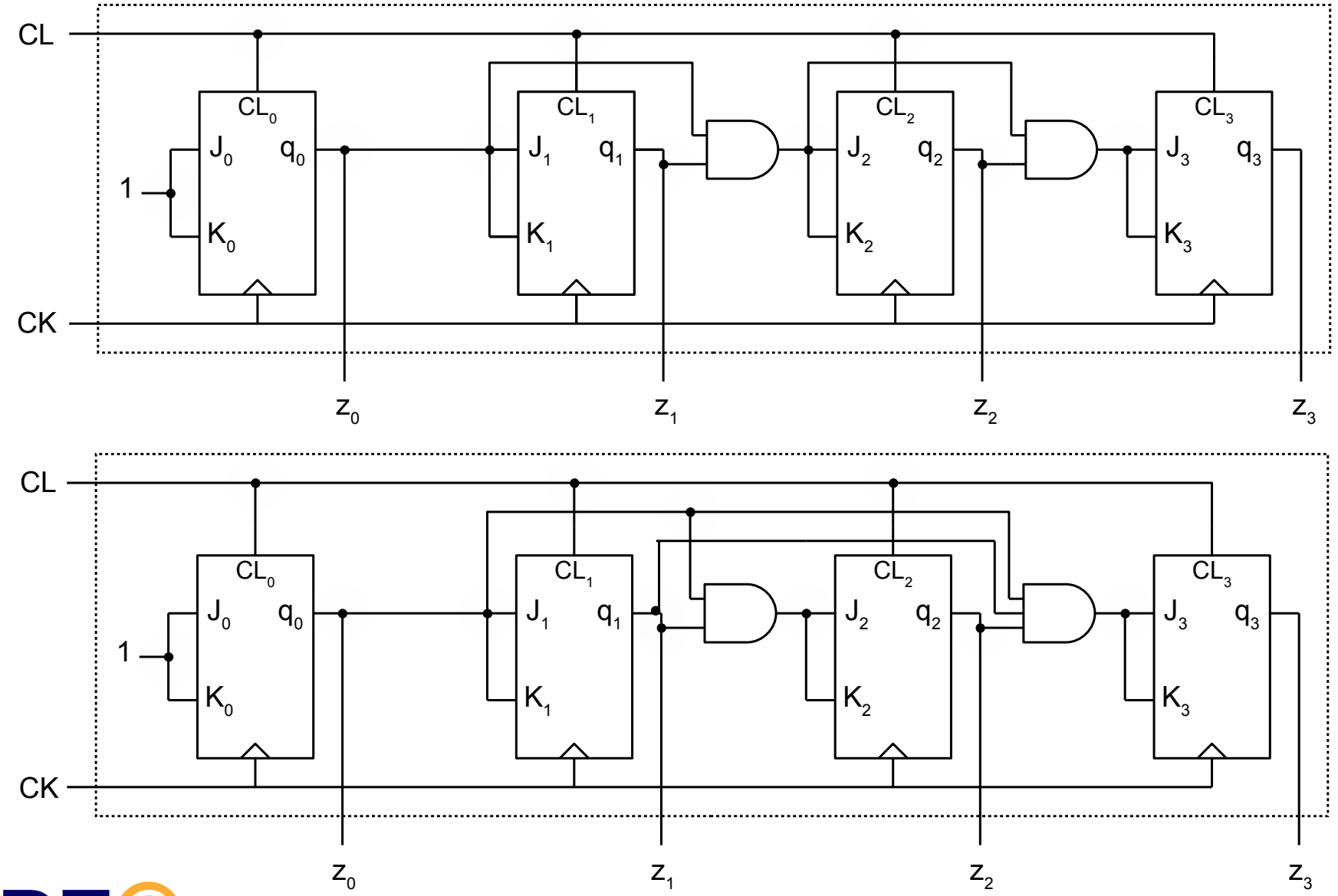



Tabla de operación síncrona

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



# Contador binario módulo $2^n$. Circuito



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

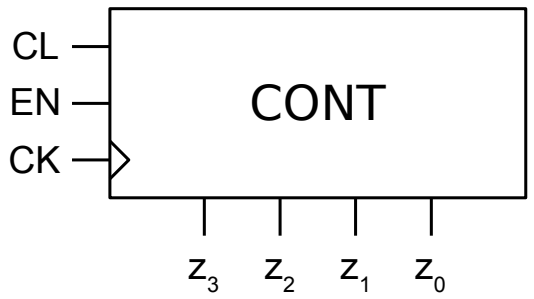



Tabla de operación

| CL EN | Operación | Tipo |
|---|---|---|
| 1 x | CONT ← 0 | sínc. |
| 0 1 | CONT ← CONT+1$\|_{\text{mod 16}}$ | sínc. |
| 0 0 | CONT ← CONT | sínc. |

Salida: $z_i = q_i$

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

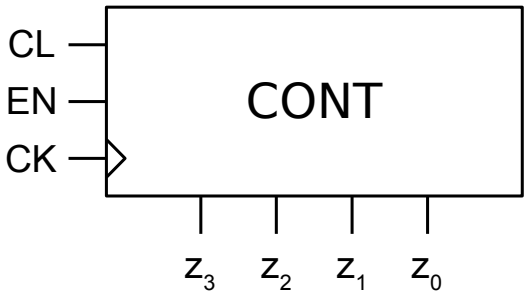
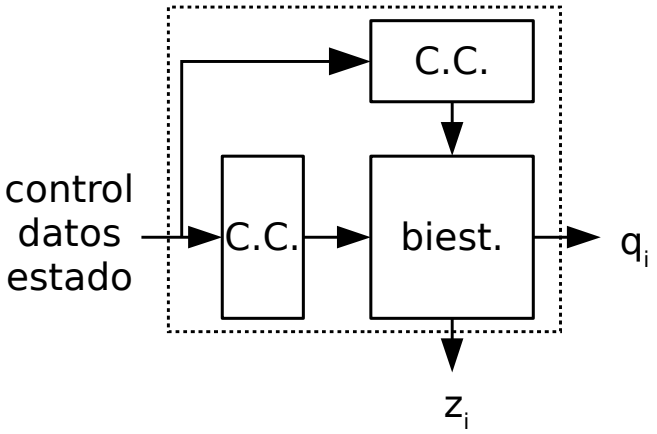
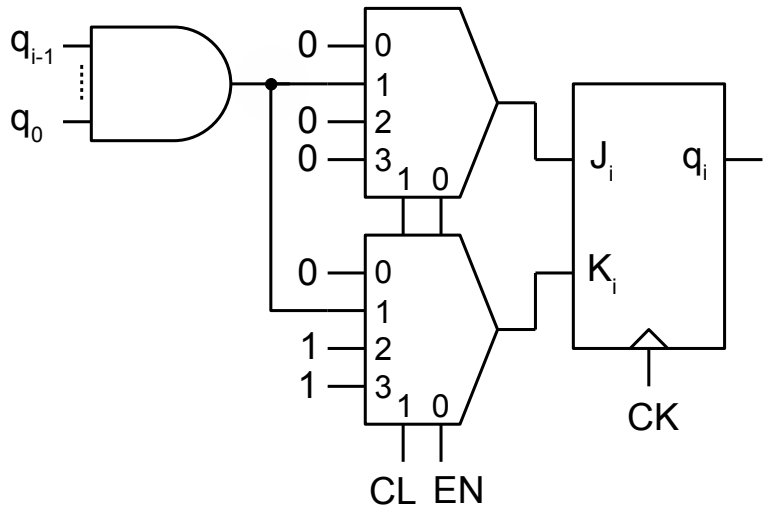



Tabla de operación síncrona

| CL EN | Operación | Et. típica | Et. 0 |
|---|---|---|---|
| 1 x | CONT ← 0 | $J_i$=0, $K_i$=1 | $J_0$=0, $K_0$=1 |
| 0 1 | CONT ← CONT+1$\|_{mod\ 16}$ | $J_i$=$K_i$=$q_{i-1}$ ... $q_0$ | $J_0$=$K_0$=1 |
| 0 0 | CONT ← CONT | $J_i$=$K_i$=0 | $J_0$=$K_0$=0 |

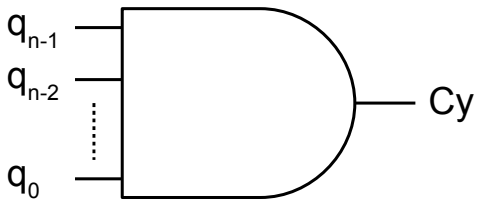


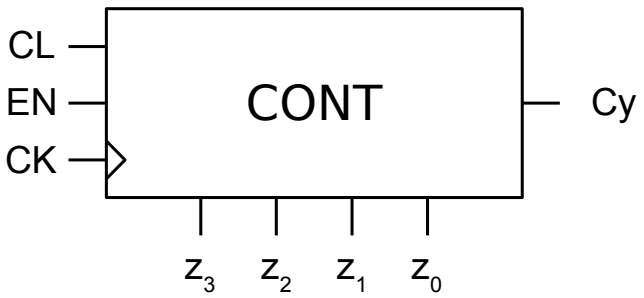
# Salida de fin de cuenta

- Cuenta ascendente (acarreo – carry)
  - Cy = 1 sii q = $2^n$ – 1

$$Cy = q_{n-1}\, q_{n-2} \cdots q_0$$



# Salida de fin de cuenta




Código Verilog

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

    reg [3:0] q;

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

    assign z = q;
    assign cy = &q;

endmodule
```

# Unión de contadores

- Se combinan 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*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

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

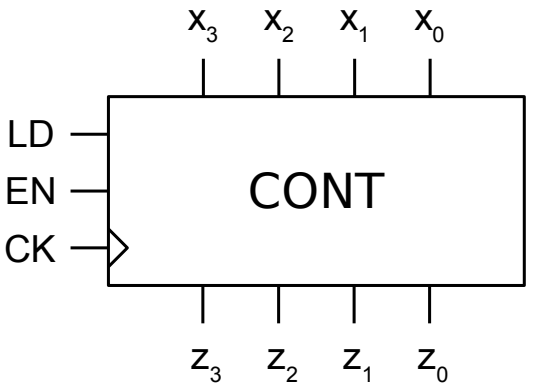



Tabla de operación

| LD EN | Operación | Tipo |
|---|---|---|
| 1 x | CONT ← X | sínc. |
| 0 1 | CONT ← $\text{CONT+1}\|_{\text{mod 16}}$ | sínc. |
| 0 0 | CONT ← CONT | sínc. |

Salida: $z_i=q_i$

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 puesta a cero, habilitación y carga

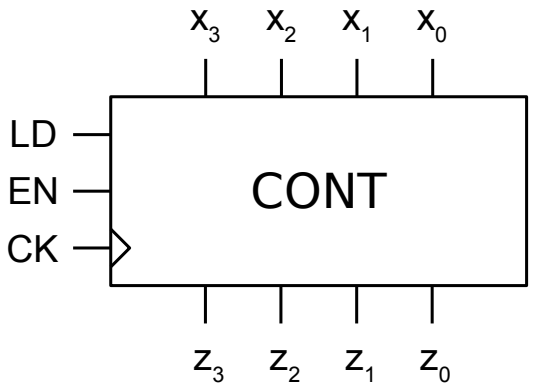
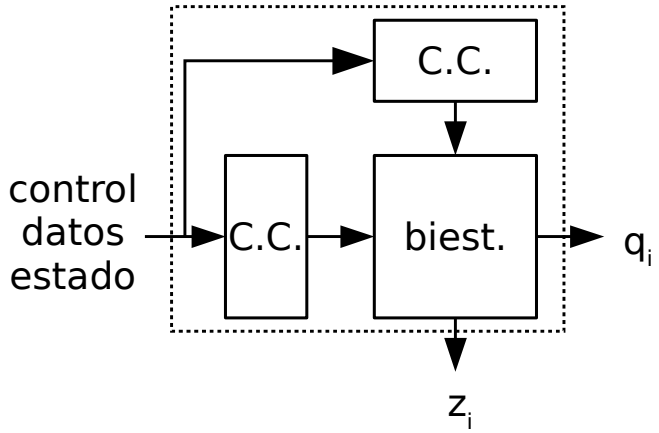
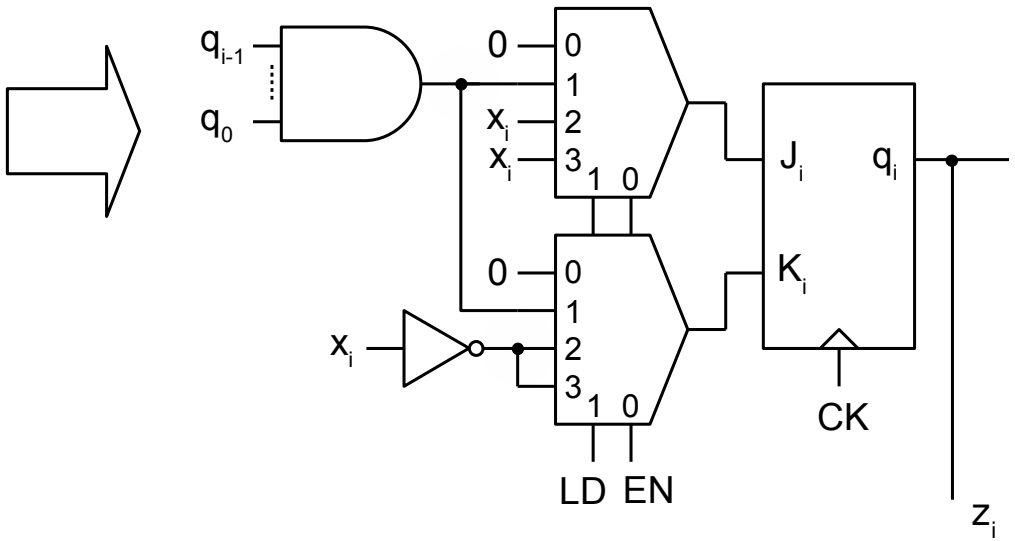



Tabla de operación síncrona

| LD EN | Operación | Et. típica | Et. 0 |
|---|---|---|---|
| 1 x | CONT ← X | $J_i=x_i$, $K_i=\overline{x}_i$ | $J_0=x_0$, $K_0=\overline{x}_0$ |
| 0 1 | CONT ← CONT+1$\|_{mod\ 16}$ | $J_i=K_i=q_{i-1}\ldots q_0$ | $J_0=K_0=1$ |
| 0 0 | CONT ← CONT | $J_i=K_i=0$ | $J_0=K_0=0$ |



# Contador de módulo $<2^n$

- Los contadores módulo $< 2^n$ se obtienen normalmente limitando los estados de cuenta de un contador módulo $2^n$.
- Casos
  - Límite superior: 0 ... k, $k<2^n$
  - Límite inferior: k ... $2^n$-1, $k > 0$
  - Límites inferior y superior: k ... l, $k>0$, $l<2^n$
- 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.

# Ejemplo: contador BCD

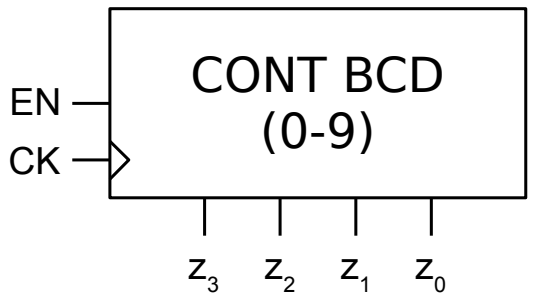



Tabla de operación

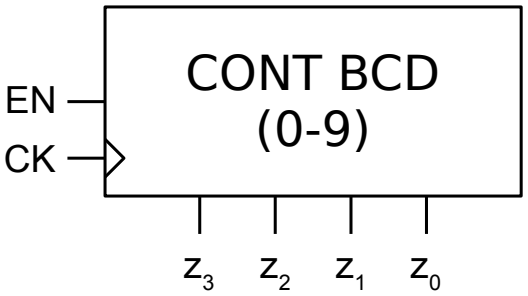
| EN | Operación | Tipo |
|---|---|---|
| 1 | $CONT \leftarrow CONT+1\|_{mod\ 10}$ | sínc. |
| 0 | $CONT \leftarrow CONT$ | sínc. |

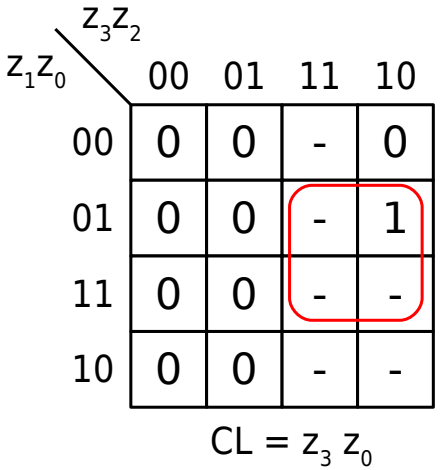
Salida: $z_i=q_i$

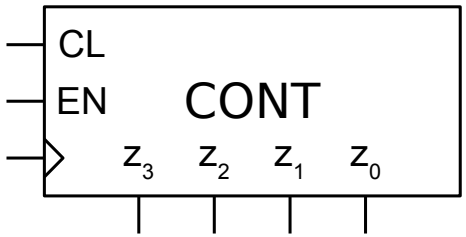
Código Verilog

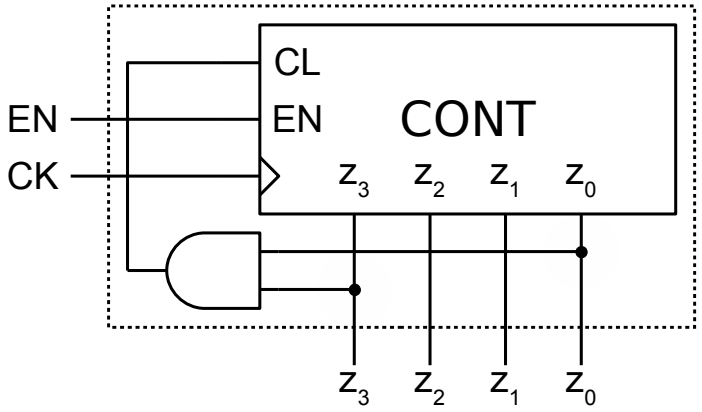
```
module count_mod10(
    input ck,
    input cl,
    input en,
    output [3:0] z,
    );

    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

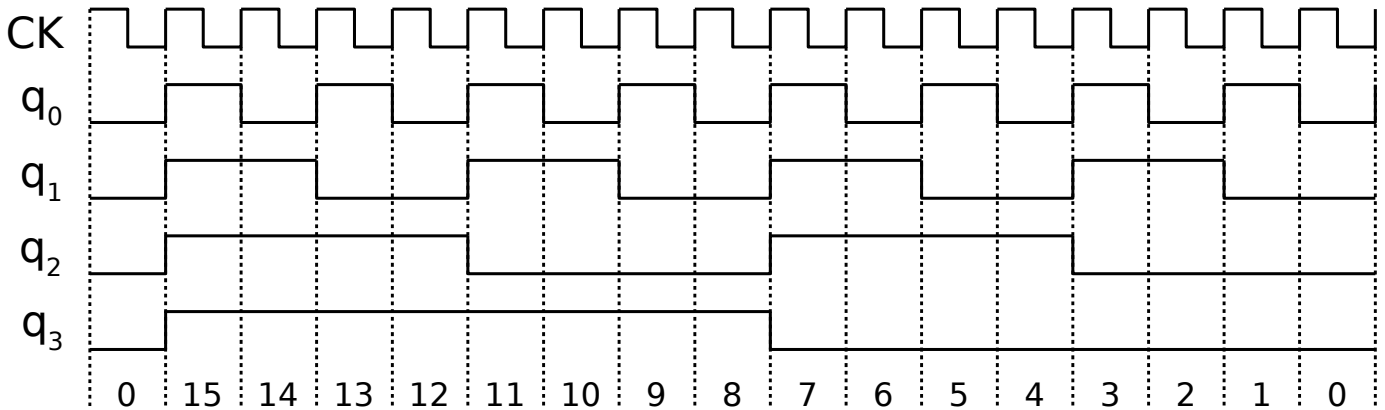


# Contador descendente módulo $2^n$




- Cuenta descendente
  - $Q_0 = \overline{q_0}$
  - $Q_i = \overline{q_i}$ si $q_j=0$, $\forall j<i$
  - Si no, $Q_i = q_i$

$$J_i = K_i = \overline{q_{i-1} + q_{i-2} + \ldots + q_0}$$

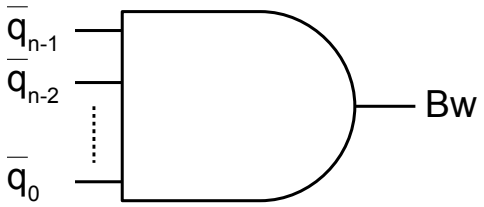
$$J_i = K_i = \overline{q_{i-1}}\,\overline{q_{i-2}} \ldots \overline{q_0}$$

# Salida de fin de cuenta

- Cuenta descendente (borrow)
  - Bw = 1 sii q = 0

$$Bw = \overline{q_{n-1} + q_{n-2} + \ldots + q_0}$$

$$Bw = \overline{q_{n-1}}\,\overline{q_{n-2}} \ldots \overline{q_0}$$



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

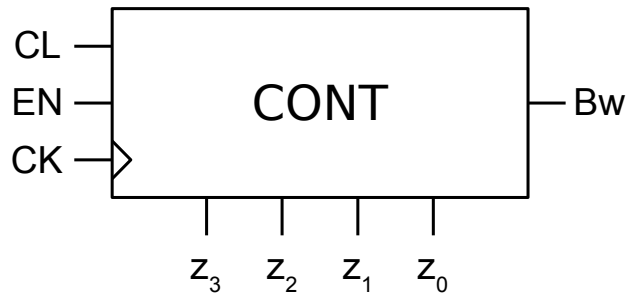



Tabla de operación

| CL EN | Operación | Tipo |
|---|---|---|
| 1 x | CONT ← 0 | sínc. |
| 0 1 | CONT ← CONT-1$\|_{\text{mod } 16}$ | sínc. |
| 0 0 | CONT ← CONT | sínc. |

Bw = 1 sii [CONT] = 0000
Zi = qi

Código Verilog

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

    reg [3:0] q;

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

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

endmodule
```

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

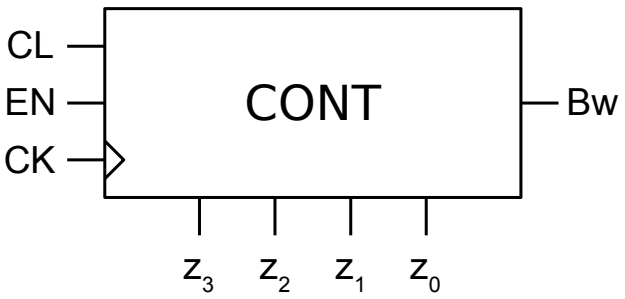
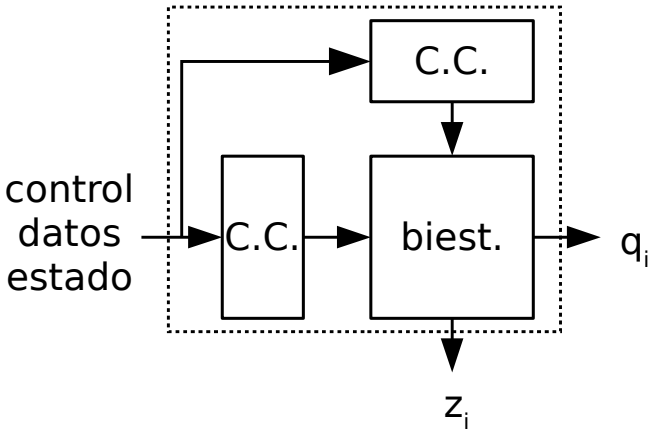
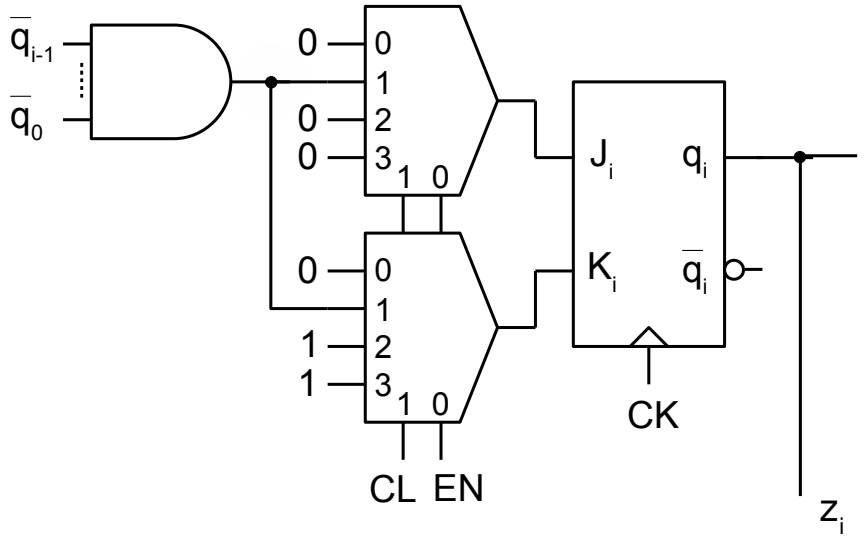



Tabla de operación síncrona

| CL EN | Operación | Et. típica | Et. 0 |
|---|---|---|---|
| 1 x | CONT ← 0 | $J_i=0, K_i=1$ | $J_0=0, K_0=1$ |
| 0 1 | CONT ← CONT-1$\|_{mod\ 16}$ | $J_i=K_i=\overline{q}_{i-1} \dots \overline{q}_0$ | $J_0=K_0=1$ |
| 0 0 | CONT ← CONT | $J_i=K_i=0$ | $J_0=K_0=0$ |

Bw = 1 sii [CONT] = 0000



# Contador reversible

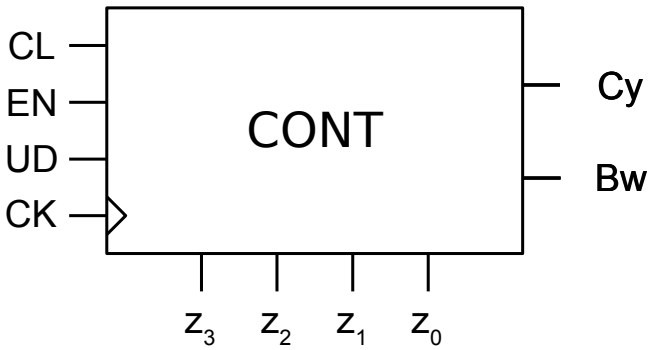



Tabla de operación

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

Cy = 1 sii [CONT] = 1111
Bw = 1 sii [CONT] = 0000

# Contador reversible

Código Verilog

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

    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 Cy = &q;
    assign Bw = ~(|q);

endmodule
```

Código Verilog

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

    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 Cy = &q;
    assign Bw = ~(|q);

endmodule
```

# Contador reversible

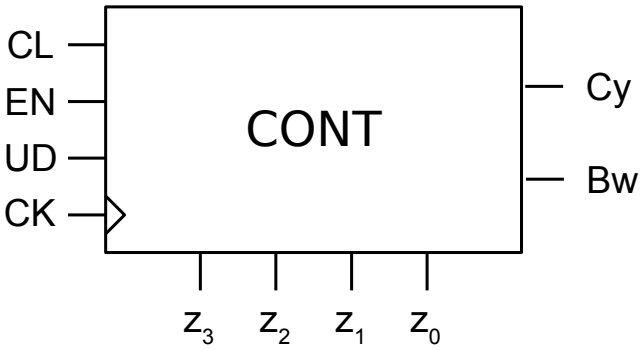
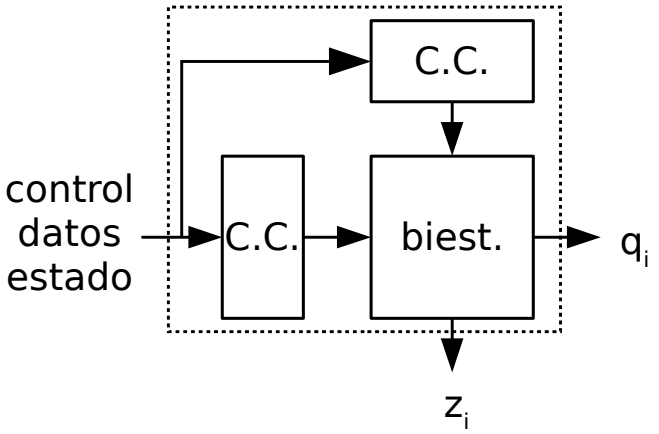
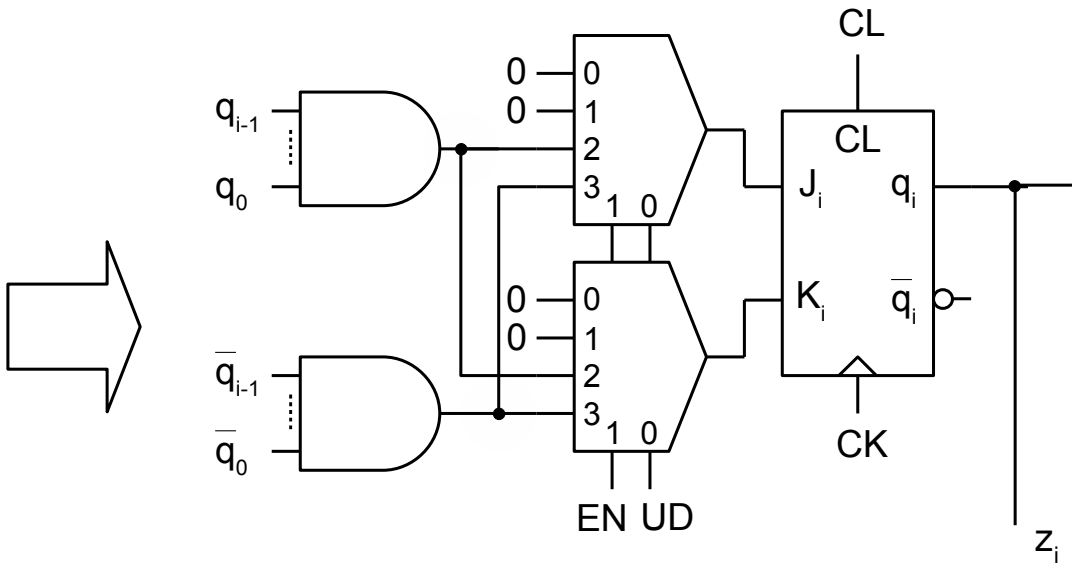



Tabla de operación síncrona

| EN UD | Operación | Et. típica | Et. 0 |
|---|---|---|---|
| 0 x | CONT ← CONT | $J_i=K_i=0$ | $J_0=K_0=0$ |
| 1 0 | CONT ← CONT+1$\|_{mod\ 16}$ | $J_i=K_i=q_{i-1} \dots q_0$ | $J_0=K_0=1$ |
| 1 1 | CONT ← CONT-1$\|_{mod\ 16}$ | $J_i=K_i=\overline{q}_{i-1} \dots \overline{q}_0$ | $J_0=K_0=1$ |

Cy = 1 sii [CONT] = 1111
Bw = 1 sii [CONT] = 0000



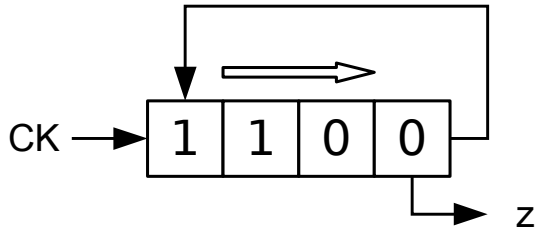



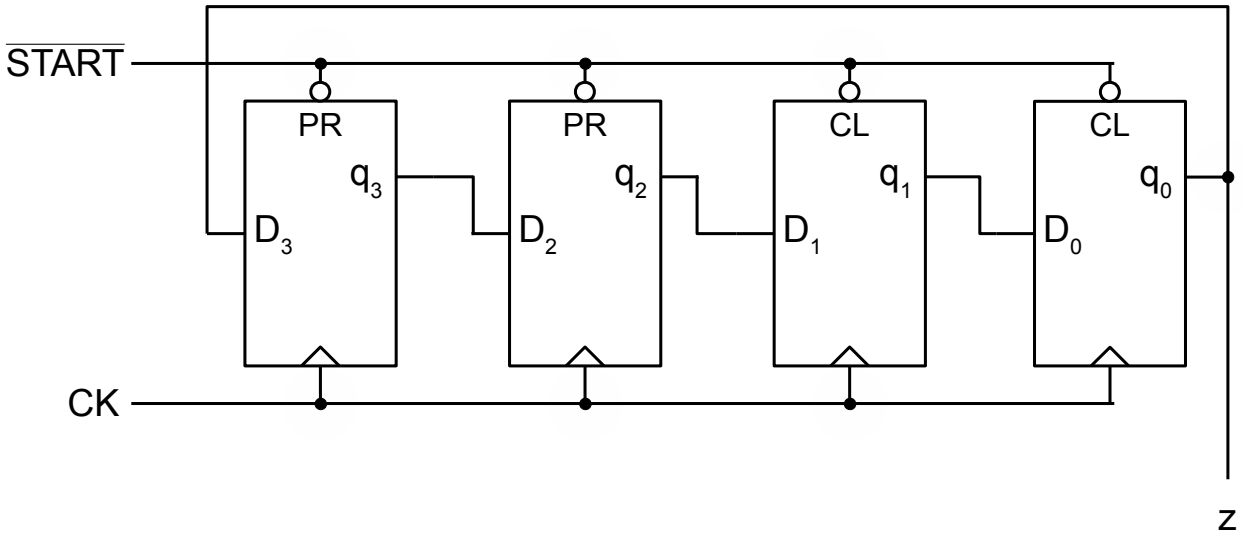
# Diseño con subsistemas secuenciales

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

# Generador de secuencia

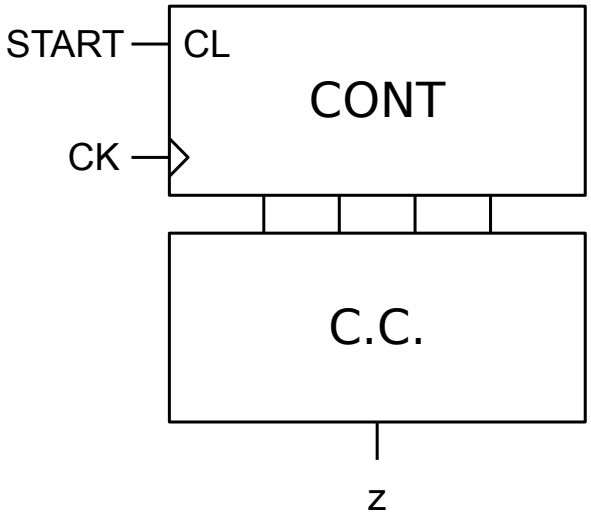
- Con registro de desplazamiento

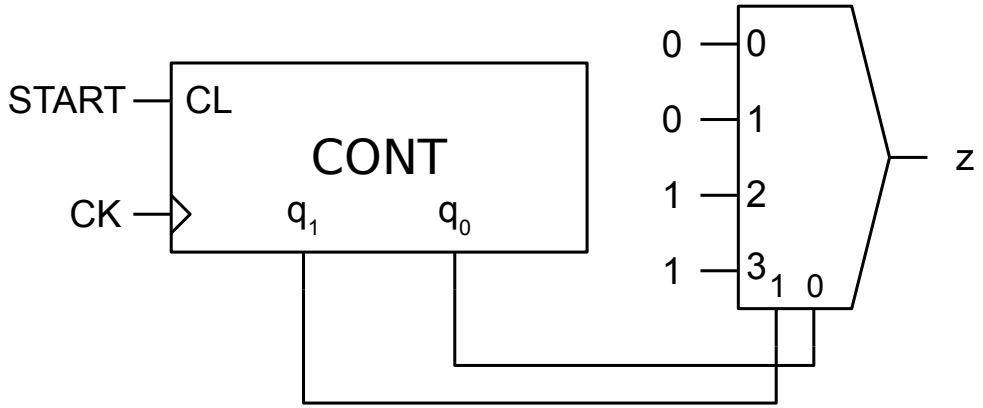


# Generador de secuencia

- Con contador y CC



# Detector de secuencia