Tema 8. Subsistemas secuenciales

Circuitos Electrónicos Digitales
E.T.S.I. Informática
Universidad de Sevilla
21/09/2010

Jorge Juan <jjchico@dte.us.es>

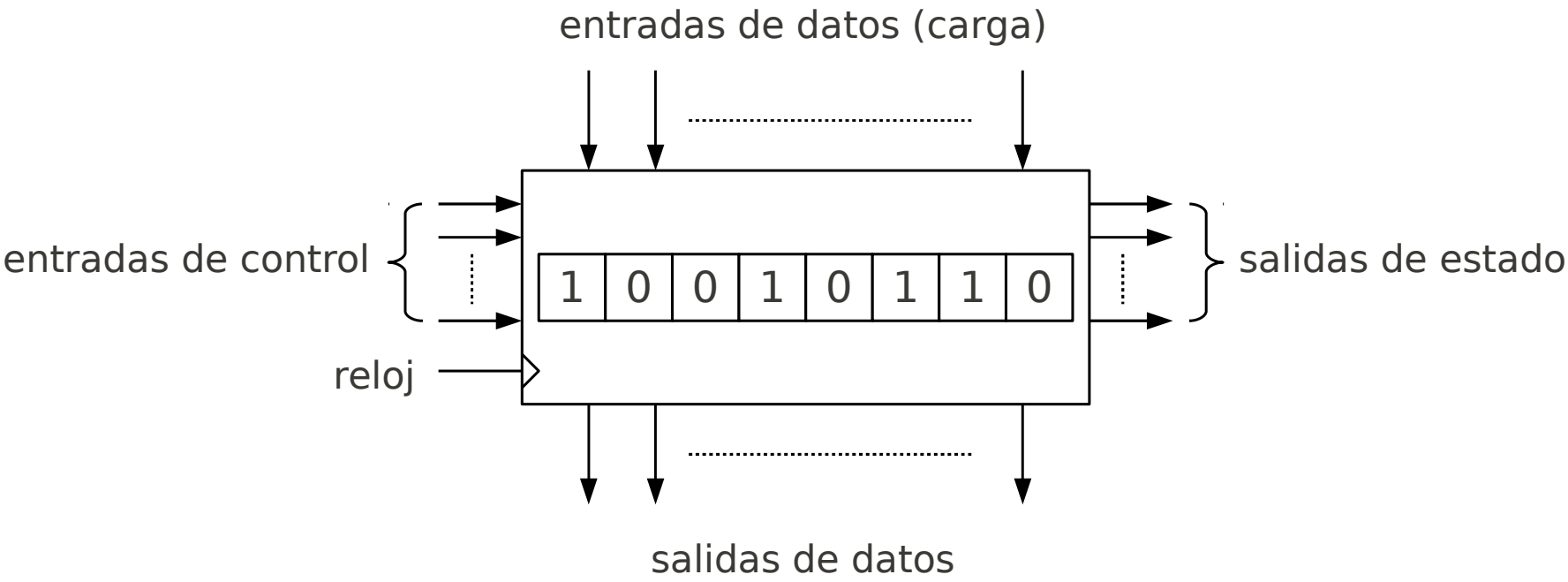
Contenidos

- 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 par 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)

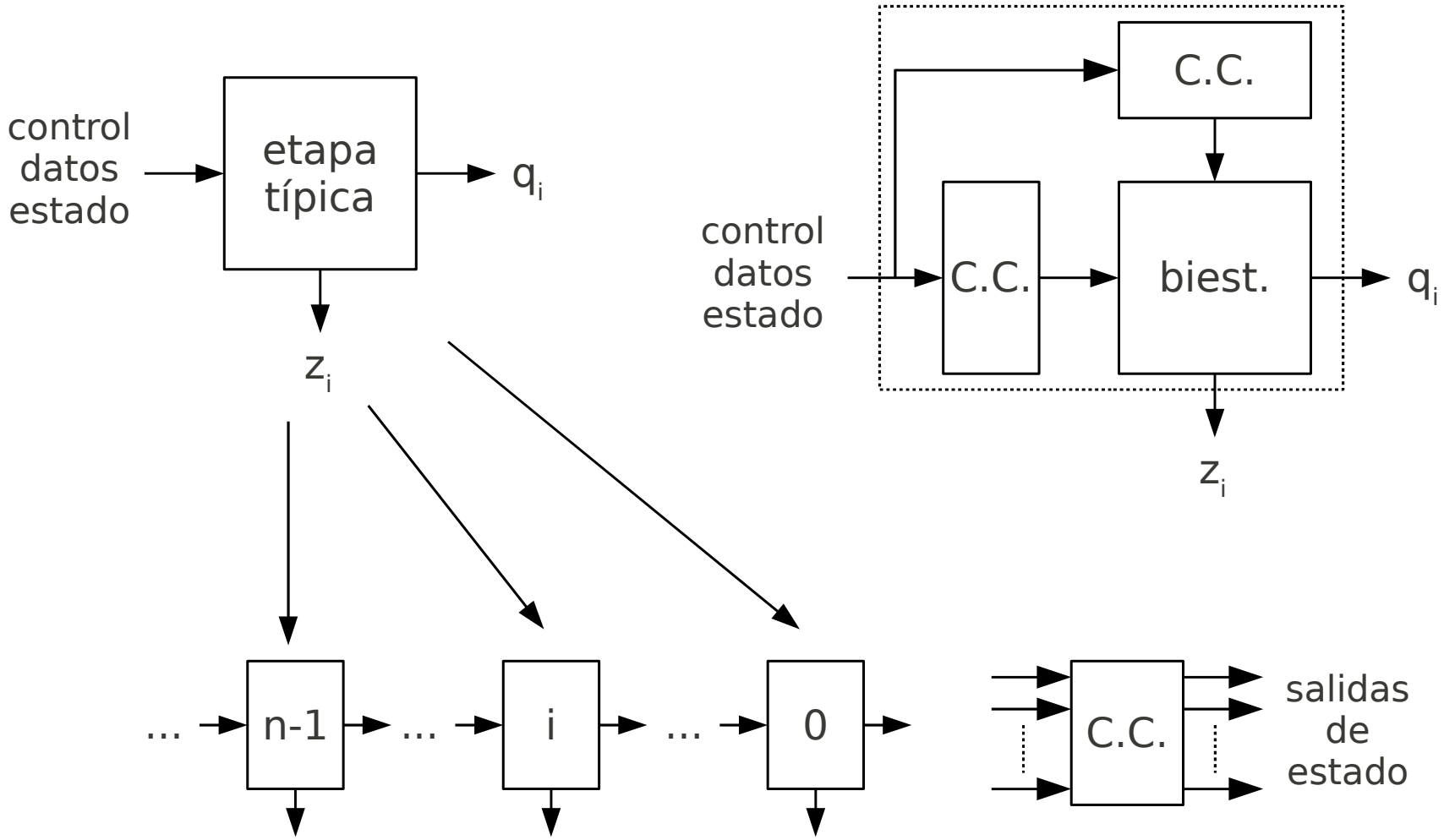
Introducción



Introducción

- Señales de control
  - Las señales de control determinan la operación a realizar.
  - Pueden ser síncronas: alteran el dato en el flanco activo de la señal de reloj
  - Pueden ser asíncronas: alteran el dato de forma inmediata tras la activación de la señal de control
  - 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



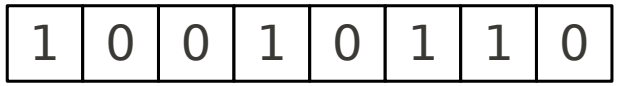
Introducción

- Estructura modular
  - Todos los bits 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 sólo 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



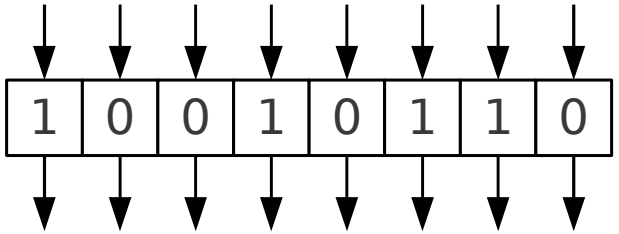
- Almacén de datos de n bits (n biestables)
  - Habitualmente nos referimos al contenido por el dato que representa y no por los bits individuales.
- Operaciones básicas:
  - Escritura (carga): modificación del dato almacenado.
  - Lectura: acceso al contenido del registro.

Registros. Clasificación

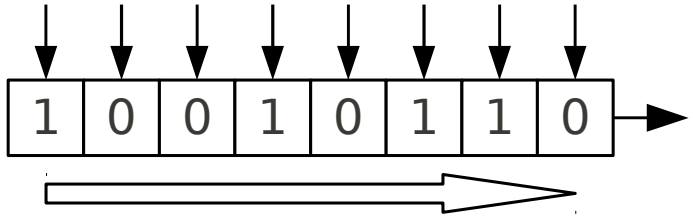
- Entrada en paralelo
  - Todos los bits pueden cargarse a la vez (en el mismo ciclo de reloj).
  - Una línea de entrada para cada bit.
- Entrada serie
  - Se carga un bit en cada ciclo de reloj.
  - Una única línea de entrada para todos los bits.
- Salida en paralelo
  - Todos los bits pueden ser leídos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

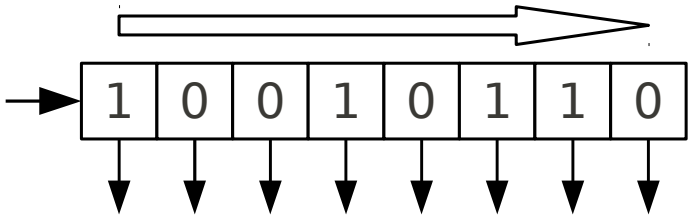
paralelo/paralelo



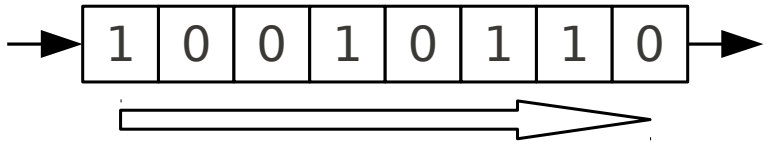
paralelo/serie



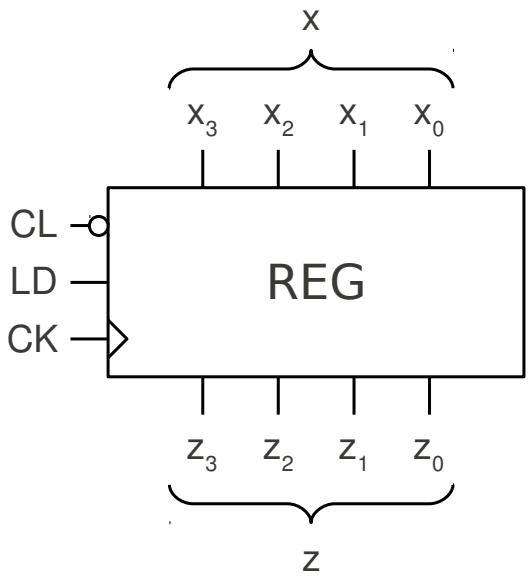
serie/paralelo



serie/serie



Registro entrada paralelo/salida paralelo



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
```

Tabla de operación

CL, LD	Operación	Tipo
0x	q ← 0	asínc.
11	q ← x	sínc.
10	q ← q	sínc.

Registro entrada paralelo/salida paralelo

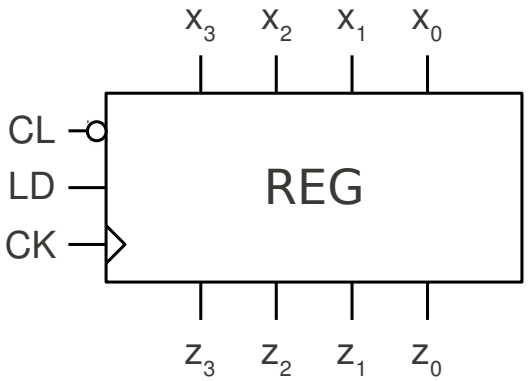
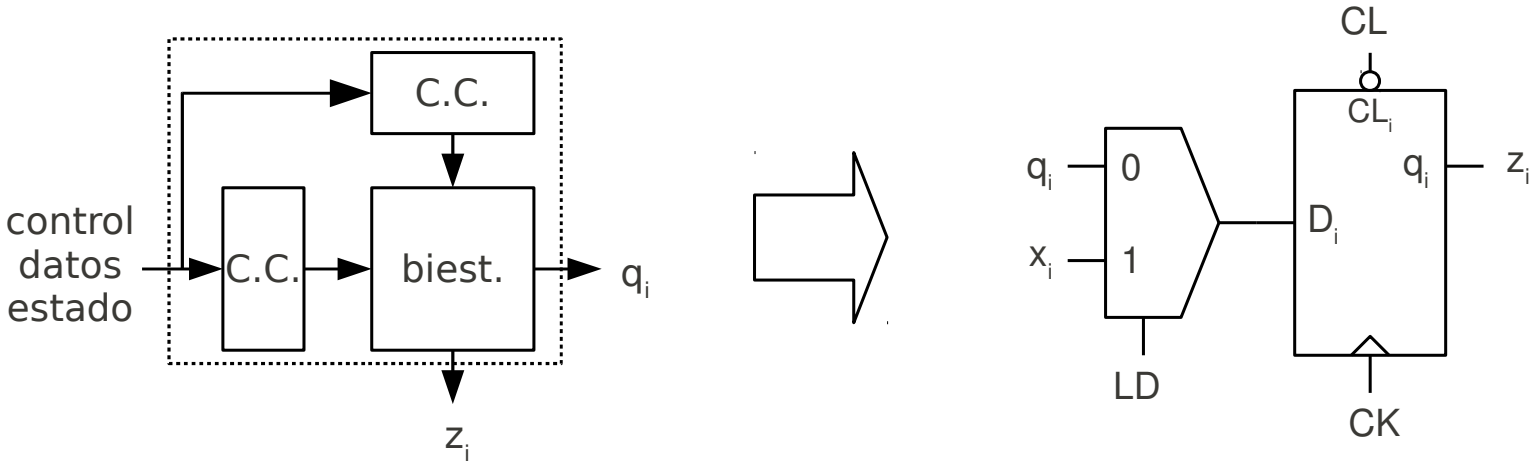


Tabla de operación asíncrona

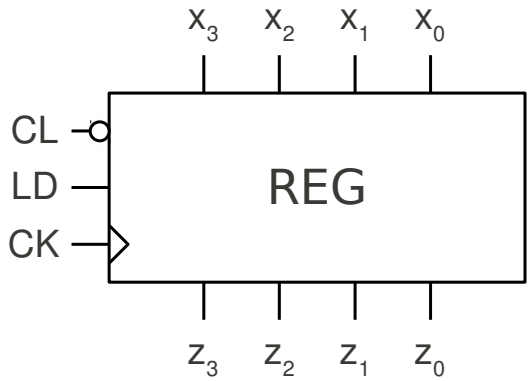
CL	Operación	Et. típica	Excit.
0	$q \leftarrow 0$	$Q_i = 0$	$CL_i = 0$
1	$q \leftarrow q$	$Q_i = q_i$	$CL_i = 1$

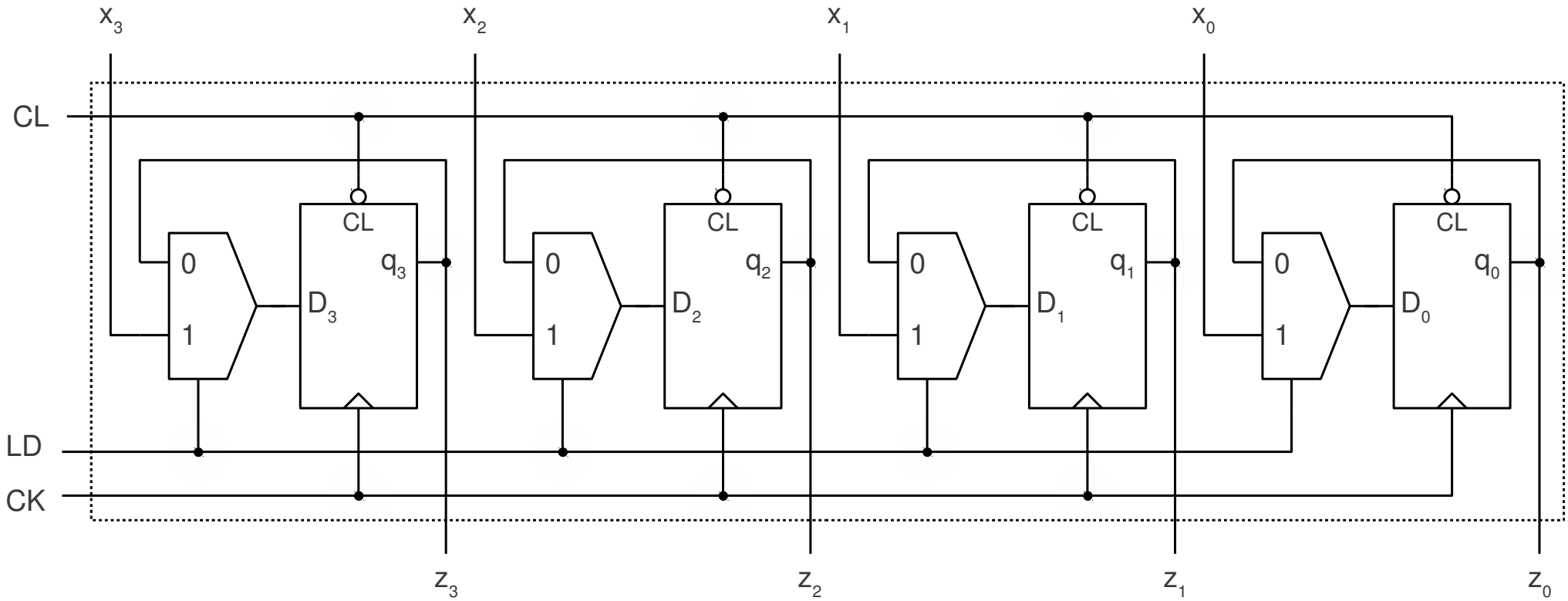
Tabla de operación síncrona

LD	Operación	Et. típica	Excit.
1	$q \leftarrow x$	$Q_i = x_i$	$D_i = x_i$
0	$q \leftarrow q$	$Q_i = q_i$	$D_i = q_i$



Registro entrada paralelo/salida paralelo





Registro de desplazamiento

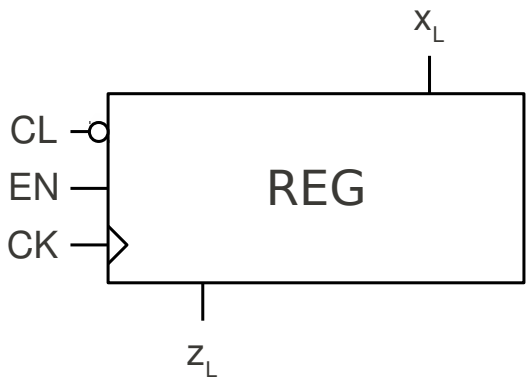


Tabla de operación

CL, EN	Operación	Tipo
0x	q ← 0	asíncrona
11	q ← SHL(q)	síncrona
10	q ← q	síncrona

Código Verilog

```
module reg_shl(
    input ck,
    input cl,
    input en,
    input xl,
    output zl
    );

    reg [3:0] q;

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

    assign zl = q[3];

endmodule
```

Registro de desplazamiento

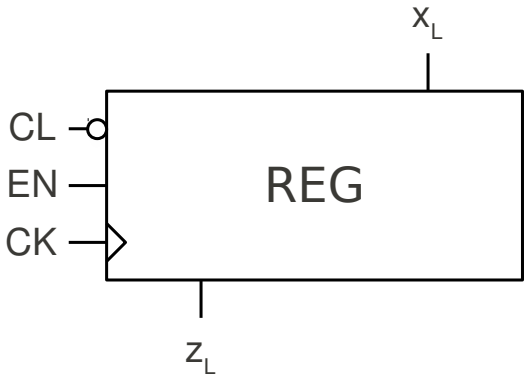
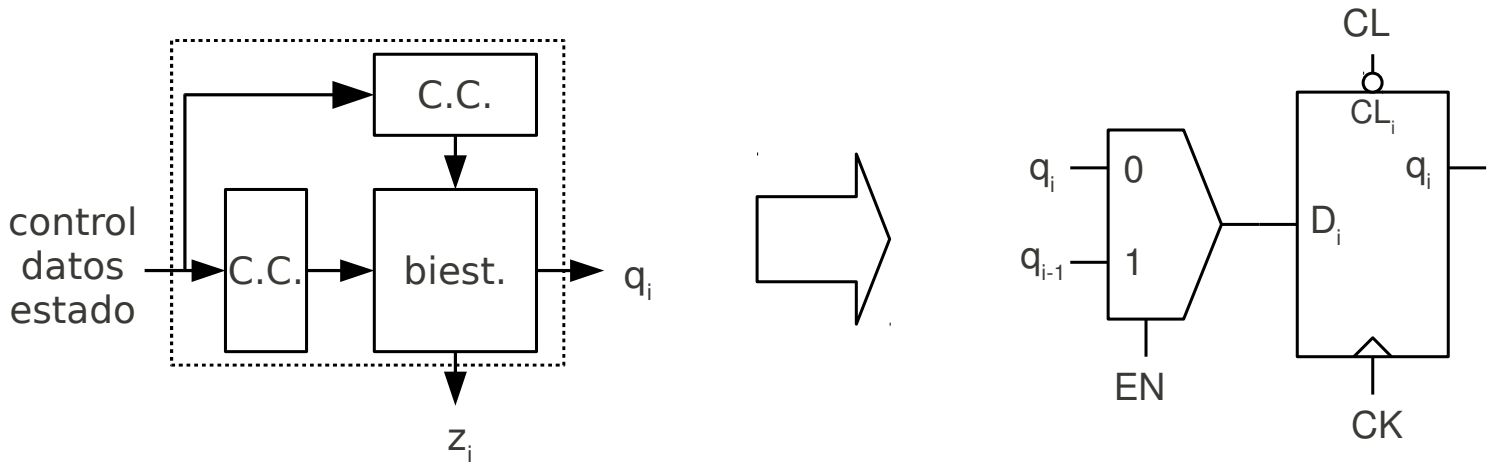
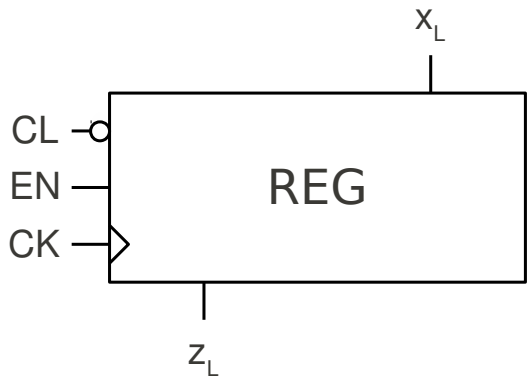


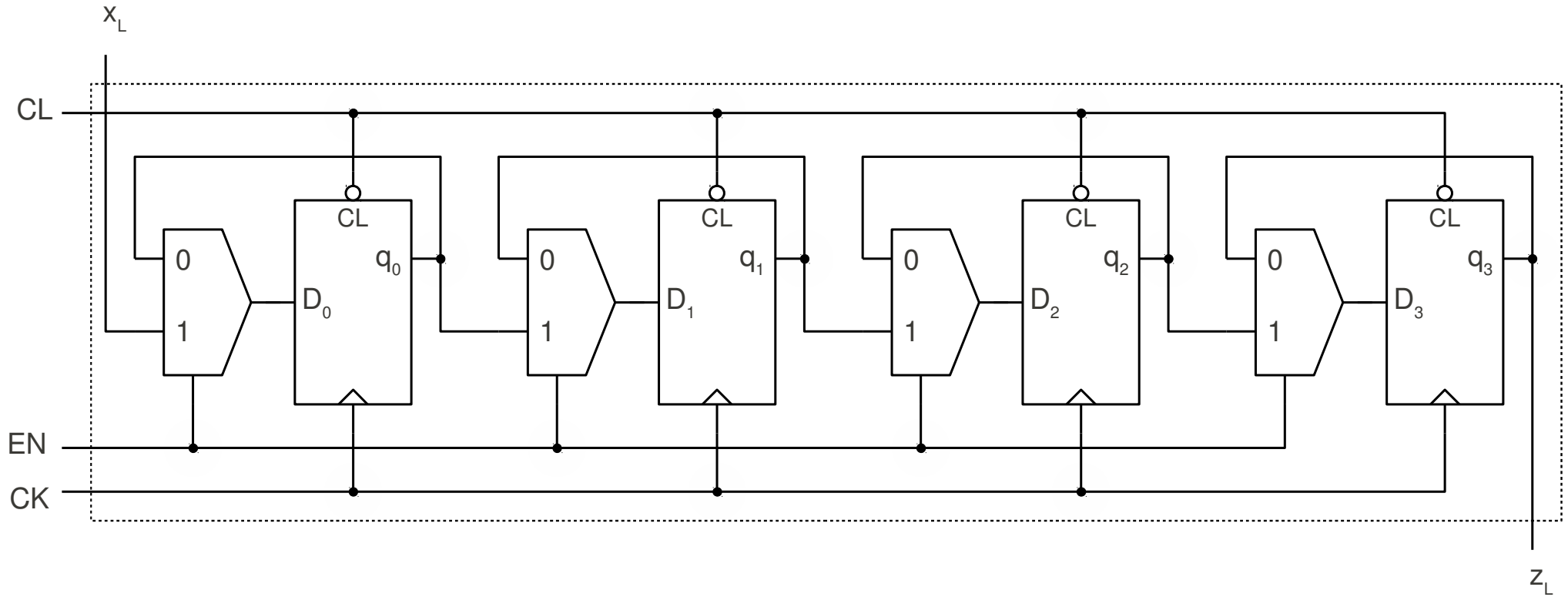
Tabla de operación síncrona

EN	Operación	Et. típica	Et. 0	Ex. típ.	Ex. et. 0
1	$q \leftarrow SHL(q)$	$Q_i = q_{i-1}$	$Q_0 = x_L$	$D_i = q_{i-1}$	$D_0 = x_L$
0	$q \leftarrow q$	$Q_i = q_i$	$Q_0 = q_0$	$D_i = q_i$	$D_0 = q_0$



Registro de desplazamiento





Registro universal

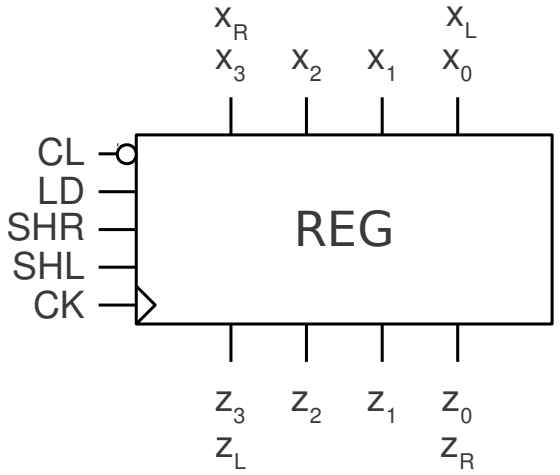


Tabla de operación

CL,LD,SHR,SHL	Operación	Tipo
0xxx	q ← 0	asínc.
11xx	q ← x	sínc.
1010	q ← SHR(q)	sínc.
1001	q ← SHL(q)	sínc.
1000	q ← q	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

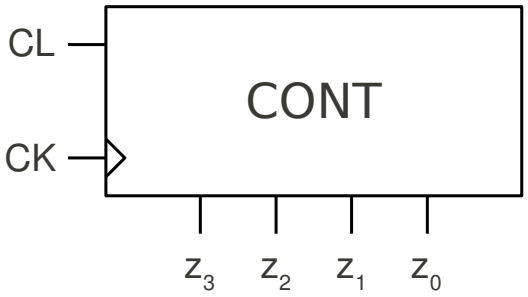
Contadores

- Similar al registro: añade operación de cuenta.
- Diseño
  - Se aplican los mismos principios del diseño modular
  - 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

- Módulo
  - Número de estados de cuenta del contador
- Binario
  - Los estados de cuenta representan números en base 2 consecutivos.
- Módulo 2^n
  - Cuenta de 0 a 2^n-1 (n bits)
- Cuenta cíclica
  - Normalmente, después del último estado de cuenta se pasa al primero (desbordamiento)

Contador binario módulo 2^n



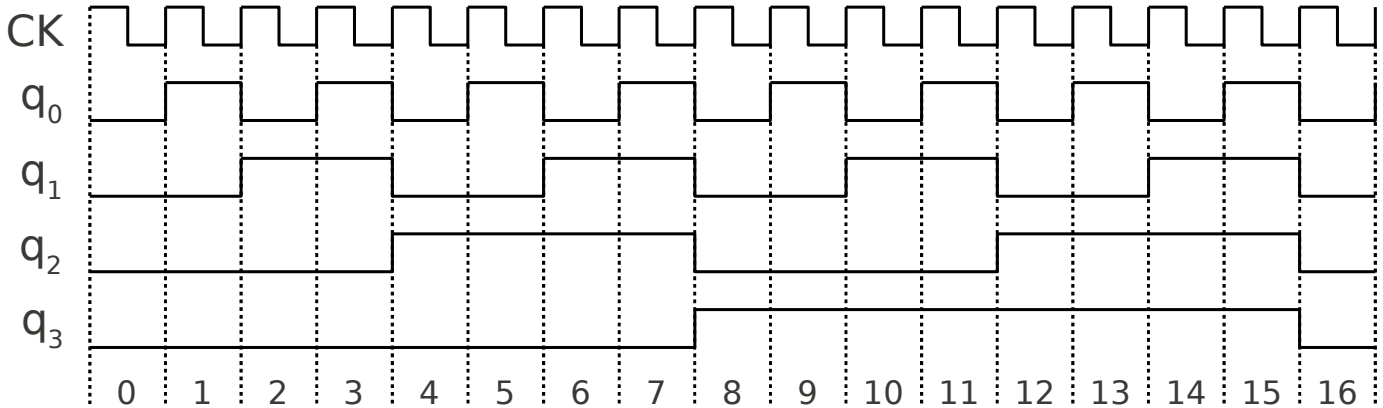
Código Verilog

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

    reg [3:0] q;

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

    assign z = q;

endmodule
```

Tabla de operación

CL	Operación	Tipo
1	$q \leftarrow 0$	asínc.
0	$q \leftarrow q+1\|_{\text{mod } 16}$	sínc.

Contador binario módulo 2^n
Operación de cuenta



- Cuenta ascendente
  - Qi = $\overline{q}_i$ si q_j=1, ∀j<i
  - Si no, Qi = q_i

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

Contador binario módulo 2^n

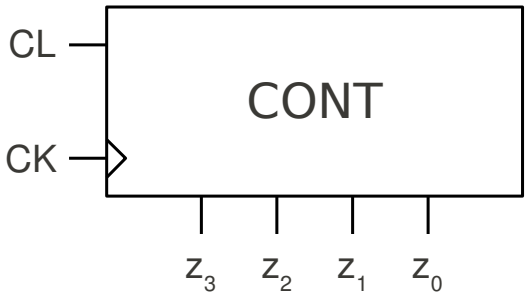
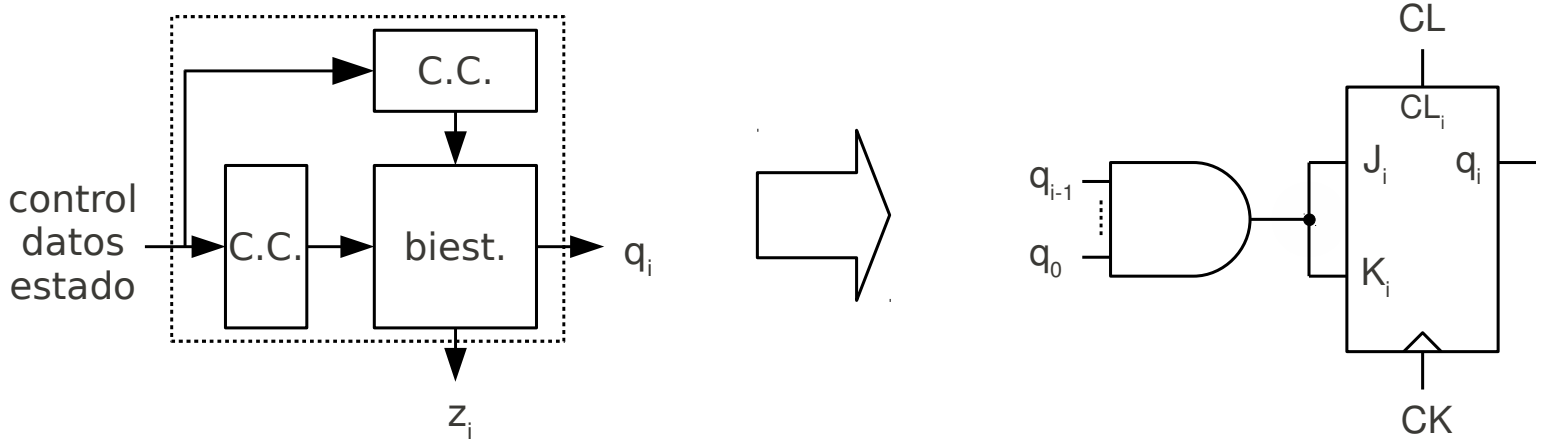
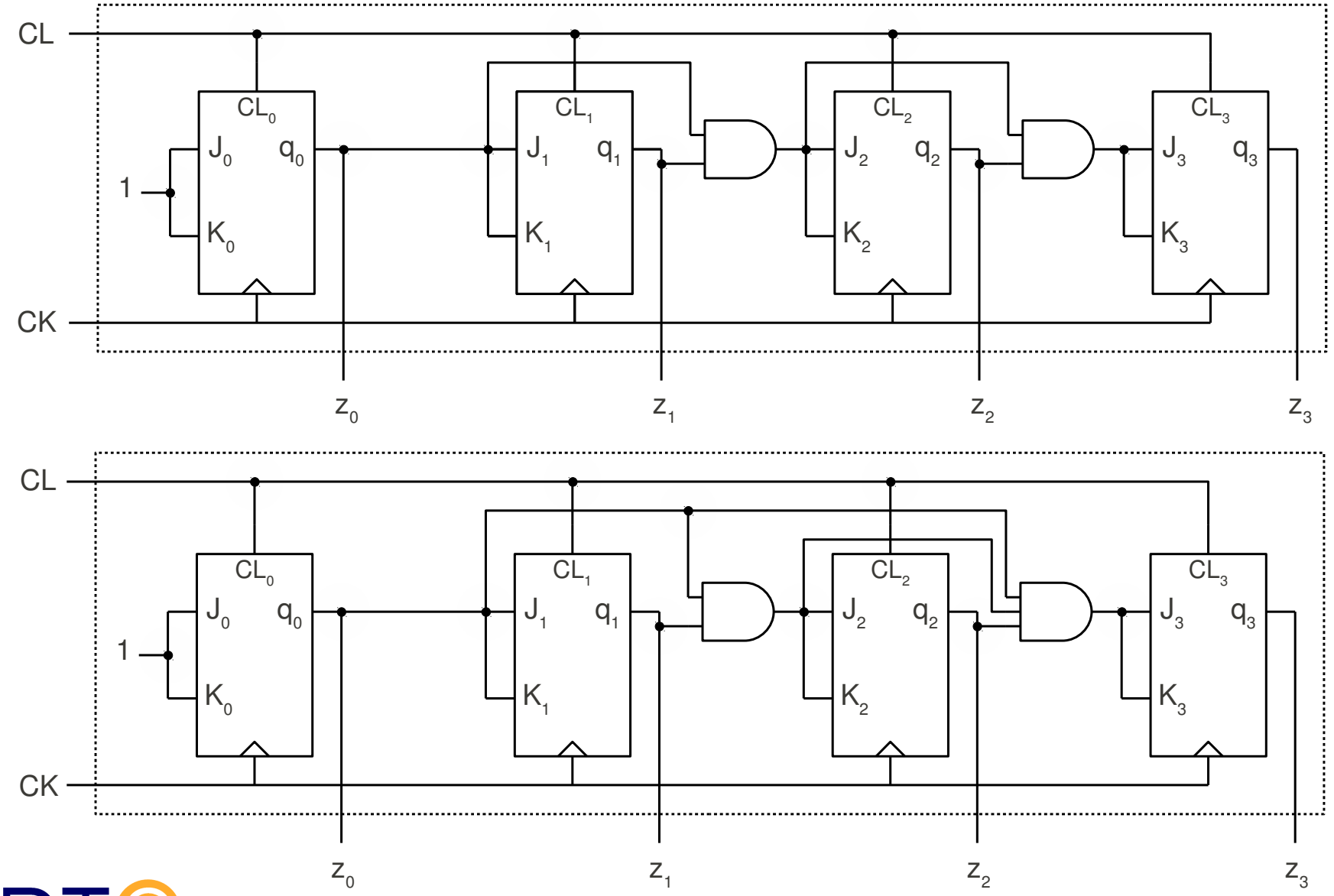


Tabla de operación síncrona

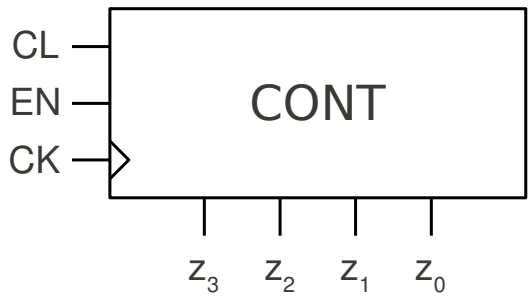
Operación	Et. típica	Et. 0
$q \leftarrow q+1\|_{mod\ 16}$	$J_i=K_i=q_{i-1} \dots q_0$	$J_0=K_0=1$



Contador binario módulo 2^n. Circuito



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



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
```

Tabla de operación

CL, EN	Operación	Tipo
1x	$q \leftarrow 0$	sínc.
01	$q \leftarrow q+1\|_{\text{mod } 16}$	sínc.
00	$q \leftarrow q$	sínc.

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

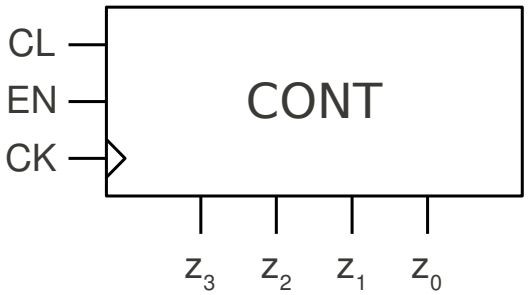
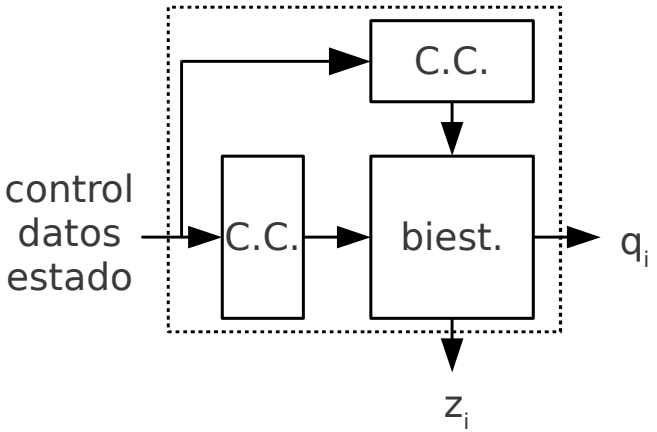
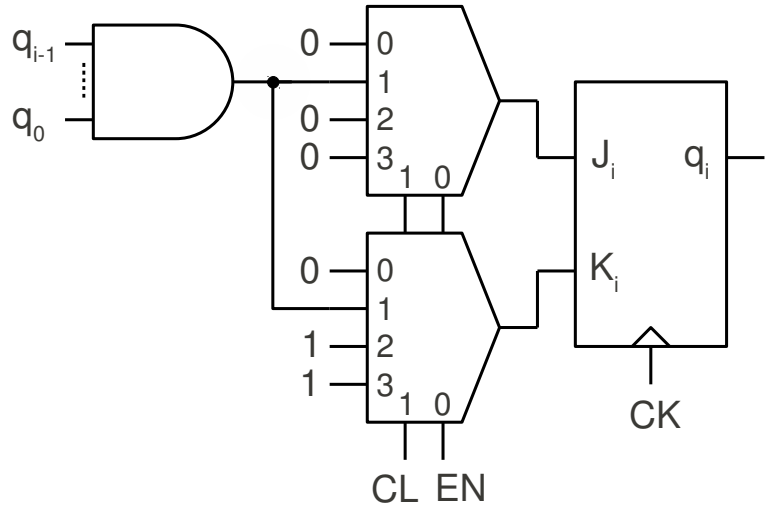


Tabla de operación síncrona

CL, EN	Operación	Et. típica	Et. 0
1x	$q \leftarrow 0$	$J_i=0, K_i=1$	$J_0=0, K_0=1$
01	$q \leftarrow q+1\|_{mod\ 16}$	$J_i=K_i=q_{i-1} \dots q_0$	$J_0=K_0=1$
00	$q \leftarrow q$	$J_i=K_i=0$	$J_0=K_0=0$





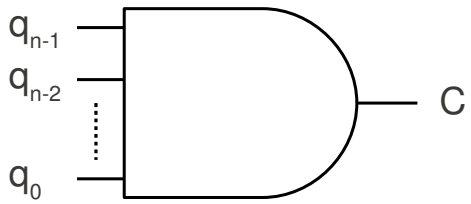
Contador binario módulo 2^n con puesta a cero y habilitación. Circuito

- (hacer como ejercicio)

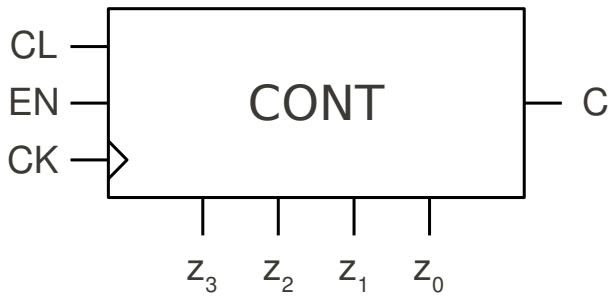
Salida de fin de cuenta

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

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



Salida de fin de cuenta



Código Verilog

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

    reg [3:0] q;

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

    assign z = q;
    assign c = &q;

endmodule
```

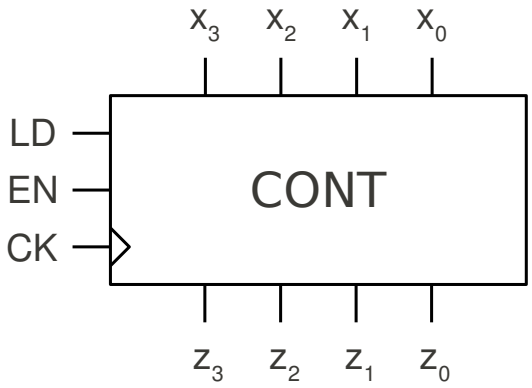
Unión de contadores

- 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*l
- Ejemplo: contador módulo 256 (8bits) a partir de dos contadores módulo 16 (4 bits)
- La combinación de los contadores se puede hacer de forma más simple (con menos componentes adicionales) si los contadores poseen entradas de control y estado adecuadas. Ej:
  - Habilitación
  - Fin de cuenta

Unión de contadores

- (contador mod 256 a partir de mod 16)
- (contador mod 4096 a partir de mod 16)

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



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
```

Tabla de operación

LD, EN	Operación	Tipo
1x	$q \leftarrow x$	sínc.
01	$q \leftarrow q+1\|_{\text{mod } 16}$	sínc.
00	$q \leftarrow q$	sínc.

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

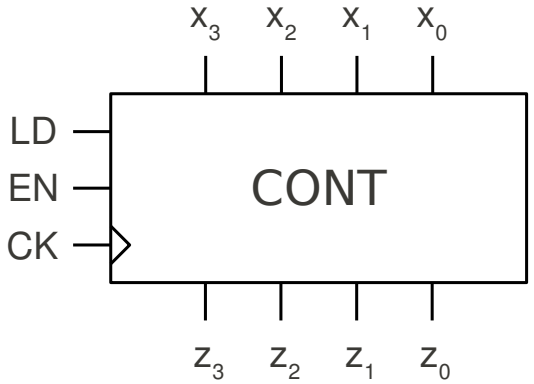
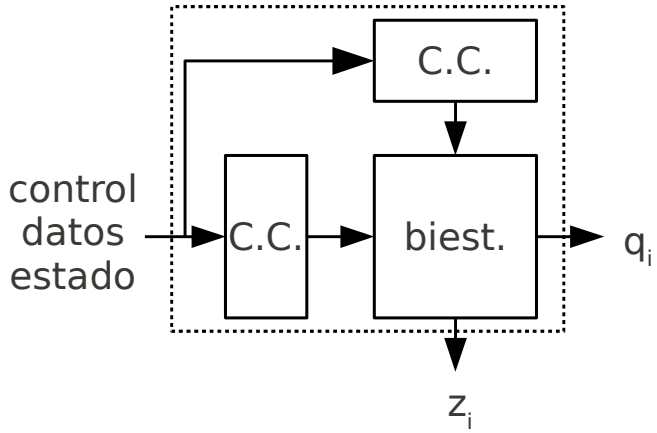
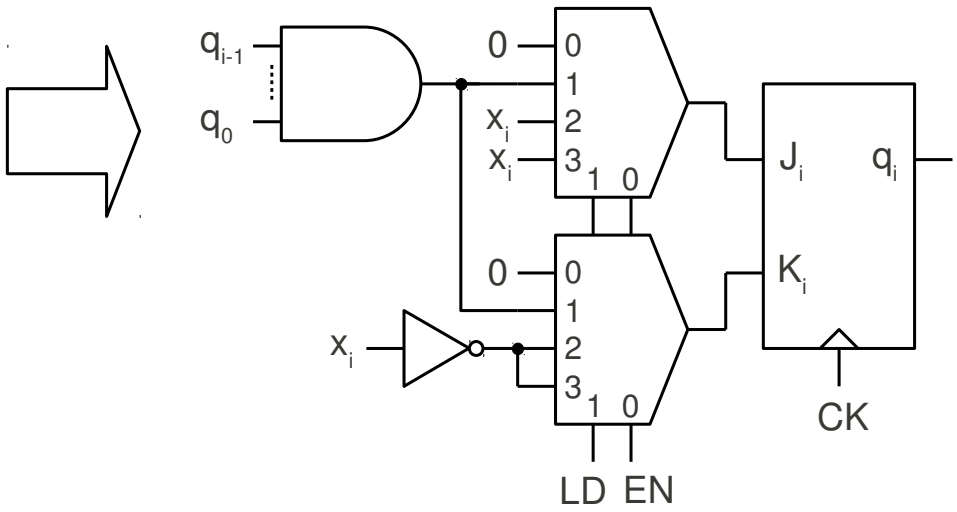


Tabla de operación síncrona

LD, EN	Operación	Et. típica	Et. 0
1x	$q \leftarrow x$	$J_i=x_i$, $K_i=\overline{x}_i$	$J_0=x$, $K_0=\overline{x}$
01	$q \leftarrow q+1\|_{mod\ 16}$	$J_i=K_i=q_{i-1} \dots q_0$	$J_0=K_0=1$
00	$q \leftarrow q$	$J_i=K_i=0$	$J_0=K_0=0$





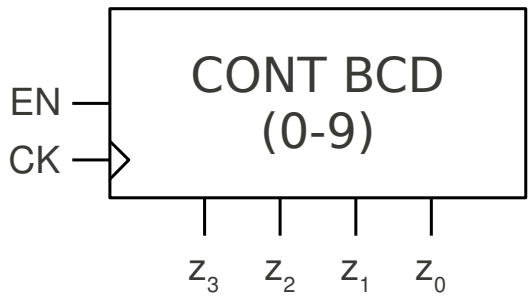
Contador binario módulo 2^n con puesta a cero, habilitación y carga

- (Esquema como ejercicio)

Límite de estado de cuenta

- Los contadores módulo < 2^n se obtienen normalmente limitando los estados de cuenta de un contador módulo 2^n.
- Casos
  - Límite superior: 0 ... l, $l<2^n$
  - Límite inferior: k ... 2^n, $k > 0$
  - Límites inferior y superior: k ... l, $k>0$, $l<2^n$
- Estrategia
  - Se detecta la llegada al estado de cuenta l 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 l+1, apareciendo un estado de cuenta transitorio a la salida.

Límite de estado de cuenta



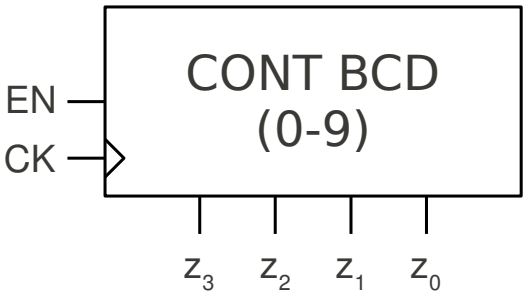
Código Verilog

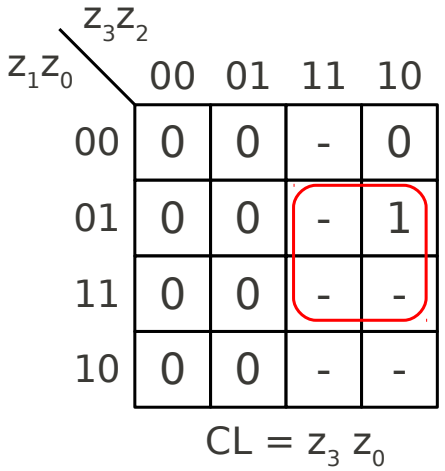
```
module count_mod10(
    input ck,
    input cl,
    input en,
    output [3:0] z,
    );

    reg [3:0] q;

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

    assign z = q;

endmodule
```

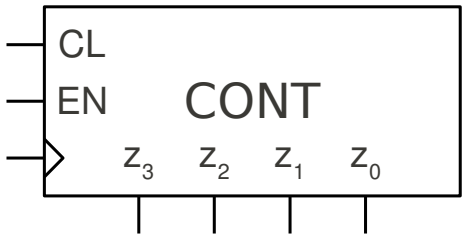
Tabla de operación

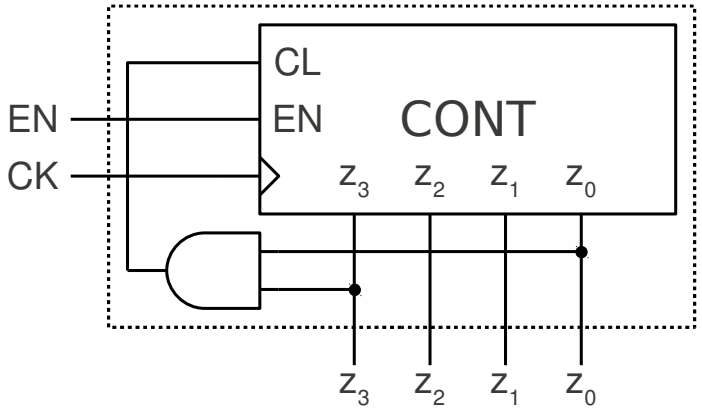
CL, EN	Operación	Tipo
1x	$q \leftarrow 0$	sínc.
01	$q \leftarrow q+1\|_{\text{mod } 10}$	sínc.
00	$q \leftarrow q$	sínc.

Límite de estado de cuenta. Contador BCD

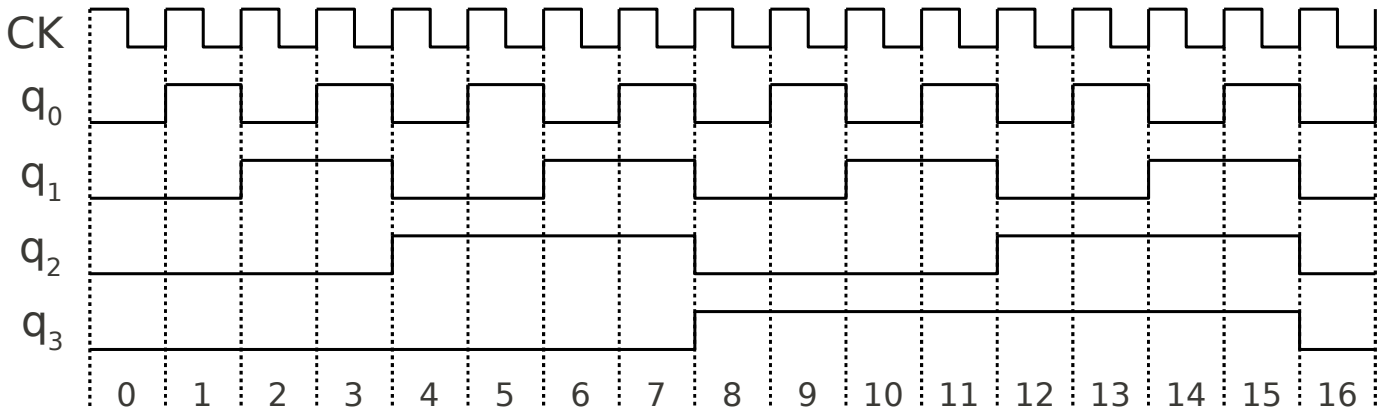








Contador descendente módulo 2^n



- Cuenta descendente
  - Qi = $\overline{q_i}$ si q_j=0, ∀j<i
  - Si no, Qi = q_i

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

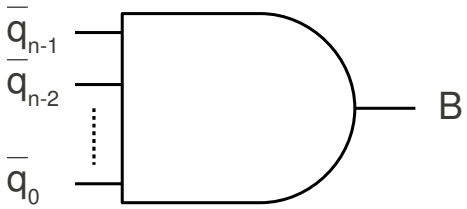
$$J_i = K_i = \overline{q_{i-1}}\,\overline{q_{i-2}} \ldots \overline{q_0}$$

Salida de fin de cuenta

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

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

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



Contador binario descendente mód. 2^n con CL, habilit. y fin de cuenta

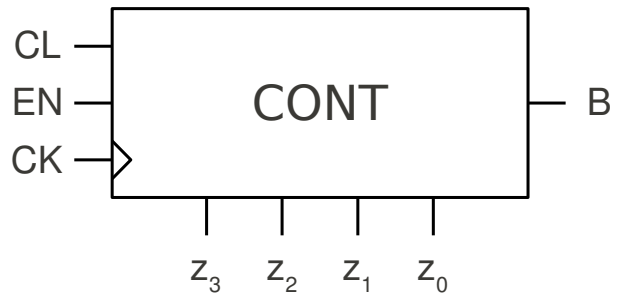


Tabla de operación

CL, EN	Operación	Tipo
1x	$q \leftarrow 0$	sínc.
01	$q \leftarrow q\text{-}1\|_{\text{mod } 16}$	sínc.
00	$q \leftarrow q$	sínc.

Código Verilog

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

    reg [3:0] q;

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

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

endmodule
```

Contador binario descendente mód. 2^n con CL, habilit. y fin de cuenta

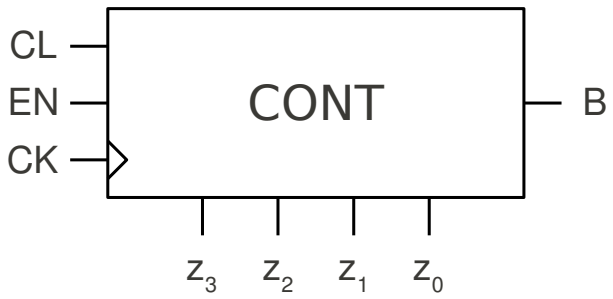
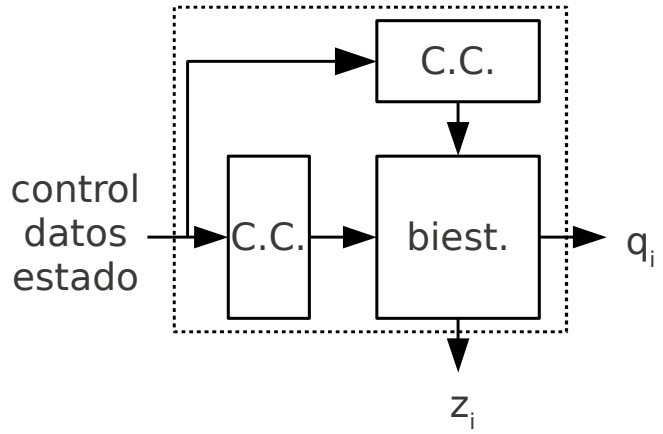
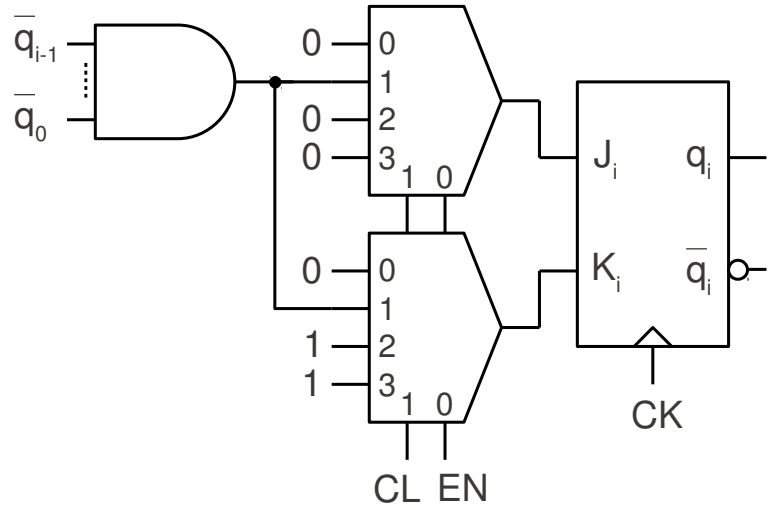


Tabla de operación síncrona

CL, EN	Operación	Et. típica	Et. 0
1x	$q \leftarrow 0$	$J_i=0, K_i=1$	$J_0=0, K_0=1$
01	$q \leftarrow q\text{-}1\|_{\text{mod } 16}$	$J_i=K_i=\overline{q}_{i-1} \dots \overline{q}_0$	$J_0=K_0=1$
00	$q \leftarrow q$	$J_i=K_i=0$	$J_0=K_0=0$





Contador reversible

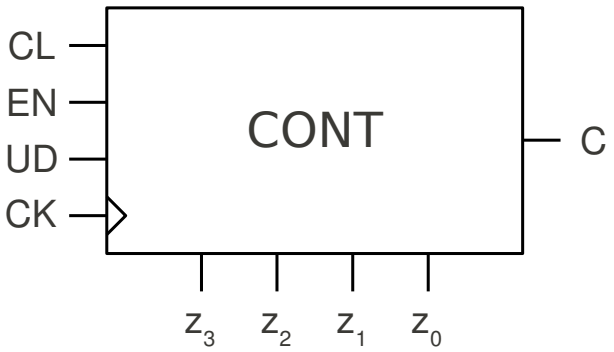


Tabla de operación

CL, EN, UD	Operación	Tipo
1xx	$q \leftarrow 0$	asínc.
00x	$q \leftarrow q$	sínc.
010	$q \leftarrow q+1\|_{\text{mod } 16}$	sínc.
011	$q \leftarrow q-1\|_{\text{mod } 16}$	sínc.

Contador reversible

Código Verilog

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

    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 c = ud ? ~(|q) : &q;

endmodule
```

Código Verilog

```
module rev_counter2(
    input ck,
    input cl, input en, input ud,
    output [3:0] z, output c
    );
    reg [3:0] q; reg c;
    always @(posedge ck, posedge cl)
    begin
        if (cl)
            q <= 0;
        else if (en)
            if (!ud)
                q <= q + 1;
            else
                q <= q - 1;
    end

    always @*
        if (ud)
            c = ~(|q);
        else
            c = &q;

    assign z = q;

endmodule
```

Contador reversible

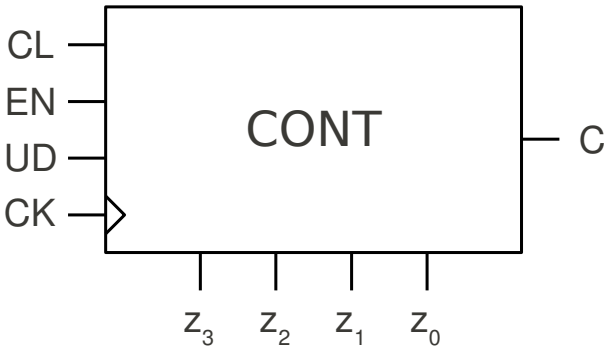
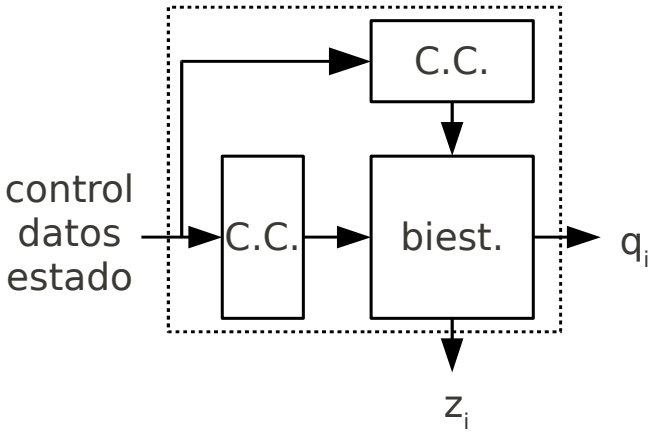
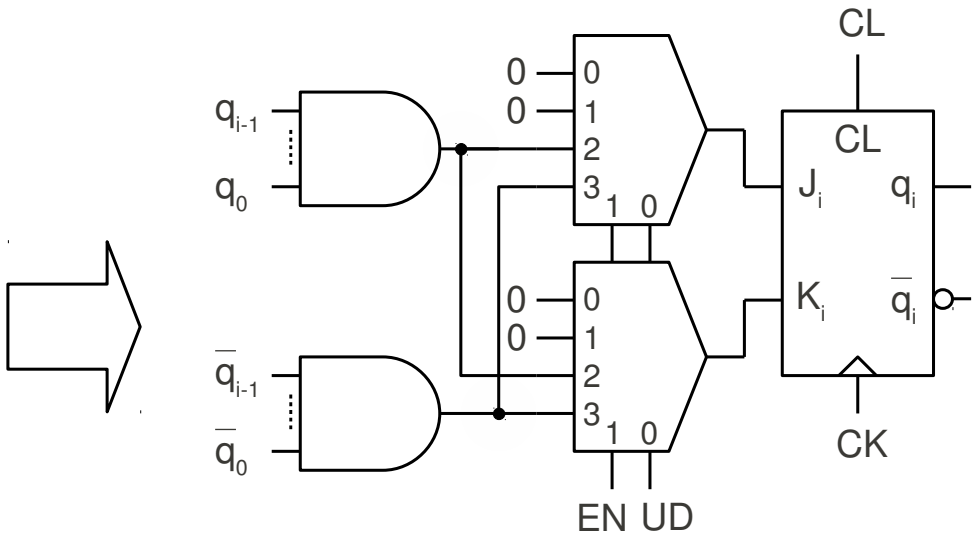


Tabla de operación síncrona

EN, UD	Operación	Et. típica	Et. 0
0x	$q \leftarrow q$	$J_i=K_i=0$	$J_0=K_0=0$
10	$q \leftarrow q+1\|_{\text{mod } 16}$	$J_i=K_i=q_{i-1} \dots q_0$	$J_0=K_0=1$
11	$q \leftarrow q-1\|_{\text{mod } 16}$	$J_i=K_i=\overline{q}_{i-1} \dots \overline{q}_0$	$J_0=K_0=1$

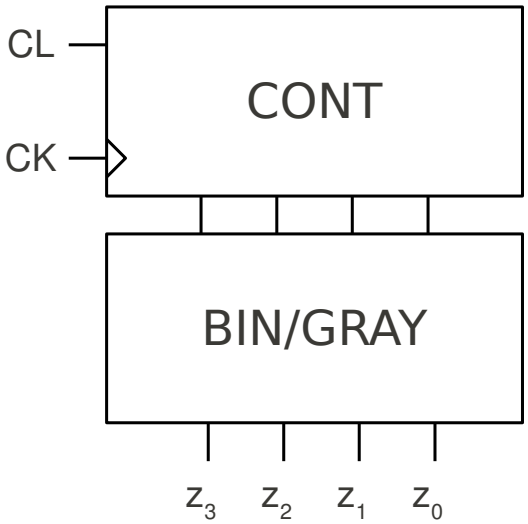




Contadores no binarios

- Secuencia no binaria (ej. Gray)
  - Nativos
  - Mediante convertidor de códigos
- Contadores de desplazamiento
  - Contador en anillo
  - Contador Johnson

Contador Gray con convertidor de código

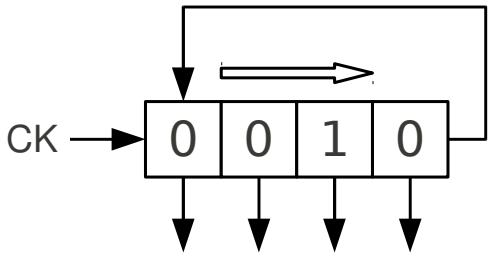


z_3 z_2 z_1 z_0
0000
0001
0011
0010
0110
0111
0101
0100
1100
1101
1111
1110
1010
1011
1001
1000

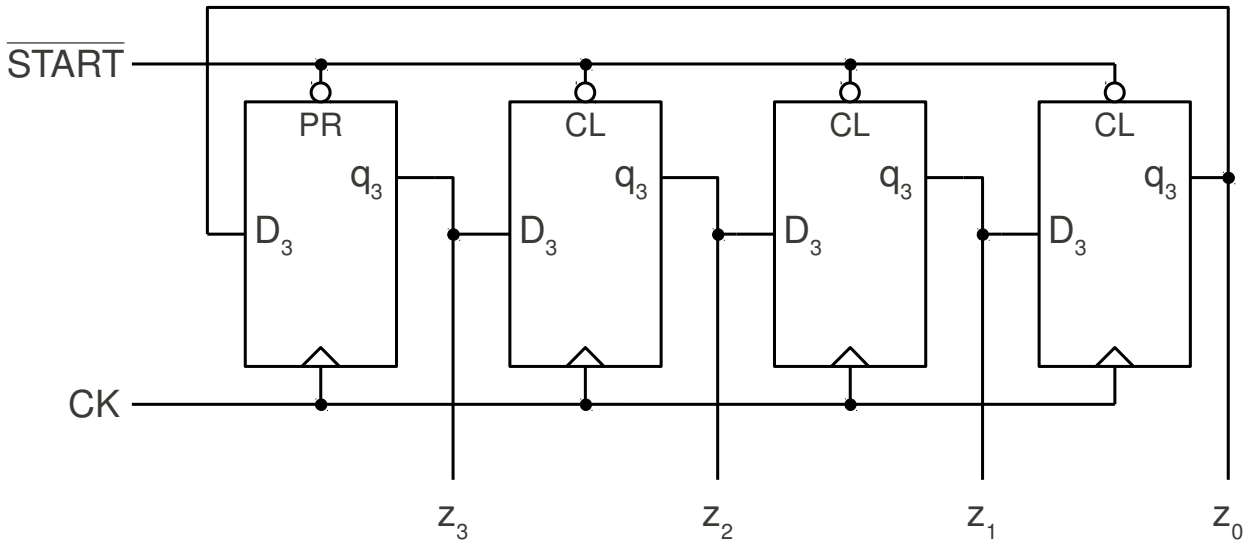
Código Verilog

```
module graycounter_mod16(
    input ck,
    input 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



z_3 z_2 z_1 z_0
1000
0100
0010
0001
1000
...

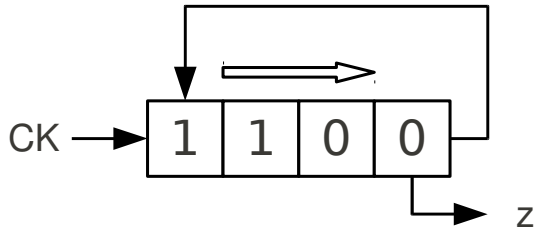


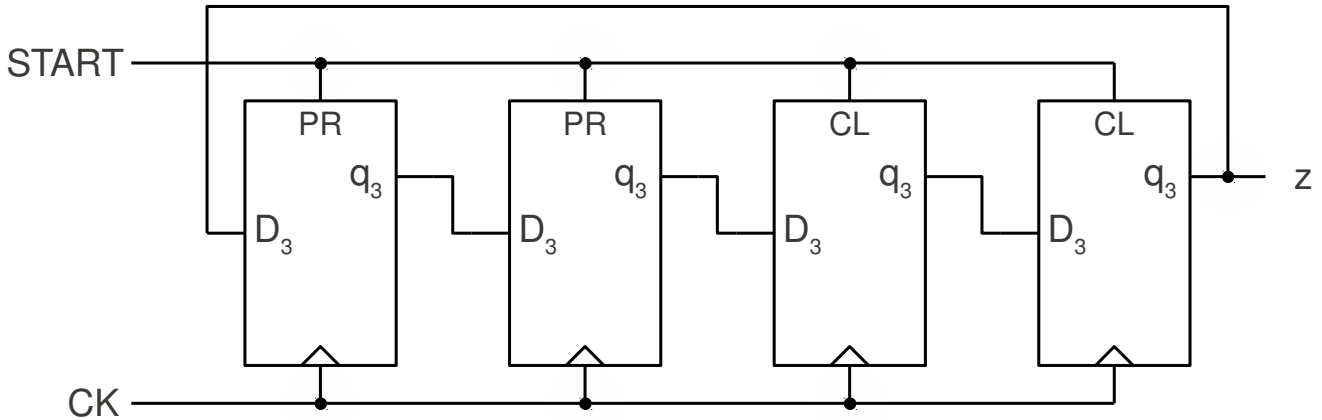
Diseño con subsistemas secuenciales

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

Generador de secuencia

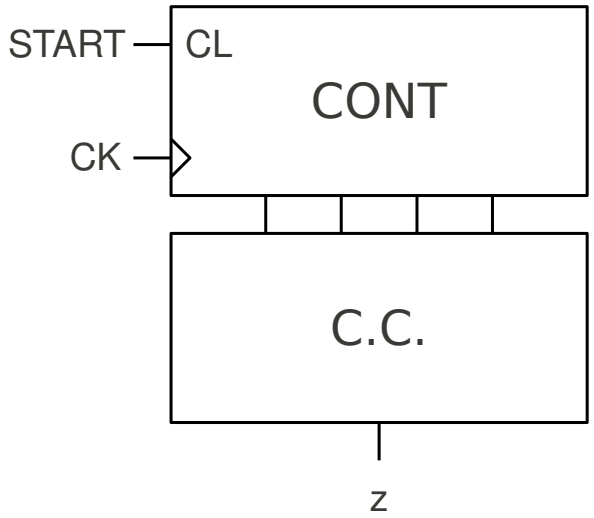
- Con registro de desplazamiento

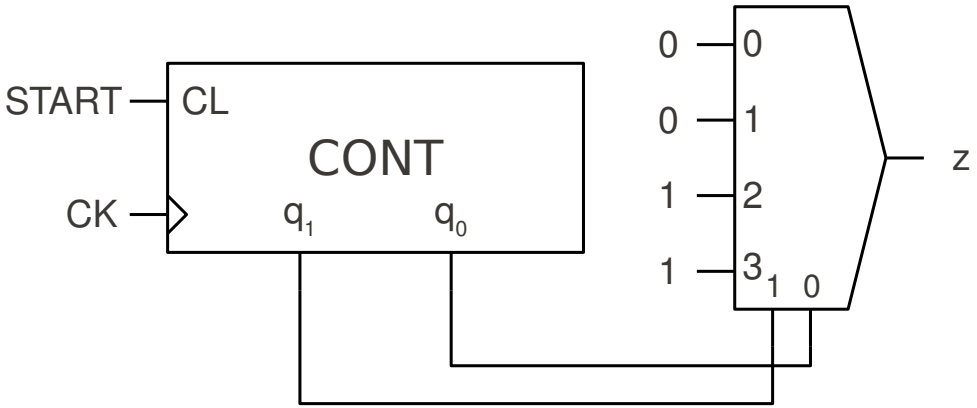




Generador de secuencia

- Con contador y CC

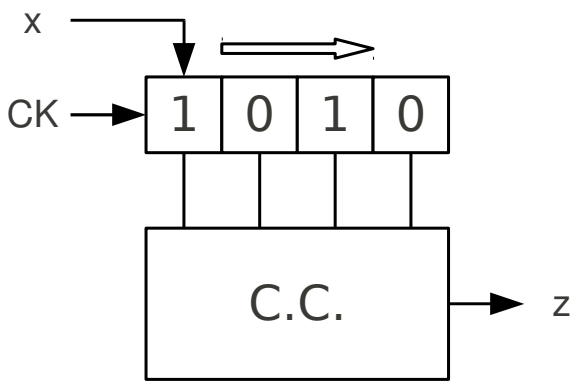




Código Verilog

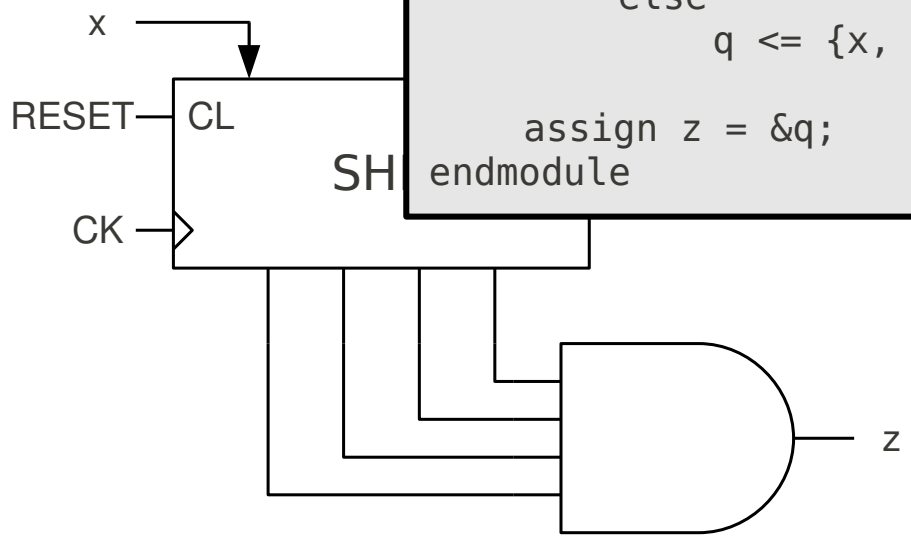
```
module seq_gen(
    input ck,
    input start,
    output z
    );

    reg [1:0] q; reg z;

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

    case (q)
      2'h0: z = 1'b0;
      2'h1: z = 1'b0;
      2'h2: z = 1'b1;
      2'h3: z = 1'b1;
    endcase

endmodule
```

Detector de secuencia



Código Verilog

```
module seq_det(
    input ck, input reset,
    input x,
    output z
    );

    reg [3:0] q;

    always @(posedge ck)
        if (reset == 1)
            q <= 0;
        else
            q <= {x, q[3:1]};

    assign z = &q;
endmodule
```



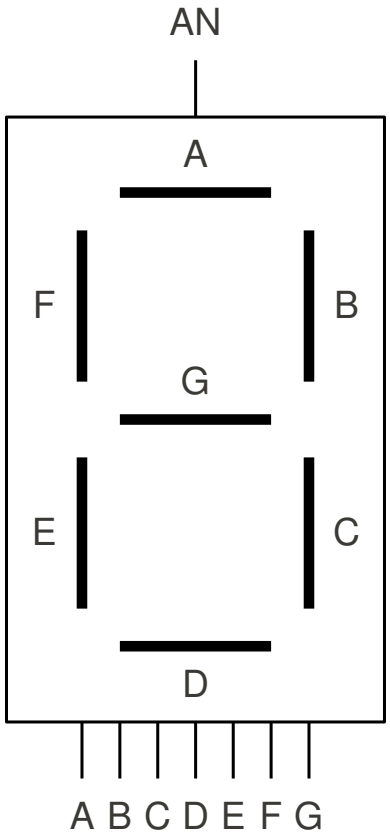
Ejemplo

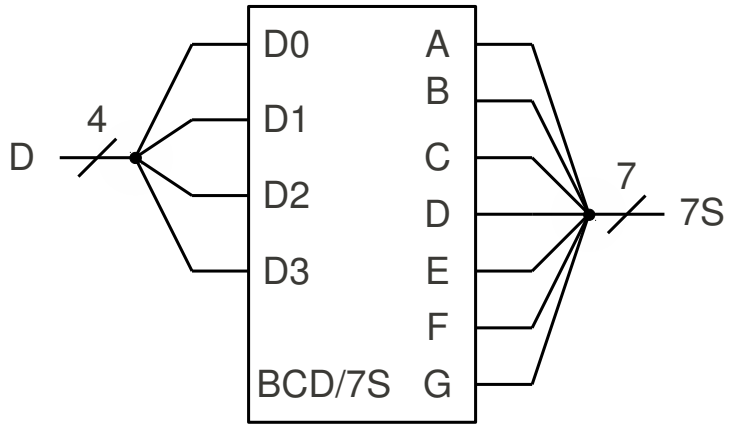
- Control de puerta con cierre automático
- (añadir sensor de seguridad)

Ejemplo

- Filtro de rebotes para pulsador/interruptor

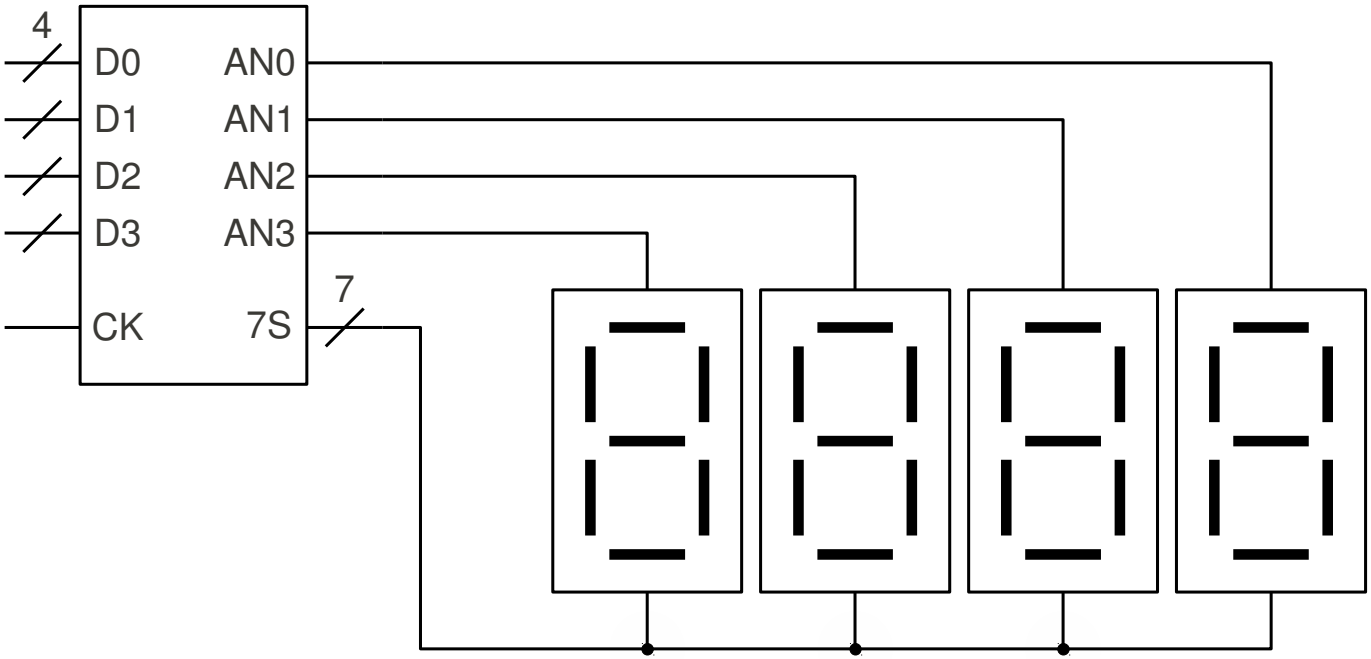
Ejemplo: controlador 7S

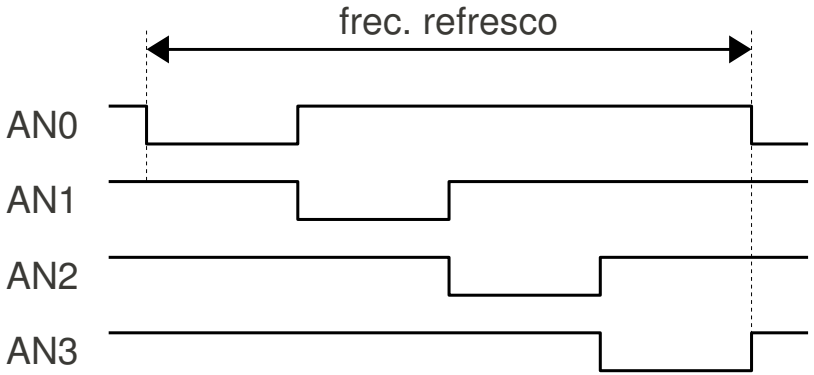




$D_3D_2D_1D_0$	7S	ABCDEFG
0000	0	0000001
0001	1	1001111
0011	2	0010010
0010	3	0000110
0110	4	1001100
0111	5	0100100
0101	6	0100000
0100	7	0001111
1100	8	0000000
1101	9	0001100

Ejemplo: controlador 7S





CK: 50MHz
F. refresco: 100Hz - 1KHz