Unidad 9. Subsistemas secuenciales

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

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

Contenido

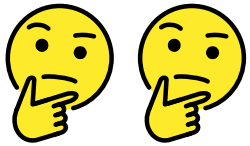
- Introducción
- Registros
- Contadores
- Diseño con subsistemas secuenciales

Lecturas y ejercicios recomendados

- Contenidos teóricos
  - LaMeres: 7.5 (contadores desde el punto de vista del diseño con máquinas de estados).
  - Floyd: 8.1 – 8.7 (contadores), 9.1 – 9.8 (registros de desplazamiento).
- Modelado en Verilog
  - curso_verilog.v: unidad 7.
  - LaMeres: 9.4, 9.5.
- Ejercicios recomendados del boletín general (boletín 8)
  - Diseño de registros: 4.
  - Diseño de contadores: 2, 3a.
  - Generador de secuencia: 5a, 5b.
  - Diseño de contadores a partir de otro básico: 6.
  - Reloj digital: 8.

Leyenda de ejercicios

Ejercicio básico/fácil



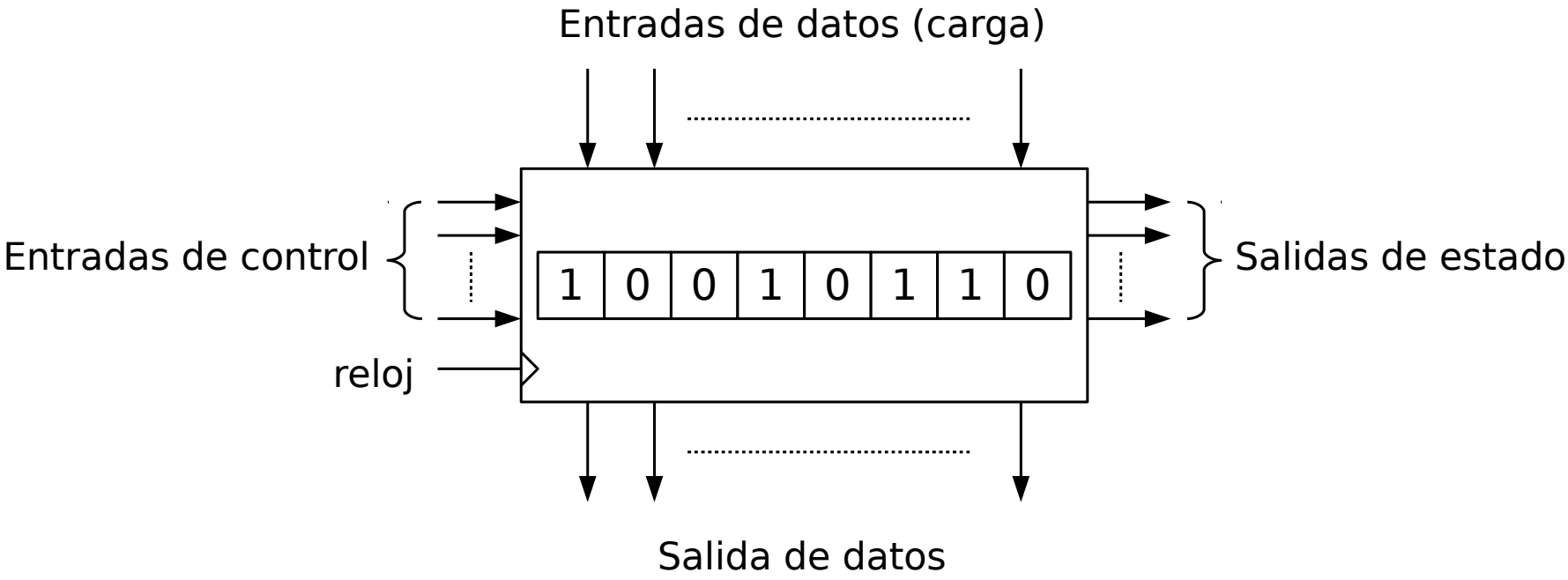
Ejercicio para pensar un poco

Ejercicio para pensar mucho

Introducción

- Subsistemas secuenciales
  - Son circuitos secuenciales con n biestables que operan en conjunto para hacer alguna tarea o tareas determinadas.
  - La operación se refiere a los n bits usados en conjunto y no a la operación de cada bit por separado.
  - La funcionalidad del circuito es lo bastante general para ser de utilidad en múltiples aplicaciones.
- Tipos básicos de subsistemas secuenciales
  - Registros: almacenan un dato de n bits para su recuperación posterior, con posibilidad, a veces, de hacer transformaciones sobre el dato.
  - Contadores: realizan operaciones de cuenta (incremento, decremento, etc.) sobre el dato almacenado.

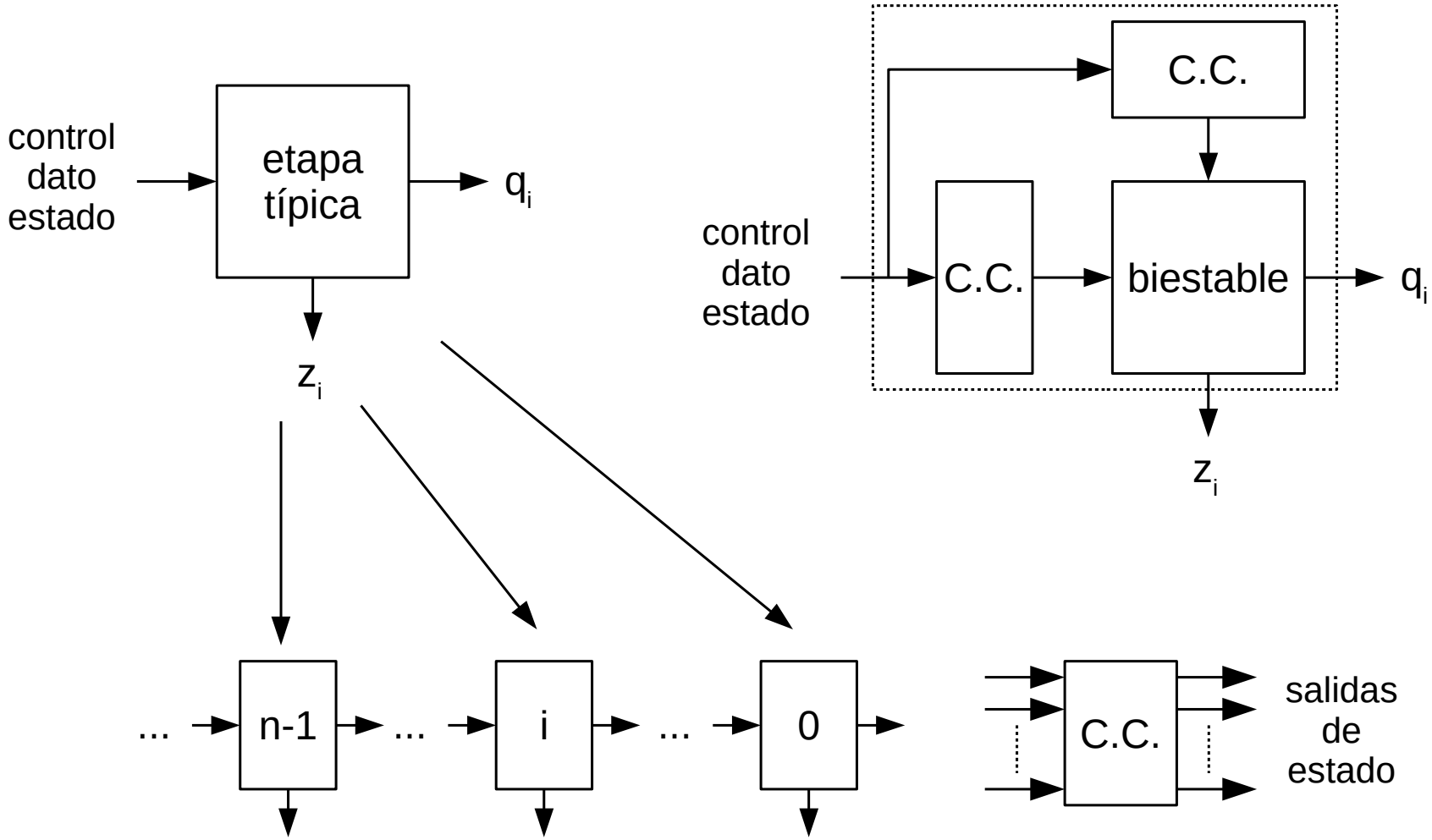
Introducción



Introducción

- Entradas de control
  - Seleccionan la operación a realizar.
  - Operaciones síncronas: ocurren con el flanco activo del reloj. Emplean las entradas síncronas de los biestables.
  - Operaciones asíncronas: ocurren inmediatamente al activarse la señal de control. Emplean las entradas asíncronas de los biestables.
  - Señales de control típicas:
    - CL: puesta a cero (clear)
    - EN: habilitación (enable)
    - LD: carga de dato (load)
- Entradas de datos: proporcionan el dato a cargar
- Salidas de datos: dan acceso al dato almacenado
- Salidas de estado: informan sobre el dato almacenado o estado del sistema
  - Ej: contenido cero, fin de cuenta, etc.

Diseño modular



Diseño modular

- 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.

Notación RT (Register Transfer)

- Notación empleada para representar asignación de datos de múltiples bits (contenido de registros).
- Operadores de asignación:
  - A ← <expresión> (asignación secuencial): el valor de la expresión se almacena en el registro A al producirse un flanco activo del reloj.
    - El nombre de un registro en la expresión corresponde al valor “actual” del registro.
    - El nombre de un registro en el lado izquierdo de la asignación corresponde al registro que será “escrito”.
    - El valor de la expresión corresponde al próximo valor a ser almacenado en el registro.
  - A = <expresión> (asignación combinacional): el valor de A se obtiene del valor de la expresión a través de una operación combinacional.

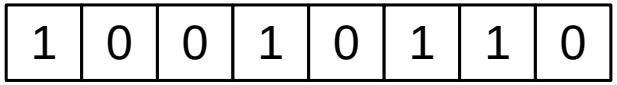
Notación RT (Register Transfer)

Tipo	Operadores	Función	Ejemplos
Aritméticos	+, -, *, /, ...	Operaciones aritméticas	$A \leftarrow B+C$ $CNT \leftarrow CNT+1$
Lógicos	&, |, ^, $\overline{\ }$, ...	Operaciones lógicas bit a bit	$A = B\&\overline{C}$
Desplazamiento	SHL(A,b), SHR(A,b)	Se desplaza un bit el dato en A. Se completa con el bit b.	$Q \leftarrow SHL(Q,1)$ $Q \leftarrow SHR(X,Y_R)$
Selección de 1 bit	A[i], A_i	Se selecciona el bit i-ésimo de A	$z = B_3$
Selección de bits	A[i..j], $A_{i..j}$	Se selecciona un rango de bits de A	$B = Q_{7..4}$
Combinación	{A,B}	Se combinan datos para formar un dato mayor	$A \leftarrow \{B,C\}$ $D = \{A_{3..0},B_{7..4}\}$

Contenidos

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

Registros



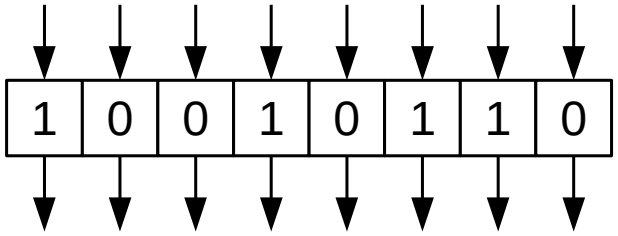
- Almacén de datos de n bits (n biestables)
  - Habitualmente nos referimos al contenido por el dato que representa y no por los bits individuales.
  - En general, emplean biestables tipo D: facilita el diseño.
  - Los biestables comparten señal de reloj y entradas asíncronas: funcionan a la vez.
- Operaciones básicas:
  - Escritura (carga): modificación del dato almacenado.
  - Lectura: acceso al contenido del registro.

Registros
Clasificación

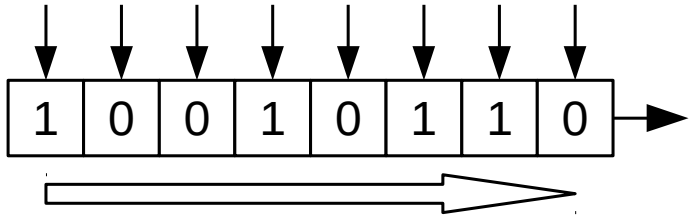
- Entrada en paralelo
  - Todos los bits pueden cargarse a la vez (en el mismo ciclo de reloj).
  - Una línea de entrada para cada bit.
- Entrada serie
  - Se carga un bit en cada ciclo de reloj.
  - Una única línea de entrada para todos los bits (necesita operación de desplazamiento de bits).
- Salida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

Registros
Clasificación

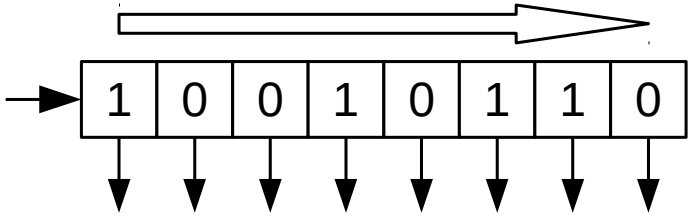
paralelo/paralelo



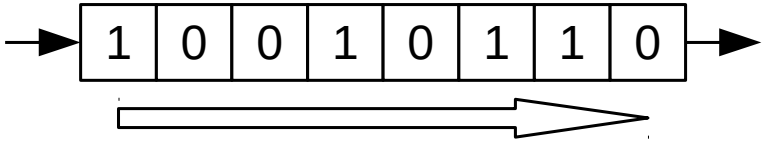
paralelo/serie



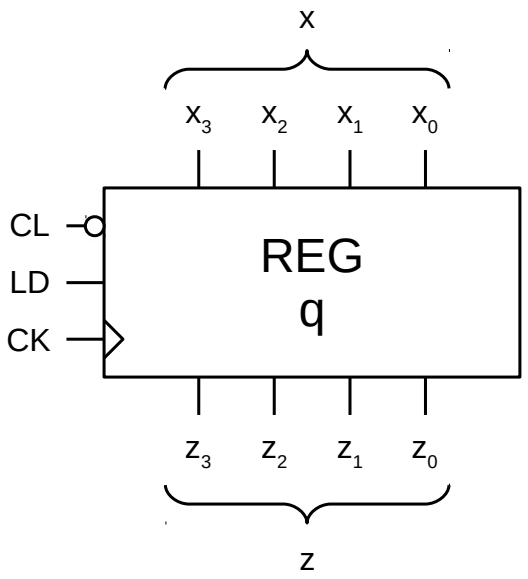
serie/paralelo



serie/serie



Registro entrada paralelo/salida paralelo



Código Verilog

```
module reg(
    input  wire ck,
    input  wire cl,
    input  wire ld,
    input  wire [3:0] x,
    output wire [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

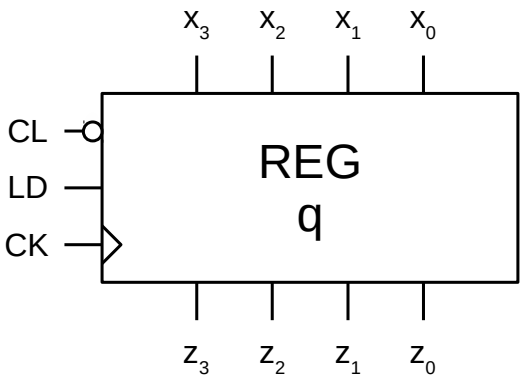
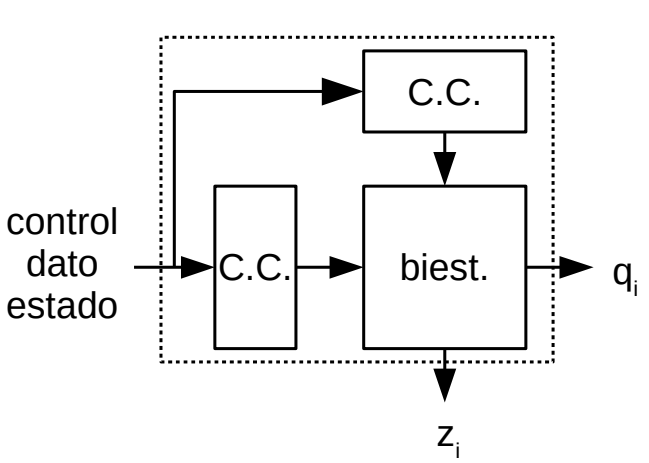


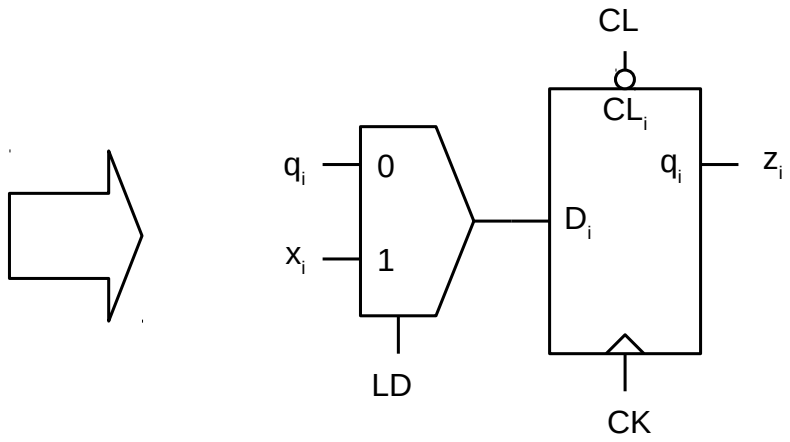
Tabla de operación asíncrona

CL	Operación	Etapa típica	Excitación
0	$q \leftarrow 0$	$q_i \leftarrow 0$	$CL_i = 0$
1	$q \leftarrow q$	$q_i \leftarrow q_i$	$CL_i = 1$

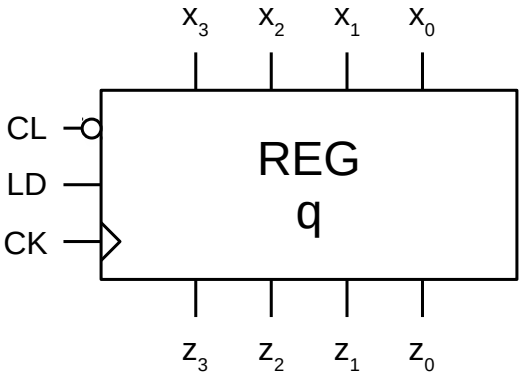
Tabla de operación síncrona

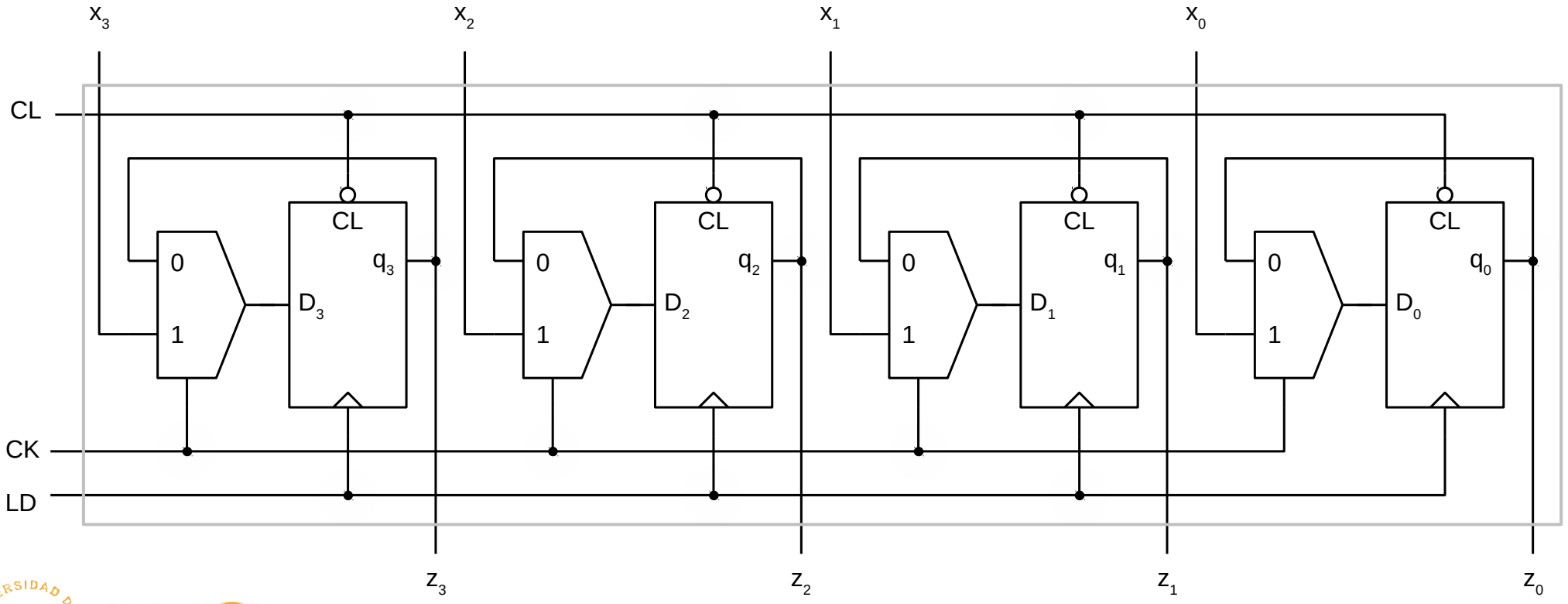
LD	Operación	Etapa típica	Excitación
1	$q \leftarrow x$	$q_i \leftarrow x_i$	$D_i = x_i$
0	$q \leftarrow q$	$q_i \leftarrow q_i$	$D_i = q_i$



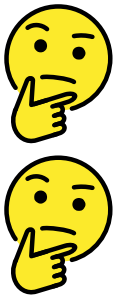


Registro entrada paralelo/salida paralelo





Ejercicio 1. 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 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

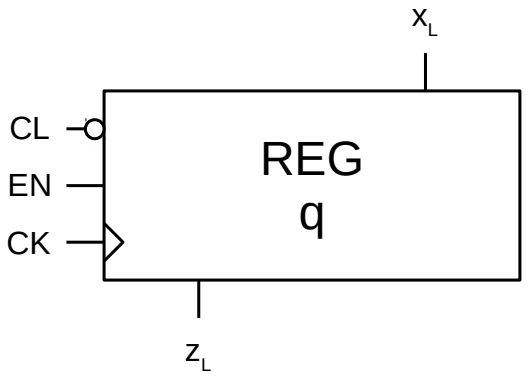


Tabla de operación

CL, EN	Operación	Tipo
0x	$q \leftarrow 0$	asínc.
11	$q \leftarrow SHL(q,x_L)$	sínc.
10	$q \leftarrow q$	sínc.

Código Verilog

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

    reg [3:0] q;

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

    assign zl = q[3];

endmodule
```

Registro de desplazamiento

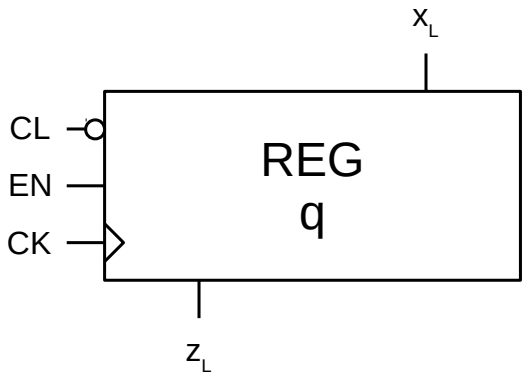
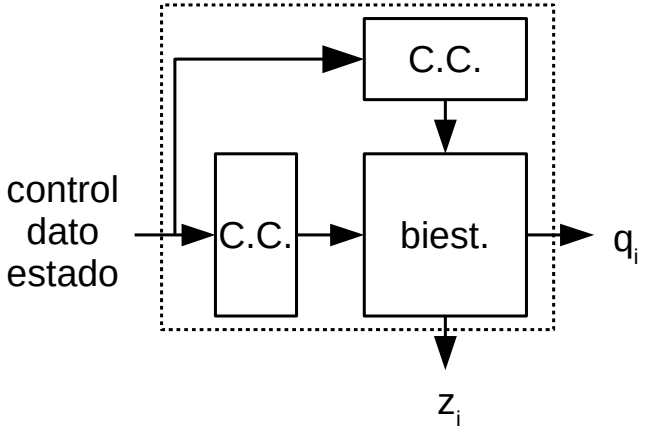
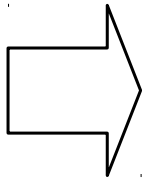
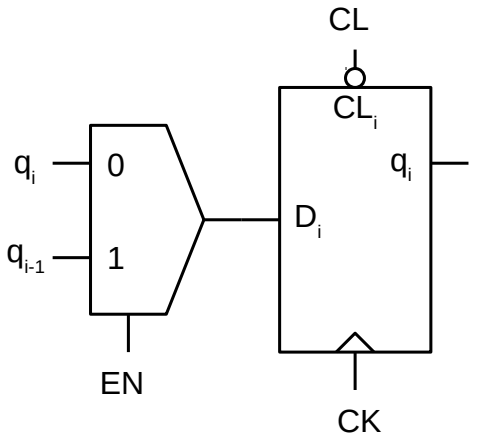


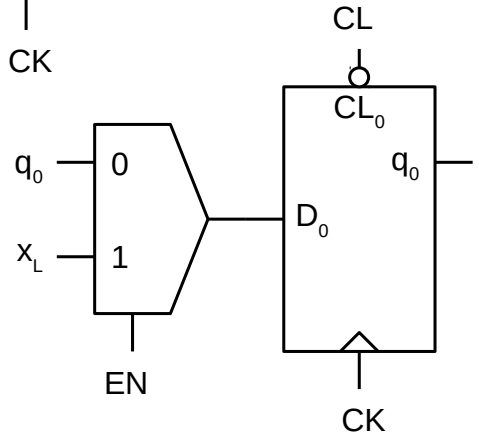
Tabla de operación síncrona

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

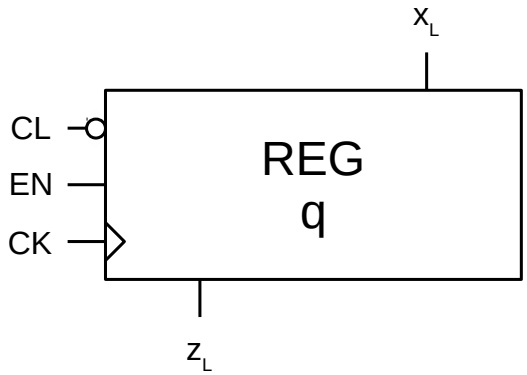


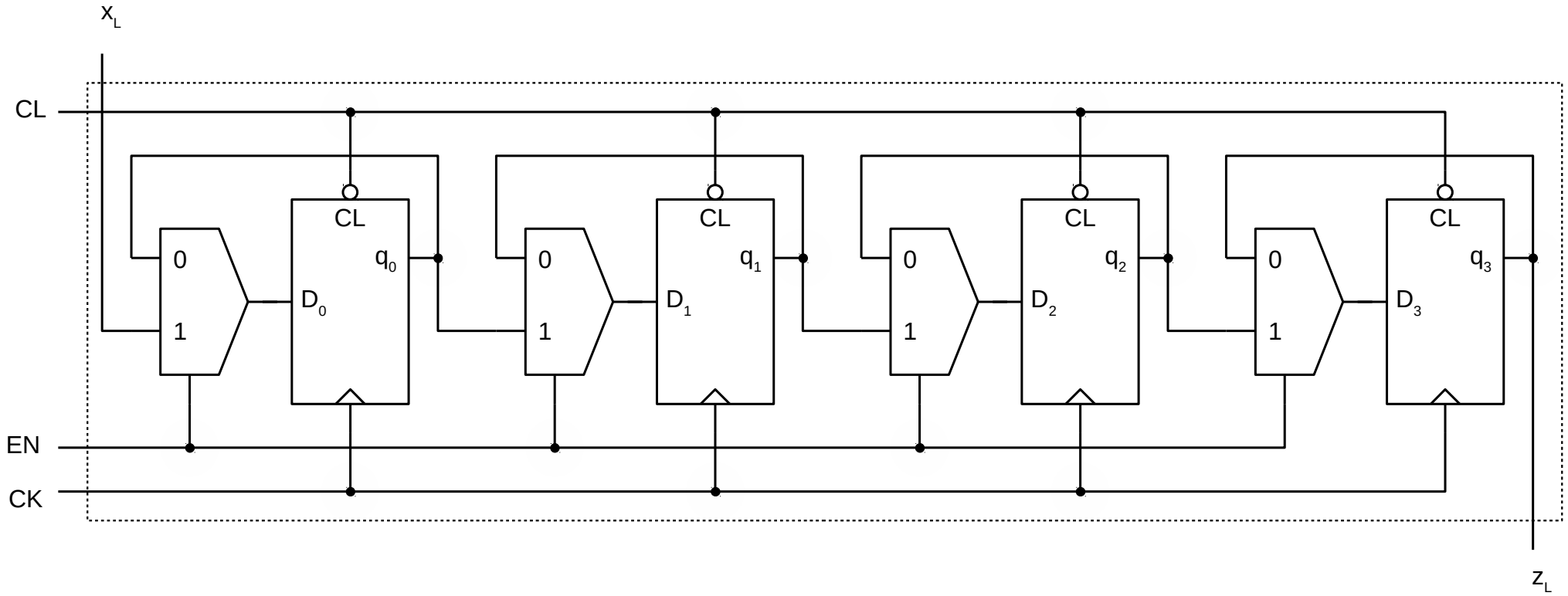






Registro de desplazamiento





Registro de desplazamiento

¿Cómo se haría un registro de desplazamiento a la derecha?

Registro universal

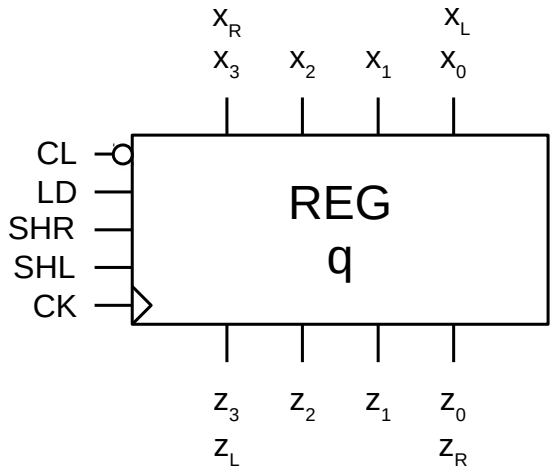


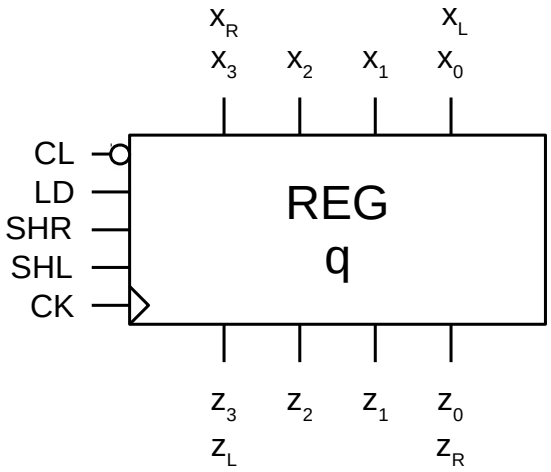
Tabla de operación

CL,LD,SHR,SHL	Operación	Tipo
0xxx	q ← 0	asínc.
11xx	q ← x	sínc.
101x	q ← SHR(q,x_R)	sínc.
1001	q ← SHL(q,x_L)	sínc.
1000	q ← q	sínc.

Código Verilog

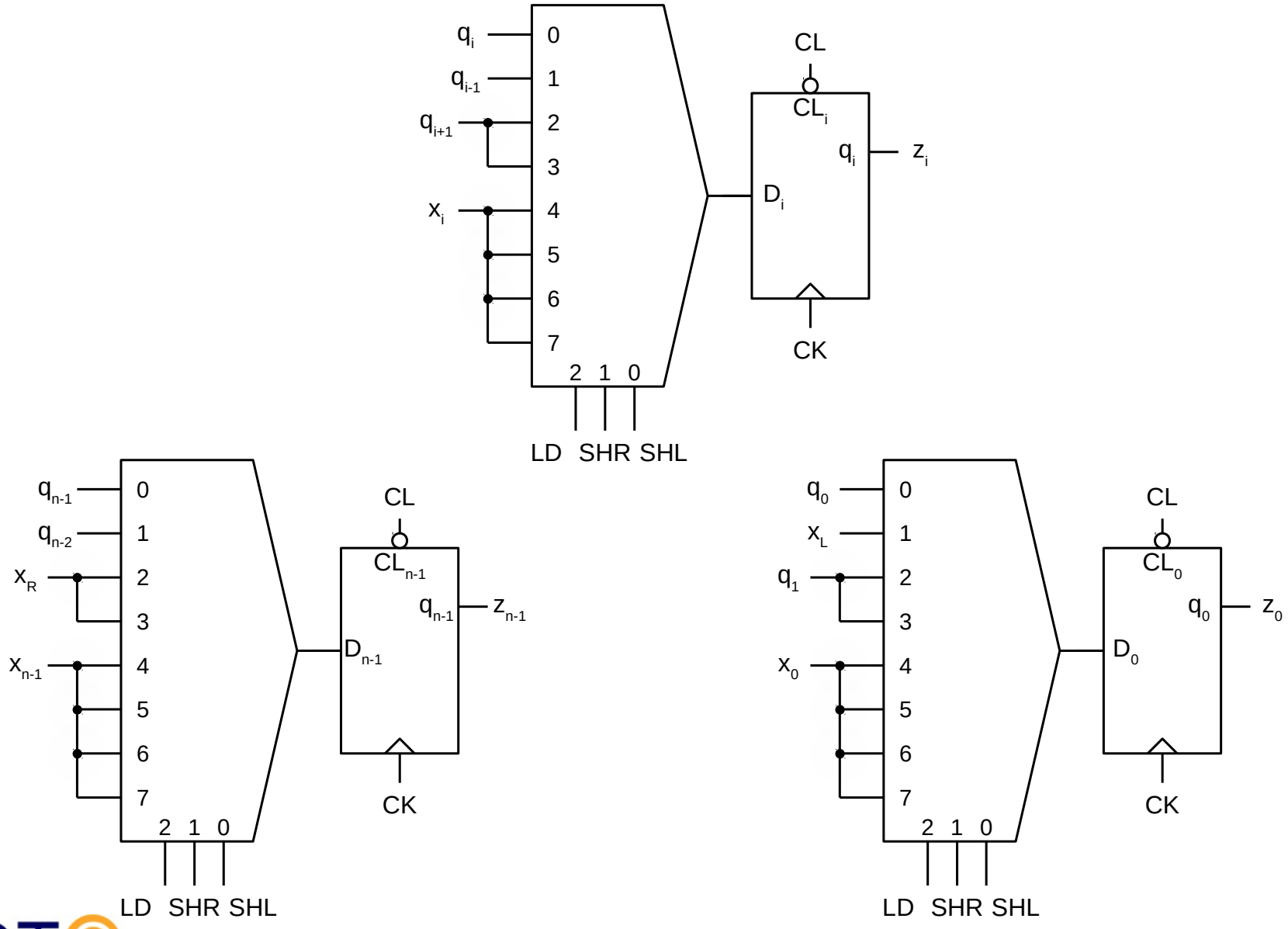
```
module ureg(
    input  wire ck,
    input  wire cl,
    input  wire ld,
    input  wire shr,
    input  wire shl,
    input  wire [3:0] x,
    output wire [3:0] z
);

    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



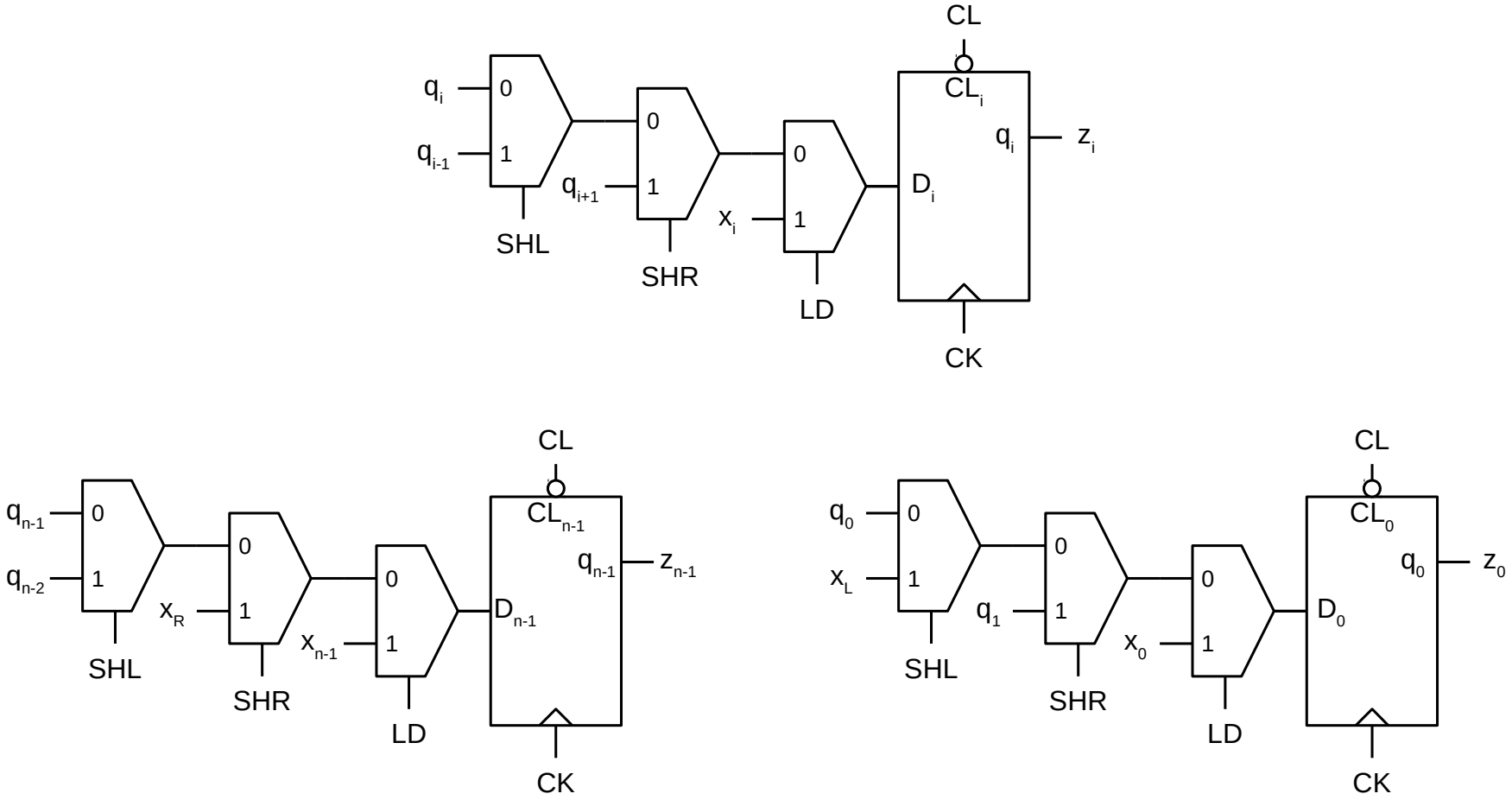
LD,SHR,SHL	Operación	Et. típica	Et. n-1	Et. 0	Exc. típ.	Exc. n-1	Exc. 0
1xx	$q \leftarrow x$	$q_i \leftarrow x_i$	$q_{n-1} \leftarrow x_{n-1}$	$q_0 \leftarrow x_0$	$D_i = x_i$	$D_{n-1} = x_{n-1}$	$D_0 = x_0$
01x	$q \leftarrow SHR(q,x_R)$	$q_i \leftarrow q_{i+1}$	$\mathbf{q_{n-1} \leftarrow x_R}$	$q_0 \leftarrow q_1$	$D_i = q_{i+1}$	$\mathbf{D_{n-1} = x_R}$	$D_0 = q_1$
001	$q \leftarrow SHL(q,x_L)$	$q_i \leftarrow q_{i-1}$	$q_{n-1} \leftarrow q_{n-2}$	$\mathbf{q_0 \leftarrow x_L}$	$D_i = q_{i-1}$	$D_{n-1} = q_{n-2}$	$\mathbf{D_0 = x_L}$
000	$q \leftarrow q$	$q_i \leftarrow q_i$	$q_{n-1} \leftarrow q_{n-1}$	$q_0 \leftarrow q_0$	$D_i = q_i$	$D_{n-1} = q_{n-1}$	$D_0 = q_0$

Registro universal



Registro universal

Implementación alternativa

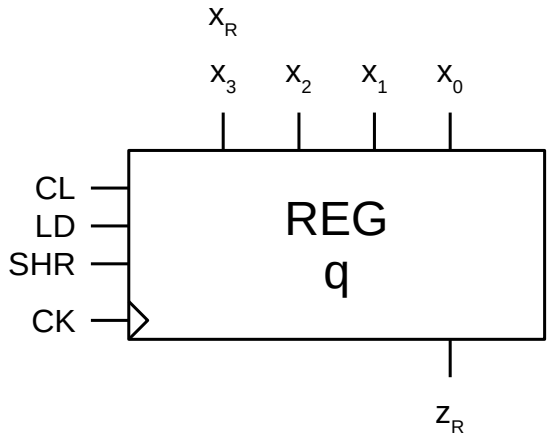


Ecuaciones de excitación (“recetas”) para hacer registros con biestables D

Descripción	Operación	Etapa	Exc. etapa
Inhibición	$q \leftarrow q$	$q_i \leftarrow q_i$	$D_i = q_i$
Carga en paralelo	$q \leftarrow x$	$q_i \leftarrow x_i$	$D_i = x_i$
Desplazamiento izquiera	$q \leftarrow SHL(x,x_L)$	$q_i \leftarrow q_{i-1}$ $q_0 \leftarrow x_L$	$D_i = q_{i-1}$ $D_0 = x_L$
Desplazamiento derecha	$q \leftarrow SHR(x,x_R)$	$q_i \leftarrow q_{i+1}$ $q_{n-1} \leftarrow x_R$	$D_i = q_{i+1}$ $D_{n-1} = x_R$
Puesta a cero síncrona	$q \leftarrow 0$	$q_i \leftarrow 0$	$D_i = 0$
Puesta a cero asíncrona	$q \leftarrow 0$	$q_i \leftarrow 0$	$CL_i = 1$

Ejercicio 2

- Diseña el siguiente registro con biestables D.
- ¡Cuidado! El clear (CL) es síncrono.



CL,LD,SHR	Operación
1xx	$q \leftarrow 0$
01x	$q \leftarrow x$
001	$q \leftarrow SHR(q,x_R)$
000	$q \leftarrow q$

Contenidos

- 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

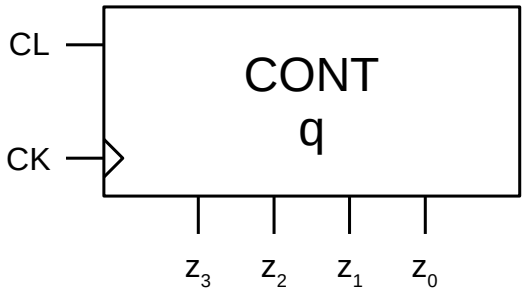
Contadores

- 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: valor actual de la cuenta
  - Fin de cuenta: indica cuenta en valor máximo

Contador binario ascendente 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 ascendente módulo 2^n



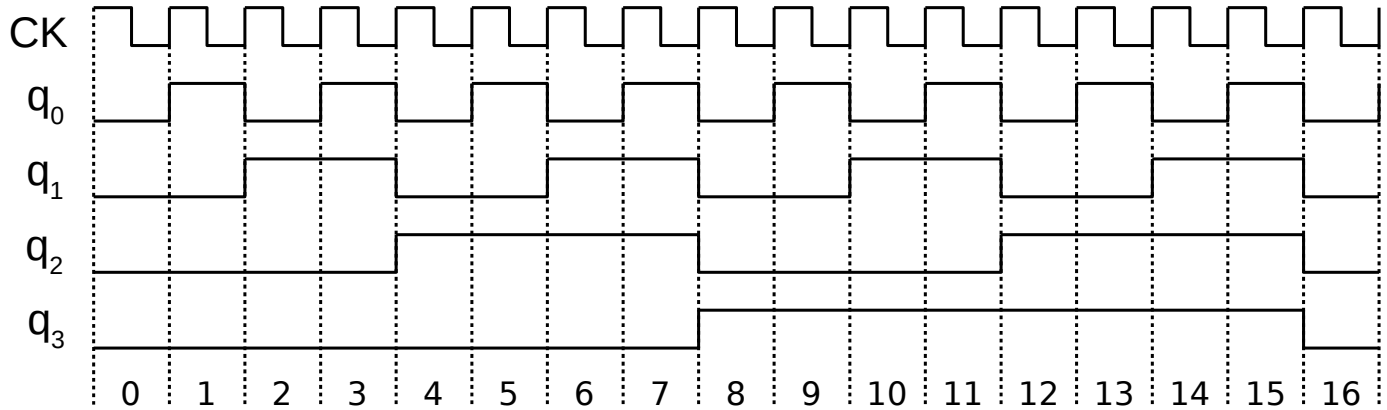
Código Verilog

```
module count_mod16(
    input  wire ck,
    input  wire cl,
    output wire [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
```

Tabla de operación

CL	Operación	Tipo
1	$q \leftarrow 0$	asínc.
0	$q \leftarrow (q+1) \bmod 2^n$	sínc.

Contador binario ascendente módulo 2^n. Operación



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

q ←

$$q_i=\begin{cases}\overline{q_i}, \text{si } q_j=1\ \forall\ j<i\\ q_i, \text{en otro caso}\end{cases}$$

$$q_i=\begin{cases}\overline{q_i}, \text{si } q_{i-1}q_{i-2}\dots q_0=1\\ q_i, \text{si } q_{i-1}q_{i-2}\dots q_0=0\end{cases}$$

$$q_i=\begin{cases}\overline{q_i}, \text{si } J_i=K_i=1\\ q_i, \text{si } J_i=K_i=0\end{cases}$$

$$q_j=1\ \forall\ j<i \Leftrightarrow q_{i-1}q_{i-2}\dots q_0=1$$

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

Operación	Etapa típica	Etapa 1	Etapa 0
q ← (q+1) mod 2^n	$q_i \leftarrow \overline{q_i}$ si $q_{i-1} \dots q_0=1$, q_i si $q_{i-1} \dots q_0=0$	$q_1 \leftarrow \overline{q_1}$ si $q_0=1$, q_1 si $q_0=0$	$q_0 \leftarrow \overline{q_0}$

Operación	Exc. etapa típica	Exc. etapa 0	Exc. etapa 0
q ← (q+1) mod 2^n	$J_i = K_i = q_{i-1} \dots q_0$	$J_1 = K_1 = q_0$	$J_0 = K_0 = 1$

Contador binario ascendente módulo 2^n

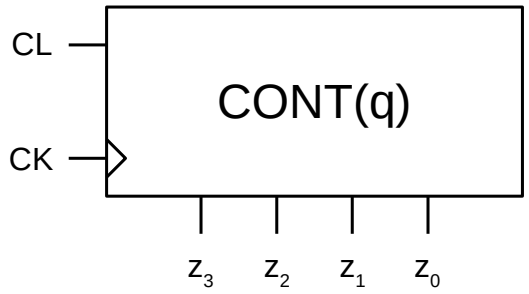
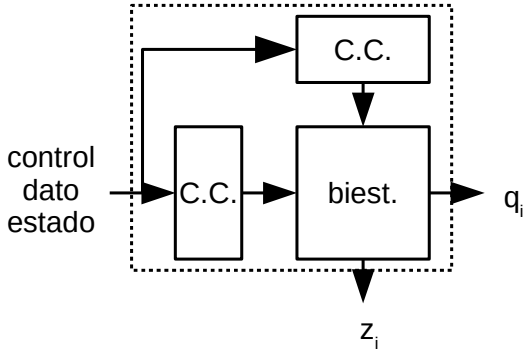
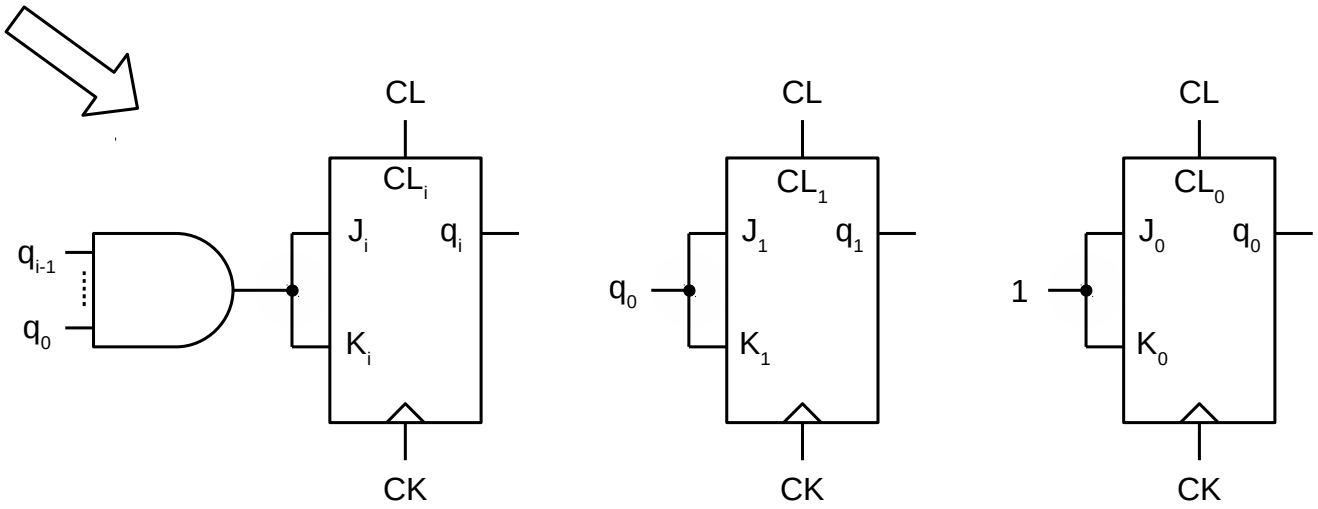


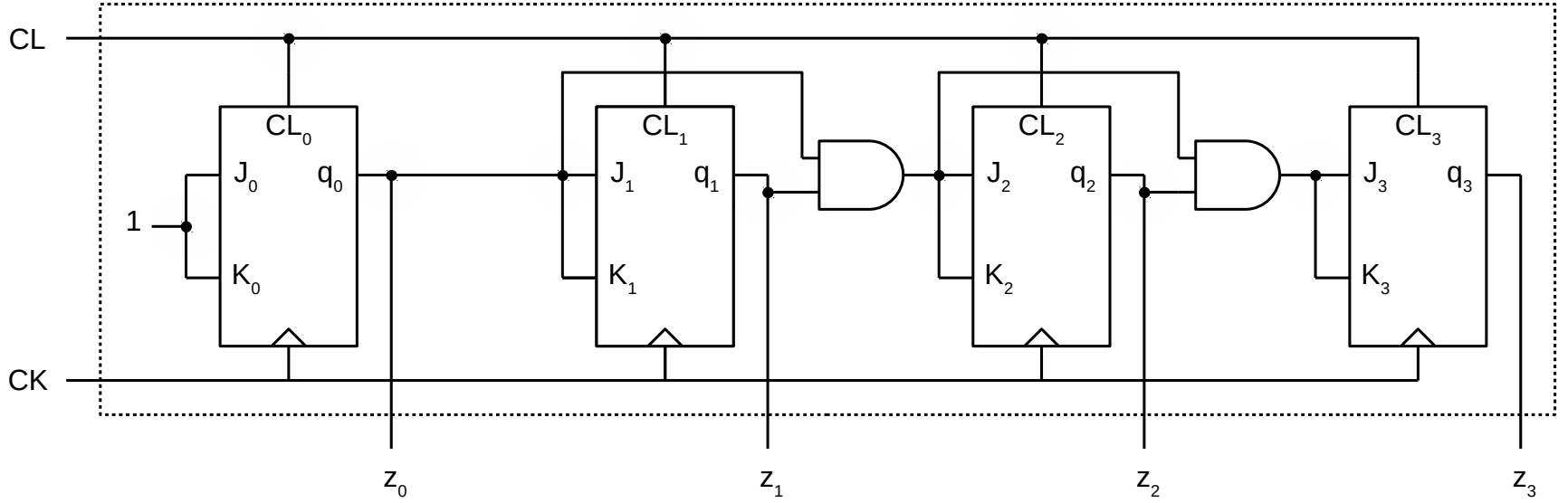
Tabla de operación síncrona

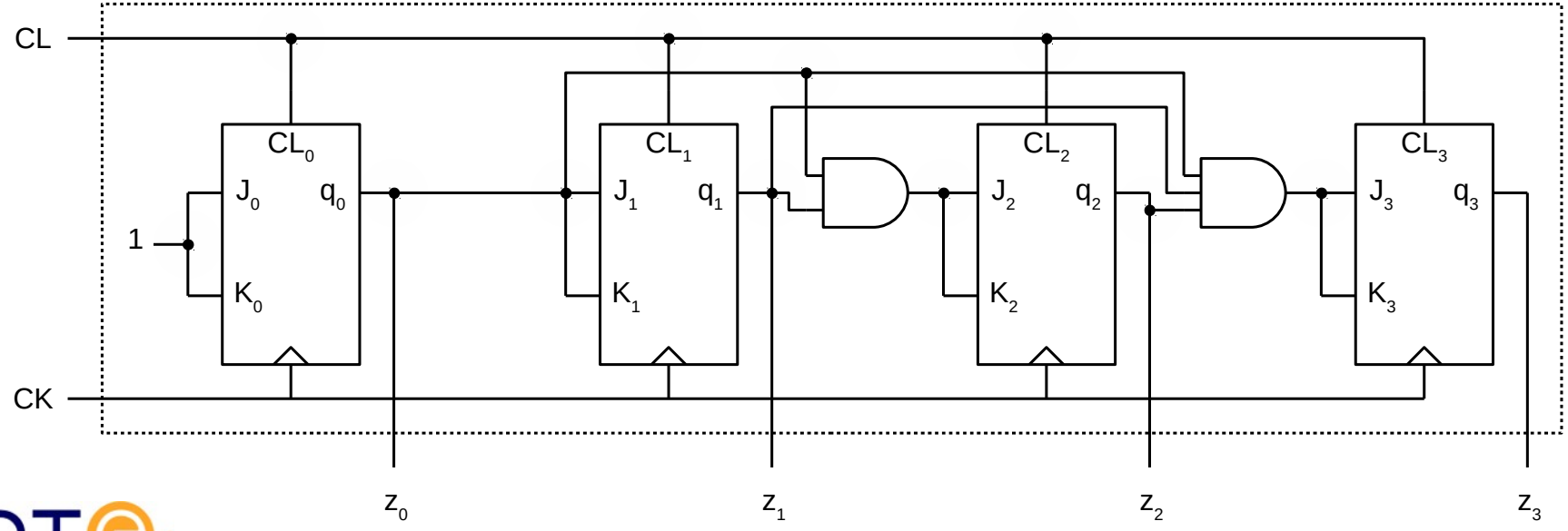
Operación	Exc. et. típica	Exc. et. 0	Exc. et. 0
$q \leftarrow (q+1) \bmod 2^n$	$J_i=K_i=q_{i-1} \dots q_0$	$J_1=K_1=q_0$	$J_0=K_0=1$



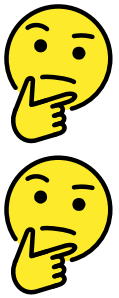


Contador binario ascendente módulo 2^n. Circuito





Alternativas de diseño. Ejercicio 3



- El biestable JK suele ser el más adecuado para diseñar contadores porque:
  - Es fácil implementar la operación de cuenta: $J_i = K_i = q_0 \ldots q_{i-1}$
  - Facilita implementar otras operaciones
- 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 ascendente módulo 2^n con “clear” y “enable”

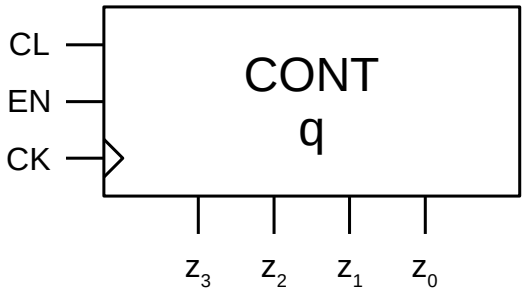


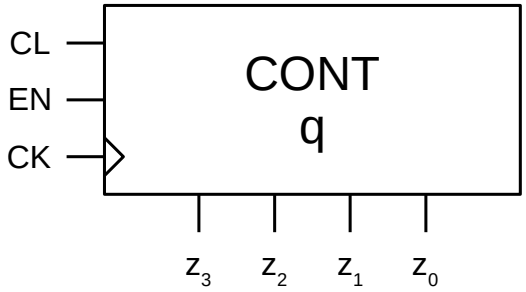
Tabla de operación

CL, EN	Operación	Tipo
1x	$q \leftarrow 0$	sínc.
01	$q \leftarrow (q+1) \bmod 2^n$	sínc.
00	$q \leftarrow q$	sínc.

Código Verilog

```
module count_mod16(
    input  wire ck,
    input  wire cl,
    input  wire en,
    output wire [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 ascendente módulo 2^n con "clear" y "enable"



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

q ←

Descripción	Operación	Etapa	Exc. etapa
Inhibición	$q \leftarrow q$	$q_i \leftarrow q_i$	$J_i=K_i=0$
Puesta a cero síncrona	$q \leftarrow 0$	$q_i \leftarrow 0$	$J_i=0, K_i=1$

Contador binario ascendente módulo 2^n con "clear" y "enable"

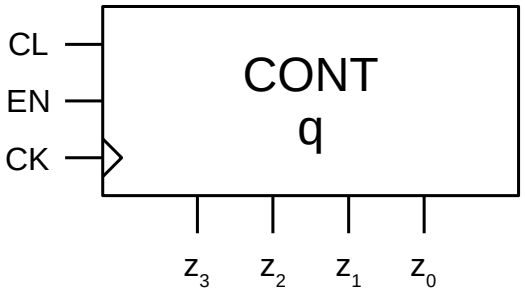
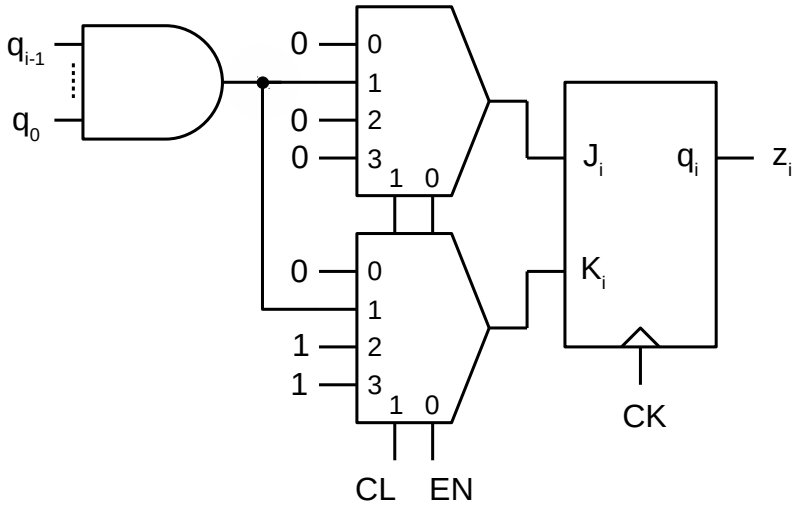
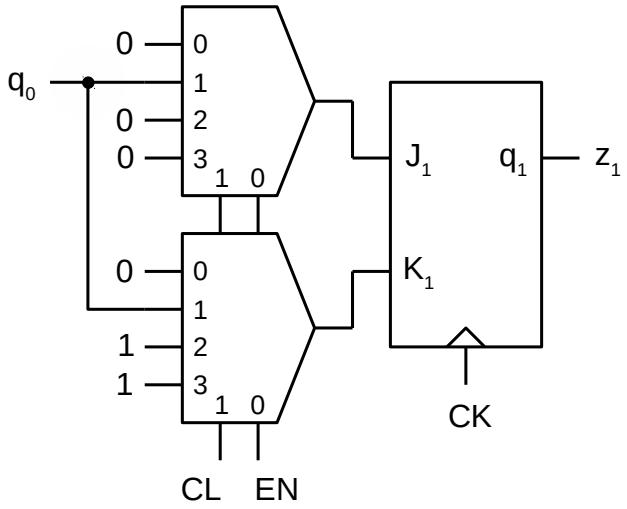


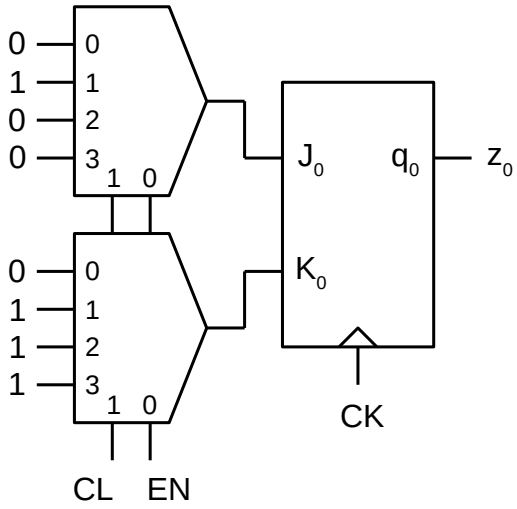
Tabla de operación síncrona

CL,EN	Operación	Etapa típica	Etapa 1	Etapa 0
1x	$q \leftarrow 0$	$J_i=0, K_i=1$	$J_1=0, K_1=1$	$J_0=0, K_0=1$
01	$q \leftarrow (q+1) \bmod 2^n$	$J_i=K_i=q_{i-1} \ldots q_0$	$\mathbf{J_1=K_1=q_0}$	$\mathbf{J_0=K_0=1}$
00	$q \leftarrow q$	$J_i=K_i=0$	$J_1=K_1=0$	$J_0=K_0=0$

Implementación con MUX 4:1

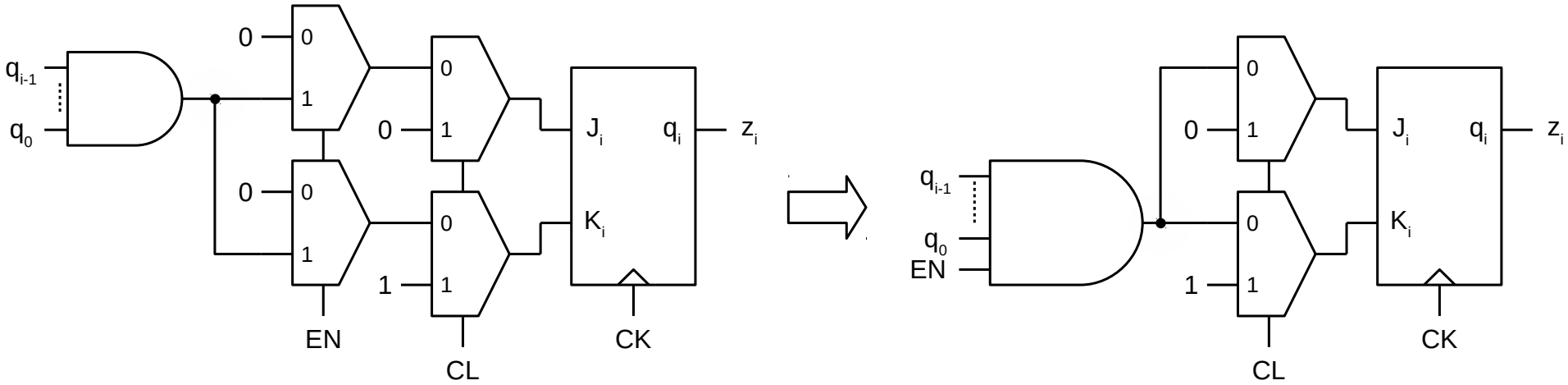


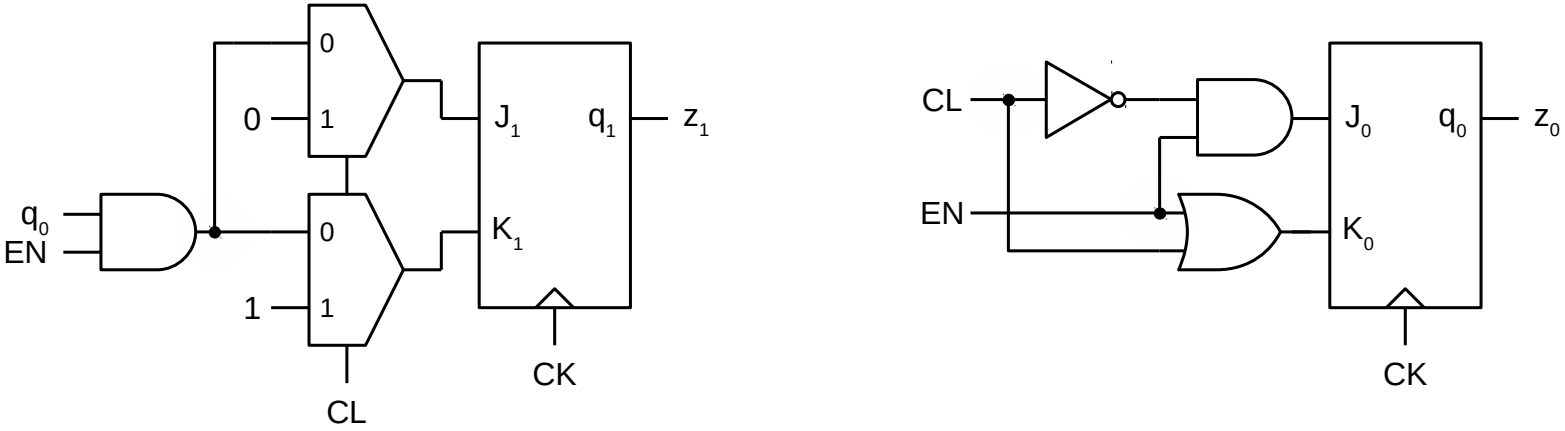




Contador binario ascendente módulo 2^n con "clear" y "enable"

Implementaciones alternativas

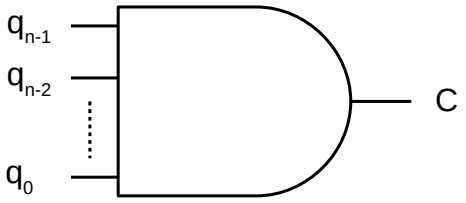




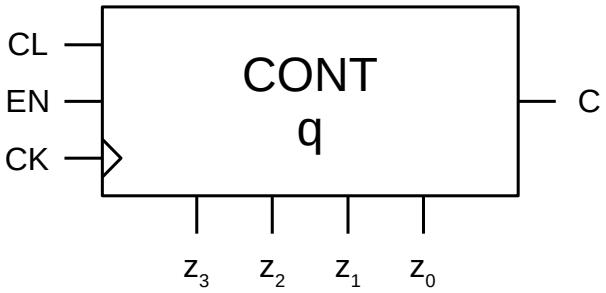
Salida de fin de cuenta

- Cuenta ascendente (acarreo – carry)
  - C = 1 si y sólo si q = $2^n - 1$

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



Salida de fin de cuenta



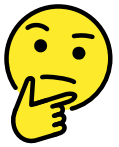
Código Verilog

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

Combinación de contadores

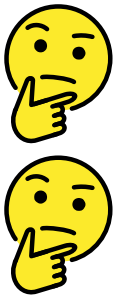
- Objetivo: combinar contadores para obtener un contador de módulo mayor.
  - Módulo k + módulo l → módulo k*l
- Diseño:
  - Incrementar el contador más significativo cuando el menos significativo alcanza el valor máximo.
  - Diseño más simple con entradas/salidas apropiadas
    - Fin de cuenta: último estado
    - Enable: habilitación

Ejercicio 4.1
Combinación de contadores



- Queremos:
  - Contador módulo 256 (8 bits) con CL (clear)
- Tenemos:
  - Contador módulo 16 (4 bits) con CL (clear), EN (enable) y C (acarreo)
- Análisis del problema:
  - Necesitamos dos contadores MOD16: q0 y q1
    - z[7:4] = q1[3:0], z[3:0] = q0[3:0]
  - Los dos contadores cambian de estado a la vez:
    - CK0 = CK1 = CK
  - Los dos contadores se ponen a cero a la vez:
    - CL0 = CL1 = CL
  - q1 se incrementa sólo cuando q0 = 15:
    - EN1 = C0

Ejercicio 4.2
Combinación de contadores



- Queremos:
  - Contador módulo 256 (8 bits) con CL (clear), EN (enable) y C (acarreo)
- Tenemos:
  - Contador módulo 16 (4 bits) con CL (clear), EN (enable) y C (acarreo)
- Análisis del problema:
  - Necesitamos dos contadores MOD16: q0 y q1
    - z[7:4] = q1[3:0], z[3:0] = q0[3:0]
  - Los dos contadores cambian de estado a la vez:
    - CK0 = CK1 = CK
  - Los dos contadores se ponen a cero a la vez:
    - CL0 = CL1 = CL
  - q1 se incrementa sólo cuando EN=1 y q0 = 15:
    - EN1 = EN C0
  - C se activa cuando q=255, esto es, q0=15 y q1=15:
    - C = C0 C1

Contador binario módulo 2^n con "load" y "enable"

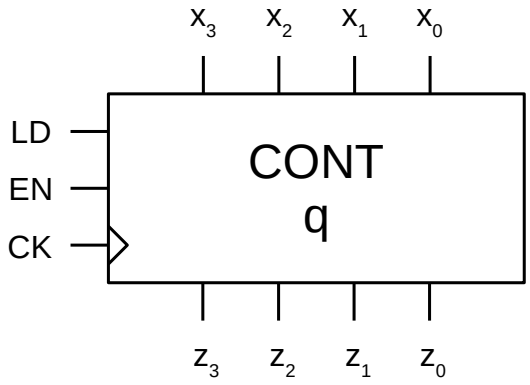


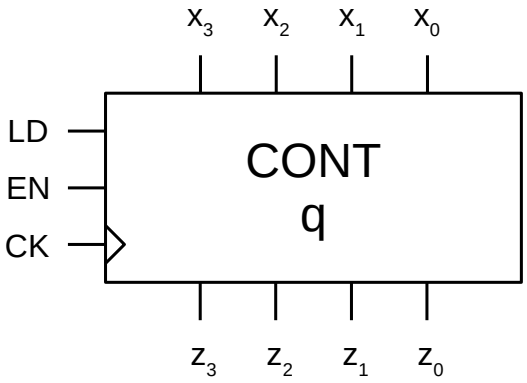
Tabla de operación

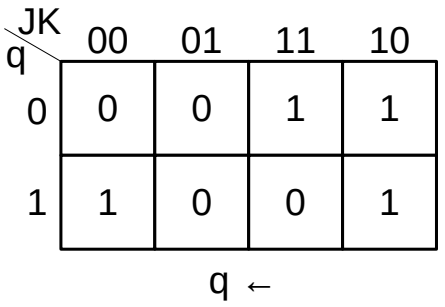
LD, EN	Operación	Tipo
1x	$q \leftarrow x$	sínc.
01	$q \leftarrow (q+1) \bmod 2^n$	sínc.
00	$q \leftarrow q$	sínc.

Código Verilog

```
module count_mod16(
    input  wire ck,
    input  wire ld,
    input  wire en,
    input  wire [3:0] x,
    output wire [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
```

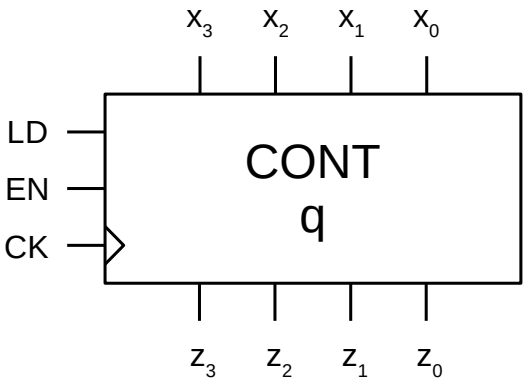
Contador binario módulo 2^n con "load" y "enable"



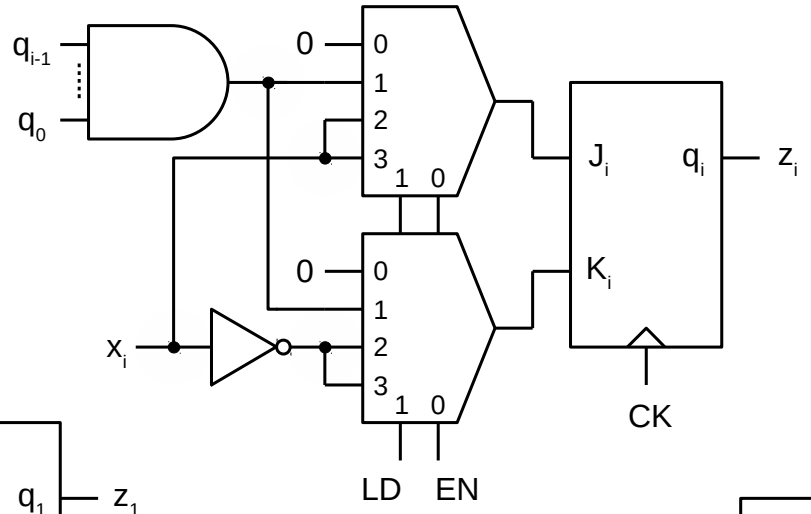


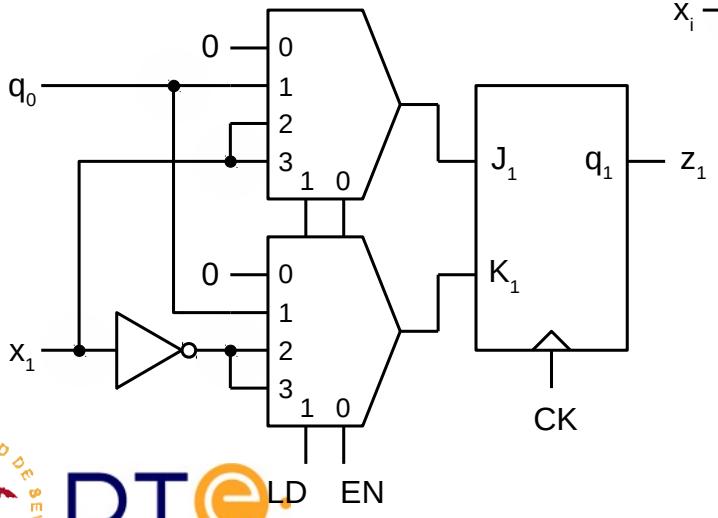
Descripción	Operación	Etapa	Exc. etapa	Exc. etapa
Carga en paralelo	$q \leftarrow x$	$q_i \leftarrow x_i$	$J_i=1, K_i=0$ si $x=1$ $J_i=0, K_i=1$ si $x=0$	$J_i = x_i$ $K_i = \overline{x_i}$

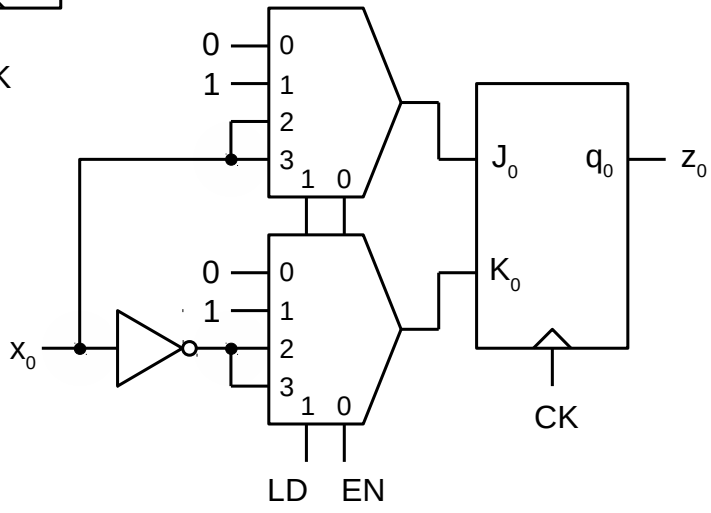
Contador binario módulo 2^n con "load" y "enable"



LD,EN	Operación	Etapa típica	Etapa 1	Etapa 0
1x	$q \leftarrow x$	$J_i=x_i$, $K_i=\overline{x_i}$	$J_1=x_1$, $K_1=\overline{x_1}$	$J_0=x_0$, $K_0=\overline{x_0}$
01	$q \leftarrow (q+1) \bmod 16$	$J_i=K_i=q_{i-1} \ldots q_0$	$\mathbf{J_1=K_1=q_0}$	$\mathbf{J_0=K_0=1}$
00	$q \leftarrow q$	$J_i=K_i=0$	$J_1=K_1=0$	$J_0=K_0=0$



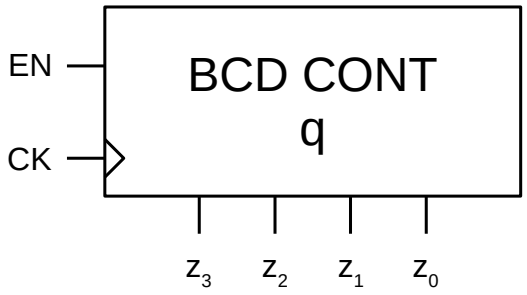




Límite del la cuenta

- Sirve para diseñar contadores de módulo < 2^n a partir de contadores módulo 2^n.
- Casos:
  - Límite superior: 0 ... l, l<2^n
  - Límite inferior: k ... 2^n, k > 0
  - Límites superior e inferior: k ... l, k>0, l<2^n
- Estrategia:
  - Emplear contadores con entradas útiles específicas:
    - LD: carga de estado de cuenta
    - CL: puesta a cero
  - Detectar el estado de cuenta y activar la operación adecuada para volver a cero (CL) o a un valor inicial k (LD)
  - Mejor si las señales de control son síncronas: evita salidas transitorias.

Límite del la cuenta
Contador BCD



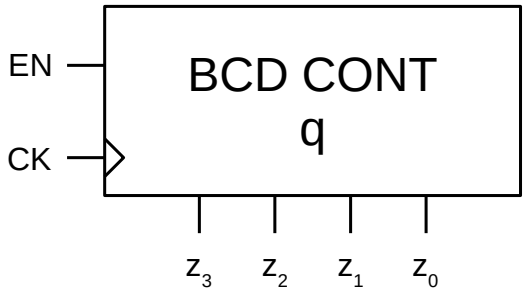
Código Verilog

```
module count_mod10(
    input  wire ck,
    input  wire cl,
    input  wire en,
    output wire [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

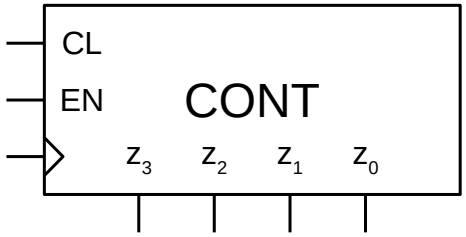
CL, EN	Operación	Tipo
1x	q ← 0	sínc.
01	q ← (q+1) mod 10	sínc.
00	q ← q	sínc.

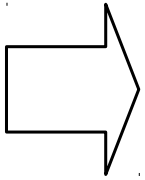
Límite del la cuenta
Contador BCD

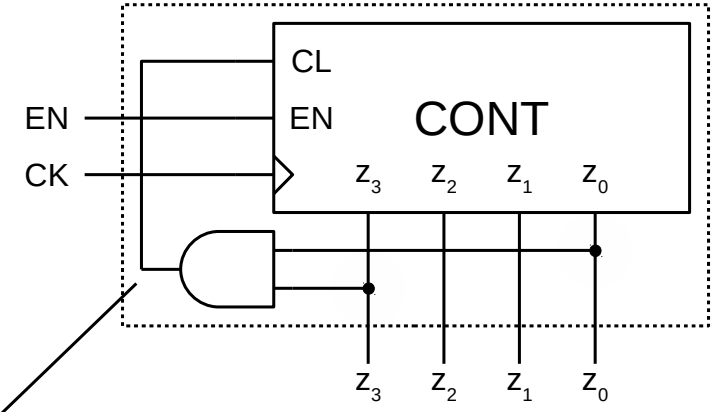


z_1z_0 \ z_3z_2	00	01	11	10
00	0	0	-	0
01	0	0	-	1
11	0	0	-	-
10	0	0	-	-

$CL = z_3\, z_0$

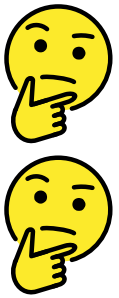






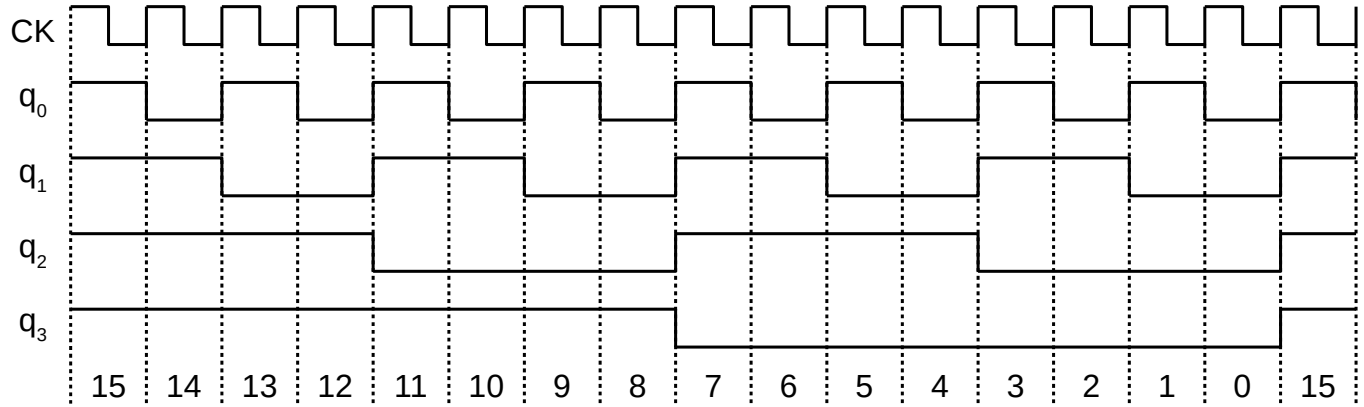
Puede usarse como señal fin de cuenta

Ejercicio 5. Contador de 3 a 12



- Diseñar un contador cíclico de 3 a 12 con las funciones:
  - Puesta a cero (CL)
  - 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).
- Propón otra solución empleando un contador BCD y circuitos aritméticos.

Contador binario descendente módulo 2^n. Operación



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

q ←

$$q_i = \begin{cases} \overline{q_i}, \text{si } q_j = 0 \; \forall \; j<i \\ q_i, \text{en otro caso} \end{cases} \longrightarrow q_i = \begin{cases} \overline{q_i}, \text{si } \overline{q_{i-1}}\,\overline{q_{i-2}}\ldots\overline{q_0} = 1 \\ q_i, \text{si } \overline{q_{i-1}}\,\overline{q_{i-2}}\ldots\overline{q_0} = 0 \end{cases} \qquad q_i = \begin{cases} \overline{q_i}, \text{si } J_i = K_i = 1 \\ q_i, \text{si } J_i = K_i = 0 \end{cases}$$

$$q_j = 1 \; \forall \; j<i \Leftrightarrow \overline{q_{i-1}}\,\overline{q_{i-2}}\ldots\overline{q_0} = 1 \qquad J_i = K_i = \overline{q_{i-1}}\ldots\overline{q_0}$$

Operación	Etapa típica	Etapa 1	Etapa 0
q ← (q-1) mod 2^n	$q_i \leftarrow \overline{q_i}$ si $\overline{q_{i-1}} \ldots \overline{q_0}=1$, q_i si $\overline{q_{i-1}} \ldots \overline{q_0}=0$	$q_1 \leftarrow \overline{q_1}$ si $\overline{q_0}=1$, q_1 si $\overline{q_0}=0$	$q_0 \leftarrow \overline{q_0}$

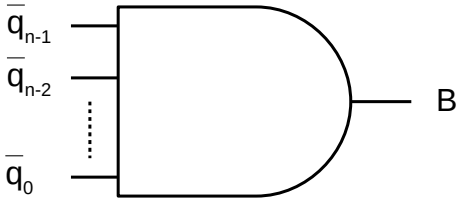
Operación	Exc. etapa típica	Exc. etapa 0	Exc. etapa 0
q ← (q-1) mod 2^n	$J_i = K_i = \overline{q_{i-1}} \ldots \overline{q_0}$	$J_1 = K_1 = \overline{q_0}$	$J_0 = K_0 = 1$

Salida de fin de cuenta (borrow)

- Cuenta descendente (borrow)
  - B = 1 si y sólo si 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 descendente módulo 2^n con "clear", "enable" y "borrow"

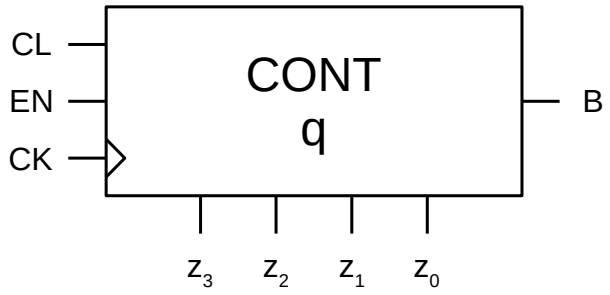


Tabla de operación

CL, EN	Operación	Tipo
1x	q ← 0	sínc.
01	q ← (q-1) mod 16	sínc.
00	q ← q	sínc.

Código Verilog

```
module count_mod16(
    input  wire ck,
    input  wire cl,
    input  wire en,
    output wire [3:0] z,
    output wire 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 descendente módulo 2^n con "clear", "enable" y "borrow"

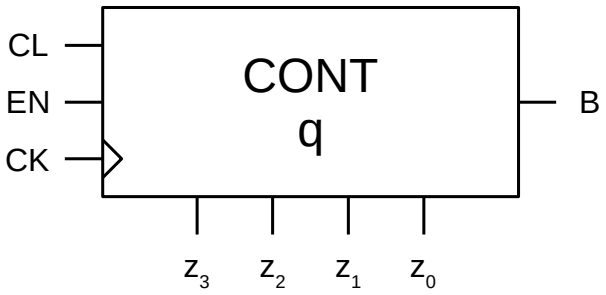
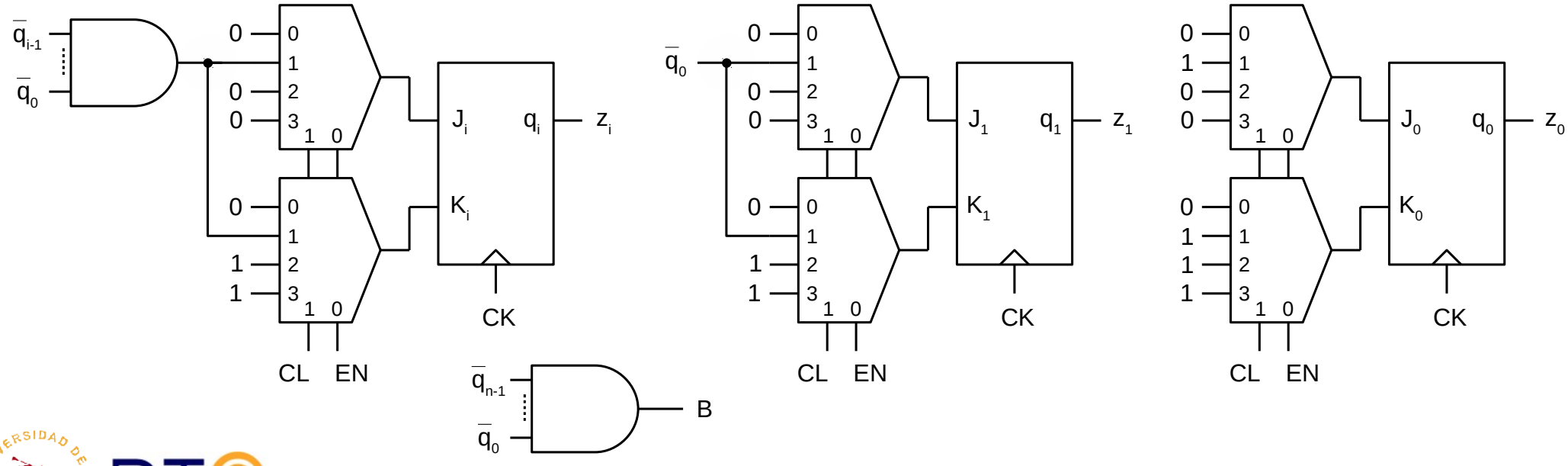


Tabla de operación síncrona

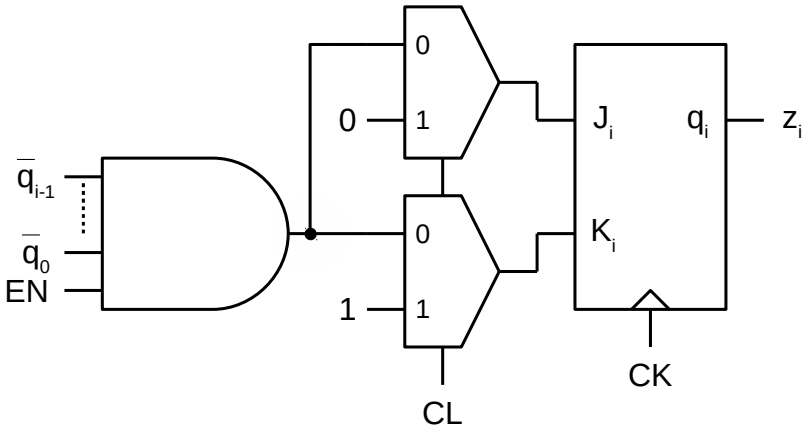
CL,EN	Operación	Etapa típica	Etapa 1	Etapa 0
1x	$q \leftarrow 0$	$J_i=0, K_i=1$	$J_1=0, K_1=0$	$J_0=0, K_0=1$
01	$q \leftarrow (q+1) \bmod 16$	$J_i=K_i=\bar{q}_{i-1} \dots \bar{q}_0$	$\mathbf{J_1=K_1=\bar{q}_0}$	$\mathbf{J_0=K_0=1}$
00	$q \leftarrow q$	$J_i=K_i=0$	$J_1=K_1=0$	$J_0=K_0=0$

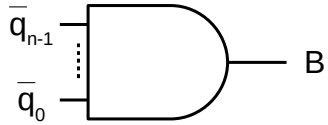
Implementación con MUX 4:1

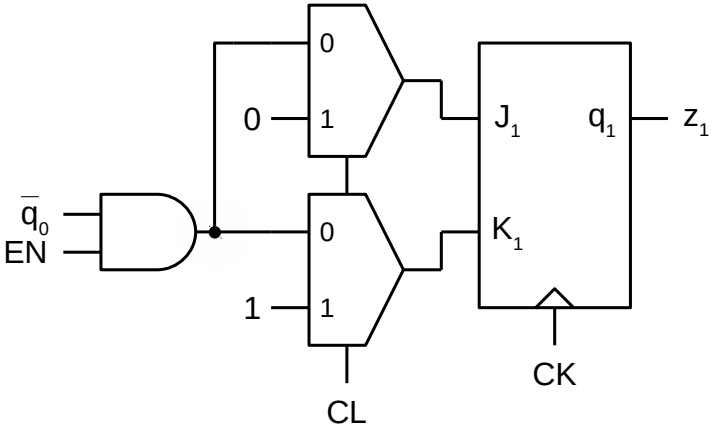


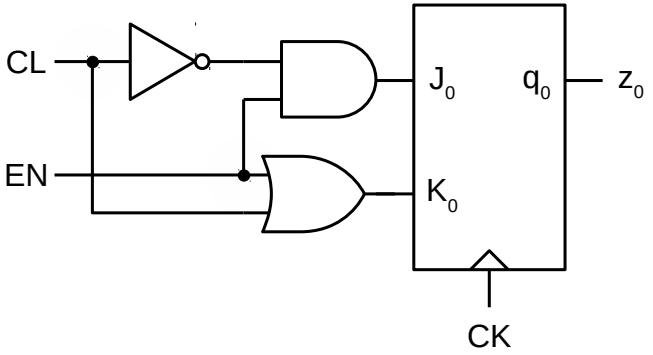
Contador descendente módulo 2^n con "clear", "enable" y "borrow"

Implementación alternativa





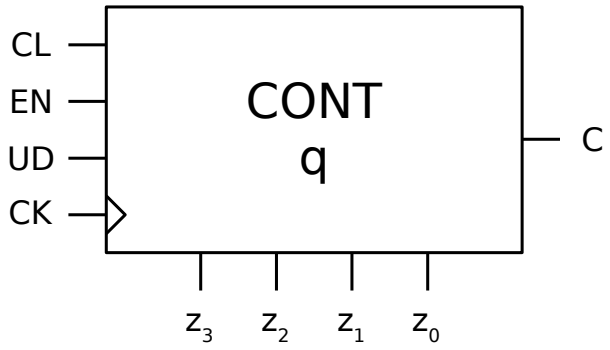




Ecuaciones de excitación (“recetas”) para hacer contadores con biestables JK

Descripción	Operación	Etapa	Exc. etapa
Inhibición	$q \leftarrow q$	$q_i \leftarrow q_i$	$J_i = K_i = 0$
Carga en paralelo	$q \leftarrow x$	$q_i \leftarrow x_i$	$J_i = x_i;\ K_i = \overline{x}_i$
Cuenta ascendente	$q \leftarrow (q+1) \bmod 2^n$	$q_i \leftarrow \overline{q}_i$ sii $q_{i-1} \dots q_0 = 1$ $q_1 \leftarrow \overline{q}_1$ sii $q_0 = 1$ $q_0 = \overline{q}_0$	$J_i = K_i = q_{i-1} \dots q_0$ $J_1 = K_1 = q_0$ $J_0 = K_0 = 1$
Cuenta descendente	$q \leftarrow (q+1) \bmod 2^n$	$q_i \leftarrow \overline{q}_i$ sii $\overline{q}_{i-1} \dots \overline{q}_0 = 1$ $q_1 \leftarrow \overline{q}_1$ sii $\overline{q}_0 = 1$ $q_0 = \overline{q}_0$	$J_i = K_i = \overline{q}_{i-1} \dots \overline{q}_0$ $J_1 = K_1 = \overline{q}_0$ $J_0 = K_0 = 1$
Puesta a cero síncrona	$q \leftarrow 0$	$q_i \leftarrow 0$	$J_i = 0;\ K_i = 1$
Puesta a cero asíncrona	$q \leftarrow 0$	$q_i \leftarrow 0$	$CL_i = 1$

Contador reversible



CL, EN, UD	Operación	Tipo
1xx	q ← 0	asínc.
00x	q ← q	sínc.
010	q ← (q+1) mod 16	sínc.
011	q ← (q-1) mod 16	sínc.

Contador reversible en Verilog

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

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

    assign z = q;

    always @*
        if (ud == 1)
            c = ~(|q);
        else
            c = &q;
endmodule
```

```
module rev_counter2(
    input wire ck,
    input wire cl,
    input en,
    input ud,
    output [3:0] z,
    output wire c
);
    reg [3:0] q;

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

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

endmodule
```

Contador reversible

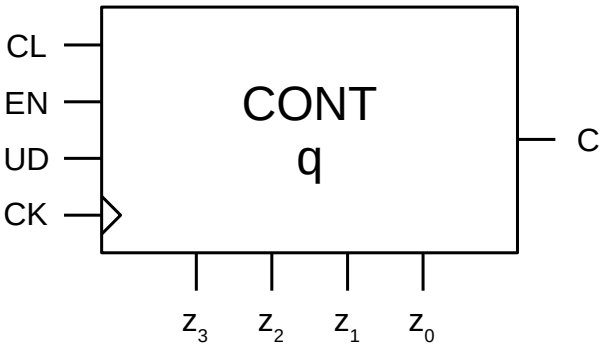
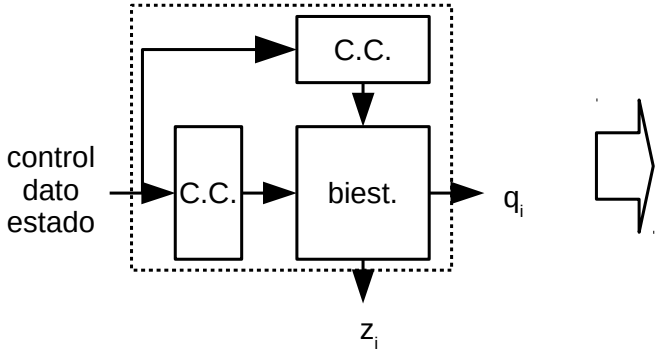
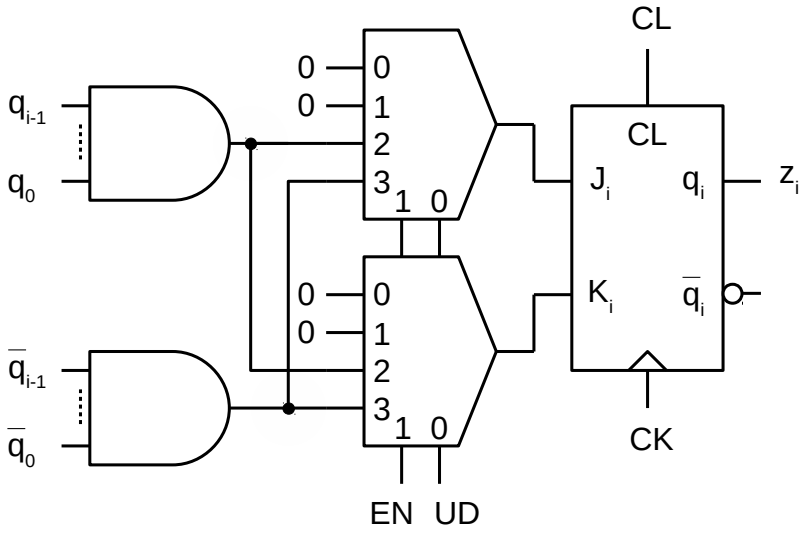
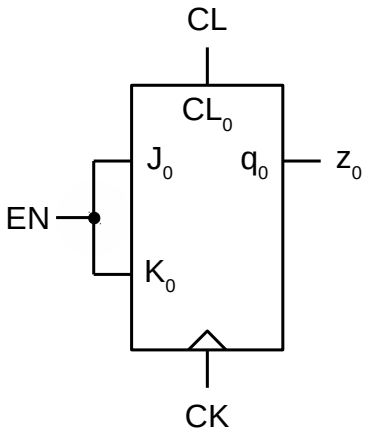


Tabla de operación síncrona

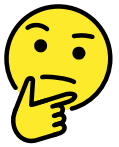
EN, UD	Operación	Etapa típica	Etapa 0
0x	$q \leftarrow q$	$J_i=K_i=0$	$J_0=K_0=0$
10	$q \leftarrow (q+1) \bmod 16$	$J_i=K_i=q_{i-1} \dots q_0$	**$J_0=K_0=1$**
11	$q \leftarrow (q-1) \bmod 16$	$J_i=K_i=\overline{q}_{i-1} \dots \overline{q}_0$	**$J_0=K_0=1$**



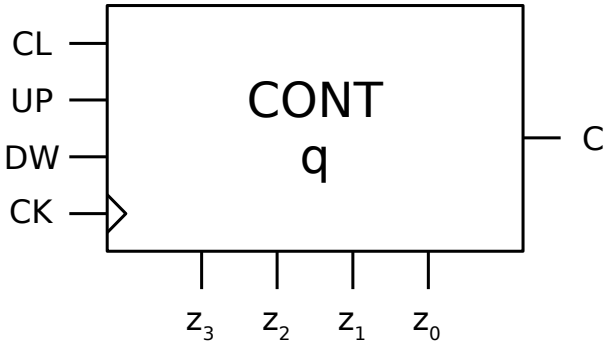




Ejercicio 6. Contador reversible



- Diseñar un contador reversible con las siguientes funciones

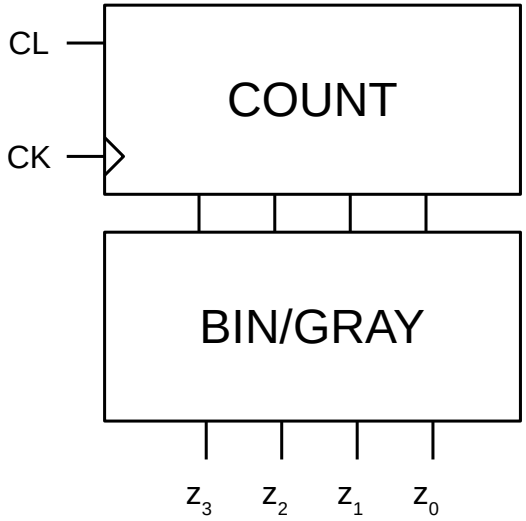


CL, UP, DW	Operación	Tipo
1xx	$q \leftarrow 0$	sínc.
01x	$q \leftarrow (q+1) \bmod 16$	sínc.
001	$q \leftarrow (q-1) \bmod 16$	sínc.
000	$q \leftarrow q$	sínc.

Contadores no binarios

- Secuencia no binaria (ej. Gray)
  - Nativos
  - A partir de contador binario y convertidor de código
- Contadores de desplazamiento
  - Contador en anillo

Contador Gray con convertidor de código

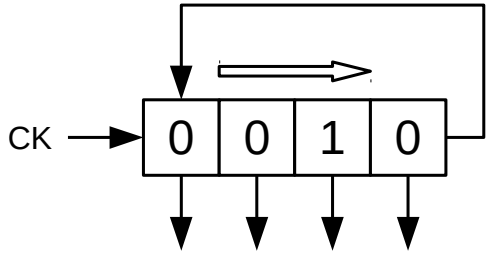


z_3 z_2 z_1 z_0
0000
0001
0011
0010
0110
0111
0101
0100
1100
1101
1111
1110
1010
1011
1001
1000

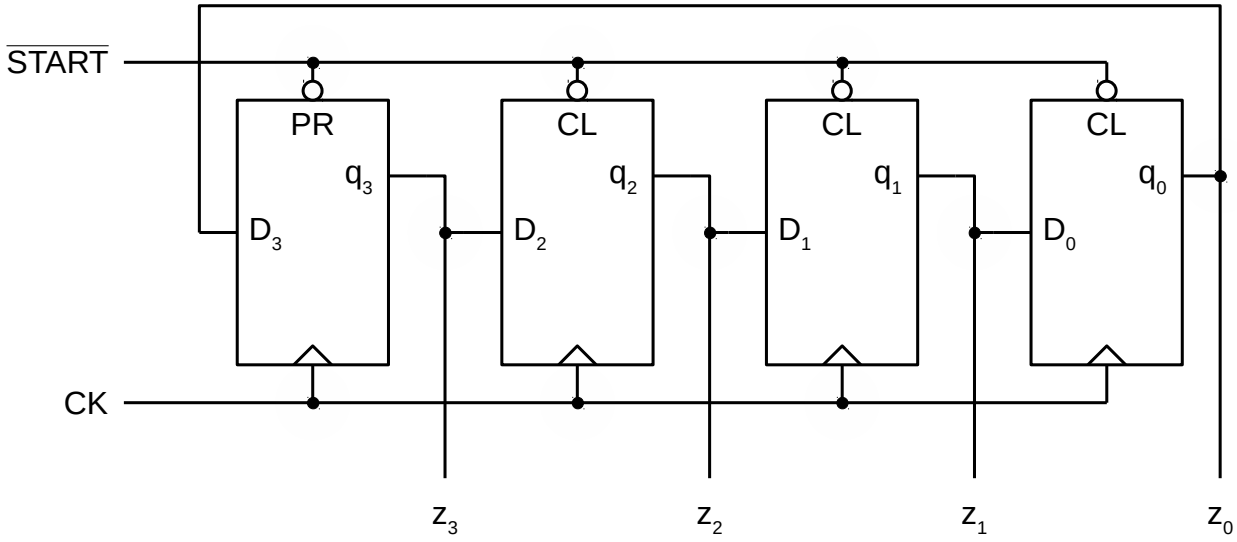
Código Verilog

```
module graycounter_mod16(
    input  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
```

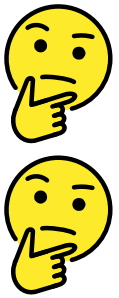
Contador en anillo



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



Ejercicio 7



- 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.

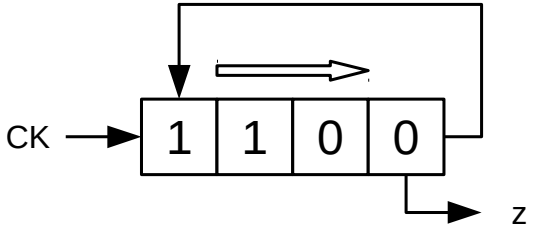
START,EN	Operación	Tipo
1x	$q \leftarrow 0x01$	sínc.
01	$q \leftarrow SHL(q,q_7)$	sínc.
00	$q \leftarrow q$	sínc.

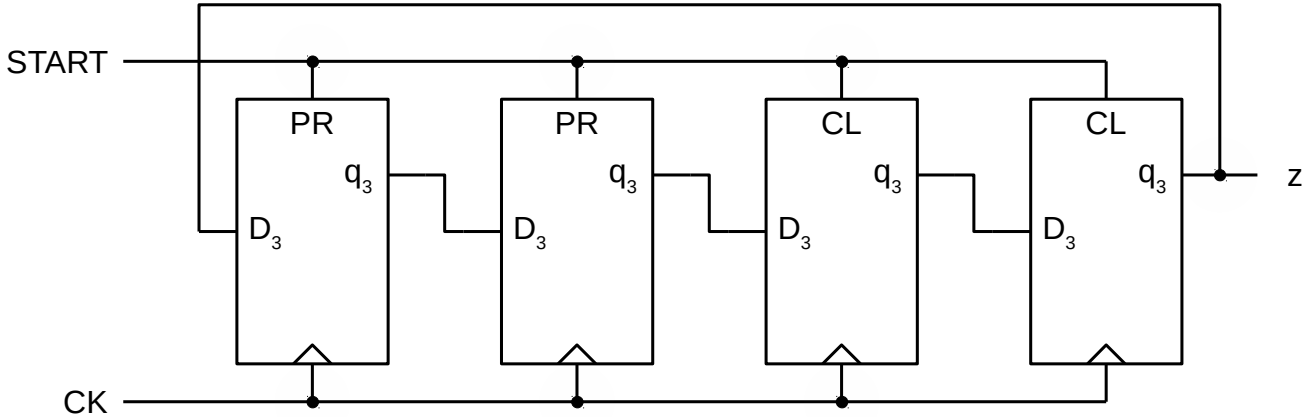
Contenido

- Introducción
- Registros
- Contadores
- Diseño con subsistemas secuenciales
  - Generadores y detectores de secuencia
  - Generador de pulsos
  - Ejemplos

Generador de secuencia

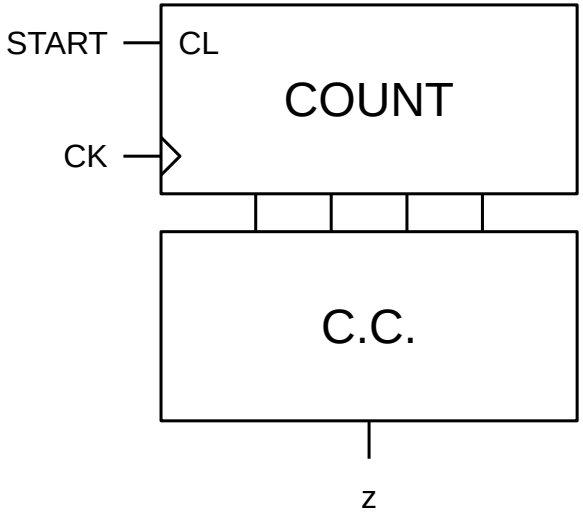
- Generador de secuencia “0011” con registro de desplazamiento.

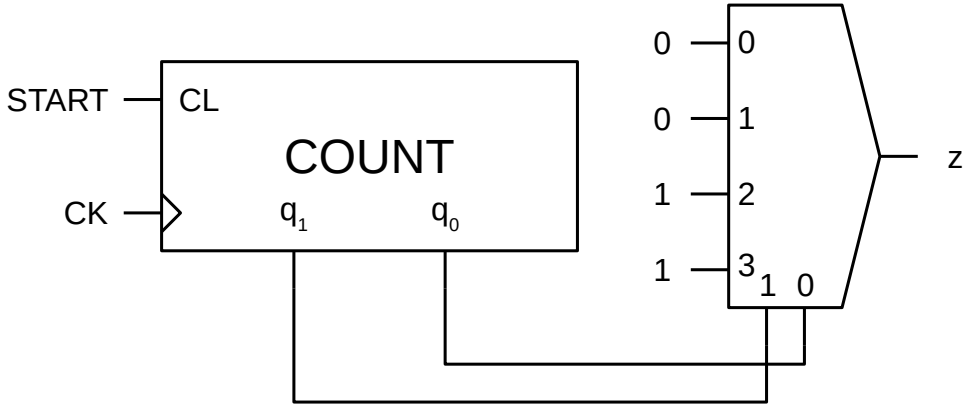




Generador de secuencia

- Con contador y C.C. (ej. MUX)





Código Verilog

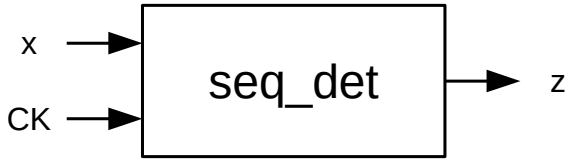
```
module seq_gen(
    input  wire ck,
    input  wire start,
    output wire z
);

    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
```

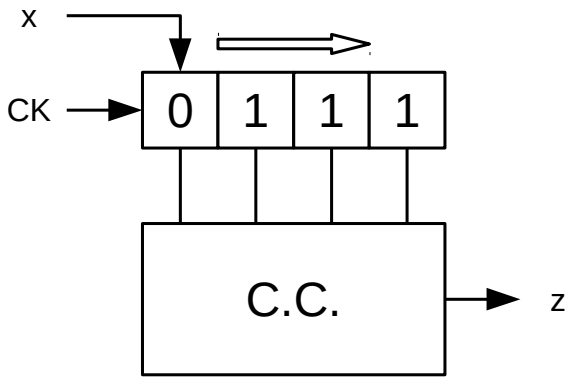
Ejercicio 8

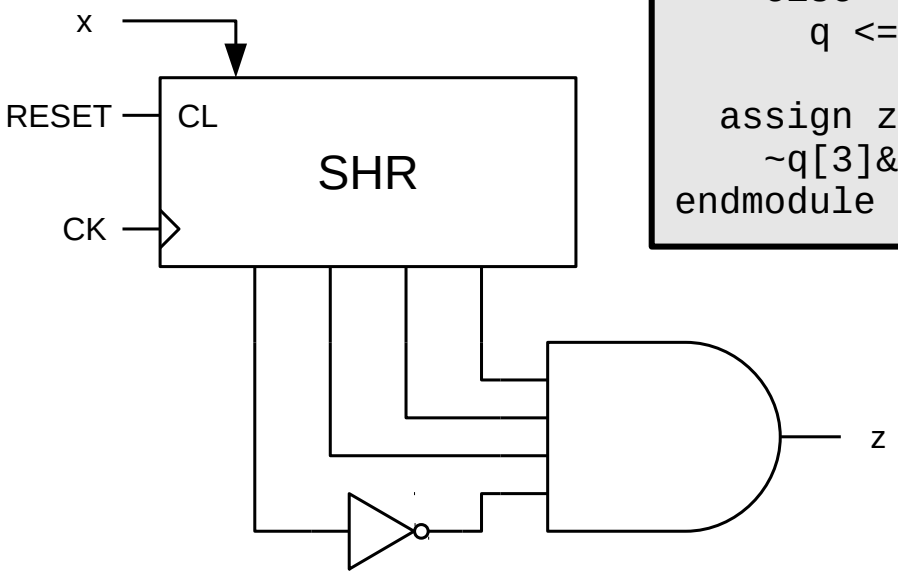
- Diseña un generador de la secuencia “101001” con una entrada:
  - START: regresa al valor inicial.
- Emplea un registro universal de 8 bits como el visto en el tema.

Detector de secuencia

- Detector de secuencia “1110” (3 o más unos consecutivos) con registro de desplazamiento y C.C.

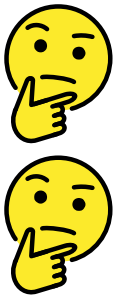






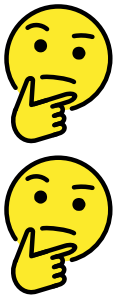

```
module seq_det(
  input ck, reset, x,
  output z
);
  reg [3:0] q;

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

  assign z =
    ~q[3]&q[2]&q[1]&q[0];
endmodule
```

Ejercicio 9



- Diseña un detector de la secuencia 101001 con entrada de puesta a cero síncrona (reset).
- Tenemos un registro de desplazamiento con señales:
  - CL: puesta a cero síncrona
  - SHL: desplazamiento a la izquierda
  - x_L: entrada serie
  - z: salida en paralelo
- a) Empleando el registro y puertas lógicas
- b) Empleando el registro y un comparador

Ejercicio 10. Generador de pulsos



- En muchas ocasiones es necesario activar un sistema o realizar una acción cada cierto tiempo con una frecuencia menor que la frecuencia de reloj.
- Para ello se puede usar un contador del módulo adecuado para que genere una señal de fin de cuenta a intervalos regulares.
- La señal de fin de cuenta se usará como señal de “habilitación” de la acción deseada.
- El generador de pulsos es una forma de “divisor de frecuencia” o “*prescaler*”.
- Ejercicio:
  - Un sistema posee un reloj del sistema de 1MHz de frecuencia. Se desea generar una señal de habilitación que se active durante un periodo de reloj cada 20ms.
  - Diseñe un circuito que genere dicha señal basándose en un contador.

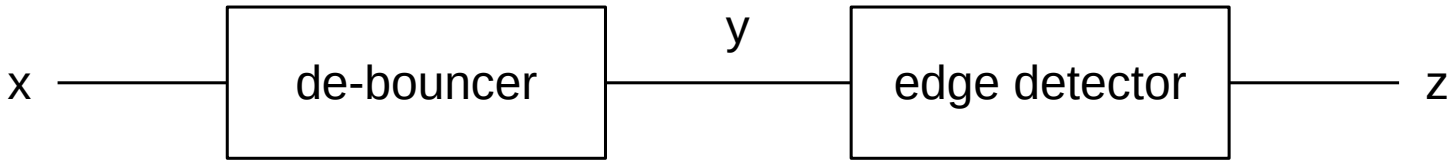
Ejercicio 11. Cronómetro de 1 minuto

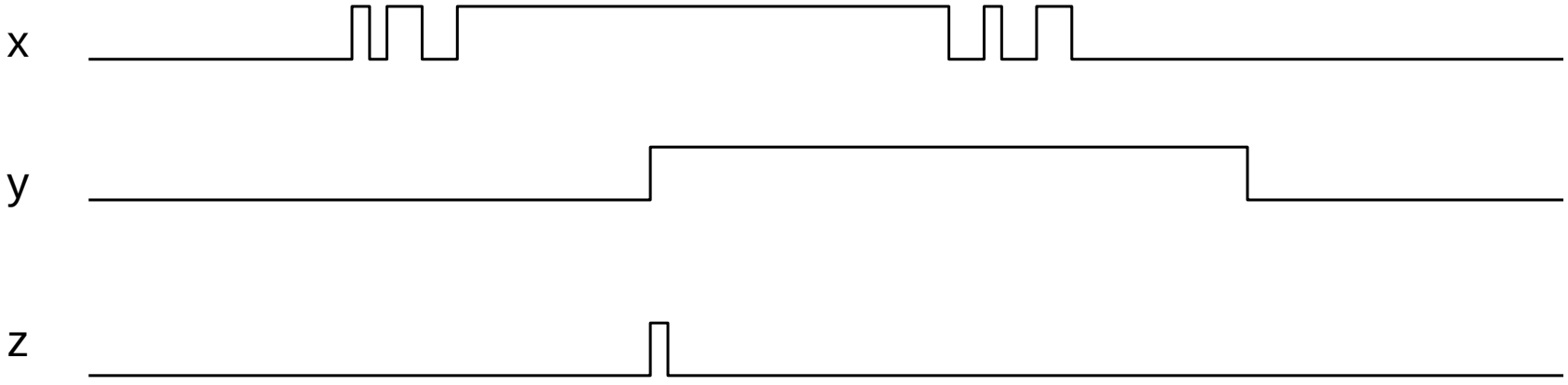
- Diseña un cronómetro de un minuto con resolución de milisegundos, con salidas BCD.
- Especificaciones:
  - Reloj del sistema: 1MHz
  - Entradas:
    - start: continúa la cuenta.
    - stop: detiene la cuenta.
    - clear: pone la cuenta a cero y detiene la cuenta.
  - Salidas BCD:
    - m0: unidades de milisegundo
    - m1: decenas de milisegundos
    - s0: unidades de segundo
    - s1: decenas de segundo
- Realice el diseño empleando contadores módulo 16 con entradas de habilitación (EN) y puesta a cero (CL) síncronas.
- ¿Cómo harías para mostrar el valor de cuenta del cronómetro?

Ejercicio 12
Control de semáforo

- Diseña un circuito de control de un semáforo.
- Especificaciones:
  - Reloj del sistema (clk): 50MHz
  - Entradas: ninguna
  - Salidas:
    - v: luz verde
    - a: luz amarilla
    - r: luz roja
  - Operación: el semáforo activa las luces cíclicamente en la secuencia siguiente y por los tiempos indicados:
    - rojo: 5s
    - verde: 10s
    - amarillo: 1s
- Diseña el circuito empleando contadores y elementos combinacionales.
- Mejora: añade un botón para peatones que haga cambiar el semáforo a rojo.
- (Pista: ¿qué tal un contador en anillo?)

Filtro de rebotes y detector de flancos





Filtro de rebotes

- Filtro de rebotes
  - Entrada: x, Salida: z
  - Operación: z=x sólo si x tiene el mismo valor durante un tiempo “largo”.
- Estrategia 1
  - Usar un contador para contar ciclos de reloj.
  - Cambiar la salida sólo si la salida es diferente a la entrada durante cierto número de ciclos de reloj.
  - Ciclos=tiempo (relacionado por la frecuencia de reloj).
  - Tiempo: 1 – 10ms.
- Estrategia 2 (más simple)
  - Usar un contador para comprobar la entrada sólo cada cierto tiempo (1 – 10ms).
  - Actualizar la salida con el valor detectado.

Ejercicio 13

- Diseñar un filtro de rebotes usando la estrategia 2
- Especificaciones
  - El reloj del sistema es de 8MHz
  - La entrada debe comprobarse cada 20ms aproximadamente.

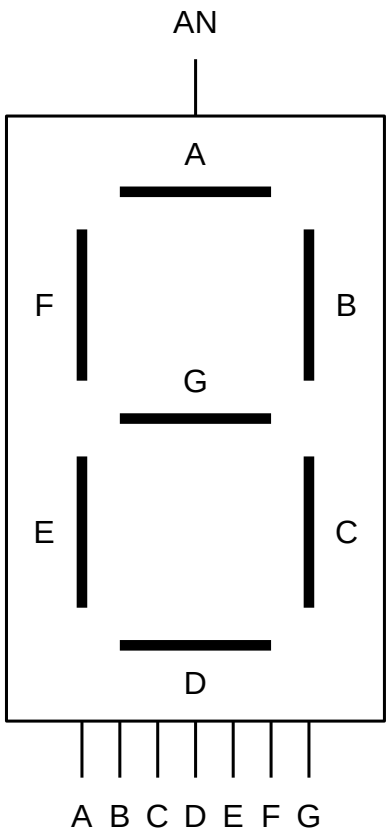
Detector de flancos

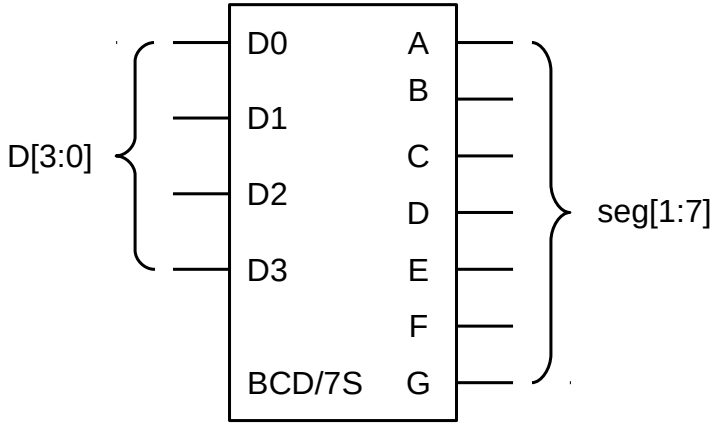
- Detector de flancos
  - La salida se activa sólo durante un ciclo de reloj cuando la entrada cambia de valor.
  - Puede activarse sólo en un tipo de flanco (subida o bajada) o en los dos.
  - Simplifica el diseño de sistemas controlados por botones: detectar si un botón se ha pulsado consiste en detectar un pulso de un solo ciclo de duración.
- Estrategia 1
  - Diseñar usando una máquina de estados.
- Estrategia 2
  - Comparar la entrada en los dos últimos ciclos: salida igual a “1” si valor diferente.

Ejercicio 14

- Diseña un detector de flancos de subida usando la estrategia 2.

Controlador visor 7 segmentos de 4 cifras

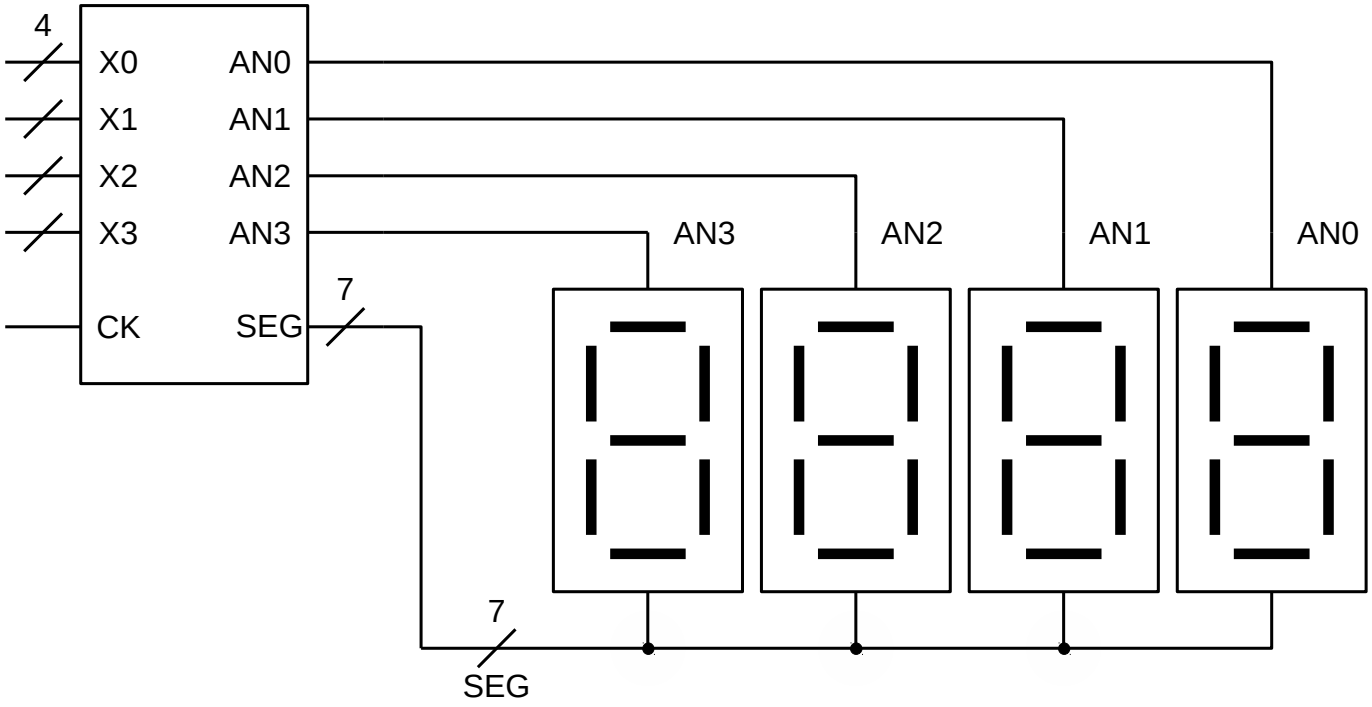




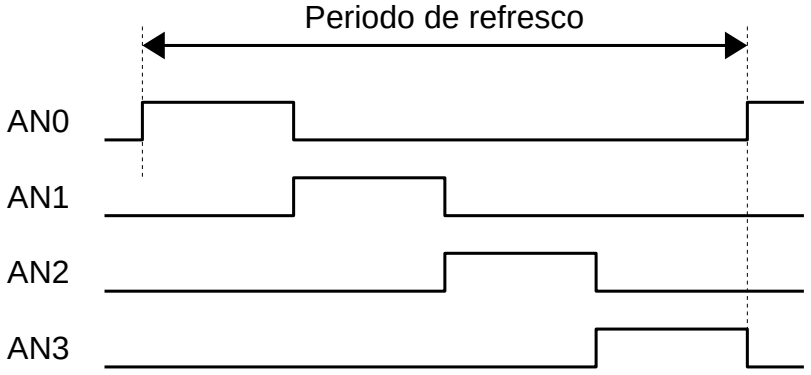
$D_3D_2D_1D_0$	D	seg[1:7] ABCDEFG
0000	0	0000001
0001	1	1001111
0011	2	0010010
0010	3	0000110
0110	4	1001100
0111	5	0100100
0101	6	0100000
0100	7	0001111
1100	8	0000000
1101	9	0001100

AN debe ser '0' para activar el dispositivo

Controlador visor 7 segmentos de 4 cifras



CK: 50MHz
Frecuencia de refresco: 50Hz – 100Hz



Controlador visor 7 segmentos de 4 cifras

- Estrategia
  - Tomar como base un convertidor BCD/7-segmentos o BIN/7-segmentos (representación hexadecimal)
  - Seleccionar secuencialmene las entradas X0, X1, X2 y X3 y pasarlas al convertidor.
  - Con cada selección, activar el visor 7-segmentos correspondiente.
  - Calcular la frecuencia de la selección para que se hagan de 50 a 100 barridos completos por segundo, para no notar el parpadeo. Usar un contador para ajustar la frecuencia de la selección.

Ejercicio 15

- Diseña un controlador de visor de 7 segmentos de 4 cifras
- Especificaciones
  - El reloj del sistema es de 50MHz
  - El valor mostrado debe refrescarse con una frecuencia entre 50 y 100Hz
- Emplea únicamente subsistemas combinacionales y secuenciales.