simplyjarod.com

EDIG

Carpeta Montero

Apuntes y exámenes ETSIT UPM

Si alguna vez estos apuntes te sirvieron de ayuda, piensa que tus apuntes pueden ayudar a muchas otras personas.

Comparte tus apuntes en simplyjarod.com

EDIG
Teoría

PROGRAMA DE LA ASIGNATURA EDIG

1. **Codificación de la Información**
   Introducción Electrónica Digital. Abstracción digital (analógico vs. digital). Sistemas de numeración. Representación números negativos. Álgebra de Boole. Axiomas. Operadores básicos. Tabla de Verdad. Puertas Lógicas simples y complejas. Mapas de Karnaugh.

2. **Dispositivos de Lógica Programable (VHDL)**
   Introducción a los dispositivos lógicos programables y a los lenguajes de descripción hardware (VHDL). Estructura código VHDL. Sintaxis básica.

3. **Circuitos combinacionales**
   Multiplexores. Interconexión de varios MUX. Implementación de funciones con MUX. Codificadores y Decodificadores. Interconexión de varios codificadores. Comparadores. Sumadores. Memorias no volátiles.

4. **Circuitos secuenciales**
   Elemento biestable básico. Báscula Set-Reset. Biestables activos por nivel (latch). Biestables activos por flanco de CLK (flip-flops): tipo-D, po J-K y tipo-T. Temporización. Registros de almacenamiento. Contadores. Registros de desplazamiento.

5. **Autómatas** (1,0 crédito)
   Máquinas Moore y Mealy. Diagrama de estados. Tabla de transiciones autómatas.

TEMA 1: CODIFICACIÓN DE LA INFORMACIÓN

1.1 Sistemas de numeración

Los símbolos numéricos con los que más familiarizados estamos en la actualidad se conocen como **dígitos arábigos**, ya que se desarrollaron en la cultura árabe medieval. Como ya sabemos, son:

1,2,3,4,5,6,7,8,9 y 0

Un sistema de numeración se define por sus símbolos básicos, llamados **dígitos** ó **cifras**, y las formas de combinar los mismos para representar toda la gama de números que requerimos.

Decimos que nuestra **notación** de los números es **posicional**, puesto que cada dígito de un número tiene un valor fijo determinado por su posición. Este valor viene dado por el dígito, multiplicado por una potencia de la **base de numeración** (con su posición como exponente, empezando por cero).

Así, un número "N" se representa en base "b" como "$...d_7d_6d_5d_4d_3d_2d_1$" donde "d_i" son los dígitos.

$$N = ... + \boxed{d_7 b^7} + d_6 b^6 + d_5 b^5 + d_4 b^4 + d_3 b^3 + d_2 b^2 + d_1 b^1 + \boxed{d_0 b^0}$$

- d_0 es el bit menos significativo (LSB)
- d_7 es el bit más significativo (MSB)

Los sistemas más habituales son el **decimal** (base 10), el **binario** (base 2), el **octal** (base 8) y el **hexadecimal** (base 16) de los que se presentan los primeros valores en la siguiente tabla:

Decimal	Binario	Octal	Hexadecimal
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

1.1.1 Sistemas digitales. Tamaño de palabra

Bit = "Binary Digit"

Los tamaños de palabra normalmente son potencias de 2.

Como veremos en epígrafes posteriores, los sistemas digitales trabajan con numeración binaria, en la que a cada dígito lo llamaremos **bit**. Un aspecto importante de estos sistemas es que sólo pueden manejar números con una cantidad fija de dígitos *n*. Así, por ejemplo, los computadores tienen un **tamaño de palabra** específico, que es la longitud (número de bits) de los números binarios procesados por las instrucciones internas del computador.

1.2 Conversión entre las bases más comunes

De binario a:

→ **Octal**: basta con agrupar los digitos binarios de tres en tres empezando por la derecha y sustituir cada terna por el digito octal correspondiente (ver tabla de la página anterior). Si el número de dígitos binarios no es multiplo de tres rellenamos con ceros por la izquierda.

El subíndice indica en qué base está el número

Ejemplo: $10111011001_2 = 10\ 111\ 011\ 001_2 = 2731_8$

→ **Hexadecimal**: basta con agrupar los digitos binarios de cuatro en cuatro empezando por la derecha y sustituir cada grupo por el digito hexadecimal correspondiente (ver tabla de la página anterior). Si el número de dígitos binarios no es multiplo de cuatro rellenamos con ceros por la izquierda.

Es igual que el octal pero agrupando de cuatro en cuatro

Ejemplo: $10111011001_2 = 101\ 1101\ 1001_2 = 5D9_{16}$

→ **Decimal**: multiplicamos cada dígito binario por 2 elevado a la posición que ocupa. Tras ello sumamos todo y sale el número en decimal

Ejemplo:

$$10111011001_2 = 1\cdot 2^{10} + 0\cdot 2^9 + 1\cdot 2^8 + 1\cdot 2^7 + 1\cdot 2^6 + 0\cdot 2^5 + 1\cdot 2^4 + 1\cdot 2^3 + 1\cdot 2^2 + 0\cdot 2^1 + 1\cdot 2^0 = 1497_{10}$$

De octal a:

→ **Binario**: basta con sustituir los digitos en octal por los tres digitos equivalentes en binario (ver tabla de la página anterior).

Ejemplo: $1234_8 = 001\ 010\ 011\ 100_2$

→ **Hexadecimal**: Primero pasamos el dígito octal a binario y después de binario a hexadecimal (ambos procedimientos ya están explicados)

Ejemplo: $1234_8 = 001\ 010\ 011\ 100_2 = 0010\ 1001\ 1100_2 = 29C_{16}$

→ **Decimal**: multiplicamos cada dígito octal por 8 elevado a la posición que ocupa. Tras ello sumamos todo y sale el número en decimal

Ejemplo: $1234_2 = 1\cdot 8^3 + 2\cdot 8^2 + 3\cdot 8^1 + 4\cdot 8^0 = 668_{10}$

De hexadecimal a:

→ **Binario**: basta con sustituir los digitos en hexadecimal por los cuatro digitos equivalentes en binario (ver tabla de la página anterior).

Ejemplo: $C0DE_{16} = 1100\ 0000\ 1101\ 1110_2$

→ **Octal**: Primero pasamos el dígito hexadecimal a binario y después de binario a octal (ambos procedimientos ya están explicados)

Ejemplo: $C0DE_{16} = 1100\ 0000\ 1101\ 1110_2 = 1\ 100\ 000\ 011\ 011\ 110_2 = 140336_8$

→ **Decimal**: multiplicamos cada dígito hexadecimal por 16 elevado a la posición que ocupa. Tras ello sumamos todo y sale el número en decimal

Ejemplo: $C0DE_{16} = 12\cdot 16^3 + 0\cdot 16^2 + 13\cdot 16^1 + 14\cdot 16^0 = 49374_{10}$

De decimal a:

→ **Binario**: Se va dividiendo el dígito en decimal entre 2 tantas veces como sea posible (división entera, sin decimales) y nos vamos quedando con el resto que sobra en cada división (que siempre será 0 ó 1). Esos restos ordenados del último al primero forman el número binario:

Ejemplo: 108 ÷2 = 54 y sobra 0 (LSB)
÷2 = 27 y sobra 0
÷2 = 13 y sobra 1
÷2 = 6 y sobra 1
÷2 = 3 y sobra 0
÷2 = 1 y sobra 1
÷2 = 0 y sobra 1 (MSB)

$108_{10} = 1101100_2$

→ **Octal**: Se va dividiendo el dígito en decimal entre 8 tantas veces como sea posible (división entera, sin decimales) y nos vamos quedando con el resto que sobra en cada división (que esta-rá siempre entre 0 y 7). Esos restos ordenados del último al primero forman el número octal:

Ejemplo: 108 ÷8 = 13 y sobra 4 (LSB)
÷8 = 1 y sobra 5
÷8 = 0 y sobra 1 (MSB)

$108_{10} = 154_8$

→ **Hexadecimal**: Se va dividiendo el dígito en decimal entre 16 tantas veces como sea posible (división entera, sin decimales) y nos vamos quedando con el resto que sobra en cada división (que esta-rá siempre entre 0 y 15). Esos restos ordenados del último al primero forman el número octal:

Ejemplo: 108 ÷16 = 6 y sobra 12 (LSB)
÷16 = 0 y sobra 6 (MSB)

$108_{10} = 6C_{16}$

1.4 Códigos para números decimales

Estos sistemas se usan para excitación de dispositivos externos como relés o LEDs.

En este apartado estudiamos formas alternativas de representar los números decimales, a caballo entre el "pensamiento" de una máquina (sistema binario) y el de una persona (sistema decimal). Los más utilizados son BCD y 1-de-10.

1.4.1 Decimal codificado en binario (BCD)

BCD = "Binary-coded decimal"

BCD no es más que el sistema decimal, con cada dígito codificado en binario.

Hasta ahora hemos supuesto que los números decimales se traducen a binario para ser procesados por circuitos digitales. Un método alternativo es codificar los dígitos decimales en forma binaria, manteniendo su notación posicional. Estos números se denominan números decimales codificados en binario.

La forma de conversión es muy sencilla:
Un número decimal sin signo $N_{10} = d_{n-1}d_{n-2}...d_1d_0$ se convierte a la forma BCD estableciendo la correspondencia de cada dígito d_i a un número binario de cuatro bits B_i.

La correspondencia entre cada dígito decimal y su representación en BCD la tienes en la tabla del final de este epígrafe

Ejemplo:

Si tenemos el número decimal $N_{10} = 7109_{10}$, el proceso de conversión a BCD es:

$$N_{10} = \underbrace{7}_{0111}\ \underbrace{1}_{0001}\ \underbrace{0}_{0000}\ \underbrace{9}_{1001} = 0111000100001001_{10}$$

Observa que el subíndice del resultado es 10, para representar que, efectivamente, no está representado en binario, sino en BCD.

Por supuesto, la conversión de BCD a la forma decimal ordinaria se efectúa reemplazando los grupos de cuatro bits por el dígito decimal equivalente.

Ejemplo:

$$N_{10} = \underbrace{00111000}_{3}\ \underbrace{01001001}_{8\ \ 4}\ \underbrace{00000101}_{9\ \ 0\ \ 5} = 384905_{10}$$

1.4.2 1-de-10

Este sistema consiste simplemente en codificar cada dígito decimal como una palabra binaria de 10 bits. En esta, todos los bits están puestos a cero excepto aquel que ocupa la posición coincidente con el valor del dígito decimal representado.

Es importante recordar que en este sistema contamos las posiciones desde la izquierda, empezando por cero.

Así, por ejemplo, el dígito decimal 0 estará representado por '1000000000', y el decimal 7, corresponderá al '0000000100'. Todas las correspondencias entre dígitos puedes encontrarlas en la tabla adjunta.

Dígito decimal	BCD	1-de-10
0	0000	1000000000
1	0001	0100000000
2	0010	0010000000
3	0011	0001000000
4	0100	0000100000
5	0101	0000010000
6	0110	0000001000
7	0111	0000000100
8	1000	0000000010
9	1001	0000000001

Observa que no se utilizan las combinaciones comprendidas entre 1010 y 1111, inclusive.

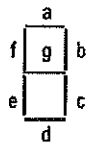
1.4.3 Ejemplo de uso de BCD: excitación de los LEDs de "displays" de 7 segmentos

LS247
FUNCTION TABLE

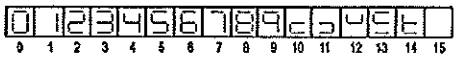
INPUTS				OUTPUTS						
D	C	B	A	a	b	c	d	e	f	g
L	L	L	L	ON	ON	ON	ON	ON	ON	OFF
L	L	L	H	OFF	ON	ON	OFF	OFF	OFF	OFF
L	L	H	L	ON	ON	OFF	ON	ON	OFF	ON
L	L	H	H	ON	ON	ON	ON	OFF	OFF	ON
L	H	L	L	OFF	ON	ON	OFF	OFF	ON	ON
L	H	L	H	ON	OFF	ON	ON	OFF	ON	ON
L	H	H	L	ON	OFF	ON	ON	ON	ON	ON
L	H	H	H	ON	ON	ON	OFF	OFF	OFF	OFF
H	L	L	L	ON	ON	ON	ON	ON	ON	ON
H	L	L	H	ON	ON	ON	ON	OFF	ON	ON
H	L	H	L	OFF	OFF	OFF	ON	ON	OFF	ON
H	L	H	H	OFF	OFF	ON	ON	OFF	OFF	ON
H	H	L	L	OFF	ON	OFF	OFF	OFF	ON	ON
H	H	L	H	ON	OFF	OFF	ON	OFF	ON	ON
H	H	H	L	OFF	OFF	OFF	ON	ON	ON	ON
H	H	H	H	OFF	OFF	OFF	OFF	OFF	OFF	OFF

74LS247: Decodificador de BCD a 7 segmentos

Display de 7 segmentos:



SEGMENT
IDENTIFICATION



NUMERICAL DESIGNATIONS AND RESULTANT DISPLAYS

Para comprender mejor lo anterior, demostremos la primera ley del álgebra de Boole, llamada **ley de absorción**; su expresión es la que sigue:

$$a + a \cdot b = a$$

Su demostración se encuentra en la tabla siguiente:

a	b	$a + a \cdot b$	a
0	0	$0 + 0 \cdot 0 = 0$	0
0	1	$0 + 0 \cdot 1 = 0$	0
1	0	$1 + 1 \cdot 0 = 1$	1
1	1	$1 + 1 \cdot 1 = 1$	1

Existen infinidad de teoremas en el álgebra de Boole, tantos como puedan ser demostrados por el método ya referido; sin embargo, hay una serie de ellos que, dada su utilidad, es importante conocer. La tabla siguiente muestra los más importantes.
Por otra parte, siempre que se cumple una ley o teorema en el álgebra de Boole, se cumple también su llamada **forma dual**; es decir, *se cumple también la expresión que se obtiene cambiando solamente las operaciones de suma por las de producto y las de producto por las de suma.* Las formas duales de las leyes y teoremas básicos también se indican en la tabla siguiente.

Nombre de la ley	Forma básica	Forma dual	
Ley de absorción	$a + a \cdot b = a$	$a \cdot (a + b) = a$	(1)
Teorema de De Morgan	$\overline{(a + b + c + \cdots)} = \bar{a} \cdot \bar{b} \cdot \bar{c} \cdot \cdots$	$\overline{(a \cdot b \cdot c \cdot \cdots)} = \bar{a} + \bar{b} + \bar{c} + \cdots$	(2)
Leyes de transposición	$a \cdot b + \bar{a} \cdot c = (a+c) \cdot (\bar{a}+b)$ $\bar{a} \cdot \bar{b} + a \cdot b = (\bar{a}+b) \cdot (a+\bar{b})$	$(a+b) \cdot (\bar{a}+c) = a \cdot c + \bar{a} \cdot b$ $(\bar{a}+\bar{b}) \cdot (a+b) = \bar{a} \cdot b + a \cdot \bar{b}$	(3) (4)
Leyes varias	$a + \bar{a} \cdot b = a + b$ $\bar{a} + a \cdot b = \bar{a} + b$ $a \cdot b + a \cdot \bar{b} \cdot c = a \cdot b + a \cdot c$ $a \cdot b + \bar{a} \cdot c + b \cdot c = a \cdot b + \bar{a} \cdot c$ $a \cdot b + a \cdot \bar{b} = a$ $a \cdot b + a \cdot c = a \cdot (b + c)$	$a \cdot (\bar{a} + b) = a \cdot b$ $\bar{a} \cdot (a + b) = \bar{a} \cdot b$ $(a+b) \cdot (a+\bar{b}+c) = (a+b) \cdot (a+c)$ $(a+b) \cdot (\bar{a}+c) \cdot (b+c) = (a+b) \cdot (\bar{a}+c)$ $(a+b) \cdot (a+\bar{b}) = a$ $(a+b) \cdot (a+c) = a + (b \cdot c)$	(5) (6) (7) (8) (9) (10)

1.8 Formas canónicas de una función booleana

Ejercicio de clase. 1, Tema 1

Las ecuaciones o expresiones booleanas pueden adoptar dos estructuras o formas típicas, denominadas **formas canónicas.** Dichas formas son:

- **Primera forma canónica – Ecuaciones con estructura minterms:** Esta ecuación está estructurada como una suma de términos en forma de producto de las diferentes variables que intervienen en la ecuación.

  Ejemplo:

  $$x = a \cdot \bar{b} \cdot c + \bar{a} \cdot b \cdot \bar{c} + a \cdot b \cdot c$$

- **Segunda forma canónica – Ecuación con estructura maxterms:** Se dispone como un producto de términos en forma de suma de las diferentes variables que intervienen en la ecuación.

  Ejemplo:

  $$y = \left(\bar{a} + b + c\right) \cdot \left(a + \bar{b} + \bar{c}\right) \cdot \left(\bar{a} + \bar{b} + \bar{c}\right) \cdot \left(a + \bar{b} + c\right)$$

IMPORTANTE: Tanto en una estructura como en la otra, todos los términos han de contener todas las variables que intervienen en la ecuación.

Esta forma canónica se utiliza para implementar una función empleando sólo puertas NAND.

- **Tercera forma canónica – Ecuación con producto de productos:** Se dispone como un producto de términos en forma de producto de las diferentes variables que intervienen en la ecuación.

  Ejemplo:

$$z = \overline{\overline{\bar{a}bc} \cdot \overline{\bar{a}\bar{b}c} \cdot \overline{ab\bar{c}}}$$

Esta forma canónica se utiliza para implementar una función empleando sólo puertas NOR.

- **Cuarta forma canónica – Ecuación con suma de sumas:** Se dispone como una suma de términos en forma de suma de las diferentes variables que intervienen en la ecuación.

  Ejemplo:

$$w = \overline{\left(\overline{a}+b+c\right)+\left(a+\overline{b}+\overline{c}\right)+\left(\overline{a}+\overline{b}+\overline{c}\right)+\left(a+\overline{b}+c\right)}$$

1.9 Obtención de la ecuación de una función lógica partiendo de su tabla de verdad. *ejercicio de clase 1, Tema 1.*

Con el ejercicio 1 de clase comprenderemos mejor estos métodos.

Dada la tabla de verdad,que representa la respuesta binaria de una función lógica, existen dos métodos para obtener su ecuación en primera y segunda forma canónica, respectivamente. Estos métodos están expresados y resumidos en la tabla siguiente, en la que también explicamos el método para obtener a partir de estas, la tercera y la cuarta forma canónica.

Tipo de ecuación	Método de obtención	Convenio a aplicar
Primera forma canónica	Obtener la suma de productos de variables cuyas combinaciones hacen 1 la función	0 variable negada 1 variable sin negar
Segunda forma canónica	Obtener el producto de las sumas de variables cuyas combinaciones hacen 0 la función	0 variable sin negar 1 variable negada
Tercera forma canónica	Negar dos veces la ecuación en la primera forma canónica	
Cuarta forma canónica	Negar dos veces la ecuación en la segunda forma canónica	

Ecuación minterms

Ecuación maxterms

1.10 Simplificación de ecuaciones booleanas

Existen dos procedimientos básicos a la hora de simplificar las ecuaciones booleanas:

- **Método de simplificación algebraico:** Se realiza aplicando las leyes y teoremas del álgebra.
- **Métodos tabulares y gráficos:** Destacamos entre estos, el método de Karnaugh, que estudiamos a continuación.

1.10.1 Simplificación utilizando el Método de Karnaugh Ejercicio 2, Tema 1.

Se han propuesto diversos métodos para minimizar de modo sistemático. Algunos son numéricos, y conducen a un algoritmo que puede programarse. Aquí veremos un método gráfico que es el más sencillo y también el más conocido (aunque prácticamente deja de tener utilidad para formas booleanas con más de seis variables).

Una tabla de Karnaugh no es otra cosa que una presentación alternativa de la misma información contenida en una tabla de verdad. La tabla de Karnaugh es de doble entrada, y tiene las asignaciones colocadas de tal modo que las que corresponden a productos canónicos adyacentes están físicamente contiguas.

En la siguiente figura pueden verse las disposiciones de las tablas de Karnaugh para los casos de tres, cuatro y cinco variables booleanas. Cada casilla corresponde a una línea de la tabla de verdad, y se pondrá en ella un "0" ó un "1". En la figura hemos numerado las casillas con los números de fila correspondiente de la tabla de verdad.

z \ xy	00	01	11	10
0	0	2	6	4
1	1	3	7	5

x_3x_4 \ x_1x_2	00	01	11	10
00	0	4	12	8
01	1	5	13	9
11	3	7	15	11
10	2	6	14	10

x_1=0

x_4x_5 \ x_2x_3	00	01	11	10
00	0	4	12	8
01	1	5	13	9
11	3	7	15	11
10	2	6	14	10

x_1=1

x_4x_5 \ x_2x_3	00	01	11	10
00	16	20	28	24
01	17	21	29	25
11	19	23	31	27
10	18	22	30	26

IMPORTANTE: de una casilla a otra adyacente sólo puede cambiar una variable.

Es por esto que, por ejemplo, representemos la entrada $x_4x_5 = 11$ justo debajo de la casilla con entrada $x_4x_5 = 01$

Una vez rellena la tabla, pasamos a agrupar términos para obtener la forma canónica simplificada que buscamos y después expresar finalmente la función. Según sea esta la primera o la segunda, actuaremos de formas diferentes:

- **Primera forma canónica – Simplificación por unos:**

  Las reglas para agrupar los "1" son las que se enuncian a continuación:

  1) Cada grupo debe tener 2^n elementos (o sea, 1, 2, 4, 8, etc...).
  2) Los grupos deben ser lo más grandes posibles.
  3) Los grupos siempre deben ser rectángulos o cuadrados.
  4) Está permitido pasar los bordes del mapa (en vertical y horizontal).
  5) Cada grupo debe tener al menos un elemento en exclusividad (que no pertenezca a ningún otro grupo).
  6) Todos los "1" deben pertenecer al menos a un grupo.
  7) Hay que intentar agrupar todos los "1" en el menor número posible de grupos.
  8) No es obligatorio que estén superpuestos unos grupos con otros.

Una vez agrupados todos los unos, pasamos a expresar la función buscada.

- Obtendremos tantos términos como grupos de unos (términos que se sumarán entre sí)
- Cada uno de los términos se obtiene multiplicando las variables que quedan constantes.
- Complementamos las variables que valen 0.

- **Segunda forma canónica – Simplificación por ceros:**

Las reglas para agrupar los "0" son las que se enuncian a continuación:

Observa que son las mismas reglas que para las agrupaciones de unos, solo que referidas a los ceros.

1) Cada grupo debe tener 2^n elementos (o sea, 1, 2, 4, 8, etc...).
2) Los grupos deben ser lo más grandes posibles.
3) Los grupos siempre deben ser rectángulos o cuadrados.
4) Está permitido pasar los bordes del mapa (en vertical y horizontal).
5) Cada grupo debe tener al menos un elemento en exclusividad (que no pertenezca a ningún otro grupo).
6) Todos los "0" deben pertenecer al menos a un grupo.
7) Hay que intentar agrupar todos los "0" en el menor número posible de grupos.
8) No es obligatorio que estén superpuestos unos grupos con otros.

Una vez agrupados todos los ceros, pasamos a expresar la función buscada.

- Obtendremos tantos términos como grupos de ceros (términos que se multiplicarán entre sí)
- Cada uno de los términos se obtiene sumando las variables que quedan constantes.
- Complementamos las variables que valen 1.

Febrero 2006
Problema 5

VHDL

- **Entidad:**

```
ENTITY nombre_circuito is
    PORT (nombre_entradas: IN tipo_entradas;
          nombre_salidas: OUT tipo_salidas);
END nombre_circuito;
```

- **Arquitectura:**

```
ARCHITECTURE nombre_arquitectura OF nombre_entidad I
-- Zona de declaraciones
BEGIN
-- Descripción lógica del circuito [cuerpo ("body")]
END nombre_arquitectura;
```

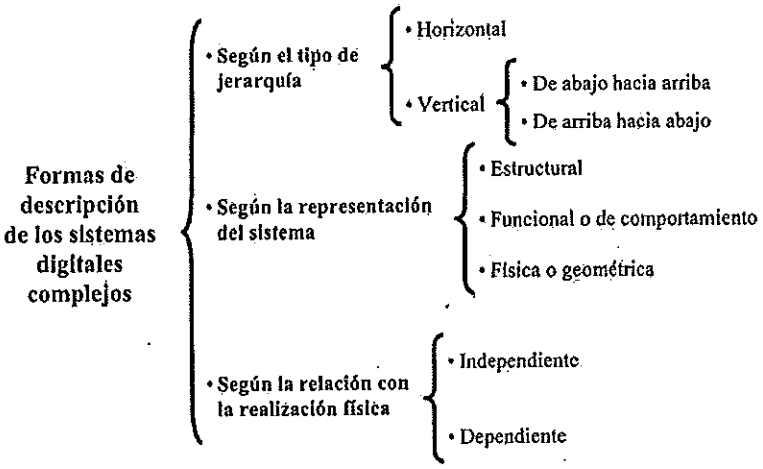
- **Tipos de datos:** *tipo_entradas*, *tipo_salidas*
  - Bit: puede tomar los valores '0' y '1'
  - Bit_vector: conjunto del tipo bit.
  - Std_logic: puede tomar los valores: '0', '1', 'X', 'Z'...
  - Std_logic_vector: conjunto del tipo Std_logic
- **Tipos de objetos:**
  - Constantes: CONSTANT nombre_constante : tipo_de_dato := {};
  - Variables: VARIABLE nombre_variable: tipo_de_dato;
  - Señales: SIGNAL nombre_señal: tipo_de_dato;

TEMA 2: DISP. DE LÓGICA PROGRAMABLE (VHDL).

2.1 Descripción de sistemas digitales complejos

La descripción de los sistemas digitales complejos se puede realizar, tal y como se indica en el esquema siguiente, de diferentes maneras según tres parámetros característicos interrelacionados entre sí:

- El tipo de jerarquía.
- La representación del sistema.
- La relación con la realización física.



En inglés "*structural design*".

A pesar de que existen, tal y como hemos visto, tantas formas de describir un circuito, para esta asignatura sólo vamos a centrarnos en la segunda de las clasificaciones y más concretamente, sólo en las descripciones estructural y funcional.

2.1.1 Formas de descripción según la representación del sistema

2.1.1.1 Descripción estructural

La descripción estructural consiste en especificar la totalidad de los elementos que componen el sistema digital y las interconexiones entre ellos. Inicialmente se realizó representando los elementos mediante símbolos e interconectándolos para dar lugar a un esquema.

Las dificultades que presenta la descripción del esquema de un sistema digital complejo mediante transistores o incluso mediante puertas lógicas hizo que a los programas de diseño de esquemas se les dotase de bibliotecas (en inglés "*libraries*").

Estas bibliotecas contienen componentes o módulos denominados macros que pueden ser de dos tipos: **Macros hardware** ("*hard macros*") y **Macros software** ("soft macros").

La edición de esquemas fue, hasta la década de 1980, la única forma de realizar la descripción estructural asistida por computador de los sistemas digitales. Pero la elevación de su complejidad propició el desarrollo de lenguajes orientados a la descripción de sistemas digitales, que reciben el nombre de **HDL**, uno de los cuales (**VHDL**) estudiaremos en apartados posteriores.

2.1.1.2 Descripción funcional o de comportamiento

La descripción funcional consiste en especificar el funcionamiento del sistema digital en lugar de detallar los elementos que lo forman.

Los formatos habituales para describir el funcionamiento de los sistemas digitales sencillos son las **tablas de verdad, las ecuaciones lógicas y los grafos de estado**. Para sistemas más complejos estos formatos habituales son el flujo de datos y los algoritmos de comportamiento.

Para potenciar esta forma de descripción fue necesario desarrollar lenguajes de descripción como el que estudiamos a continuación.

2.2 Introducción a los lenguajes de descripción

HDL = "*Hardware Description Language*".

Como se ha dicho, los fabricantes de circuitos integrados iniciaron durante la década de 1970 el desarrollo de lenguajes que permitiesen realizar la descripción estructural y funcional de los sistemas digitales complejos. Estos lenguajes, que reciben el nombre genérico de HDL, evolucionaron siguiendo dos caminos paralelos:

Ejemplos de lenguajes HDL no estructurados son ABEL, CUPL y PALASM.

- Algunos fabricantes de circuitos digitales configurables ó de equipos de instrumentación electrónica desarrollaron lenguajes HDL sencillos, denominados **no estructurados**, ya que están orientados a la realización de un único circuito o módulo por fichero.

Ejemplos de lenguajes HDL estructurados son RTL y VHDL.

- Los fabricantes de circuitos integrado a medida desarrollaron lenguajes HDL complejos, denominados **estructurados** porque permiten definir submódulos y enlazarlos jerárquicamente en un único fichero.

  Estos lenguajes se caracterizan además por el hecho de que las herramientas de CAD asociadas con ellos permiten utilizar un fichero HDL como nivel superior de la jerarquía, lo que propicia la creación de bibliotecas de circuitos y operadores frecuentemente utilizados (puertas lógicas, bloques funcionales, operadores aritméticos...).

2.3 Lenguaje VHDL

2.3.1 Introducción

VHSIC – "Very High Speed Integrated Circuits".

VHDL es un lenguaje de descripción de sistemas digitales que fue desarrollado en el marco del programa **VHSIC** del Departamento de Defensa de los Estados Unidos.

VHDL – "VHSIC Hardware Description Language".

Se trata de un lenguaje complejo y estructurado que permite la descripción de cualquier circuito combinacional o secuencial. Por ser un **lenguaje universal**, permite la implementación posterior del circuito descrito en cualquier tipo de circuito integrado disponible en la actualidad, como los circuitos digitales configurables (**PLDs y FPGAs**). Actualmente, todas las herramientas de CAD para el diseño con PLDs y FPGAs incluyen una versión del VHDL.

Además, cabe destacar que el lenguaje VHDL permite la descripción de un circuito de distintas formas, incluyendo las descripciones Estructural y Funcional, descritas en apartados anteriores.

Cabe resaltar que VHDL es independiente de la tecnología empleada.

Para realizar la descripción de un circuito o sistema digital en VHDL, debe incluirse ésta en un fichero de texto, que posteriormente será compilado y sintetizado, obteniendo el código necesario para programar el circuito configurable elegido.

En esto radica que se diga que VHDL es un lenguaje estructurado.

En un fichero escrito en VHDL se puede incluir la descripción de un único circuito o bloque funcional o la de varios, cuyas interconexiones se definirán a su vez en el mismo fichero o en otro distinto.

2.3.2 Sintaxis del fichero VHDL

A continuación se describe la sintaxis que debe presentar cualquier fichero fuente de VHDL. El fichero debe constar de las siguientes secciones, que se muestran en el ejemplo de la tabla siguiente (en negrita):

Sintaxis básica del fichero VHDL

- Cabecera

use ≡ import

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
```

- Declaraciones

En este fichero se describe un biestable de tipo D activado por flancos ascendentes.

```
entity BIESTABLED is
port
(
-- Entradas
reloj : in std_logic;
dato : in std_logic;
-- Salidas
salida_Q : out std_logic
);
end BIESTABLED
```

- Descripción lógica

```
architecture ejemplo of BIESTABLED is
begin
   process
   begin
       wait until reloj'event and reloj = '1';
           salida_Q <= dato;
   end process;
end ejemplo;
```

- Fin del fichero

Si bien en sucesivos apartados de describen con más detalle cada una de las secciones del fichero, vamos a comentar algunas generalidades:

- Cada instrucción debe terminar generalmente con punto y coma, pero no cada línea, puesto que algunas instrucciones ocupan varias líneas.
- Los comentarios, que deben ir precedidos de dos guiones(--), deben situarse al final de la línea o bien ocupar toda una línea

2.3.3 Bibliotecas y paquetes

Una **biblioteca** ("library") es la agrupación de un conjunto de componentes y elementos descritos en VHDL. Las bibliotecas están formadas por **paquetes** ("package") que contienen a su vez definiciones de tipos de datos, funciones, procedimientos, componentes (circuitos), etc, descritos en VHDL.

Existen dos tipos de bibliotecas:

- Predefinidas: suelen venir incorporadas en las herramientas de CAD para diseño de circuitos con VHDL.
- Definidas por el usuario: éste puede incluir en ellas los circuitos que vaya a utilizar habitualmente en sus diseños.

En estos apuntes no desarrollaremos la definición de bibliotecas por el usuario

En los ficheros VHDL de descripción de circuitos, es habitual incluir en su cabecera la utilización de bibliotecas y paquetes, que permiten utilizar en todo el fichero las definiciones que incluyen.

La sintaxis para el uso de bibliotecas y paquetes es:

```
LIBRARY nombre_biblioteca;
USE nombre_biblioteca.nombre_paquete.elemento_paquete
```

Aquí se especifica el uso de todos los componentes del paquete "std_logic_1164 2 de la biblioteca "IEEE", y el uso del componente "dff" del paquete "registros" de la biblioteca "mi_biblioteca".

Ejemplo:

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all

LIBRARY mi_biblioteca;
USE mi_biblioteca.registros.dff
```

Algunos paquetes típicos, disponibles en la biblioteca IEEE, existente en la mayoría de las herraimentas CAD, son: *std_logic_1164.vhd* (estándar), *std_logic_arith.vhd* (aritmética), *std_logic_unsigned.vhd* (aritmética sin signo), *std_logic_components.vhd*...

2.3.4 Entidad

Una entidad ("entity") en VHDL consiste en la definición del nombre del circuito y de sus señales de entrada y salida (**puertos**), indicando el tipo de cada una de ellas.

Para cada circuito que se utilice en VHDL, es necesario definir un bloque entidad.

La sintaxis es la siguiente

```
ENTITY nombre_circuito is
     PORT (nombre_entradas: IN tipo_entradas;
     nombre_salidas: OUT tipo_salidas);
END nombre_circuito;
```

Esos puertos pueden se de los siguientes tipos:
- IN → entrada
- OUT → salida (que no se puede leer desde el propio componente).
- BUFFER → salida (que se puede leer desde dentro)
- INOUT → salida ó entrada

No los usaremos

Más adelante explicaremos los tipos de datos de VHDL.

Ejemplo:

```
ENTITY sumador_total is
     PORT (a, b, cin: IN bit;
     sum, cout: OUT bit);
END sumador_total;
```

En este ejemplo se especifica la entidad correspondiente a un circuito sumador total de dos números de un bit.

2.3.5 Arquitectura

En la arquitectura daremos la Descripción del circuito, en cualquiera de las formas que hemos visto del circuito.

Una arquitectura ("architecture") en VHDL es la definición del funcionamiento de un circuito, de cualquiera de las formas permitidas en VHDL, descritas ya anteriormente.

Una arquitectura siempre debe asociarse con una entidad, aunque una entidad puede tener varias arquitecturas, es decir, varias formas de realización o descripción. Cuando posteriormente se utilice el circuito será necesario indicar cuál es la arquitectura que se utilizará.

Cada circuito en VHDL debe incluir una declaración de entidad y al menos una declaración de arquitectura.

El conjunto entidad – arquitectura es la unidad mínima de diseño.

Un mismo fichero puede contener la declaración de muchos circuitos, lo que permite diseñar un sistema digital complejo mediante un solo fichero VHDL, aunque también se pueda utilizar un fichero distinto para cada circuito. Esto lo convierte en un lenguaje estructurado, que permite realizar fácilmente la descripción jerárquica de un sistema digital.

La sintaxis para la descripción de una arquitectura es la siguiente:

```
ARCHITECTURE nombre_arquitectura OF nombre_entidad IS
-- Zona de declaraciones
BEGIN
-- Descripción lógica del circuito [cuerpo("body")]
END nombre_arquitectura
```

Ejemplo:

Aquí se especifica una posible arquitectura correspondiente al circuito sumador total de dos números de 1 bit, cuya entidad se ha definido en el apartado anterior.

```
ARCHITECTURE comportamiento OF sumador_total IS
SIGNAL auxiliar: bit_vector (1 DOWNTO 0);
BEGIN
        auxiliar <= a+b+cin;
        sum  <= auxiliar(0)
        cout <= auxiliar(1)
END comportamiento
```

Merecen especial mención los **Enumerados**, forma con la que podemos definir nuestros propios tipos de datos. La sintaxis para hacerlo es:

`TYPE nombre tipo IS (lista de elementos)`

2.3.6 Descripción lógica

2.3.6.1 Tipos de datos

Los tipos de datos son las diferentes representaciones de información que vamos a manejar con circuitos lógicos.

Como en otros lenguajes, podríamos hacer una clasificación muy amplia de los tipos de datos existentes en VHDL. No obstante, sólo diremos que los tipos que posteriormente vamos a ver se pueden catalogar en: *Enumerados* (conjunto de literales separados por comas), Numéricos (incluye los tipos natural, entero, real...) y *Compuestos* (conjuntos homogéneos o heterogéneos de elementos.

Obviando esta clasificación, podemos decir que los **tipos de datos más importantes** para la descripción de circuitos son:

- `Bit` Puede tomar los valores '0' y '1'.
- `Bit_vector` Conjunto de señales (o variables) de tipo bit.
- `Std_logic` Puede tomar los valores:

  '0'.....0 fuerte.
  '1'.....1 fuerte.
  'X'.....desconocido fuerte.
  'Z' .....alta impedancia.
  '-' .....indiferente.
  'U'.....no inicializado.
  'L' .....0 débil.
  'H'.....1 débil.
  'W'.....desconocido débil

- `Std_logic_vector` Conjunto de señales (o variables) de tipo std_logic.

Un bit_vector se puede entender como un número binario de varios bits.

No se debe confundir el símbolo utilizado para un valor desconocido fuerte ("X") con el símbolo de la indirefencia ("-").

2.3.6.2 Tipos de objetos

Los tipos de objetos se refieren al tratamiento que va a tener la información manejada.

La sintaxis para declarar señales es:

`SIGNAL nombre: tipo;`

– Constantes: mantienen siempre el mismo valor.

  Ejemplo: `CONSTANT ciclos: integer:=3;`

– Variables: se puede cambiar su valor.

  Ejemplo: `VARIABLE registro: bit_vector (3 down to 0);`

El uso de variables está restringido a las estructuras secuenciales (dentro de procesos y subprogramas). Sin embargo, el uso de señales NO está restringido.

– Señales: su valor puede cambiar a lo largo del tiempo. Equivalen a una conexión física, por lo que es el tipo de datos más utilizado para la descripción de circuitos

  Ejemplo: `SIGNAL entrada: bit;`

2.3.6.3 Notación

IMPORTANTE: los espacios ó caracteres de subrayado "_" no son tenidos en cuenta y sólo sirven para aumentar la legibilidad del número.

- 2, 8, 10, 16 .....binario, octal, decimal o hexadecimal, respectivamente.

  La base debe preceder al valor, dispuesto entre caracteres #, para tipos numéricos.

  Ejemplo: `2#1100_0010#`

- B, X, O.....binario, hexadecimal u octal, respectivamente.

  Este símbolo debe preceder al valor de tipos "bit_vector" o "bit string literals" (literales de cadenas de bits).

  Por defecto, la base es binaria.

  Ejemplo: `X"2E"`

2.3.6.4 Operadores

Operadores Lógicos		Operadores aritméticos	
`and`	función Y lógica	`+`	suma aritmética
`or`	función O lógica	`-`	- resta ritmética (dos operandos)
`xor`	función O-exclusiva		- signo negativo (un operando)
`xnor`	función O-exclusiva negada	`*`	multiplicación
`not`	negación	`/`	división
`nand`	función Y lógica negada	`mod`	módulo (resto de la división entera)
`nor`	función O lógica negada	`**`	exponenciación
`sll`	desplazamiento lógico a izquierda	`sla`	desplazamiento aritmético a izda.
`srl`	desplazamiento lógico a derecha	`sra`	desplazamiento aritmético a dcha.
`&`	concatenación		

La concatenación con & permite unir varios operandos del mismo tipo

Operadores de asignación		Operadores de relación	
`:=`	- asignación de valores a constantes y variables. - asignación de valor inicial a señales	`=`	igual
		`/=`	distinto
		`<`	menor que
`<=`	asignación a señales	`>`	mayor que
		`<=`	menor o igual que
		`>=`	mayor o igual que

Dentro de un if o para expresiones booleanas.

Vamos a distinguir entre dos tipos de sentencias de descripción: las concurrentes y las secuenciales.

Las sentencias concurrentes son las que distinguen al VHDL de los lenguajes de programación y son las que permiten modelar el comportamiento real de los circuitos.

MUY IMPORTANTE: los procesos se utilizan para modelar el comportamiento de cualquier circuito ante el cambio en señales de entrada, cuando este comportamiento requiere una reacción secuencial. Esto es, aunque se trata de un bloque que en conjunto funciona de forma combinacional (su relación con otros procesos no es secuencial, e incluso puede ser simultáneo con otros), un proceso está compuesto por sentencias secuenciales, que sí dependen del orden de aparición. Es por esto que los procesos sirven para describir circuitos secuenciales. Cuando necesitemos sentencias secuenciales, tendrán que ir dentro de un proceso.

En el ejemplo se especifica un proceso que describe una puerta lógica AND de dos entradas.

Importante: sólo usamos uno de los tipos de sentencia WAIT aquí presentadas (por eso comentamos con –o el final de esas sentencias).

Aquí se especifica un proceso que describe un biestable de tipo D activado por flancos.

2.3.6.5 Sentencias concurrentes

Las sentencias concurrentes son aquellas que se ejecutan de forma simultánea a otras sentencias concurrentes, independientemente del orden en que estén escritas en el fichero VHDL. Se puede decir que las sentencias concurrentes modelan un comportamiento combinacional.

a) Process

Los procesos siempre se ejecutan una vez durante la inicialización, pero posteriormente sólo se ejecutan cuando se produce un "evento" (cambio) en alguna de las señales indicadas en el proceso.

Existen dos formas de indicar las señales cuyos eventos producen la activación de un proceso.

– Mediante una **lista de "sensibilidad"**, que enumera entre paréntesis y separados por comas, todas las señales cuyos eventos deben activar el proceso.

  La sintaxis de esta forma es:

```
PROCESS (lista de señales)
BEGIN
     Sentencias secuenciales;
END PROCESS
```

  Ejemplo:

```
PROCESS (a,b)
BEGIN
     Salida <= a AND b;
END PROCESS
```

– Mediante una **sentencia "wait"**, que obliga al proceso a esperar a que se cumpla una determinada condición para activarse. La condición puede ser de tipo booleano, un evento, o una condición temporal.

  La sintaxis de esta forma es:

```
PROCESS
BEGIN
     WAIT ON señal; --o
     WAIT UNTIL expresión; --o
     WAIT FOR expresión temporal;
     Sentencias secuenciales;
END PROCESS
```

  Ejemplo:

```
PROCESS
BEGIN
     -- Espera flanco ascendente de reloj
     WAIT UNTIL reloj'event and reloj = '1';
     Salida <= entrada;
END PROCESS
```

Esta sentencia concurrente es análoga a la sentencia secuencial condicional case, que explicaremos más adelante.

MUY IMPORTANTE: esta sentencia sólo se utiliza para describir comportamientos combinacionales.

b) With ... select (Solo sirven para asignar valores a una señal)

Se utiliza normalmente para las expresiones en las que intervienen varias señales. Se deben cubrir todos los posibles valores de la expresión, de forma excluyente.

La sintaxis es la siguiente:

```
WITH expresión SELECT
    señal <= nuevo_valor WHEN valor_expresión
    ...
    UNAFFECTED WHEN OTHERS;
```

Observa que se trata de una sentencia de asignación, sólo, como When ... else

Ejemplo:

Aquí se especifica el comportamiento de un circuito combinacional cuya salida es una combinación de 3 bits (resultado) y cuya entrada es un vector de 2 bits (operandos)

```
WITH operandos SELECT
   resultado <= "100" WHEN "00"
                "010" WHEN "01"
                "000" WHEN OTHERS;
```

c) When ... else (Solo sirven para asignar valores a una señal)

Esta sentencia permite la asignación condicional de valores a señales. Se utiliza normalmente para las condiciones en las que intervienen varias señales.

Esta es otra sentencia concurrente análoga a la sentencia secuencial condicional If...Then...Else

MUY IMPORTANTE: esta sentencia sólo se utiliza para describir comportamientos combinacionales.

La sintaxis es la siguiente:

```
Señal <= nuevo_valor_1 WHEN expresión_1 ELSE
         nuevo_valor_2 WHEN expresión_2 ELSE
                                        ELSE
         ...
UNAFFECTED;
```

Ejemplo:

Aquí se especifica el comportamiento del mismo circuito combinacional anterior.

```
Resultado <= "100" WHEN (operandos = "00") ELSE
             "010" WHEN (operandos = "01") ELSE
             "000"
```

2.3.6.6 Sentencias secuenciales

Las sentencias secuenciales son aquellas que se ejecutan de forma secuencial, es decir, en el orden en que se han escrito en el fichero VHDL.

Las sentencias secuenciales sólo pueden aparecer dentro de procesos ("process") y subprogramas (funciones y procedimientos).

Las sentencias secuenciales se comportan como las de cualquier lenguaje de programación.

a) If...then...else ≡ if... else if (java)

Se utiliza generalmente para aquellas condiciones de transición que dependen de una sola expresión o, a lo sumo, dos. Las expresiones no tienen por qué ser excluyentes, pero si al evaluar una, ésta es verdadera, las demás no se comprueban.

Esta sentencia secuencial es análoga a la sentencia concurrente when...else

MUY IMPORTANTE: esta sentencia sólo se utiliza para describir comportamientos secuenciales.

La sintaxis es la siguiente:

```
IF expresión_1 THEN
    Sentencias secuenciales;
ELSIF expresión_2 THEN
    Sentencias secuenciales;
ELSE
    Sentencias secuenciales;
END IF
```

Ejemplo: (Contador, no entra para el parcial)

Aquí se especifica el comportamiento de un contador ascendente con carga en paralelo y señal de puesta en estado inicial.

```
IF RESET = '1' THEN
    Ctr <= "0000";
ELSIF (clk'EVENT AND clk = '1') THEN
    IF LOAD = '1' THEN
        Ctr <= entrada;
    ELSE
        Ctr <= Ctr + 1;
    END IF
END IF
```

b) Case...when ≡ switch... case (java)

Esta sentencia secuencial es análoga a la sentencia concurrente **with...select**

Se utiliza generalmente para las expresiones en las que intervienen varias variables o señales y debe cubrir todos los posibles valores de la expresión, de forma excluyente.

MUY IMPORTANTE: esta sentencia sólo se utiliza para describir comportamientos secuenciales.

La sintaxis es la siguiente:

```
CASE expresión IS
    WHEN valor¡expresión1|valor_expresión2|... =>
        Sentencias secuenciales;
    ...
    WHEN OTHERS => NULL;
END CASE;
```

El símbolo | representa la función "o bien".

Ejemplo:

```
CASE seleccion IS
    WHEN "0000"|"0001" =>
        y <= '0';
    WHEN "0010"|"0011" =>
        y <= '1';
    WHEN OTHERS => y <= '0';
END CASE;
```

c) For ≡ for de matlab

Desde aquí comenzamos las sentencias de bucle.

Permite repetir la ejecución de las sentencias incluidas en el bucle mientras el índice del bucle se encuentre en el rango predefinido.

La sintaxis es la siguiente:

```
FOR i IN rango_índice LOOP
    Sentencias secuenciales;
END LOOP;
```

Ejemplo: *** Importante

Aquí se especifica un bucle que calcula el bit de paridad par de una combinación x(i) de 16 bits.

```
Bit_paridad_par <= '0';
FOR i IN 0 TO 15 LOOP
    IF x(i)='1' THEN
        Bit_paridad_par <= NOT bit_paridad_par;
    END IF;
END LOOP;
```

d) While ≡ while (java)

Permite repetir la ejecución de las sentencias incluidas en el bucle mientras se cumpla la condición definida

La sintaxis es la siguiente:

```
WHILE expresion LOOP
    Sentencias secuenciales;
END LOOP;
```

Ejemplo:

```
producto <= 0;
WHILE (q/=0) LOOP
    producto <= producto + p;
    q <= q - 1;
END LOOP;
```

Aquí se especifica un bucle que multiplica dos números "p" y "q" mediante el método de sumas sucesivas.

2.3.7 Ejemplos de circuitos combinacionales

```
library IEEE;
use IEEE.std_logic_1164.all;

entity rol16 is
 port (
  DIN: in STD_LOGIC_VECTOR(15 downto 0); -- Data inputs
  S: in STD_LOGIC_VECTOR (3 downto 0); -- Shift amount, 0-15
  DOUT: out STD_LOGIC_VECTOR(15 downto 0) -- Data bus output
 );
end rol16;

architecture rol16_arch of rol16 is
begin
process(DIN, S)
 variable X, Y, Z: STD_LOGIC_VECTOR(15 downto 0);
 begin
  if S(0)='1' then X := DIN(14 downto 0) & DIN(15); else X := DIN; end if;
  if S(1)='1' then Y := X(13 downto 0) & X(15 downto 14); else Y := X; end if;
  if S(2)='1' then Z := Y(11 downto 0) & Y(15 downto 12); else Z := Y; end if;
  if S(3)='1' then DOUT <= Z(7 downto 0) & Z(15 downto 8); else DOUT <= Z; end if;
 end process;
end rol16_arch;
```

2.3.8 Ejemplos de circuitos secuenciales

2.3.8.1 Báscula R-S

```
library IEEE;
use IEEE.std_logic_1164.all;

entity Vsrlatch is
 port (S, R: in STD_LOGIC;
       Q, QN: buffer STD_LOGIC );
end Vsrlatch;

architecture Vsrlatch_arch of Vsrlatch is
begin
 QN <= S nor Q;
 Q <= R nor QN;
end Vsrlatch_arch;
```

2.3.8.2 Biestable D con flanco positivo

```
library IEEE;
use IEEE.std_logic_1164.all;

entity Vdff is
 port (D, CLK: in STD_LOGIC;
       Q: out STD_LOGIC );
end Vdff;

architecture Vdff_b of Vdff is
begin
process(CLK)
 begin
  if (CLK'event and CLK='1') then Q <= D;
  end if;
 end process;
end Vdff_b;
```

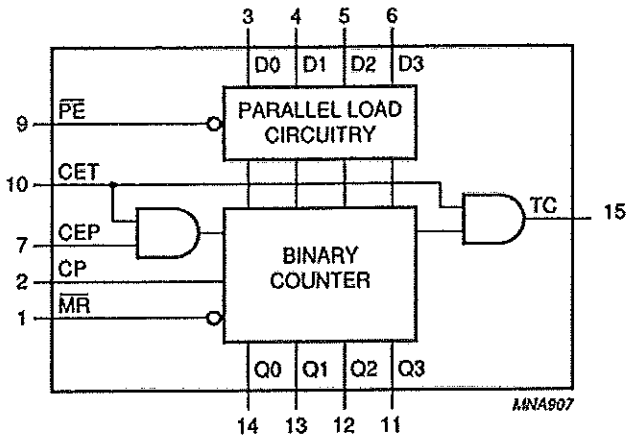
2.3.8.3 Biestable D con Preset y Clear 74xx74

```
library IEEE;
use IEEE.std_logic_1164.all;

entity Vdff74 is
 port (D, CLK, PR_L, CLR_L: in STD_LOGIC;
       Q, QN: out STD_LOGIC );
end Vdff74;

architecture Vdff74_b of Vdff74 is
signal PR, CLR: STD_LOGIC;
begin
process(CLR_L, CLR, PR_L, PR, CLK)
 begin
  PR <= not PR_L; CLR <= not CLR_L;
  if (CLR and PR) = '1' then Q <= '0'; QN <= '0';
  elsif CLR = '1' then Q <= '0'; QN <= '1';
  elsif PR = '1' then Q <= '1'; QN <= '0';
  elsif (CLK'event and CLK='1') then Q <= D; QN <= not D;
  end if;
 end process;
end Vdff74_b;
```

2.3.8.4 Contador binario de cuatro bits 74xx163



FUNCTION TABLE

OPERATING MODE	INPUTS						OUTPUTS	
	$\overline{MR}$	CP	CEP	CET	$\overline{PE}$	D_n	Q_n	TC
reset (clear)	l	↑	X	X	X	X	L	L
parallel load	h h	↑ ↑	X X	X X	l l	l h	L H	L (1)
count	h	↑	h	h	h	X	count	(1)
hold (do nothing)	h h	X X	l X	X l	h h	X X	q_n q_n	(1) L

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

entity V74x163 is
  port ( CLK, CLR_L, LD_L, ENP, ENT: in STD_LOGIC;
         D: in UNSIGNED (3 downto 0);
         Q: out UNSIGNED (3 downto 0);
         RCO: out STD_LOGIC );
end V74x163;

architecture V74x163_arch of V74x163 is
signal IQ: UNSIGNED (3 downto 0);
begin
process (CLK, ENT, IQ)
  begin
    if (CLK'event and CLK='1') then
      if CLR_L='0' then IQ <= (others => '0');
      elsif LD_L='0' then IQ <= D;
      elsif (ENT and ENP)='1' then IQ <= IQ + 1;
      end if;
    end if;
    if (IQ=15) and (ENT='1') then RCO <= '1';
    else RCO <= '0';
    end if;
    Q <= IQ;
  end process;
end V74x163_arch;
```

2.3.8.5 Ejemplo de máquina de estados

```
library IEEE;
use IEEE.std_logic_1164.all;

entity smexamp is
  port ( CLOCK, A, B: in STD_LOGIC;
         Z: out STD_LOGIC );
end;

architecture smexamp_arch of smexamp is
type Sreg_type is (INIT, A0, A1, OK0, OK1);
signal Sreg: Sreg_type;
begin

process (CLOCK) -- state-machine states and transitions
begin
  if CLOCK'event and CLOCK = '1' then
    case Sreg is
      when INIT => if A='0' then Sreg <= A0;
                   elsif A='1' then Sreg <= A1; end if;
      when A0   => if A='0' then Sreg <= OK0;
                   elsif A='1' then Sreg <= A1; end if;
      when A1   => if A='0' then Sreg <= A0;
                   elsif A='1' then Sreg <= OK1; end if;
      when OK0  => if A='0' then Sreg <= OK0;
                   elsif A='1' and B='0' then Sreg <= A1;
                   elsif A='1' and B='1' then Sreg <= OK1; end if;
      when OK1  => if A='0' and B='0' then Sreg <= A0;
                   elsif A='0' and B='1' then Sreg <= OK0;
                   elsif A='1' then Sreg <= OK1;          end if;
      when others => Sreg <= INIT;
    end case;
  end if;
end process;

with Sreg select -- output values based on state
  Z <= '0' when INIT | A0 | A1,
       '1' when OK0 | OK1,
       '0' when others;

end smexamp_arch;
```

2.3.8.6 Máquina de estados: intermitente de Thunderbird

```
entity Vtbird is
 port ( CLOCK, RESET, LEFT, RIGHT, HAZ: in STD_LOGIC;
       LIGHTS: buffer STD_LOGIC_VECTOR (1 to 6) );
end;

architecture Vtbird_arch of Vtbird is
constant IDLE: STD_LOGIC_VECTOR (1 to 6) := "000000";
constant L3  : STD_LOGIC_VECTOR (1 to 6) := "111000";
constant L2  : STD_LOGIC_VECTOR (1 to 6) := "110000";
constant L1  : STD_LOGIC_VECTOR (1 to 6) := "100000";
constant R1  : STD_LOGIC_VECTOR (1 to 6) := "000001";
constant R2  : STD_LOGIC_VECTOR (1 to 6) := "000011";
constant R3  : STD_LOGIC_VECTOR (1 to 6) := "000111";
constant LR3 : STD_LOGIC_VECTOR (1 to 6) := "111111";

begin
 process (CLOCK)
 begin
  if CLOCK'event and CLOCK = '1' then
   if RESET = '1' then LIGHTS <= IDLE; else
    case LIGHTS is
     when IDLE => if HAZ='1' or (LEFT='1' and RIGHT='1') then LIGHTS <= LR3;
                  elsif LEFT='1' then LIGHTS <= L1;
                  elsif RIGHT='1' then LIGHTS <= R1;
                  else LIGHTS <= IDLE;
                  end if;
     when L1 => if HAZ='1' then LIGHTS <= LR3; else LIGHTS <= L2; end if;
     when L2 => if HAZ='1' then LIGHTS <= LR3; else LIGHTS <= L3; end if;
     when L3 => LIGHTS <= IDLE;
     when R1 => if HAZ='1' then LIGHTS <= LR3; else LIGHTS <= R2; end if;
     when R2 => if HAZ='1' then LIGHTS <= LR3; else LIGHTS <= R3; end if;
     when R3 => LIGHTS <= IDLE;
     when LR3 => LIGHTS <= IDLE;
     when others => null;
    end case;
   end if;
  end if;
 end process;

end Vtbird_arch;
```

TEMA 3: CIRCUITOS COMBINACIONALES

3.1 Definiciones básicas

3.1.1 Circuitos combinacionales y circuitos secuenciales

Un ciclo de retroalimentación es la trayectoria de una señal en un circuito que permite que la salida de una puerta se propague de regreso hacia la entrada de esa misma puerta. Un ciclo de esta naturaleza generalmente genera un comportamiento secuencial en un circuito.

Los circuitos lógicos se clasifican en dos tipos, "combinacional" y "secuencial". Un circuito lógico combinacional es aquel cuyas salidas dependen solamente de sus entradas de corriente. Por el contrario, las salidas de un circuito lógico secuencial dependen no sólo de las entradas de corriente sino también de la secuencia anterior de las entradas, posiblemente arbitrarias, que sucedieron en el pasado.

Un circuito combinacional puede contener una cantidad arbitraria de puertas lógicas e inversores, pero NO ciclos de retroalimentación.

3.1.2 Análisis VS Síntesis

En el análisis de un circuito combinacional comenzamos con un diagrama lógico y procedemos hasta una descripción formal de la función que realiza el circuito, tal como una tabla de verdad o una expresión lógica.

En la síntesis hacemos lo contrario, comenzamos con una descripción formal y procedemos hasta un diagrama lógico.

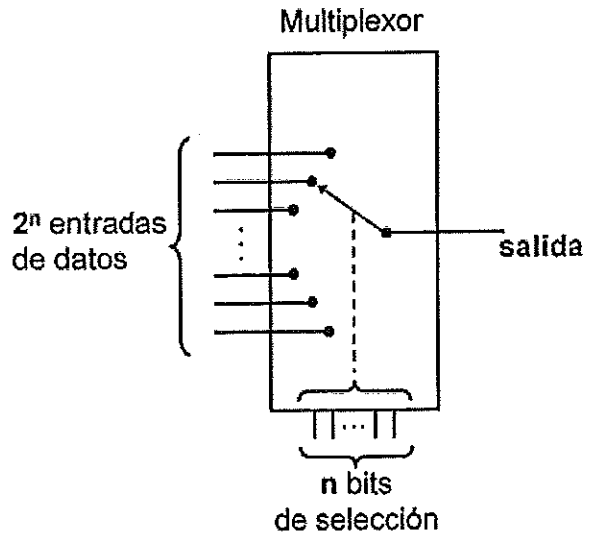
3.2 Multiplexores

Son circuitos combinacionales que poseen las siguientes entradas y salidas:

- N entradas de información o canales de datos.
- n entradas de selección o control.
- Una salida de información
- Una entrada de autorización.

Los canales de entrada están relacionados con las entradas de selección por la siguiente ecuación:

$$\text{número de canales} = 2^{\text{número de entradas de selección}} \Rightarrow N = 2^n$$

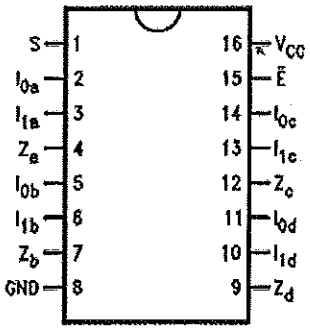


En los esquema representativos de estos circuitos se suele denominar a dichas entradas y salidas con los símbolos que se exponen a continuación:

- D_0 o I_0 a D_N o I_N a las entradas de información.
- S_0 a S_n a las entradas de direccionamiento.
- E a la entrada de autorización o *enable*.
- W o Z a la salida del circuito.

Ejemplo de esquema de un multiplexor:

El componente aquí descrito es el **74AC157**, multiplexor de dos entradas de datos (de 4 bits cada una).



Pin Names	Description
I_{0a}–I_{0d}	Source 0 Data Inputs
I_{1a}–I_{1d}	Source 1 Data Inputs
$\overline{E}$	Enable Input
S	Select Input
Z_a–Z_d	Outputs

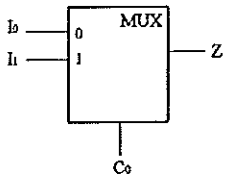
3.2.1 Funcionamiento de un multiplexor

El principio de funcionamiento del multiplexor es el siguiente: cuando una combinación binaria aparece en las entradas de selección, la información de entrada presente en el canal por ella definido aparece en la salida.

Por tanto, se puede considerar a un multiplexor como un conmutador de múltiples entradas cuya única salida se controla electrónicamente mediante las entradas de selección.

La estructura interna de estos circuitos puede llegar a ser relativamente compleja, y como, por otra parte, nosotros los vamos a encontrar en el mercado bajo la forma de *chips* integrados, no realizaremos su estudio interno.

3.2.2 Multiplexor 2 x 1



Esta es la tabla de verdad de un multiplexor de 2 canales de entrada (I_0 e I_1) con 1 entrada de control (C_0)

C_0	Z
0	I_0
1	I_1

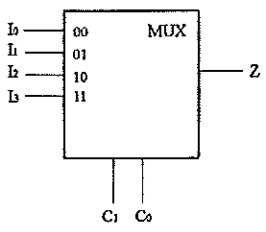
Y esta es la función que implementa:

$$Z = \overline{C}_0 I_0 + C_0 I_1 \qquad (1^a \text{ forma canónica})$$

$$Z = (C_0 + I_0)\cdot(\overline{C}_0 + I_1) \qquad (2^a \text{ forma canónica})$$

3.2.3 Multiplexor 4 x 2

La tabla de verdad de un multiplexor de 4 canales de entrada (I_0, I_1, I_2 e I_3) con 2 entradas de control (C_0 y C_1) es la siguiente:



C_1	C_0	Z
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

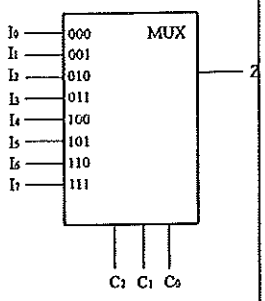
Y esta es la función que implementa:

$$Z = \overline{C}_1 \overline{C}_0 I_0 + \overline{C}_1 C_0 I_1 + C_1 \overline{C}_0 I_2 + C_1 C_0 I_3 \quad (1^a \text{ forma canónica})$$

$$Z = (C_1 + C_0 + I_0)\cdot(C_1 + \overline{C}_0 + I_1)\cdot(\overline{C}_1 + C_0 + I_2)\cdot(\overline{C}_1 + \overline{C}_0 + I_3) \quad (2^a \text{ forma canónica})$$

3.2.4 Multiplexor 8 x 3

La tabla de verdad de un multiplexor de 8 canales de entrada ($I_0, I_1, I_2, I_3, I_4, I_5, I_6$ e I_7) con 3 entradas de control (C_0, C_1 y C_2) es la siguiente:



C_2	C_1	C_0	Z
0	0	0	I_0
0	0	1	I_1
0	1	0	I_2
0	1	1	I_3
1	0	0	I_4
1	0	1	I_5
1	1	0	I_6
1	1	1	I_7

Y esta es la función que implementa:

$$Z = \overline{C}_2\overline{C}_1\overline{C}_0 I_0 + \overline{C}_2\overline{C}_1 C_0 I_1 + \overline{C}_2 C_1 \overline{C}_0 I_2 + \overline{C}_2 C_1 C_0 I_3 + C_2\overline{C}_1\overline{C}_0 I_4 + C_2\overline{C}_1 C_0 I_5 + \\ + C_2 C_1 \overline{C}_0 I_6 + C_2 C_1 C_0 I_7$$

Análogamente se pueden deducir las tablas de verdad y la forma canónica de multiplexores de mayor número de canales y entradas de control.

3.2.5 Realización de funciones lógicas con multiplexores Ejercicio 1, Tema 3

La circuitería interna que posee un multiplexor permite la implementación de funciones lógicas mediante su adecuado conexionado externo. Existen dos métodos de emplear multiplexores cuando se trata de implementar funciones lógicas:

a) Implementación de funciones booleanas de *n* variables con un multiplexor de *n* entradas de control

- En este caso basta con escribir la tabla de verdad de la función booleana y asignar cada una de las líneas de esa tabla a uno de los canales de datos. Las *n* entradas de control se reservan para las *n* variables de la función.
- Una opción más mecánica es escribir la 1ª forma canónica estandar de una función a la salida de un multiplexor, e identificar cada término (teniendo en cuenta que las entradas de control las identificamos directamente con las variables de la función. Por lo tanto, sólo nos queda saber qué valores, '1' ó '0', ponemos en las entradas de datos).

  Vamos a detallar la explicación para la realización de una función F de dos variables (X e Y) con un multiplexor 4x2:

  Para este multiplexor, la forma desarrollada de su función de salida es:

$$Z = \overline{C_1 C_o} I_0 + \overline{C_1} C_o I_1 + C_1 \overline{C_o} I_2 + C_1 C_o I_3$$

Veremos este método en la práctica en el ejercicio 1 de clase, de este tema, apartado a.

Recuerda que un multiplexor 4x2 tiene 4 canales de entrada (I_0, I_1, I_2 e I_3) con 2 entradas de control (C_0 y C_1)

En este momento empezamos ya a completar el dibujo de nuestro circuito con el multiplexor.

Así, por ejemplo, si en la función del enunciado no aparece el término $\overline{XY}$, entonces I_0 tiene que valer '0'.

Sobre esta función de salida, identificamos ya las entradas de control con las variables, y Z con F, la función que queremos implementar:

$$F = \overline{XY}\, I_0 + \overline{X}Y\, I_1 + X\overline{Y}\, I_2 + XY\, I_3$$

Donde sólo queda ya sustituir cada entrada de datos I_0, I_1, I_2 e I_3 por '1' ó '0' según aparezca o no el término en cuestión en la función del enunciado.

b) Implementación de funciones booleanas de n + 1 variables con un multiplexor de n entradas de control y un negador

Veremos este método en la práctica en el ejercicio 1 de clase, de este tema, apartado b.

En este caso una de las variables de la función booleana será enchufada al multiplexor por las entradas de datos mientras que las otras n – 1 variables se siguen enchufando en las entradas de control.

Nuevamente vamos a seguir la explicación a partir de un multiplexor 4x2, sólo que ahora la función *F* que queremos implementar es de tres variables, *X*, *Y* y *W*.

- Para poder implementar la función vamos a seguir el segundo de los métodos del apartado **a)**, pero realizando unos pasos previos.

  Lo primero que debemos hacer es elegir qué dos variables van a estar conectadas a las entradas de control. Vamos a suponer que en nuestro caso van a ser *X* e *Y*.

  Acto seguido tenemos que transformar la expresión que nos den de la función, para que en cada término de la misma aparezcan SIEMPRE estas dos variables elegidas. Para ello, cuando en algún término falte alguna, multiplicaremos dicho término por $\left(X + \overline{X}\right)$ ó $\left(Y + \overline{Y}\right)$, según cuál sea la variable que falta.

  Una vez expresadas estas multiplicaciones, deshacemos los paréntesis aplicando la propiedad distributiva, simplificamos cuando se pueda, y reordenamos la expresión.

  De esta forma, podremos ya escribir nuestra salida estandar (en la 1ª forma canónica) de un multiplexor de estas características:

$$Z = \overline{C_1 C_o} I_0 + \overline{C_1} C_o I_1 + C_1 \overline{C_o} I_2 + C_1 C_o I_3$$

  Para después identificar:

$$F = \overline{XY}\, I_0 + \overline{X}Y\, I_1 + X\overline{Y}\, I_2 + XY\, I_3$$

  Donde la única diferencia será que las entradas I_0, I_1, I_2 e I_3 no quedarán sólo identificadas con '1' ó '0', sino también por W ó $\overline{W}$, la tercera de las variables de la función.

Recuerda dos propiedades importantes:

$\boxed{a + \overline{a} = 1}$

$\boxed{a \cdot 1 = a}$

Nota:

Veremos un ejemplo de este tipo en el ejercicio 1 de clase, de este tema, apartado c.

También es posible, en algunas ocasiones, implementar funciones con $n + 2$ variables o más mediante un múltiplexor de n entradas de control pero para ello el problema debe estar preparado.

3.3 Decodificadores

Genéricamente, un **decodificador** es un circuito lógico con varias entradas y salidas que convierte las entradas codificadas en salidas codificadas, donde los códigos de entrada y de salida son diferentes.

En ocasiones un código binario de n bits es truncado para representar menos de 2^n valores. Es el caso del código BCD, en el que las combinaciones 0000 hasta 1001 representan los dígitos decimales 0 a 9, pero las combinaciones 1010 hasta 1111 no se usan.

- El código de entrada que se utiliza con mayor frecuencia es un código binario de n bits, donde una palabra de n bits representa uno de 2^n diferentes valores codificados, normalmente los enteros de 0 hasta $2^n - 1$.
- El código de salida que se utiliza con mayor frecuencia es un *código 1 fuera de m*, que contiene m bits, donde un solo bit se activa en un determinado momento.

Son circuitos combinacionales provistos de n entradas y un número de salidas menor o igual 2^n.

3.3.1 Decodificaror binario: funcionamiento

Efectivamente, son decodificadores provistos de n entradas y un número de salidas menor o igual 2^n.

El circuito decodificador más común es un decodificador de n a 2^n ó **decodificador binario.** Un decodificador de esta clase tiene un código de entrada binario de n bits y un código de salida *1 fuera de* 2^n.

Se utiliza un decodificador binario cuando se necesita activar exactamente una de las 2^n salidas basado en un valor de entrada de n bits. Dicho de otra forma, funcionan de manera que, al aparecer una combinación binaria en sus entradas, se activa una sola de sus salidas.

No todos los decodificadores poseen la misma asignación de estados lógicos.; de hecho, hay muchos que trabajan tomando un nivel alto (1) como nivel activo.

Normalmente, la salida activada presenta un 0, mientras que las demás permanecen a 1. Los decodificadores se emplean en los sistemas digitales para convertir las informaciones binarias, con las cuales trabajan, otros tipos de informaciones digitalizadas, pero no binarias, empleadas por otros dispositivos, por ejemplo, los visualizadores alfanuméricos.

A continuación presentamos algunos ejemplos de circuitos de estas características.

3.3.2 Decodificador binario 2 a 4

Un ejemplo real de decodificador (un integrado con esta funcionalidad) es el 74AC179, solo que tanto sus salidas como su entrada enable son activas a nivel bajo.

Describimos aquí un decodificador de 2 a 4 líneas con entrada de inhibición que activa la salida en nivel bajo.

La descripción completa de sus entradas y salidas es:

- I_0 e I_1 son las dos entradas.
- Y_0 a Y_3 son las cuatro salidas. Una de ellas se activará en función del número binario representado a la entrada.
- EN es la entrada de autorización o *enable*. Observa que se activa a nivel alto, esto es, el decodificador funcionará cuando esta entrada esté a 1.

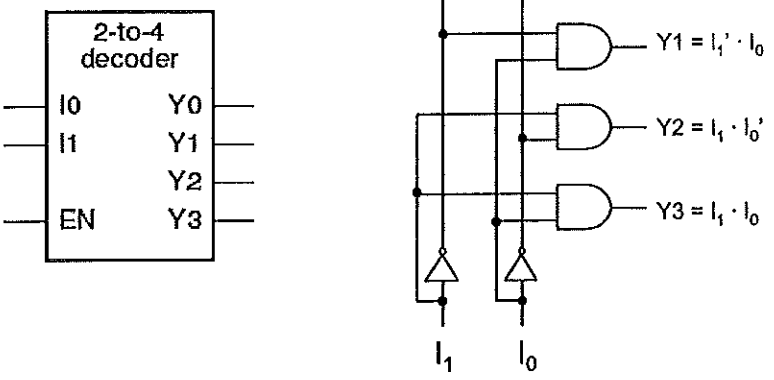
Su tabla de verdad es la siguiente:

La tabla de verdad del decodificador binario introduce una notación "sin importancia" para combinaciones de entrada. Si uno o más valores de entrada no afectan a los valores de salida para alguna combinación de las entradas restantes, se marcan con una X para esa combinación de entrada.

Inputs			Outputs			
EN	I_1	I_0	Y_3	Y_2	Y_1	Y_0
0	x	x	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

Su esquema y su circuito equivalente (implementado con puertas) son:

Es importante entender que en el número binario que pongamos en $I_1 I_0$, I_1 es el bit más significativo. Aun así, mira SIEMPRE la tabla de verdad de cada circuito integrado, para conocer con certeza estas cuestiones.



En el circuito equivalente, observa que cada salida consiste en un minterm de las variables de entrada

3.3.3 Decodificador 3 a 8

Otro ejemplo de decodificador es el de 3 a 8 líneas.

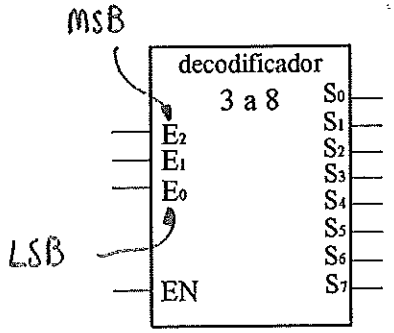
El que presentamos tiene las siguientes entradas y salidas:

- E_0 a E_2 son las tres entradas.
- S_0 a S_7 son las ocho salidas. Una de ellas se activará en función del número binario representado a la entrada.
- *EN* es la entrada de autorización o *enable*.

Su tabla de verdad es la siguiente:

EN	E_2	E_1	E_0	S_0	S_1	S_2	S_3	S_4	S_5	S_6	S_7
0	X	X	X	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	0	0	0	0
1	0	0	1	0	1	0	0	0	0	0	0
1	0	1	0	0	0	1	0	0	0	0	0
1	0	1	1	0	0	0	1	0	0	0	0
1	1	0	0	0	0	0	0	1	0	0	0
1	1	0	1	0	0	0	0	0	1	0	0
1	1	1	0	0	0	0	0	0	0	1	0
1	1	1	1	0	0	0	0	0	0	0	1

Y su esquema:



En este caso, E_2 es el bit más significativo de la entrada

3.3.4 Funciones de un decodificador

- La función que más nos interesa de un decodificador es la de implementar funciones lógicas. La forma de hacerlo es muy sencilla:
  - Necesitamos la tabla de verdad de la función.
  - Basta con unir en una puerta OR aquellas salidas del decodificador donde la función valga uno.
  - Otra posibilidad es unir justo las contrarias (donde la función vale 0) con una puerta NOR.

Observa que así es como formamos los minterms a la salida.

- Otras funciones importantes de un decodificador son la de hacer de decodificador estricto (que, en principio, es para lo que se inventaron) y la de hacer de demultiplexor (si se cumplen las características antes enunciadas en el margen).

Un decodificador es lo mismo que un demultiplexor si cumple que:

- Tiene 2^n salidas.
- Tiene entrada de validación.

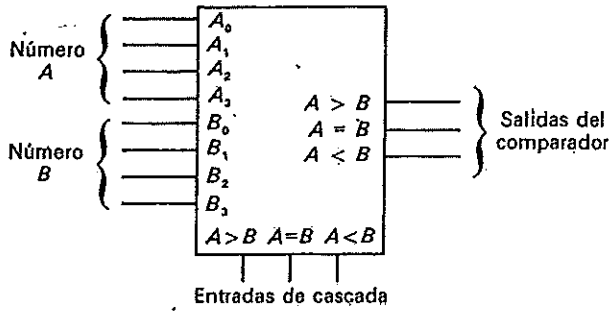
3.4 Comparadores binarios

El comparador más sencillo que podemos encontrar es el de igualdad entre dos bits, que es implementado simplemente por una puerta XOR.

Los circuitos comparadores son circuitos combinacionales que indican la relación de igualdad o desigualdad existente entre dos números binarios A y B de n bits cada uno. Además suelen disponer de una serie de entradas de acoplamiento en cascada para poder comparar con mayor número de bits que los permitidos por el comparador que usamos.

Ejemplo:

En la figura se muestra el diagrama esquemático de un comparador del tipo 74x85:



Su tabla de funcionamiento la siguiente.

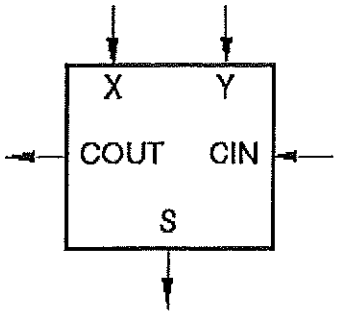
Entradas de comparación				Entradas de cascada			Salidas		
A3, B3	A2, B2	A1, B1	A0, B0	A > B	A < B	A = B	A > B	A < B	A = B
A3 > B3	X	X	X	X	X	X	1	0	0
A3 < B3	X	X	X	X	X	X	0	1	0
A3 = B3	A2 > B2	X	X	X	X	X	1	0	0
A3 = B3	A2 < B2	X	X	X	X	X	0	1	0
A3 = B3	A2 = B2	A1 > B1	X	X	X	X	1	0	0
A3 = B3	A2 = B2	A1 < B1	X	X	X	X	0	1	0
A3 = B3	A2 = B2	A1 = B1	A0 > B0	X	X	X	1	0	0
A3 = B3	A2 = B2	A1 = B1	A0 < B0	X	X	X	0	1	0
A3 = B3	A2 = B2	A1 = B1	A0 = B0	1	0	0	1	0	0
A3 = B3	A2 = B2	A1 = B1	A0 = B0	0	1	0	0	1	0
A3 = B3	A2 = B2	A1 = B1	A0 = B0	0	0	1	0	0	1
A3 = B3	A2 = B2	A1 = B1	A0 = B0	X	X	1	0	0	1
A3 = B3	A2 = B2	A1 = B1	A0 = B0	1	1	0	0	0	0
A3 = B3	A2 = B2	A1 = B1	A0 = B0	0	0	0	1	1	0

3.5 Sumadores – Full Adder

Tema3, ejercicio de clase 2

El bloque elemental de los sumadores es el sumador completo o "*full adder*", con tratamiento de acarreos de entrada (CIN) y salida (COUT). A continuación se muestra su esquema y su tabla de verdad.

Al full adder lo abreviaremos FA

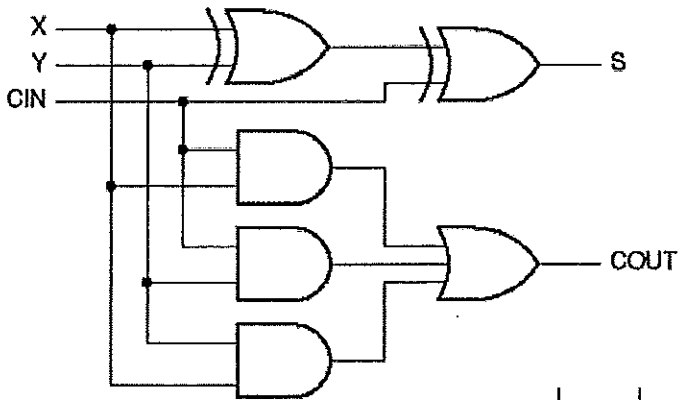


X	Y	Cin	S	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

El FA sólo suma dos bits más el acarreo de entrada

El circuito interno del sumador completo se muestra a continuación:

Para más detalles ver el ejercicio 1 de clase de este tema



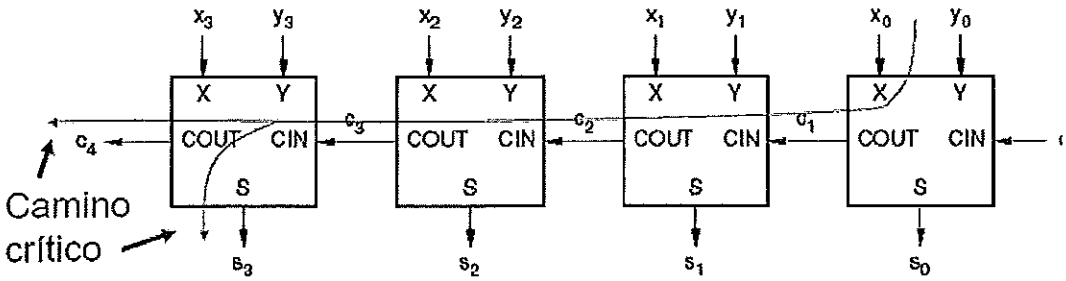
A partir del bloque elemental se pueden construir sumadores de más bits (*n* bits).

Estos sumadores de dos palabras de *n* bits, formados por full-adders, se suelen denominar **sumadores de rizo.**

Solo necesitamos una cascada de n etapas de sumadores completos, si cada una de las cuales maneja un bit.

Por ejemplo, aquí mostramos un sumador de cuatro bits construido a partir de cuatro sumadores de un bit:

Ver febrero 2000- ej2



En este tipo de sumadores la velocidad de cálculo está limitada por el camino del acarreo. Si este se propaga en toda la cuenta, ralentiza la operación

3.6 Memorias ROM

matriz = memoria

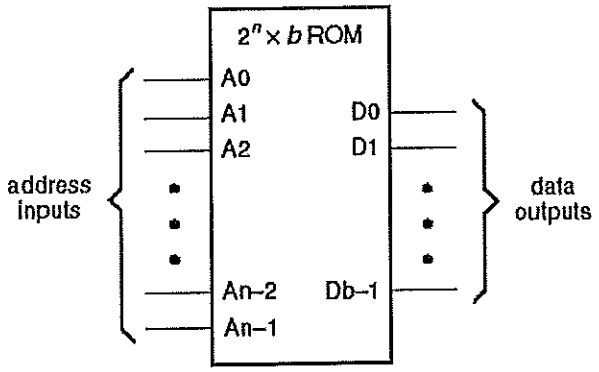
fila = palabra

Podemos entender una memoria ROM (Read Only Memory) como una matriz de dimensión $2^n \times b$, es decir una matriz de 2^n filas y b columnas. A las filas de esta matriz las vamos a llamar palabras y diremos que cada palabra está compuesta de b bits. Por tanto tenemos que mirar siempre esta matriz por filas (nunca por columnas).

Nota: La longitud b de la palabra puede ser cualquiera, independientemente del número de filas

Para acceder a cada fila de la matriz tenemos una dirección que no es más que un número binario que identifica cada fila. Por tanto si nuestra matriz tiene 2^n filas necesitaremos un dirección de n bits. A su vez cada fila alberga una información de b bits (que son las columnas que tiene la matriz).

El esquema genérico de una memoria ROM de $2^n \times b$ es el siguiente:



Resumen

- En la memoria caben 2^n palabras de b bits, o sea, en total $2^n \times b$ bits.
- Consta de n entradas (llamadas Bus de direcciones) y b salidas (llamadas Bus de datos)
- Por cada combinación binaria de las entradas (2^n) existe un dato de longitud b bits

3.6.1 Memoria ROM de 4 x 5

Recuerda que b es totalmente independiente de n

En este caso nuestra memoria tiene $2^2 = 4$ palabras (es decir, una matriz con 4 filas), las direcciones para referirnos a cada palabra tendrán 2 bits. En concreto las direcciones de las cuatro palabras de la memoria serán 00, 01, 10, 11. Cada una de estas palabras tendrá una información de 5 bits.

3.6.2 Memoria ROM de 4 x 2

En este caso nuestra memoria tiene $2^3 = 8$ palabras (es decir, una matriz con 8 filas) las direcciones para referirnos a cada palabra tendrán 3 bits. En concreto las direcciones de las ocho palabras de la memoria serán 000, 001, 010, 011, 100, 101, 110, 111. Cada una de estas palabras tendrá una información de 2 bits.

3.6.3 Funcionamiento de las memorias.

El método de funcionamiento de una memoria es la siguiente: para elegir una fila de la memoria debo introducir por las entradas (bus de direcciones o *address inputs*) el código de esa fila. La memoria "va" a esa fila, extrae la información que hay en ella y la "saca" por las salidas (bus de datos o *data outputs*).

3.6.4 Uso de las memorias ROM

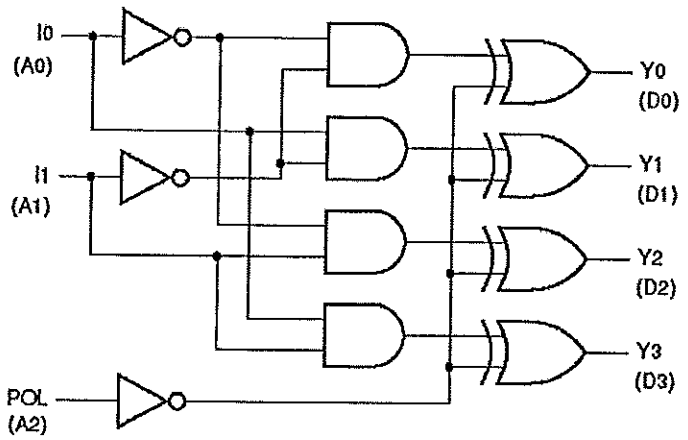
Una Memoria ROM puede implementar cualquier función lógica combinacional.

Para ello basta con hacer los siguiente:

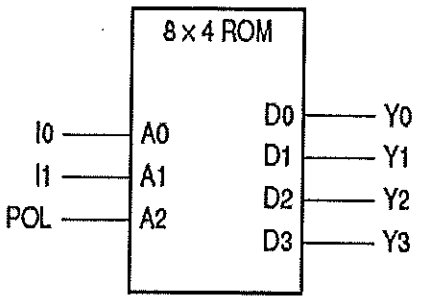
- Las entradas de la función se conectan al bus de direcciones
- Las salidas de la función se conectan al bus de datos

Ejemplo de implementación de una función lógica en una ROM:

Sea el circuito combinacional siguiente,



Podemos implementarlo mediante una memoria ROM de 8 x 4 :



Inputs			*Outputs*			
A2	*A1*	*A0*	*D3*	*D2*	*D1*	*D0*
0	0	0	1	1	1	0
0	0	1	1	1	0	1
0	1	0	1	0	1	1
0	1	1	0	1	1	1
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

TEMA 4: CIRCUITOS SECUENCIALES

Recuerda lo explicado en el tema 3 sobre la diferencia entre circuitos combinacionales y secuenciales:

- Un circuito lógico combinacional es aquel cuyas salidas dependen solamente de sus entradas.
- Un circuito lógico secuencial es aquel cuyas salidas dependen no sólo de sus entradas actuales, sino también de la secuencia pasada de entradas, posiblemente retrasada en el tiempo de manera arbitraria.

4.1 Introducción

Los circuitos secuenciales se caracterizan por su capacidad para memorizar información; en consecuencia, los valores de las salidas, en un determinado momento, no dependen exclusivamente de los valores de las entradas en ese instante, sino que dependen también de los que tuvieran presentes con anterioridad.

Definiremos el estado actual de un circuito secuencial como una colección de "variables de estado" cuyos valores en cualquier tiempo contienen toda la información acerca del pasado necesario para explicar el comportamiento futuro del circuito.

Dicho de otra forma, vamos a llamar "variables de estado" a las variables que guardan toda la información sobre la historia del circuito y permiten predecir la salida actual en base a su contenido y al de las señales de entrada actuales.

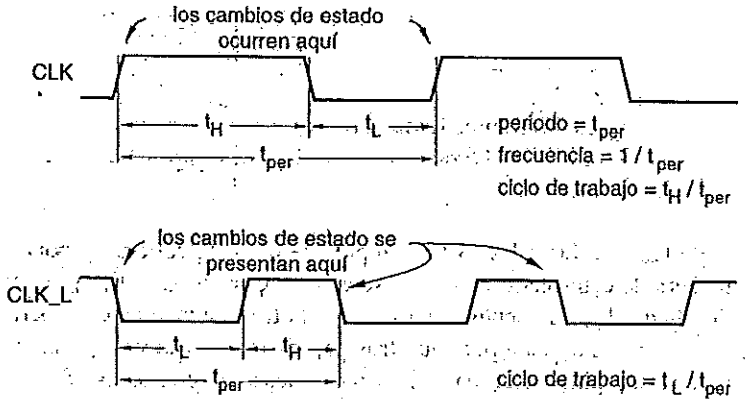
- Las variables de estado se guardan en uno o más bits de información.
- Considerando como entradas las entradas del circuito y las variables de estado, el diseño de un circuito secuencial es igual al de uno combinacional.

Tan grande como parezca, 2^n siempre será finito, nunca infinito, de modo que los circuitos secuenciales en ocasiones se conocen como **máquinas de estado finito.**

En un circuito de lógica digital, las variables de estado son valores binarios, correspondientes a ciertas señales lógicas en el circuito, como veremos en secciones posteriores. Un circuito con n variables de estado binarias tiene 2^n estados posibles.

4.1.1 Señales de reloj

Los cambios de estado en la mayoría de los circuitos secuenciales se presentan en tiempos especificados por una señal de reloj de funcionamiento libre. En la siguiente figura presentamos los diagramas de temporización y nomenclatura para señales de reloj típicas.

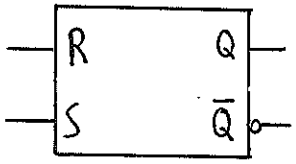


Los sistemas digitales típicos, desde relojes digitales hasta supercomputadoras, utilizan un oscilador de cristal de cuarzo para generar una señal de reloj de funcionamiento libre.

Por convención, una señal de reloj es de estado activo alto si los cambios de estado se presentan en el flanco de subida del reloj o cuando el reloj está en ALTO, y de estado activo bajo en el caso complementario.

- El **periodo del reloj** es el tiempo entre transiciones sucesivas en la misma dirección.
- La **frecuencia del reloj** es el inverso del periodo.
- El primer flanco o pulso en un periodo de reloj (o en ocasiones el periodo mismo) se denomina una **marca** (*"tic"*) de reloj.
- El **ciclo de trabajo** (*duty cycle*) es el porcentaje de tiempo que la señal de reloj se encuentra en su nivel asertivo.

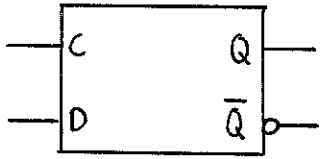
Báscula R-S



R	S	Q_t	Q_t^*	Q_{t+1}
0	0	0	1	0
0	0	1	0	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	?

R	S	Q_{t+1}
0	0	
0	1	
1	0	
1	1	?

Latch D



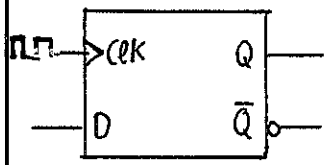
modo retención

C	D	Q_t	Q_{t+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

modo transparente

C	D	Q_{t+1}
0	X	
1	0	
1	1	

Flip-Flop D



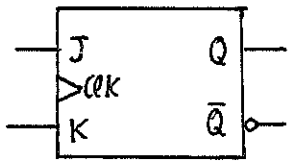
D	Q_t	Q_{t+1}
0	0	0
0	1	0
1	0	1
1	1	1

Solo deja pasar el dato D cuando se lo permite el reloj.

D	Q_{t+1}
0	0
1	1

$Q_{t+1} = D$

Flip-Flop J-K



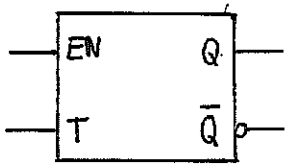
J	K	Q_t	Q_{t+1}	
0	0	0	0	*
0	0	1	1	
0	1	0	0	*
0	1	1	0	
1	0	0	1	
1	0	1	1	
1	1	0	1	
1	1	1	0	

Solo opera cuando se lo permite el reloj

J	K	Q_{t+1}
0	0	
0	1	
1	0	
1	1	

Q_t	Q_{t+1}	J	K	
0	0	0	X	*
0	1	1	X	
1	0	X	1	
1	1	X	0	

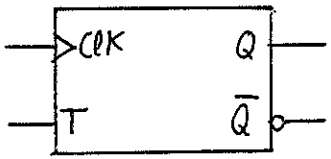
Biestable T asíncrono



EN	T	Q_t	Q_{t+1}
0	X	0	0
0	X	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

EN	T	Q_{t+1}
0	X	
1	0	
1	1	

Flip-Flop T

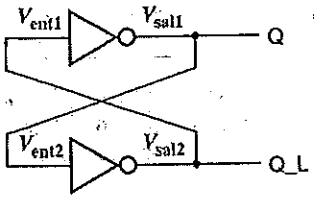


T	Q_t	Q_{t+1}
0	0	0
0	1	1
1	0	1
1	1	0

T	Q_{t+1}
0	
1	

4.2 Biestables

El circuito secuencial más sencillo consiste en un par de inversores que forman un ciclo de retroalimentación, como se muestra en la siguiente figura:



Como puedes ver, no tiene entradas y tiene dos salidas, *Q* y *Q_L*

Este circuito se denomina con frecuencia biestable, puesto que en un análisis estrictamente digital tiene dos estados estables:

- Si *Q* es ALTO, entonces el inversor inferior tiene una entrada en ALTO y una salida en BAJO, lo que fuerza un nivel ALTO a la salida del inversor superior, como supusimos en primer lugar.
- Pero si *Q* está en BAJO, entonces el inversor inferior tiene una entrada en BAJO y una salida en ALTO, lo que fuerza *Q* a BAJO, otra situación estable.

Podríamos por tanto utilizar una variable de estado simple, el estado de la señal *Q*, para describir el estado del circuito. Así, existen dos estados posibles: $Q = 0$ y $Q = 1$.

Un biestable tiene mucho más que mostrar si consideramos su funcionamiento desde el punto de vista analógico, estudiando conceptos de metaestabilidad, que se escapan del temario de EDIG.

Este elemento es tan simple que no tiene entradas y así no hay manera de controlar o modificar su estado. Cuando se aplica primero la energía al circuito, llega aleatoriamente a un estado o al otro y se mantiene ahí permanentemente.

Por supuesto, este elemento biestable que presentamos en un comienzo, evoluciona a otras formas, controlables mediante entradas, que exponemos a continuación.

4.2.1 Cerrojos y flip-flops: generalidades

Los cerrojos y flip-flops son circuitos secuenciales constituidos por puertas lógicas, capaces de almacenar un bit, la información binaria más elemental. Son los bloques de construcción básicos de la mayoría de los demás circuitos secuenciales.

Estos circuitos pueden ser síncronos o asíncronos:

Se dice que los flip-flops son dispositivos síncronos.

- Todos los diseñadores digiales utilizan el nombre de **flip-flop** para un dispositivo secuencial que necesita una señal de **reloj** (CLK) para ser activado.

  Esto significa que, aun-que cambie el valor de alguna de las entradas, la salida no cambia hasta un determinado momento marcado por el reloj. Normalmente esta activación sevproduce en el **flanco** de subida o bajada de la señal de reloj.

Se dice que los cerrojos son dispositivos asíncronos.

- Por otra parte, la mayoría de los diseñadores digitales utilizan el nombre de **cerrojo** (*latch*) para un dispositivo secuencial que no necesitan una señal de reloj.

  Esto es, la salida cambia cada vez que cambia una de las entradas.

Es importante saber que algunos textos y diseñadores digitales pueden emplear (incorrectamente) el nombre "flip-flop" para un dispositivo que en realidad es un cerrojo.

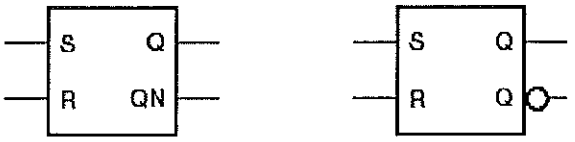
La clasificación de los biestables, desde el punto de vista de su constitución y del número de entradas puede resumirse en: **biestable R-S, biestable J-K, biestable T y biestable D.**

En EDIG a los cerrojos les llamamos básculas.

Encontrarás también estas básculas con el nombre de "Báscula S-R" (*set – reset*, de *establecimiento – restablecimiento*)

4.2.2 Báscula R-S

Una báscula R-S tiene como símbolo lógico es cualquiera de los siguientes:



El circuito tiene dos entradas, S y R, y dos salidas, etiquetadas como Q y QN, donde normalmente QN es el complemento de Q. La señal QN se representa en ocasiones como $\overline{Q}$ ó Q_L

Un poco más adelante comprobaremos que esta definición de QN no es completamente exacta.

Se trata de un biestable asíncrono (no está monitoreado por un reloj) . Su tabla de transiciones entre dos estados es la siguiente:

R	S	Q_t	$\overline{Q}_t$	Q_{t+1}
0	0	0	1	0
0	0	1	0	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	-

Q_t representa la salida Q en un determinado instante t.

$\overline{Q}_t$ representa la salida $\overline{Q}$ en un determinado instante t.

Q_{t+1} representa la salida Q en el siguiente instante, $t+1$.

Como se puede observar en la tabla si tanto S como R son 0, el circuito se comporta como el elemento biestable: tendremos un ciclo de retroalimentación que retiene uno o dos estados lógicos, $Q = 0$ ó $Q = 1$. Pero tanto S como R pueden ser asertivas (ponerse a 1) para forzar al ciclo de retroalimentación a un estado deseado:

- S establece la salida Q a 1.
- R restablece o limpia la salida Q a 0.

Recuerda que cuando decimos "*elemento biestable*" nos referimos al biestable básico, sin entradas, presentado en el comienzo de este punto 5.2

Después de que la entrada S ó R es negada de nuevo, la báscula permanece en el estado al cual fue forzado.

Toda esta información queda resumida en la tabla de transición siguiente, más concisa:

R	S	Q_{t+1}
0	0	Q_t
0	1	1
1	0	0
1	1	-

Inestabilidades:

Con esta observación queda patente que en un caso determinado, $\overline{Q}$ no es el complemento de Q, en la salida de una báscula R-S.

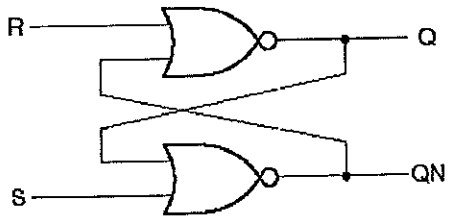
Cuando ambas entradas, S y R, están a 1, ambas salidas son forzadas a cero. Una vez que negamos cualquiera de las entradas, las salidas regresan a la operación complementaria como es habitual.

Sin embargo, si negamos ambas entradas de manera simultánea, la báscula se encamina a un estado siguiente impredecible y de hecho puede oscilar o entrar a un estado conocido como metaestable.

La metaestabilidad no es objeto de esta asignatura.

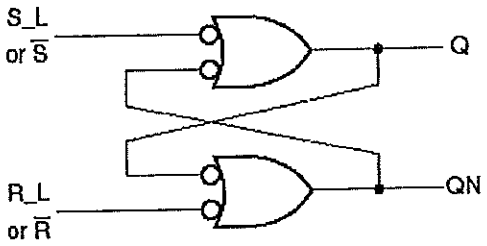
La metaestabilidad también puede presentarse se un pulso 1 que sea demasiado breve se aplica a S ó R.

4.2.2.1 Implementación de una báscula R-S con puertas NOR



Para este circuito valen todas las explicaciones de la página anterior, dado que esta es justo su implementación

4.2.2.2 Implementación de una báscula R-S con puertas NAND



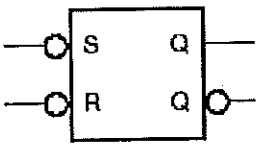
La báscula resultante de esta implementación suele recibir el nombre de Báscula $\overline{R}-\overline{S}$

Observa que las puertas OR con las entradas negadas son, en realidad, puertas NAND:

$$\overline{X}+\overline{Y}=\overline{X\cdot Y}$$

El resultado de esta implementación es una báscula con entradas de establecimiento y restablecimiento a nivel bajo.

A continuación tenemos su símbolo y su tabla de verdad:



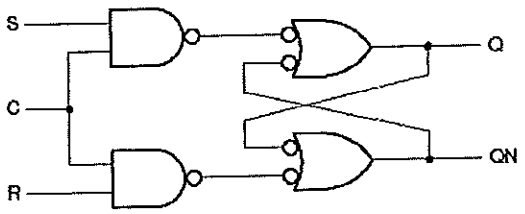
S_L	R_L	Q	QN
0	0	1	1
0	1	1	0
1	0	0	1
1	1	last Q	last QN

El funcionamiento de esta báscula $\overline{R}-\overline{S}$ es semejante al de la $R-S$, con dos diferencias principales:

- $\overline{R}$ y $\overline{S}$ son de nivel activo bajo, de modo que el cerrojo recuerda su estado anterior cuando $\overline{R}=\overline{S}=1$.
- Cuando tanto $\overline{R}$ como $\overline{S}$ son afirmadas de manera simultánea, ambas salidas de la báscula se van a 1, no a 0 como en la báscula $R-S$.

Las entradas de nivel bajo se indican claramente en el símbolo lógico de este circuito.

4.2.2.3 Báscula R-S con "enable"



C permite o impide que las entradas R y S lleguen a la báscula

Este circuito se comporta como una báscula R – S cuando C = 1, y retiene su estado anterior cuando C = 0.

Nota:

Si tanto S como R son 1 cuando C cambia de 1 a 0, el circuito se comporta como una báscula R – S en la cual S y R son negadas simultáneamente.

Esto es, el estado siguiente es impredecible y la salida puede llegar a ser metaestable

S	R	C	Q	QN
0	0	1	last Q	last QN
0	1	1	0	1
1	0	1	1	0
1	1	1	1	1
x	x	0	last Q	last QN

4.2.3 Latch D

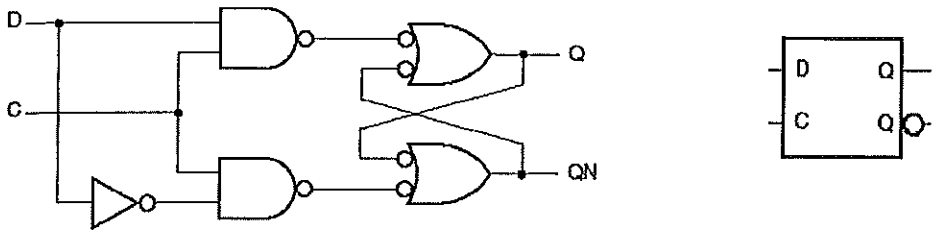
Los cerrojos tipo D son útiles en aplicaciones de control, donde con frecuencia pensamos en términos de establecer una marca en respuesta a alguna condición y restablecerla cuando cambian las condiciones.

Con frecuencia necesitamos cerrojos simplemente para almacenar bits de información: cada bit es presentado en una línea de señal, y nos gustaría almacenarlo en alguna parte.

Un cerrojo D puede emplearse en una aplicación de esta naturaleza.

La figura muestra un cerrojo D. Su diagrama lógico se reconoce como el de una báscula R-S con habilitación (enable), con un inversor agregado para generar entradas S y R a partir de la entrada simple D (de datos).

Esto elimina la situación problemática en los cerrojos R-S, donde R y S pueden afirmarse de manera simultánea.



La entrada de control de un cerrojo D, etiquetada C, se llama en ocasiones ENABLE, CLK, ó G.

La tabla de verdad de un latch D es la siguiente:

C	D	Q	QN
1	0	0	1
1	1	1	0
0	x	last Q	last QN

- Cuando C es afirmada, la salida Q sigue la entrada D. En esta situación se dice que el cerrojo está abierto y la trayectoria desde la entrada D a la salida Q es transparente.

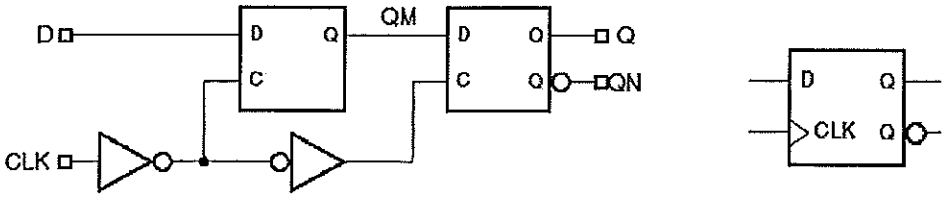
El circuito se denomina con frecuencia un "cerrojo transparente" por esta razón.

- Cuando la entrada C es negada, el cerrojo se cierra y la salida Q retiene su último valor y ya no cambia en respuesta a D, mientras que C permanezca negada.

4.2.4 Flip-flop D disparado por flanco

El primer cerrojo de la figura se conoce con el nombre de maestro; se abre y sigue la entrada cuando CLK es 0. Cuando CLK sube a 1, el cerrojo maestro se cierra y su salida se transfiere al segundo cerrojo, conocido como esclavo. Éste se encuentra abierto todo el tiempo que CLK es 1, pero solamente cambia al principio de este intervalo, porque el maestro está cerrado y sin modificaciones durante el resto del intervalo.

Un flip-flop D disparado por flanco positivo combina un par de cerrojos D como se ilustra en la siguiente figura:



El triángulo en la entrada CLK de los flip-flops D (en su símbolo lógico) indica un comportamiento de disparo por flanco, y se denomina **indicador de entrada dinámica.**

Así tenemos un circuito que muestrea su entrada D y cambia sus salidas Q y QN sólo para el flanco ascendente de la señal CLK de control.

Su tabla de verdad es la siguiente:

D	CLK	Q	QN
0	↓	0	1
1	↓	1	0
x	0	last Q	last QN
x	1	last Q	last QN

Podemos entender mejor el comportamiento de este biestable con la tabla de transiciones del circuito:

C	D	Q_t	Q_{t+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

La ecuación característica de un biestable D es:

$$Q_{t+1} = D$$

A las cuatro primeras filas se les llama modo retención y las cuatro últimas modo transparente

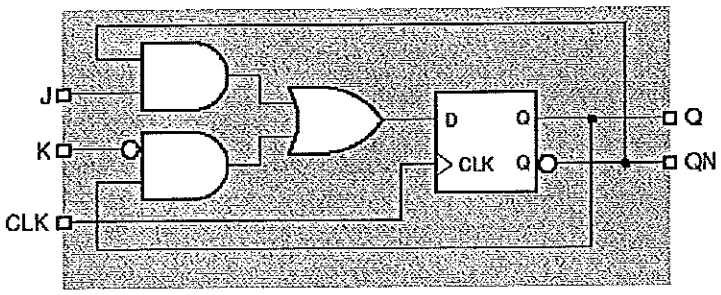
Observaciones:

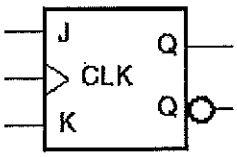
- Un flip-flop D disparado por flanco negativo simplemente invierte la entrada de reloj, de modo que toda la acción se desarrolla sobre el flanco descendente de una entrada CLK_L.
- Algunos flip-flops D tienen entradas asíncronas que se pueden utilizar para llevar al flip-flop a un estado particular, independientemente de las entradas CLK y D. Estas entradas, generalmente se identifican como PR (preset, prestablecimiento) y CLR (clear, borrado), se comportan como las entradas de establecimiento y restablecimiento de una báscula R-S.

4.2.5 Flip-flop J-K

El problema de qué hacer cuando en una báscula R-S, las entradas R y S se afirman de manera simultánea, se resuelve en un flip-flop J-K.

Se trata de un biestable síncrono, con la siguiente configuración:





Observa que este circuito utiliza un flip-flop D disparado por flanco.

Su tabla de verdad es:

J	K	CLK	Q	QN
x	x	0	last Q	last QN
x	x	1	last Q	last QN
0	0	↓	last Q	last QN
0	1	↓	0	1
1	0	↓	1	0
1	1	↓	last QN	last Q

Por supuesto, recuerda que la báscula RS es un circuito asíncrono.

- Las entradas J y K son análogas a S y R (de la báscula R-S).
- La novedad es que en este circuito, si J y K se afirman de manera simultánea, el flip-flop se irá al valor opuesto de su estado actual.

Así, su tabla de transiciones es la siguiente:

J	K	Q_t	Q_{t+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

La ecuación característica de un biestable J-K es:

$$Q_{t+1} = J\overline{Q}_t + \overline{K}Q_t$$

Esta información se puede expresar de manera más concisa en la siguiente tabla:

J	K	Q_{t+1}
0	0	Q_t
0	1	0
1	0	1
1	1	$\overline{Q}_t$

La aplicación más común de los flip-flops J-K se encuentra en las máquinas de estado síncronas temporizadas

De la anterior tabla de transiciones se puede extraer la siguiente tabla que es de gran utilidad para los ejercicios de autómatas:

Q_t	Q_{t+1}	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

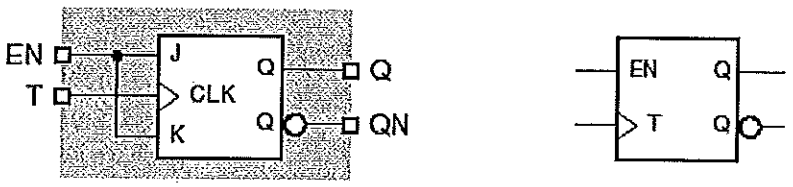
En esta tabla se interpreta que:

x = 0 ó 1

4.2.6 Flip-flop T

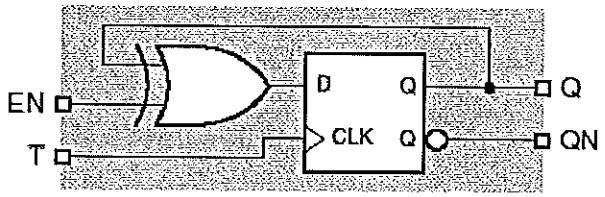
El circuito que aquí presentamos es en realidad un flip-flop T con habilitación (ENABLE).

Se trata de un biestable J-K al que se han cortocircuitado las entradas:



Este circuito es muy importante para realizar contadores.

Construido con un flip-flop D, el circuito queda:



En el ejercicio 1 de clase veremos otra modalidad de biestable T, muy parecida a esta.

Un flip-flop T cambia de estado con cada pulso de T, siempre y cuando la entrada de habilitación EN esté activa. Dicho de otra forma, se puede entender un biestable T como un circuito con la siguiente tabla de transición:

T viene de "*toggle*", conmutación

T	Q_t	Q_{t+1}
0	0	0
0	1	1
1	0	1
1	1	0

La tabla de la derecha muestra la misma información pero de forma más concisa.

T	Q_{t+1}
0	Q_t
1	$\overline{Q}_t$

La ecuación característica de un biestable T es:

$$Q_{t+1} = T \oplus Q_t$$

Estos son dos cálculos típicos en ejercicios de examen.

Los puntos 5.3.4.1 y 5.3.4.2 explican cómo debemos actuar con ante circuitos con un único biestable (además de puertas lógicas).

Observa que no interviene el tiempo de hold.

Observa que no interviene el tiempo de setup.

4.3.4 Cálculos de temporización

4.3.4.1 Periodo mínimo (ó frecuencia máxima) del reloj del circuito.

Para hallar este valor, calcularemos cuánto tarda en "volver" (**por el camino más largo**) una señal, desde la salida de un biestable hasta una de sus entradas. Este camino incluye:

- Tiempo de propagación del biestable, t_{FF}.
- Tiempos de propagación t_d de todas las puertas que se encuentre a su paso.
- Tiempo de setup del biestable, t_{setup}, necesario para que el biestable "se prepare" para su actuación en el siguiente flanco activo del reloj.

4.3.4.2 Máximo tiempo de hold de un biestable.

Para calcular el máximo tiempo de hold de un biestable, calcularemos cuánto tarda en "volver" (**por el camino más corto**) una señal, desde la salida del biestable hasta una de sus entradas.

Este será el tiempo real que van a tardar en cambiar sus entradas después del flanco activo, y por lo tanto este es el **tiempo máximo que debe necesitar de hold el biestable**. Si necesitase más, nuestro circuito no funcionaría con la topología actual.

Este camino incluye:

- Tiempo de propagación del biestable, t_{FF}.
- Tiempos de propagación t_d de todas las puertas que se encuentre a su paso.

Observaciones para circuitos con más de un biestable:

- Para circuitos con más de un biestable, los caminos que buscaremos serán desde la salida de cualquier biestable, hasta la entrada de cualquier biestable. Por supuesto, podemos llegar al mismo biestable del que partimos.

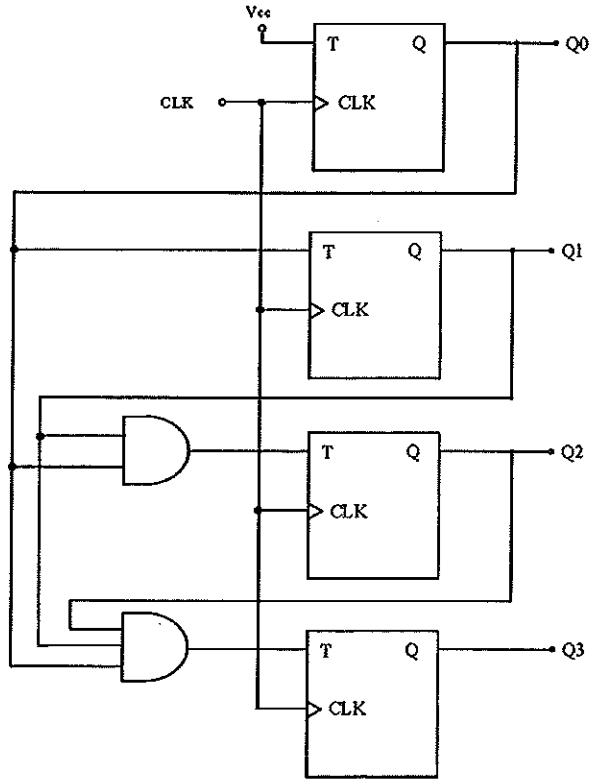
- Lo importante es que exploremos TODOS los caminos posibles y elijamos:

  - El más largo para los cálculos de frecuencia máxima.
  - El más corto para los cálculos del máximo tiempo de hold.

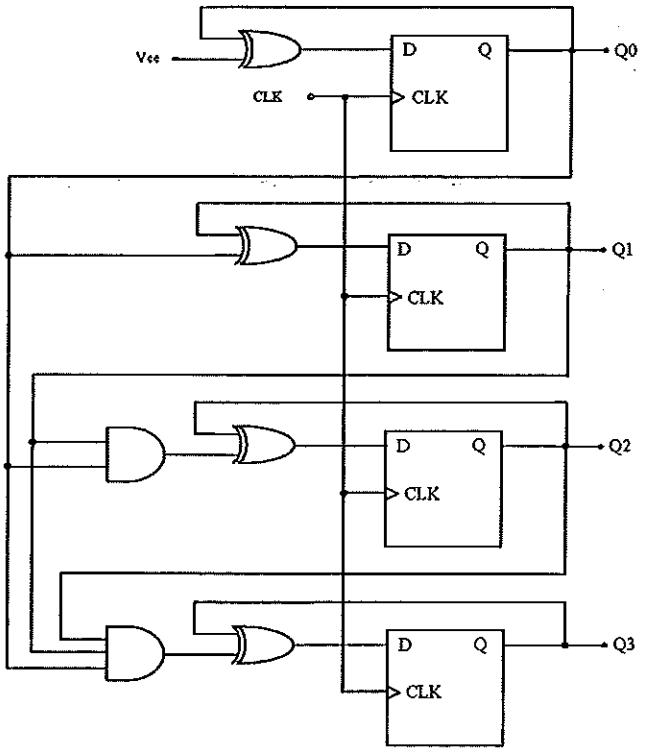
4.4 Contadores

Los contadores son circuitos secuenciales síncronos implementados internamente mediante biestables. Si bien su función varía según su topología, se puede decir en general que estos circuitos realizan una cuenta binaria, que podremos programar para forzar su inicio, su final, y su sentido de cuenta.

4.4.1 Implementación mediante flip-flops T



4.4.2 Implementación mediante flip-flops D



4.4.3 Contadores comerciales

En el caso de los contadores comerciales (como cualquier integrado comercial) el fabricante siempre proporciona una hoja de especificaciones donde aparece un esquema del patillaje y un cronograma (*Timing Diagram*). Este cronograma proporciona normalmente toda la información necesaria para conocer el funcionamiento del contador.

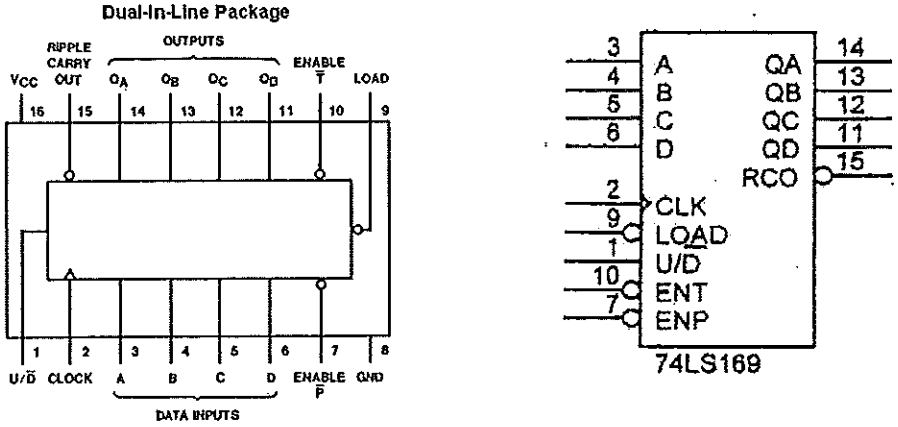
4.4.3.1 Contador 74LS169

El resto de contadores comerciales tienen funcionamientos similares

A modo de ejemplo, vamos a estudiar con detalle un modelo determinado, el 74LS169, cuyo esquema y cronograma mostramos a continuación.

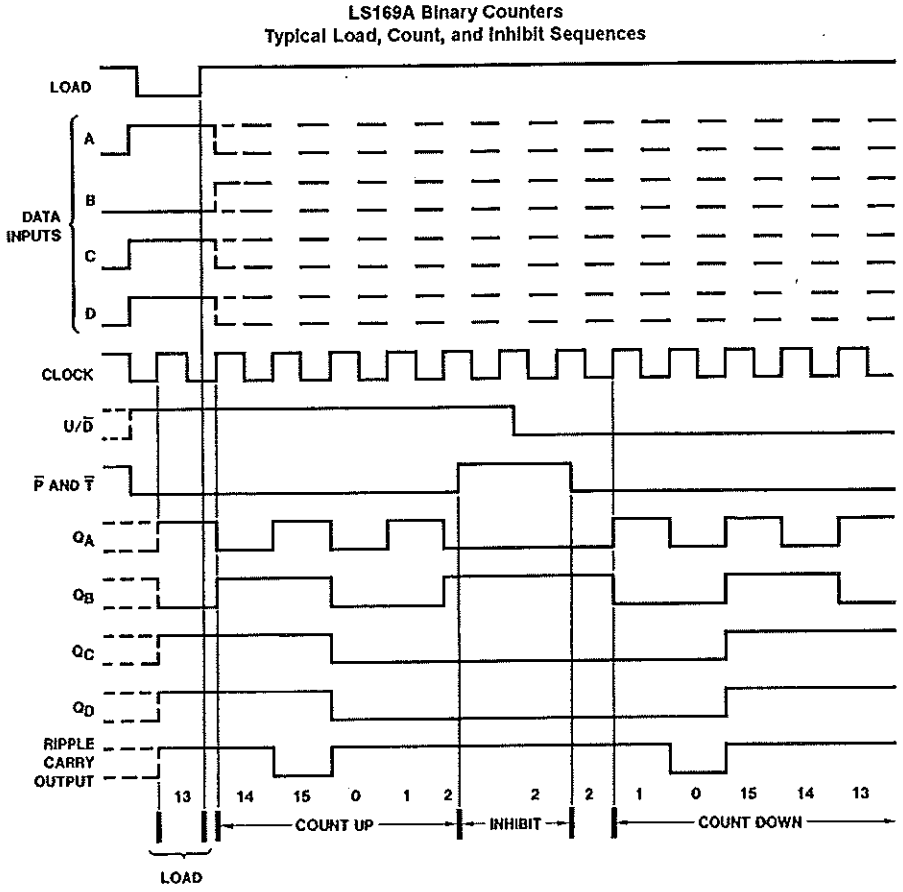
A la izquierda mostramos el patillaje del integrado, tal y como se comercializa. A la derecha, su símbolo lógico.

Observa que en el símbolo lógico se dibujan las entradas a la izquierda y las salidas a la derecha



Timing Diagram

Las líneas a trozos de las entradas A B C D significan que pueden tomar el valor "0" ó "1" indistintamente.



Descripción del patillaje del 74LS169

Entradas A B C D

D' = decimal
B' = binario

En estas entradas podemos poner el número que nosotros queramos en binario desde el D'0=B0000 hasta el D'15=B'1111. siempre teniendo en cuenta que A es el bit menos significativo y D es el más significativo.

Este número sólo pasa a las salidas Q_A Q_B Q_C Q_D cuando se activa el LOAD.

Entrada CLK

Es la entrada del reloj. El contador sólo "funciona", es decir, solo lee las entradas y saca las salidas, cuando se produce el flanco activo del reloj (que en nuestro caso, es el de subida). El resto del tiempo el contador no hace nada.

Entrada LOAD

La bolita que hay en esta entrada señala que es activa a nivel bajo. Esto quiere decir que el LOAD solo se activa cuando ponemos un "0" en esta entrada.

Cuando el LOAD se activa el valor de las entradas A B C D es transferido a las salidas Q_A Q_B Q_C Q_D de forma inmediata y con independencia de lo que hubiese en las salidas anteriormente.

Mientras LOAD permanezca a "1" las entradas A B C D no se utilizan para nada.

Entrada U/D (*Up / Down*)

Indica al contador si tiene que contar hacia arriba o hacia abajo. Si ponemos un "1" en esta entrada entonces el contador, en cada flanco activo del reloj, aumentará en una unidad el valor anterior de las salidas. Sin embargo si ponemos un "0" el contador disminuirá en una unidad el valor anterior de las salidas.

Entradas ENT y ENP

Estas son las entradas de ENABLE y las activaremos o desactivaremos siempre conjuntamente (es decir, las dos a "1" o las dos a "0").

La bolita significa que son activas a nivel bajo.

Mientras ambas valen "0" el contador funciona correctamente pero si las dos valen "1" entonces el contador se inhibe (*inhibit*) y la salida ya no cambia más (se queda fija) en los sucesivos flancos del reloj; esto equivale a "apagar" el contador.

Salidas Q_A Q_B Q_C Q_D

En estas salidas veremos aparecer, en cada flanco del reloj, un número entre el D'0=B0000 hasta el D'15=B'1111.

Si el contador está inhibido (ENT=ENP="1") aparecerá siempre el mismo número y si no lo está ese número irá cambiando siguiendo la secuencia que hayamos programado.

La salida de un contador puede servir, por ejemplo, para iluminar un display de siete segmentos como el de un reloj digital.

Salida RCO (*Ripple Carry Output*)

Esta salida avisa de cuándo el contador ha llegado al tope. En nuestro caso avisa si el contador llega a D'15=B'1111 si está subiendo o avisa si el contador llega a D'0=B'0000 si el contador está bajando.

La bolita significa que es activo a nivel bajo.

Por ser activo a nivel bajo, saldrá un "1" mientras la cuenta no haya llegado al tope y sale un "0" si llega al tope (es decir, 15 si está subiendo o 0 si está bajando).

Estos circuitos también reciben el nombre de "Registros de corrimiento".

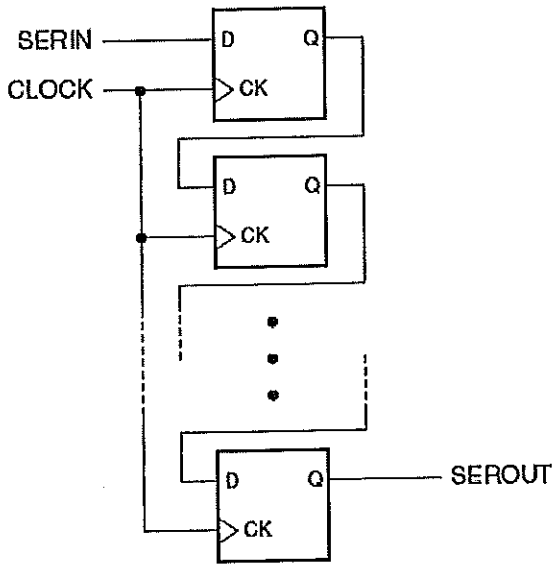
4.5 Registros de desplazamiento

Un registo de desplazamiento es un registro de n bits con una disposición para recorrer sus datos almacenados por una posición de bit en cada tic del reloj. Al igual que los contadores los registros son circuitos secuenciales síncronos implementados internamente mediante biestables.

4.5.1 Registros de desplazamiento de entrada serie y salida serie

La figura siguiente ilustra la figura de un registro de estas características:

Un registro de desplazamiento de n bits de entrada serie y salida serie puede emplearse para retardar una señal en n tics del reloj.

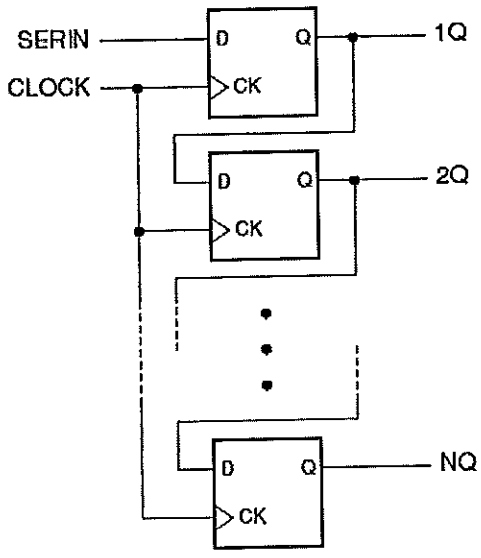


- La entrada serie SERIN especifica un nuevo bit que será desplazado en un extremo para cada tic del reloj.
- Este bit aparece en la salida serie, SEROUT, después de n tics del reloj, y se pierde un tic más tarde.

4.5.2 Registros de desplazamiento de entrada serie y salida paralelo

La figura siguiente ilustra la figura de un registro de estas características:

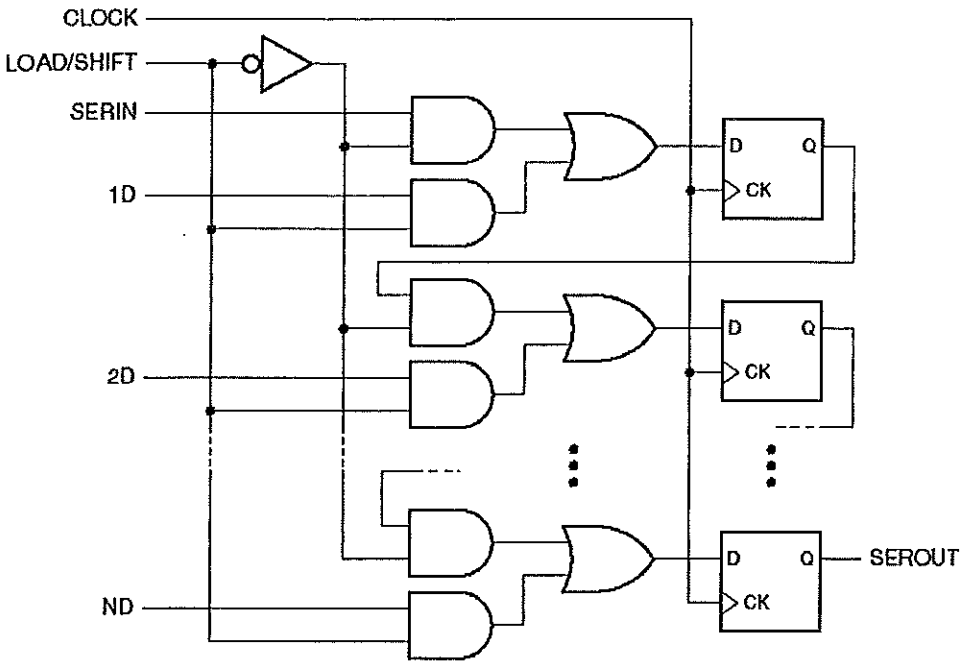
Un registro así puede utilizarse para efectuar la conversión serie a paralelo, muy utilizada en electrónica digital.



Como se puede observar, tiene salidas para todos sus bits almacenados, haciéndolos quedar disponibles para otros circuitos.

4.5.3 Registros de desplazamiento de entrada paralelo y salida serie

Este circuito funciona a la inversa que el anterior, como se puede observar en la figura siguiente:

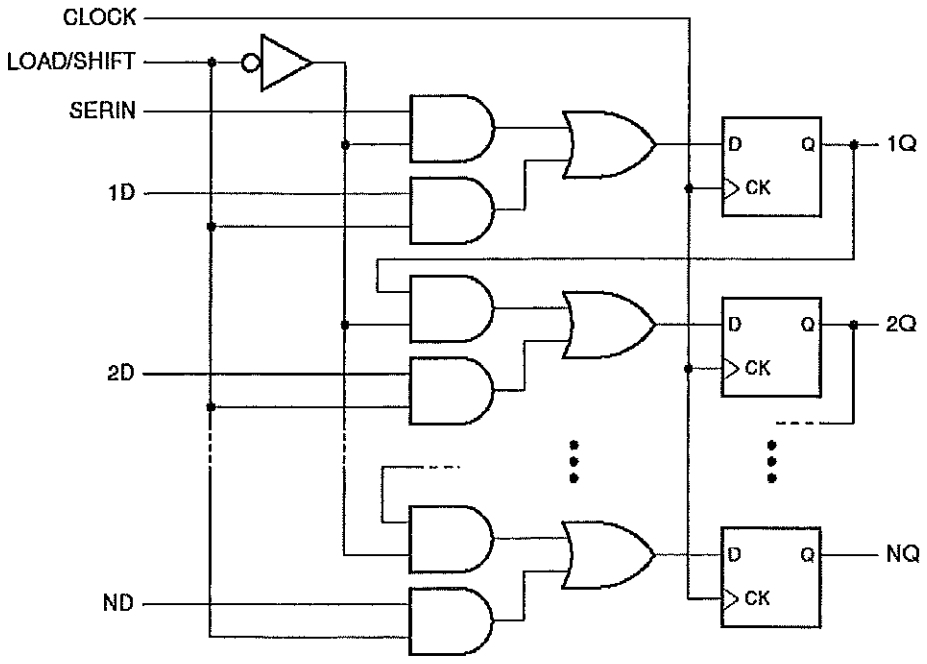


Un registro así puede utilizarse para efectuar la conversión paralelo a serie.

Para cada tic del reloj el registro carga nuevos datos de las entradas 1D – ND, ó recorre su contenido actual, dependiendo del valor de la entrada de control LOAD/SHIFT.

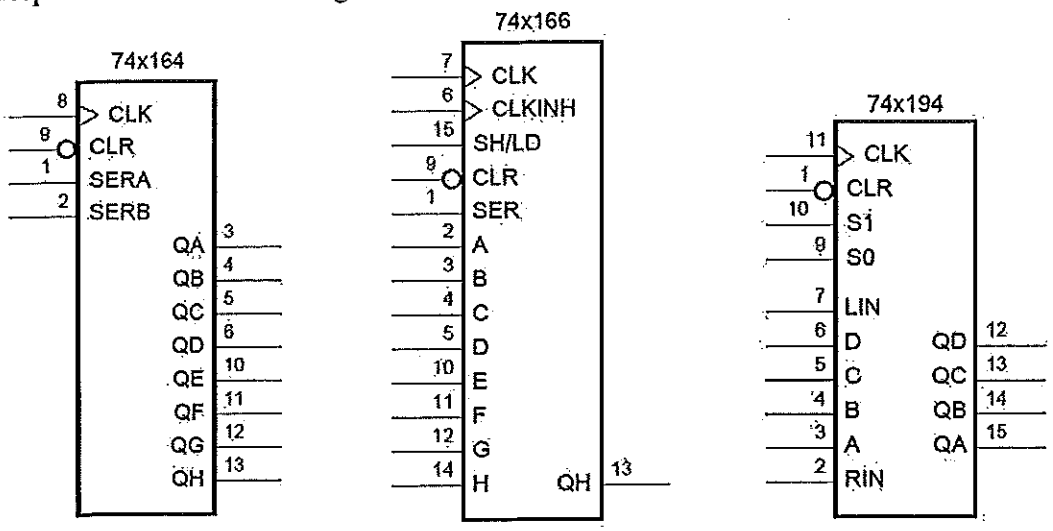
4.5.4 Registros de desplazamiento de entrada paralelo y salida paralelo

Al proporcionar salidas para todos los bits almacenados en un registro de desplazamiento de entrada en paralelo, obtenemos el registro de desplazamiento de entrada paralelo y salida paralelo de la siguiente figura:



4.5.5 Registros comerciales

La siguiente figura muestra los símbolos lógicos para tres populares registros de desplazamiento de 8 bits integrados:



4.5.5.1 Registro 74x164

Los integrados '164 y '166 son registros de desplazamiento unidireccionales, dado que efectúan el desplazamiento en una sola dirección.

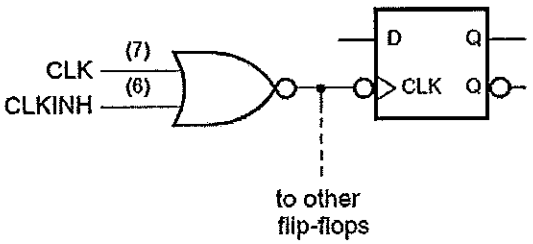
Es un dispositivo de entrada serie y salida paralelo con una entrada de borrado asíncrona (CLR_L). Tiene dos entradas serie que se operan con AND de manera interna, es decir, tanto SERA como SERB deben ser 1 para ser recorridos en el primer bit del registro.

4.5.5.2 Registro 74x166

Es un registro de desplazamiento de entrada paralelo y salida serie, también con entrada asíncrona de borrado. El dispositivo efectúa el desplazamiento cuando SH/LD es 1 y de otro modo carga nuevos datos.

Los diseñadores del '166 lo destinaron para que CLK sea colocada a un reloj de sistema de carrera libre y para que CLKINH sea asertiva para inhibir CLK.

El '166 tiene un arreglo de temporización poco habitual llamado "reloj de compuerta": tiene dos entradas de reloj conectadas a los flip-flops internos como se muestra en la figura de la derecha:



4.5.5.3 Registro 74x194

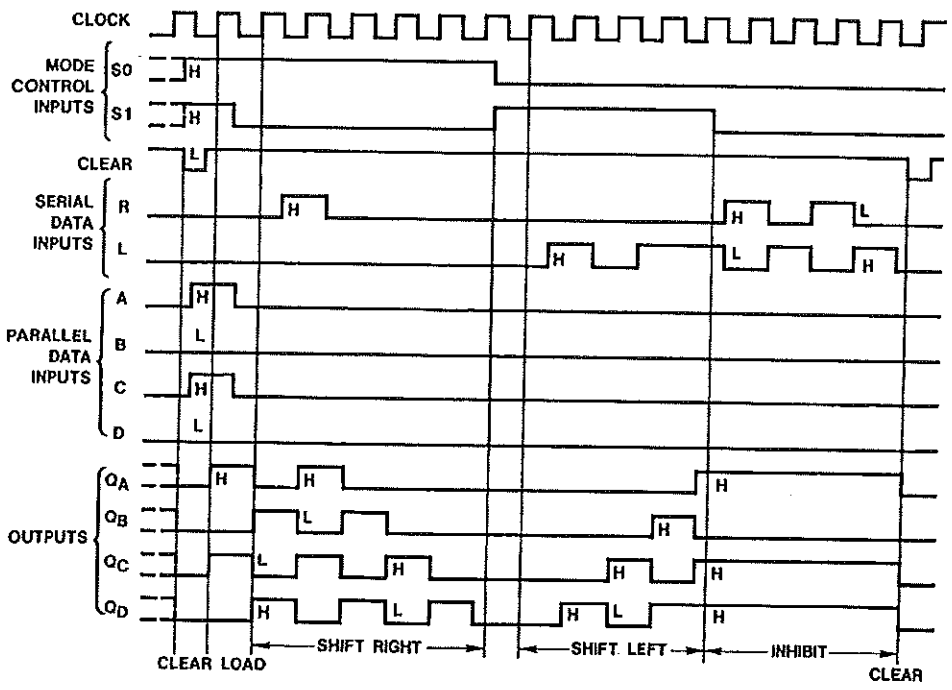
El '194 es un registro de desplazamiento bidireccional porque su contenido puede ser desplazado en cualquiera de dos direcciones, dependiendo de una entrada de control.

A modo de ejemplo, nosotros vamos a estudiar con más detalle detalle un modelo determinado, el 74LS194. Se trata de un registro de desplazamiento bidireccional de entrada en paralelo y salida en paralelo de 4 bits.

A continuación, mostramos su function table y cronograma:

Function Table

Inputs										Outputs			
Clear	Mode		Clock	Serial		Parallel				Q_A	Q_B	Q_C	Q_D
	S1	S0		Left	Right	A	B	C	D				
L	X	X	X	X	X	X	X	X	X	L	L	L	L
H	X	X	L	X	X	X	X	X	X	Q_{A0}	Q_{B0}	Q_{C0}	Q_{D0}
H	H	H	↑	X	X	a	b	c	d	a	b	c	d
H	L	H	↑	X	H	X	X	X	X	H	Q_{An}	Q_{Bn}	Q_{Cn}
H	L	H	↑	X	L	X	X	X	X	L	Q_{An}	Q_{Bn}	Q_{Cn}
H	H	L	↑	H	X	X	X	X	X	Q_{Bn}	Q_{Cn}	Q_{Dn}	H
H	H	L	↑	L	X	X	X	X	X	Q_{Bn}	Q_{Cn}	Q_{Dn}	L
H	L	L	X	X	X	X	X	X	X	Q_{A0}	Q_{B0}	Q_{C0}	Q_{D0}



Descripción del patillaje del 74LS194

Entradas A B C D

D' = decimal
B' = binario

En estas entradas podemos poner el número que nosotros queramos en binario desde el D'0=B0000 hasta el D'15=B'1111. Este número solo pasa a las salidas Q_A Q_B Q_C Q_D cuando se activa S1="1", S0="1".

Entrada CLK

Es la entrada del reloj. El registro solo "funciona", cuando se produce el flanco activo del reloj (que en nuestro caso, es el de subida). El resto del tiempo el registro no hace nada.

Entrada CLEAR

Si ponemos esta patilla a "1" no hace nada y si la ponemos a "0" ponemos todas las salidas a cero Esta entrada es asíncrona, esto quiere decir que las salidas se ponen a "0" en el mismo instante en que poenemos un "0" en la entrada CLEAR sin necesidad de esperar al flanco activo.

Entradas S0 S1

Vemos las cuatro combinaciones posibles:

Se comporta como el LOAD de un contador

- S1 = "1" S0 = "1"

  Cuando llega el flanco activo el valor de las entradas A B C D es transferido a las salidas Q_A Q_B Q_C Q_D con independencia de lo que hubiese en las salidas anteriormente.

Se comporta como el ENABLE de un contador

- S1 = "0" S0 = "0"

  En este caso el registro se inhibe (*inhibit*) y las salidas ya no cambian más (se quedan fijas) en los sucesivos flancos del reloj; esto equivale a "apagar" el registro.

En la Function Table al valor anterior de las salidas los llaman Q_{An}, Q_{Bn}, Q_{Cn}, Q_{Dn}

- S1 = "1" S0 = "0"

  Cada vez que llega el flanco activo pone el valor de Serial_Left en la salida Q_D, traslada el anterior valor de Q_D a la salida Q_C, el anterior valor de Q_C a Q_B y el anterior valor de Q_B a Q_A. Es decir, *desplaza* las salidas hacia la izquierda (o hacia arriba, según se mire). Importante: El valor anterior de Q_A se pierde.

Es análogo al caso anterior pero en el otro sentido

- S1 = "0" S0 = "1"

  Cada vez que llega el flanco activo pone el valor de Serial_Right en la salida Q_A, traslada el anterior valor de Q_A a la salida Q_B, el anterior valor de Q_B a Q_C y el anterior valor de Q_C a Q_D. Es decir, desplaza las salidas hacia la hacia la derecha (o hacia abajo, según se mire). Importante: El valor anterior de Q_D se pierde.

Entrada Serial Left

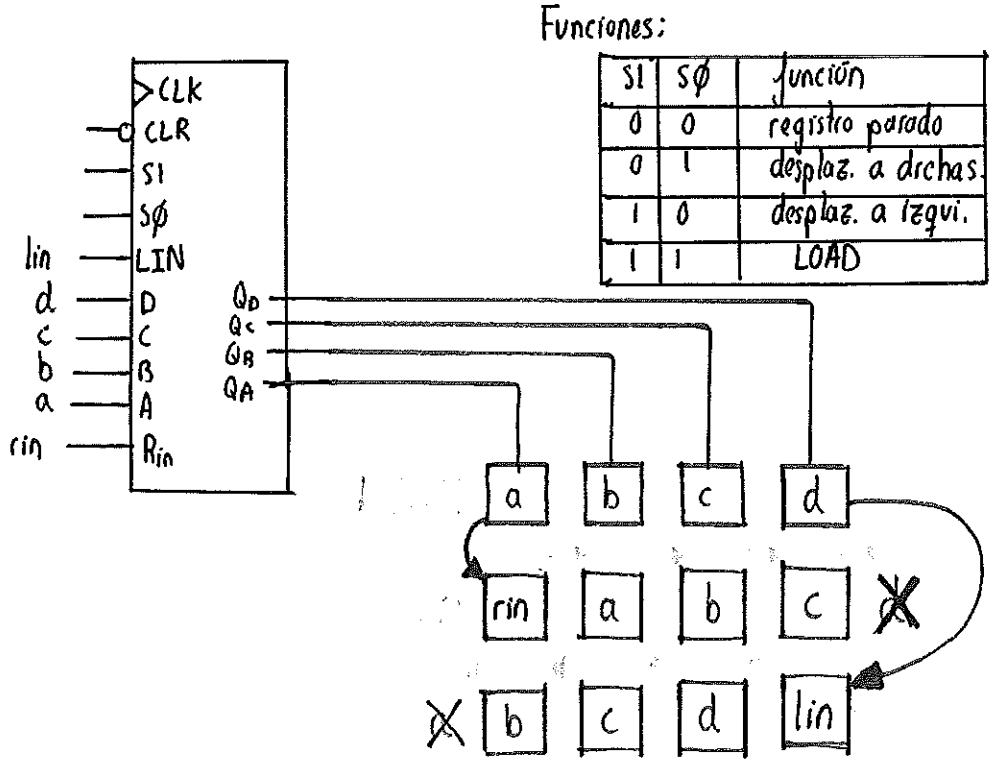
En esta entrada podemos poner el valor que nosotros queramos (0 ó 1) que solo se utilizará en el caso en que tengamos S1="1", S0="0" (desplazamiento hacia la derecha). En ese caso el valor de Serial_Left será transferido a Q_D.

Entrada Serial Right

En esta entrada podemos poner el valor que nosotros queramos (0 ó 1) que solo se utilizará en el caso en que tengamos S1="0", S0="1" (desplazamiento hacia la), en ese caso el valor de Serial_Right será transferido a Q_A.

Salidas Q_A Q_B Q_C Q_D

En estas salidas veremos aparecer, en cada flanco del reloj, un valor entre el D'0=B0000 hasta el D'15=B'1111.



TEMA 5: AUTÓMATAS

Así pues, desde ahora usaremos el término de máquina de estado o autómata indistintamente.

En este tema vamos a estudiar el funcionamiento de autómatas, que son **máquinas de estado síncronas temporizadas**.

- "Máquina de estado" es un nombre genérico dado a estos circuitos secuenciales.
- "Temporizada" hace referencia al hecho de que sus elementos de almacenamiento (flip-flops) emplean una entrada de reloj.
- "Síncrona" significa que todos los flip-flops utilizan la misma señal de reloj. Como ya hemos estudiado, una máquina de estado de esta naturaleza cambia de estado solamente cuando se presenta un flanco de disparo o "pulso" en la señal de reloj.

Formalmente, se puede decir que un autómata es una quíntupla formada por:

- Un conjunto E finito de entradas
- Un conjunto S finito de salidas
- Un conjunto Q de estados
- Una función de transición (también llamada "de estado siguiente): $f : E \times Q \rightarrow Q$
- Una función de salida: $f : E \times Q \rightarrow S$

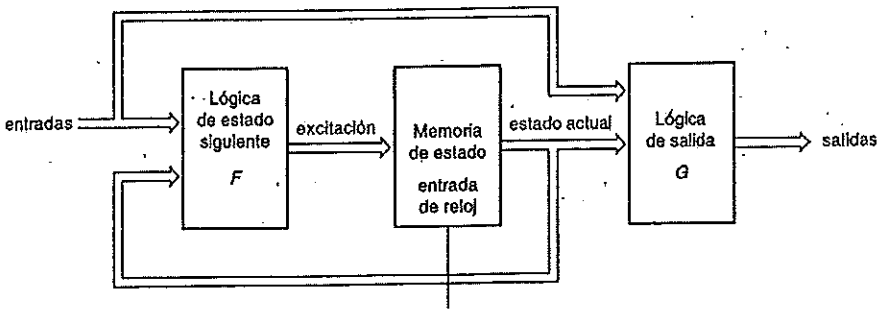
Si el conjunto de estados Q es finito entonces decimos que es un **autómata finito.** Nosotros estudiaremos solamente autómatas finitos.

A lo largo de los siguientes apuntes nos referiremos continuamente a dos de estas máquinas, que aprenderemos a analizar y diseñar: la máquina de Mealy y la de Moore.

Máquina de Mealy

En la máquina de Mealy la salida depende del estado en que se encuentra la máquina y de las entradas, por tanto la salida no es síncrona con el reloj.

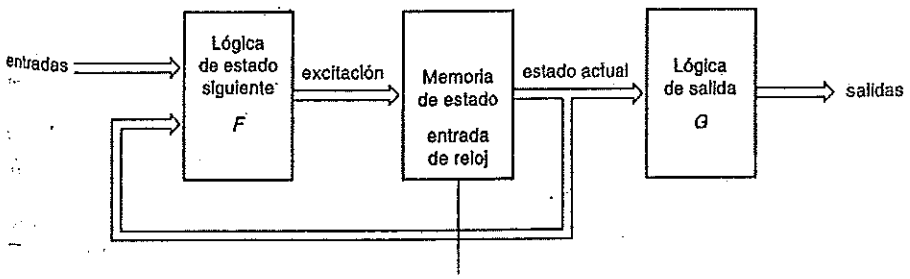
Se puede decir que en una máquina de Mealy las salidas están asociadas a las transiciones, no a los estados.



Maquina de Moore

En la máquina de Moore la salida depende exclusivamente del estado en que se encuentra la máquina.

Se puede decir que en la máquina de Moore las salidas están asociadas a los estados, o lo que es lo mismo, todas las transiciones que conducen a un mismo estado tienen asociada la misma salida.



5.1 Análisis de autómatas

5.1.1 Estructura de una máquina de estado

En las figuras de la página anterior ilustramos la estructura general de una máquina de estado síncrona temporizada. La memoria de estado es un conjunto de n flip-flops que almacenan el estado actual de la máquina, y tiene 2^n estados distintos. Los flip-flops se encuentran todos conectados a una señal de reloj común que ocasiona que cambien de estado en cada pulso del reloj.

En la máquina de Mealy:

- El estado siguiente está determinado por la *lógica de estado siguiente F*, como una función de la entrada y estado actuales.
- La *lógica de salida G* determina la salida como una función de la entrada y el estado actuales.

Tanto *F* como *G* son estrictamente circuitos lógicos combinacionales.

Estado siguiente = F(estado actual, entrada)
Salida = G(estado actual, entrada)

Máquina de Mealy

En la máquina de Moore:

- El estado siguiente está determinado por la *lógica de estado siguiente F*, como una función de la entrada y estado actuales.
- La *lógica de salida G* determina la salida como una función de la sólo del estado actual.

Estado siguiente = F(estado actual, entrada)
Salida = G(estado actual)

Máquina de Moore

Obviamente, la única diferencia entre los dos modelos de máquina de estado se encuentran en cómo son generadas las salidas. En la práctica, muchas máquinas de estado deber ser categorizadas como máquinas de Mealy, porque tienen una ó más *salidas del tipo Mealy* que dependen de la entrada así como también del estado. Sin embargo, muchas de estas máquinas también tienen una ó más *salidas de tipo Moore* que dependen solamente del estado.

Observación

Las máquinas de estado suelen emplear flip-flops D disparados por flanco positivo para su memoria de estado, aunque también es posible hacer uso de flip-flops D disparados por flanco negativo, cerrojos D ó flip-flops J-K.

5.1.2 Ecuaciones características

Como ya vimos en el tema 5, el comportamiento funcional de un cerrojo o flip-flop puede describirse formalmente mediante una ecuación característica que especifica el siguiente estado del flip-flop como una función de su estado y entradas actuales

Las ecuaciones características de los flip-flops del tema 5 se enumeran a continuación:

Tipo de dispositivo	Ecuación característica
Latch D	$Q_{t+1} = D$
Flip-flop D disparado por flanco	$Q_{t+1} = D$
Báscula R-S	$Q_{t+1} = S + \overline{R} \cdot Q_t$
Flip-flop J-K	$Q_{t+1} = J\overline{Q}_t + \overline{K} \cdot Q_t$
Flip-flop T con habilitación (teoría)	$Q_{t+1} = T \cdot \overline{Q}_t + \overline{T} \cdot Q_t$
Flip-flop T (problemas)	$Q_{t+1} = \overline{Q}_t$

5.1.3 Análisis de máquinas de estado con flip-flops D

El objetivo del análisis de circuito secuencial es determinar las funciones de salida y estado siguiente de modo que pueda predecirse el comportamiento de un circuito.

El análisis de una máquina de estado síncrona temporizada tiene siete pasos básicos:

1. Determinar las **ecuaciones de excitación** para las entradas de control del flip-flop, que son ecuaciones lógicas que expresan las señales de excitación como funciones del estado y entrada actuales. Estas ecuaciones pueden obtenerse del diagrama del circuito.

Para biestables D, recordemos que en el flanco ascendente de la señal de reloj, cada flip-flop D muestrea su entrada D y transfiere este valor a su salida Q. Por lo tanto, para determinar el siguiente valor de Q (es decir, Q_{t+1}), primero deberemos determinar el valor actual de D.

2. Sustituir las ecuaciones de excitación en las ecuaciones características del flip-flop para obtener las **ecuaciones de transición.** Estas ecuaciones expresan el valor siguiente de las variables de estado como una función del estado y entrada actuales.
3. Hacer uso de las ecuaciones de transición para construir una **tabla de transición.** Para ello, evaluamos dichas ecuaciones para cada posible combinación de estado/entrada. Tradicionalmente, una tabla de transión enumera los estados en la izquierda y las combinaciones de entrada en la parte superior de la tabla.
4. Determinar las **ecuaciones de salida.**
5. Agregar los valores de salida a la tabla de transición para cada combinación de estado (Moore) o de estado/entrada (Mealy) y crear una **tabla de transición/salida.**
6. Nombrar los estados y sustituir nombres de estado por combinaciones de variables de estado en la tabla de transición/salida para obtener una **tabla de estado/salida.**
7. Dibujar un **diagrama de estado** correspondiente a la tabla de estado/salida.

Este apartado séptimo es opcional.

Un diagrama de estado presenta la información de la tabla de estado/salida en formato gráfico.

- Tiene un círculo (ó nodo) por cada estado, y una flecha (ó arco dirigido) para cada transición.
- La letra dentro de cada círculo es un nombre de estado. Cada flecha que abandona un estado dado apunta al siguiente estado para una combinación de entrada dada.
- También muestra el valor de salida producido en el estado dado para esa combinación de entrada (si la máquina es de Mealy) ó puede mostrar los valores de salida dentro de cada círculo de estado (si la máquina es de Moore, puesto que son funciones de estado solamente).

Observación

En una máquina con n entradas, tendríamos 2^n flechas dejando cada estado. Esto es complicado si n es grande. Por convención, solamente usaremos una flecha para cada distinto estado siguiente.

5.1.4 Análisis de máquinas de estado con flip-flops J-K

Las máquinas de estado síncronas temporizadas que se construyen a partir de flip-flops J-K también se pueden analizar aplicando el procedimiento básico que se indicó en el capítulo anterior. La única diferencia es que existen dos ecuaciones de excitación para cada flip-flop: una para J y otra para K.

Para obtener las ecuaciones de transición, ambas deben sustituirse en la ecuación característica del biestable J-K, descrita en la tabla de la página anterior.

5.2 Diseño de autómatas

Los pasos de diseño de circuitos secuenciales, comenzando a partir de una especificación o descripción en palabras, son justamente el inverso de los pasos de análisis que describimos en el capítulo 6.1:

1. Pasar las especificaciones verbales a un **diagrama de estados**.

Algunos textos recomiendan que el primer paso sea la construcción de la tabla estado/salida

2. Asignar **códigos** a los estados.
3. Construir la **tabla de transiciones** entre estados.
   - Si la máquina es Mealy hace falta además incluir las salidas asociadas a cada transición.
   - Si la máquina es de Moore se puede hacer una tabla auxiliar con las salidas asociadas a cada estado en vez de incluirlas en la tabla de transiciones.
4. Seleccionar los **elementos de memoria** (biestables J-K, D, ...). Esto lo dicen en el examen.
5. Obtener las **tablas de excitación** que muestren los valores de excitación requeridos para obtener el siguiente estado deseado para cada combinación de estado/entrada.

Sólo asimilaremos bien estos pasos cuando los abordemos directamente en ejercicios.

6. Simplificar las **ecuaciones de excitación**, a partir de la tabla de excitación.
7. Implementar el circuito.

APÉNDICE 1: PUERTAS LÓGICAS

Abajo puedes encontrar ejemplos de componentes comerciales que contienen estas puertas lógicas:

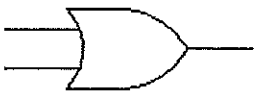
Puerta AND

74HC08

X	Y	X AND Y
0	0	0
0	1	0
1	0	0
1	1	1

Puerta OR

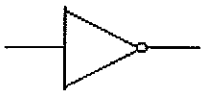
74HC32



X	Y	X OR Y
0	0	0
0	1	1
1	0	1
1	1	1

Puerta NOT

74HC04



X	NOT X
0	1
1	0

Puerta NAND

74HC00

X	Y	X NAND Y
0	0	1
0	1	1
1	0	1
1	1	0

Puerta NOR

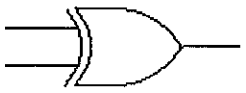
74HC02

X	Y	X NOR Y
0	0	1
0	1	0
1	0	0
1	1	0

NOTA: Las puertas NAND y NOR son típicamente más baratas y fáciles de fabricar.

Puerta XOR

74HC86



X	Y	X XOR Y
0	0	0
0	1	1
1	0	1
1	1	0

Puerta XNOR

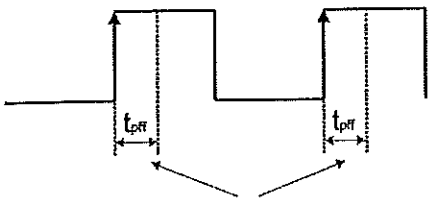
74HC86

X	Y	X XNOR Y
0	0	1
0	1	0
1	0	0
1	1	1

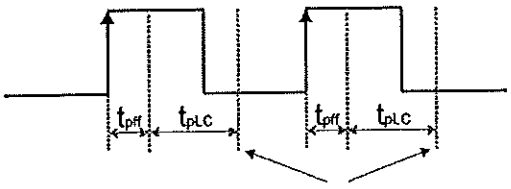
APÉNDICE 2: METAESTABILIDAD, NORMAS DE DISEÑO SÍNCRONO

Bases del funcionamiento síncrono

- Las salidas de los circuitos secuenciales sólo pueden cambiar de estado en los **flancos activos de reloj.**
- El **régimen transitorio** de los circuitos finaliza cuando ha transcurrido el tiempo de propagación máximo del circuito desde el último flanco activo de reloj.

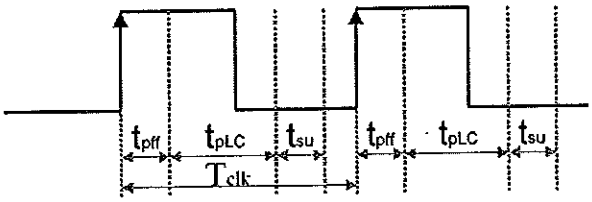


Cambio de estado



Fin del Régimen Transitorio

- Para que las salidas de los circuitos combinacionales (conectadas a la entrada de biestables) puedan registrarse correctamente deberán ser estables un tiempo antes del flanco activo de reloj, el **tiempo de set-up** de los flip-flops.



Por tanto: $T_{clk} > t_{pff} + t_{pLC} + t_{su}$

- La **frecuencia máxima** de la señal de reloj en un circuito secuencial síncrono viene dada por la expresión:

$$f_{CLK\max} = \frac{1}{t_{pff\max} + t_{pLC\max} + t_{su\min}}$$

Donde $t_{pLC\max}$ es el tiempo de propagación del bloque combinacional más lento de los existentes en el circuito.

Normas de diseño síncrono

Un circuito digital síncrono funcionando con una frecuencia de reloj menor o igual a la dada por la expresión anterior funcionará correctamente si:

- No se activan, durante la operación normal del sistema, las entradas asíncronas de los flip-flops.
- Las **entradas síncronas** de los flip-flops no están alimentadas por **señales asíncronas.**
- Las **entradas asíncronas** de los flip-flops no están alimentadas por **señales síncronas.**
- No existe lógica combinacional realimentada.
- Todas las entradas de los circuitos combinacionales, incluso las externas al sistema, están registradas.

Ver el apartado: "sincronización de entradas asíncronas".

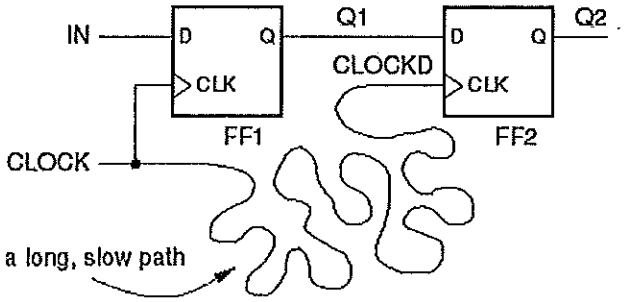
- Se emplean **flip-flops activos en el mismo tipo de flanco** como elementos de memoria del sistema.

- A todos los flip-flops les llega **de manera simultánea la señal de reloj** del circuito.

  Esto, en general, no es posible que se verifique de manera estricta; el reloj llegará con cierto desfase a las entradas de los flip-flops debido a las distintas longitudes de las pistas y a las distintas cargas que soportan los buffers del árbol de reloj.

El desfase en la llegada del reloj a los flip-flops de un circuito se denomina *skew* del reloj. Un circuito síncrono puede admitir un valor máximo de skew en la señal de reloj. Vamos a verlo con más detenimiento:

En la figura, el *skew*: es la diferencia de tiempo entre los flancos de CLOCK y CLOCKD



Cálculo del "skew"

No hay problemas en un circuito si se cumple que:

$$t_{p\text{-}ff\,(\min)} + t_{comb(\min)} - t_{hold} - t_{skew(\max)} > 0$$

- Los dos primeros términos son el mínimo tiempo después del flanco de reloj tras el cual el valor a la entrada de un biestable cambia.
- El tiempo de "hold" es el mínimo tiempo en que la entrada puede cambiar
- El "skew" se sustrae del margen existente en el tiempo de "hold"

Compesación del "skew"

Para solventar este problema, podemos acudir a varias técnicas, como lo son:

- Usar biestables con mayor tiempo de propagación.
- Realizar un ajuste específico de los retardos combinacionales.
- Usar biestables con menor tiempo de hold.

Las normas de diseño síncrono son una buena guía para la realización de diseños con un funcionamiento seguro. En su aplicación práctica es frecuente que se den casos en los que resulta inevitable vulnerarlas: en el interfaz con buses asíncronos o con memorias síncronas, por ejemplo, o en el de la sincronización de entradas asíncronas. Cuando esto ocurra es aconsejable aislar los módulos de interfaz con sistemas asíncronos y diseñar el resto del sistema ateniéndose a las reglas enunciadas.

En el diseño de circuito es aconsejable utilizar flip-flops tipo D, puesto que son los de funcionamiento más simple y facilitan la interpretación del modo de operación del circuito. Además, con los flip-flops tipo D resulta muy sencilla la incorporación de entradas síncronas de reset, preset y habilitación de reloj. Las entradas asíncronas de los flip-flops sólo deben utilizarse, si se desea, para la inicialización del circuito, pero nunca durante la operación normal del mismo.

Metaestabilidad

Recuerda que en la báscula R-S, al poner sus entradas a 1, podemos originar también un estado metaestable.

Cuando se violan los tiempos de set-up o de hold en un flip-flop, su salida puede pasar a un nivel intermedio; al cabo de un tiempo indeterminado tomará aleatoriamente el valor 0 ó 1.

La probabilidad de que un flip-flop entre en estado metaestable, así como el tiempo de permanencia en dicho estado, dependen del proceso tecnológico y de las condiciones ambientales de funcionamiento, aunque se puede generalizar que los flip-flops pasan rápidamente a un estado estable.

El problema es que si la salida del flip-flop es muestreada en el estado metaestable, se propagará un valor indefinido a la lógica a la que esté conectado.

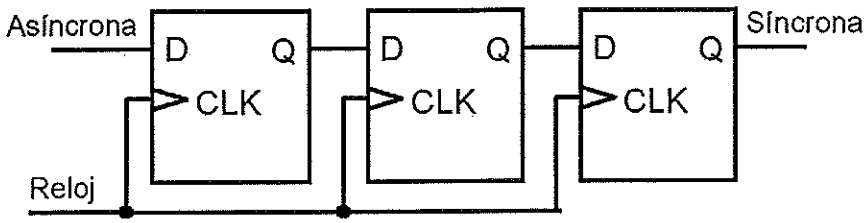
Sincronización de entradas asíncronas

A menudo existen entradas al circuito que son asíncronas respecto a su reloj y deben ser sincronizadas antes de poder ser usadas en el mismo.

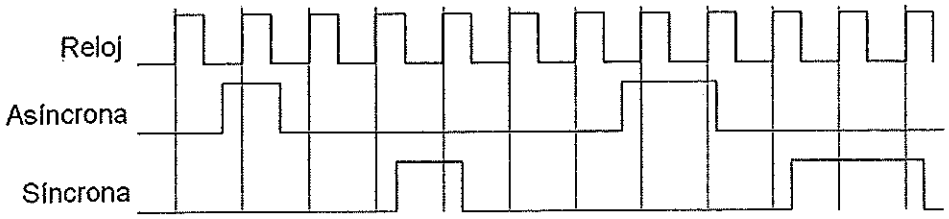
La sincronización consiste en registrar la entrada en un flip-flop conectado al reloj del circuito. Durante esta operación puede ocurrir que se violen los tiempos de set-up o de hold del flip-flop. Como consecuencia, el flip-flop puede registrar o no el evento de entrada o, lo que es peor, entrar en un estado metaestable. Para evitar esto hay muchas topologías de circuitos de sincronización.

El circuito de la figura siguiente muestra el esquema circuital típico para sincronizar entradas asíncronas, abordando el problema de metaestabilidad explicado anteriormente:

Cabe reiterar que, según el tipo de entrada que deseemos sincronizar, el circuito de sincronización será distinto. El mostrado es un ejemplo muy típico.



Este circuito provee un tiempo para que desaparezca la metaestabilidad antes de usar la señal en el circuito. A cambio, nuestro sistema tendrá un mayor tiempo de respuesta. Veamos el cronograma de la situación:



APÉNDICE 3: PUERTAS NAND Y NOR Ejercicio de clase 3, Tema

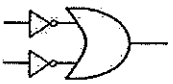
La importancia de estas puertas, NAND y NOR, radica en que son típicamente más baratas y fáciles de fabricar.

En este apéndice vamos a dedicarnos a la implementación, mediante puertas NAND y NOR, de todas las demás funciones lógicas sencillas, vistas en el tema 3. Así mismo, describiremos la función lógica que implementan, (NAND y NOR lógico), usando puertas "normales", si bien esta labor será de menor interés.

Puertas NAND

1. NAND mediante AND + NOT

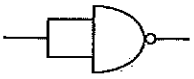
$$NOT('X'\ AND\ 'Y') = \overline{X \cdot Y} \equiv 'X'\ NAND\ 'Y'$$



2. NAND mediante OR + NOT

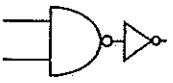
$$(NOT\ 'X')\ OR\ (NOT\ 'Y') = \overline{X} + \overline{Y} = \overline{X \cdot Y} \equiv 'X'\ NAND\ 'Y'$$

3. Implementación del resto de funciones mediante NAND



- **NOT**

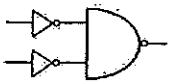
$$'X'\ NAND\ 'X' = \overline{X \cdot X} = \overline{X} = NOT\ 'X'$$



La negación podemos hacerla mediante NAND.

- **AND**

$$NOT('X'\ NAND\ 'Y') = \overline{\overline{X \cdot Y}} = X \cdot Y = 'X'\ AND\ 'Y'$$



Las negaciones podemos hacerlas mediante NAND.

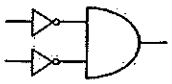
- **OR**

$$(NOT\ 'X')\ NAND\ (NOT\ 'Y') = \overline{\overline{X} \cdot \overline{Y}} = \overline{\overline{X + Y}} = X + Y = 'X'\ OR\ 'Y'$$

Puertas NOR

1. NOR mediante OR + NOT

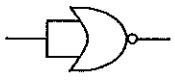
$$NOT('X'\ OR\ 'Y') = \overline{X + Y} \equiv 'X'\ NOR\ 'Y'$$



2. NOR mediante AND + NOT

$$(NOT\ 'X')\ AND\ (NOT\ 'Y') = \overline{X} \cdot \overline{Y} = \overline{X + Y} \equiv 'X'\ NOR\ 'Y'$$

3. Implementación del resto de funciones mediante NOR



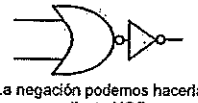
- **NOT**

$$'X'\ NOR\ 'X' = \overline{X + X} = \overline{X} = NOT\ 'X'$$

Las negaciones podemos hacerlas mediante NOR.

- **AND**

$$(NOT\ 'X')\ NOR\ (NOT\ 'Y') = \overline{\overline{X} + \overline{Y}} = \overline{\overline{X \cdot Y}} = X \cdot Y = 'X'\ AND\ 'Y'$$



La negación podemos hacerla mediante NOR.

- **OR**

$$NOT('X'\ NOR\ 'Y') = \overline{\overline{X + Y}} = X + Y = 'X'\ OR\ 'Y'$$

APÉNDICE 4:OPERACIONES ARITMÉTICAS

Los sistemas digitales realizan operaciones sobre señales expresadas mediante secuencias de números. Estas operaciones pueden ser de muy diversa índole, pero en este tema vamos a centrarnos sólo en la realización de algunas operaciones aritméticas (comparación, suma y resta).

A4.1. Comparación

Este procedimiento, conocido como ***algoritmo de comparación***, supone que los dos operandos se expresan empleando el mismo número de cifras. Si no es así, previamente habrá que extender aquel número que emplee menos cifras en su representación

La comparación es la operación de determinar, dados dos argumentos X e Y, si es $X > Y$, $X = Y$ ó $X < Y$.

El procedimiento normalmente efectuado para ello es bien conocido: se comparan las cifras individuales (x_i e y_i) de las dos cantidades comenzando por las más significativas, hasta encontrar dos que sean distintas. En este momento si $x_i > y_i$ entonces $X > Y$, y si $x_i < y_i$ entonces $X < Y$. Por supuesto, si no se encuentra ningún par de cifras de igual peso tal que $x_i \neq y_i$, entonces $X = Y$.

Este algoritmo funciona para cualquier sistema de numeración (binario, octal, decimal, hexadecimal) estudiado anteriormente.

A4.1.1. Comparación de operandos en Complemento a 2 (C2)

En el caso de que los operandos estén expresados en complemento a 2 el método de comparación es ligeramente distinto:

- Si los dos operandos son positivos, o los dos son negativos, el procedimiento expuesto es válido.
- Pero si un operando es positivo y el otro negativo, directamente el mayor es el positivo.

Ejemplo:

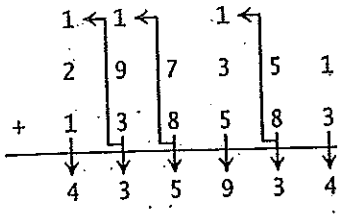
Si $X = 1101\ 0110_{C2}$ e $Y = 1101\ 1110_{C2}$, puesto que los dos operandos son negativos se procede a compararlos normalmente y se obtiene que $X < Y$.

Con $X = 0101\ 0110_{C2}$ e $Y = 0101\ 1110_{C2}$, los dos operandos son positivos, y $X < Y$ también.

En cambio, con $X = 0101\ 0110_{C2}$ e $Y = 1101\ 1110_{C2}$, puesto que X es positivo e Y es negativo, se obtiene directamente que el mayor es el positivo: $X > Y$

A4.2. Suma

El mecanismo usualmente empleado para sumar, trabajando en distintos sistemas posicionales naturales, es similiar en todo momento al empleado normalmente para sumar números en decimal. Este mecanismo está presentado en la figura de la derecha.



El algoritmo clásico de suma consiste en sumar las cifras de igual peso de los dos sumandos, comenzando por las cifras menos significativas. Cada una de estas sumas parciales da lugar a cada una de las cifras del resultado.

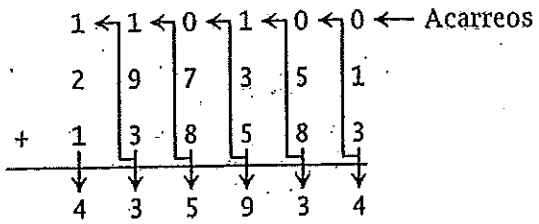
El proceso continua hasta haber sumado todas las cifras de los sumandos.

A4.2.1. Acarreo

Con frecuencia se emplea el término inglés *carry*, ó la letra *C*, para designar al acarreo.

Puede ocurrir que alguna de estas sumas parciales no se exprese en una única cifra, sino con dos. En este caso la cifra menos significativa de la suma parcial formará parte del resultado final, y la cifra más significativa se *acarrea* a la siguiente suma parcial. De esta manera, en la siguiente suma parcial se procede a sumar las cifras correspondientes, más el *acarreo* previo para obtener la siguiente cifra del resultado.

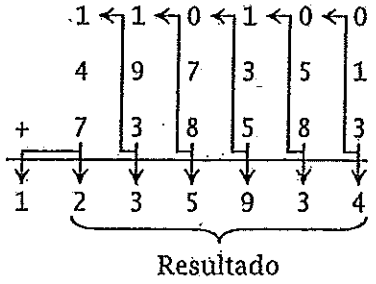
En la siguiente figura presentamos la misma suma detallando más los acarreos intermedios.



A4.2.2. Desbordamiento

Aunque trabajando con lápiz y papel el número de cifras es flexible (siempre se puede escribir una cifra más), no ocurre lo mismo con los sistemas que proceesan información. Usualmente estos se diseñan de modo que sólo son capaces de trabajar con un número de cifras determinado, por eso el desbordamiento es un punto importante sobre el que se debe prestar atención.

Al trabajar con n cifras se puede producir una situación como la dada en la siguiente figura, en la que $n = 6$. Como se ve, al sumar las últimas cifras de los sumandos junto con su acarreo se produce un nuevo acarreo. Este acarreo formaría parte, directamente, del resultado final, pero, puesto que se trabaja sólo con seis cifras, no puede ser así. En este caso, si no podemos incluir esa última cifra, el resultado mostrado será claramente incorrecto.



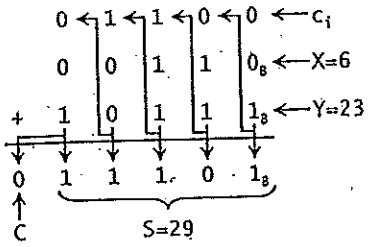
El desbordamiento suele recibir el nombre de *overflow* (del inglés)

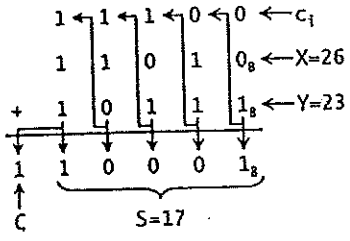
Por tanto, cuando se trabaja con un número de cifras n fijo, puede ocurrir que el resultado de la operación necesite un número mayor de cifras para expresarse. En este caso el resultado obtenido de la operación con n cifras es incorrecto, y se dice que se ha producido un **desbordamiento**.

A4.2.3. Suma en Binario Natural

El algoritmo recién presentado es de aplicación para cualquier sistema de numeración posicional natural, y particularmente cuando se trabaja en binario natural. De hecho, la suma en binario es más sencilla que en decimal, puesto que las cifras a sumar sólo pueden tomar dos valores.

Las siguientes figuras ilustran este mecanismo.





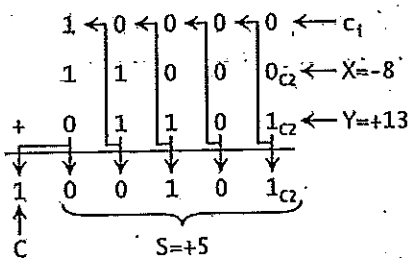
El resultado final de la primera suma es $X+Y=1\,1101_B=29_D$ y no hay acarreo ($C=1$) final ni desbordamiento.

En el segundo caso, el acarreo final es 1 ($C=1$) y, evidentemente, se ha producido desbordamiento.

A4.2.4. Suma en Complemento a 2

El algoritmo anterior también es válido para sumar números expresados según el criterio del complemento a 2.

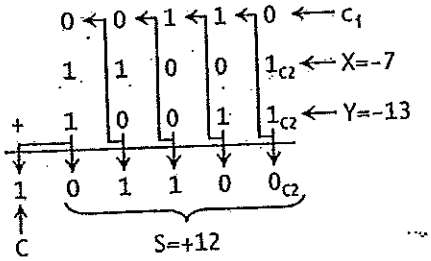
La figura siguiente muestra cómo sumar dos cantidades expresadas en C2 con cinco bits, $X=1\,1000_{C2}=-8$ e $Y=0\,1101_{C2}=+13$.



El resultado es $S=0\,0101_{C2}=+5$, se genera acarreo final y no hay desbordamiento.

Este problema introduce la necesidad de detectar desbordamiento en un sistema que realice operaciones aritméticas.

Ahora, la siguiente figura realiza otra suma de cantidades expresadas en complemento a 2 con cinco bits, $X=1\,1001_{C2}=-7$ e $Y=1\,0011_{C2}=-13$.



Vemos como mediante la aplicación directa del algoritmo descrito se obtiene $S=0\,1100_{C2}=+12$ (muy diferente del resultado que cabría esperar, -20), se genera acarreo final ($C=1$) y hay desbordamiento.

A4.2.5. Detección de desbordamiento en la suma

Recuerda que estamos hablando siempre en base 2, el proceso para detectar y tratar el desbordamiento en otras bases no es objeto de nuestro estudio.

Como ya se ha visto, es clave detectar cuándo se está produciendo desbordamiento y, si es posible, realizar las correcciones oportunas al resultado de la operación realizada.

Existe un mecanismo sencillo que permite detectar la condición de desbordamiento:

- Sumando números sin signo se produce desbordamiento si el acarreo final vale 1.
- Sumando números con signo (en complemento a 2) se produce desbordamiento si los dos operandos tienen el mismo signo y el resultado tiene el signo contrario.

  En este caso también hay otro método: habrá desbordamiento si los dos últimos acarreos son distintos.

En caso de desbordamiento, trabajando en binario, deberemos añadir el acarreo final a la izquierda, obteniendo la representación correcta del resultado con $n+1$ cifras.

A4.2.6. Suma en BCD

La suma de cantidades expresadas en BCD también se efectúa mediante el algoritmo de suma ya descrito. Sin embargo, su aplicación es más complicada que trabajando en binario natural ó decimal, puesto que, ahora, se trabaja en un sistema en base diez, pero las cifras individuales se expresan en base dos.

Para nuestros ejercicios, el mecanismo más rápido para operar en BCD será pasar los operandos a decimal, realizar la cuenta, y expresar el resultado en BCD, prestando atención, en la operación decimal, a los fenómenos de acarreo y desbordamiento.

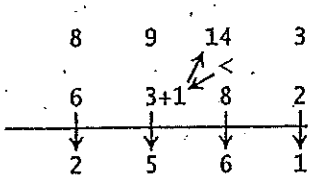
A4.3. Resta

A4.3.1. Algoritmo clásico de resta

Aunque se trata de un algoritmo conocido (y asimilado, en decimal) por todos, en este apartado no vamos a desarrollarlo, porque no será nuestra manera de operar a la hora de hacer una resta.

Así, sólo vamos a presentar una imagen recordatoria del procedimiento clásico de resta:

Este es el algoritmo preferido para realizar restas en sistemas digitales, y el más aconsejable.



A4.3.2. Algoritmo de resta por complementación

Simplemente consiste en complementar el sustraendo (hacer su complemento a 2) y realizar la suma tal y como hemos aprendido. Además, vale todo lo dicho en lo que se refiere a la detección de desbordamiento en la suma. Teniendo bien en cuenta este desbordamiento en caso de producirse, el resultado de nuestra resta será directamente el de esta suma (expresado en C2).

A4.3.3. Detección de desbordamiento en la resta

Para detectar el desbordamiento en una resta de dos cantidades expresadas en complemento a dos, podemos actuar de dos formas:

- Emplear el algoritmo de resta por complementación: esto es, en vez de restar, sumamos al minuendo el opuesto del sustraendo (obtenido complementando este a dos, es decir, complementando y sumando 1). En este caso, (cuando ya tenemos el minuendo complementado a dos) cuando nos disponemos a hacer la suma, valen las reglas descritas en el punto 2.6.2.5 acerca del desbordamiento.

- Observar directamente las cantidades del minuendo y sustraendo (sin complementar) así como de la diferencia (obtenida por el método que queramos), y aplicar:
  - Restando números con signo (en complemento a 2) se produce desbordamiento si los dos operandos tienen distinto signo y el resultado tiene el signo contrario al minuendo.

Ejemplo 1:

Vamos a relizar la resta:

$$(-7)-(-2)=-5$$

Esto es, vamos a hacer $X-Y$, donde $X=1001_{C2}$ e $Y=1110_{C2}$.

- Para ello, complementamos Y, obteniendo $-Y=0010$
- Y sumamos ambas cantidades, con el algoritmo clásico de la suma, obteniendo:

$$X+(-Y)=1001+0010=1011_{C2}$$

Donde vemos que no se ha producido desbordamiento, y que el resultado final es:

$$S=1011_{C2}=-5_D$$

Ejemplo 2:

Observa como, por producirse desbordamiento, el resultado final necesita de una cifra más para representarse

Vamos a relizar la resta:

$$(-7)-3=-10$$

Esto es, vamos a hacer $X-Y$, donde $X=1001_{C2}$ e $Y=0011_{C2}$.

- Para ello, complementamos Y, obteniendo $-Y=1101$
- Y sumamos ambas cantidades, con el algoritmo clásico de la suma, obteniendo:

$$X+(-Y)=\underline{1}001+\underline{1}101=1\underline{0}110_{C2}$$

Donde vemos que se ha producido desbordamiento (teniendo el mismo signo minuendo y sustraendo, la cifra correspondiente del resultado no coincide con este), y que el resultado final es:

$$S=10110_{C2}=-10_D$$

Estas y otras operaciones surgirán en enunciados verbales que tendremos que pasar a una tabla de verdad.

A4.4. Otras operaciones (multiplicación, división, potencias...)

Para esta asignatura, y teniendo en cuenta las preguntas de examen a las que nos vamos a enfrentar, la forma más rápida de efectuar otras operaciones aquí no descritas, como la multiplicación, división y potenciación será "pensando en decimal" y expresando el resultado en el sistema que se nos pida.

APÉNDICE 5: PRINCIPIOS BÁSICOS. CMOS

Existen muchas maneras para diseñar un circuito lógico electrónico. Un buen resumen de la evolución de la tecnología empleada en este diseño, a lo largo del tiempo, es el siguiente:

- 1930s, Bell Labs: primer circuitos lógicos usando relés
- 1940s, usando tubos de vacío; ENIAC: 18000 tubos, 31mx3mx1m
- 1950s, invención del transistor bipolar: ordenadores más rápidos
- 1960s, invención del circuito integrado: diodos+transistores+otros componentes en un solo chip
- 1960s: lógica de transistor-transistor (TTL) basada en bipolares
- 1960s. también demostración de circuitos lógicos con MOSFETs
- 1980s: desarrollo de la tecnología MOS: complementary-MOS (CMOS)
- Hoy en día CMOS usado casi únicamente en todos los circuitos integrados:
    - mayor velocidad, menor consumo que TTL
    - las tensiones de alimentación han venido decreciendo en los últimos años: desde los 5 V de TTL -> 3.3 -> 2.5 -> 1.8 V

La lógica CMOS es la tecnología de lógica digital comercial con mayor capacidad y la más fácil de comprender, y es la que vamos a estudiar en este tema.

A5.1. Lógica CMOS

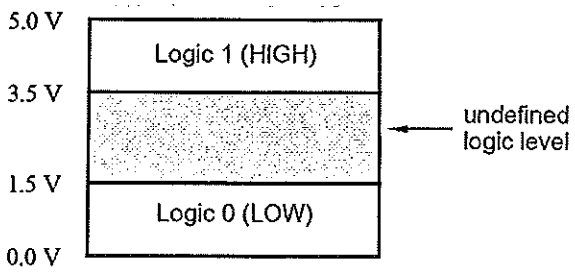
El comportamiento funcional de un circuito lógico CMOS es muy fácil de comprender, ya que los únicos bloques básicos de construcción van a ser transistores CMOS.

A5.1.1. Niveles lógicos CMOS

Los elementos lógicos abstractos procesan dígitos binarios, 0 y 1. Sin embargo los circuitos lógicos reales procesan señales eléctricas tales como niveles de voltaje. En cualquier circuito lógico existe un intervalo de voltajes que se interpreta como un 0 lógico y otro intervalo que se interpreta como un 1.

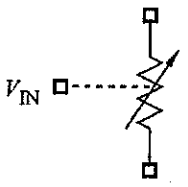
Un circuito lógico CMOS típico funciona a partir de una fuente de alimentación de 5 voltios. Un circuito de esta clase interpretará cualquier voltaje que esté entre 0 y 1.5 voltios como un 0 lógico, e interpretará a cualquier voltaje que esté entre 3.5 y 5 voltios como un 1 lógico.

Los voltajes que están en el intervalo intermedio sólo aparecerán durante las transiciones de señal, por tanto producirán valores lógicos indefinidos.



A5.1.2. Transistores MOS

Se trata de un dispositivo de tres terminales que actúa como una resistencia controlada por voltaje. El voltaje de entrada que se aplica en un terminal controla la resistencia entre las dos terminales restantes.



En aplicaciones de lógica digital, un transistor MOS opera de modo que su resistencia siempre sea muy elevada (el transistor se encuentra apagado "OFF"), ó muy baja (el transistor se encuentra encendido "ON)".

Los nombres de los dos tipos de transistores se refieren al tipo de material semiconductor que se utiliza en las terminales cuya resistencia está controlada.

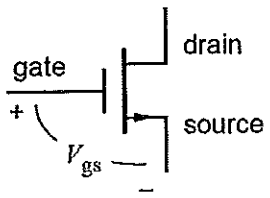
Existen dos tipos de transistores MOS, de canal-n y de canal-p.

A5.1.3. Transistores NMOS

En la siguiente figura vemos el símbolo esquemático de un transistor MOS de canal-n (NMOS).

Un aumento de la tensión V_{gs} disminuye la resistencia entre drenador y fuente, R_{ds}.

Como se observa en el circuito, el drenador se encuentra normalmente a un voltaje más alto que la fuente.



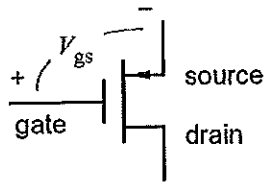
Las terminales se denominan *compuerta* (gate), *fuente* (source) y *drenador* (drain).

El voltaje de la compuerta a la fuente (V_{gs}) en un transistor NMOS es normalmente cero ó positivo. Según éste el transistor se comportará de dos formas muy sencillas:

- Si $V_{gs} = 0$ el transistor se comportará como un **circuito abierto** entre drenador y fuente (transistor "OFF").
- A medida que incrementamos V_{gs} el transistor se comportará cada vez más como un **corto circuito** entre drenador y fuente (transistor "ON").

A5.1.4. Transistores PMOS

En la siguiente figura vemos el símbolo esquemático de un transistor MOS de canal-p (PMOS).



Su funcionamiento es análogo al de un transistor NMOS, excepto porque la fuente se encuentra normalmente a un voltaje mayor que el drenador, y por lo general el voltaje de la compuerta a la fuente (V_{gs}) es negativo ó cero.

Observaciones:

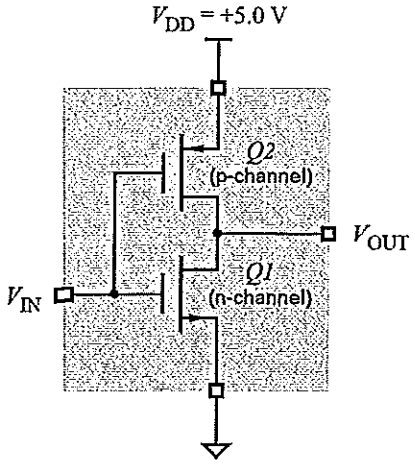
El voltaje de compuerta crea un campo eléctrico que refuerza o retarda el flujo de corriente entre las terminales de fuente y drenador. Este es el "efecto de campo" en el término MOSFET.

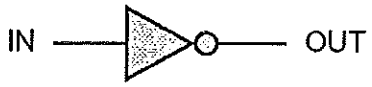
- La compuerta de un transistor MOS tiene una impedancia muy elevada. Es decir, esta compuerta está separada de la fuente y el drenador por un material aislante que tiene una resistencia muy elevada, con lo que la intensidad que corre entre la compuerta y las otras terminales del transistor es despreciable.
- Los transistores NMOS y PMOS se utilizan en forma complementaria para formar la lógica CMOS.

A5.1.5. Circuito del inversor básico CMOS – Modelo de interruptores

El circuito CMOS más sencillo, un inversor lógico, requiere solamente uno de cada tipo de transistor, conectados como se muestra a continuación:

Normalmente el voltaje de alimentación V_{DD} se encuentra en el intervalo de 2 a 6 voltios y con mucha frecuencia se establece 5V para tener compatibilidad con los circuitos TTL.





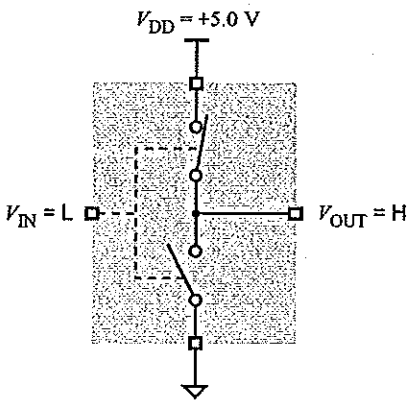
En términos ideales, el comportamiento funcional de este circuito puede caracterizarse como aparece en esta tabla:

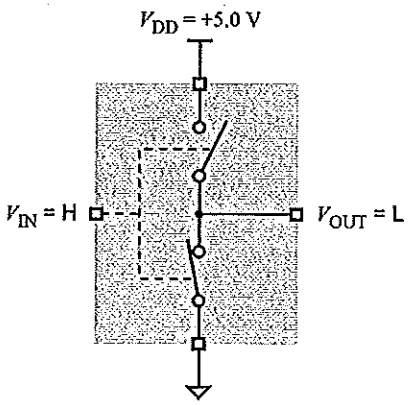
V_{IN}	Q1	Q2	V_{OUT}
0.0 (L)	off	on	5.0 (H)
5.0 (H)	on	off	0.0 (L)

- Cuando $V_{ENT} = 0$, Q1 está apagado porque su V_{gs} es 0, pero Q2 está encendido porque su V_{gs} es un valor negativo muy grande (-5.0 V). En estas condiciones $V_{OUT} = 5V$.
- Cuando $V_{ENT} = 1$, Q1 está encendido porque su V_{gs} es un valor positivo grande (5.0 V), y Q2 está apagado porque su V_{gs} es 0. En estas condiciones $V_{OUT} = 0V$.

Una manera de ver este comportamiento, tal y como haremos en los ejercicios, será usando interruptores.

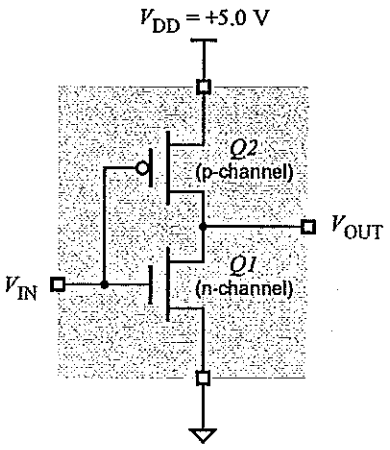
El modelo del transistor de canal-n corresponde a un interruptor normalmente abierto (cuando le ponemos un 1 lógico en la puerta), y el transistor de canal-p corresponde a un interruptor normalmente cerrado.





El modelo del interruptor nos lleva a una manera de dibujar circuitos CMOS que nos facilitará mucho su comprensión.

Como se muestra en la figura de la derecha, se emplean símbolos diferentes para los transistores de canal-n y de canal-p, a fin de reflejar su comportamiento lógico.



- El transistor de **canal-n** (Q1) se encuentra **activado** cuando se aplica voltaje **ALTO** a su compuerta.
- El transistor de **canal-p** (Q2) tiene el comportamiento opuesto: se encuentra **activado** cuando se aplica voltaje **BAJO**.

A5.1.6. Análisis de circuitos CMOS para obtener la función lógica que realizan

Para hallar la función booleana que implementa un circuito CMOS, podemos actuar de dos maneras:

a) Estudiar el circuito equivalente ideal (con modelo de interruptores) para cada combinación de las variables. Al final obtendremos la tabla de verdad de la función y sólo tendremos que expresar la función que representa.

Para entender bien estas reglas, ve a los ejercicios de examen.

b) Analíticamente: Nos plantearemos, viendo el circuito, con lógica proposicional, qué transistores necesitamos "encendidos" y cuáles "apagados", para que la función buscada valga 0 ó 1 (a nuestra elección).

Después, en la expresión obtenida cambiamos las condiciones referidas a cada transistor ("encendido" ó "apagado") por otras, referidas al valor que deben tomar sus entradas para que se produzca ese funcionamiento en cada uno. Recuerda que:

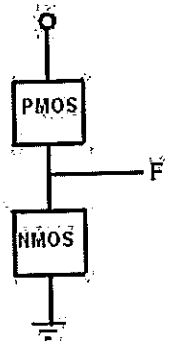
- En transistores NMOS: ON → tensión de puerta = '1'
  OFF → tensión de puerta = '0'
- En transistores PMOS: ON → tensión de puerta = '0'
  OFF → tensión de puerta = '1'

Sólo faltará complementar, en la expresión, aquellas variables que estén negadas (complementando por tanto también su valor) para después cambiar la expresión con lógica proposicional a una función booleana. Los cambios que debemos aplicar son:

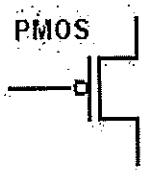
$\wedge$	→	· (AND)
$\vee$	→	+ (OR)
variable igualada a '1'	→	variable sin negar.
variable igualada a '0'	→	variable negada.

A5.1.7. Implementación de funciones mediante circuitos CMOS

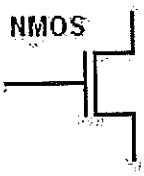
Implementaremos nuestra función siempre formando dos bloques, en cuya unión obtendremos la función booleana deseada.



- El primer bloque, conectado en su extremo superior a V_{CC}, constará de transistores PMOS (uno por cada término de la expresión de la función), realizando las siguientes reglas:
  - Cada · (AND) lo sustituiremos por una conexión SERIE.
  - Cada + (OR) lo sustituiremos por una conexión PARALELO.
  - Pondremos las variables en las puertas de los transistores, NEGADAS con respecto a como aparecen en la función booleana.



- El segundo bloque, conectado en su extremo inferior a tierra, constará de transistores NMOS (uno por cada término de la expresión de la función), realizando las siguientes reglas:
  - Cada · (AND) lo sustituiremos por una conexión PARALELO.
  - Cada + (OR) lo sustituiremos por una conexión SERIE.
  - Pondremos las variables en las puertas de los transistores, NEGADAS con respecto a como aparecen en la función booleana.



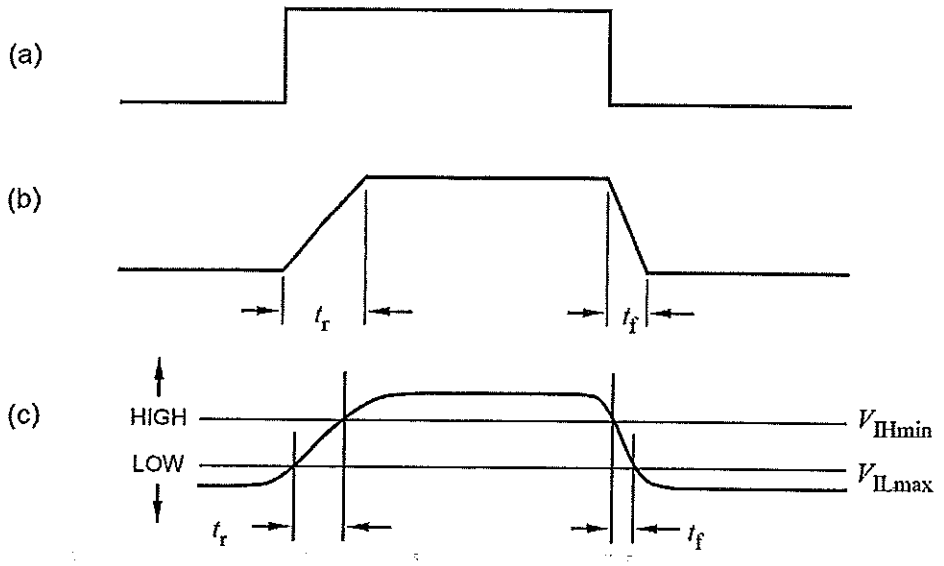
A5.2. Comportamiento dinámico de los dispositivos CMOS

Dentro del estudio del comportamiento eléctrico de un circuito CMOS, en CEDG se le da especial importancia al estudio del comportamiento dinémico del mismo, es decir, de cómo tanto la velocidad como el consumo de energía depende en gran medida de las características dinámicas (ó de CA) del dispositivo y su carga, es decir, lo que sucede cuando la salida cambia entre dos estados lógicos.

Nosotros nos vamos a centrar sólo en el estudio de la velocidad, que depende de dos características importantes: el tiempo de transición y el retardo de propagación.

A5.2.1. Tiempo de transición

Es la cantidad de tiempo que requiere la salida de un circuito lógico para cambiar de un estado a otro. Vamos a entender el concepto con la figura siguiente:



Observa que la parte inicial de una transición no se incluye en el valor del tiempo de ascenso ó de caída, sino que contribuye al valor del retardo de propagación, que estudiaremos en el punto siguiente.

a) Esta figura muestra la situación ideal para el cambio de estado en las salidas: en tiempo cero.

b) En este caso se muestra una visión más realista de la salida de un circuito: una salida necesita una cierta cantidad de tiempo, denominado tiempo de ascenso (t_r) para cambiar de BAJO a ALTO, y un tiempo posiblemente diferente, denominado tiempo de caída (t_f) para cambiar de ALTO a BAJO.

Las salidas no pueden cambiar simultáneamente, porque necesitan tiempo para cargar la capacidad parásita del cableado y otros componentes que controlan.

c) Incluso la figura B no es bastante precisa, porque la razón del cambio del voltaje de salida no cambia instantáneamente. En su lugar, el comienzo y el final de una transición son suaves, como se ve en la figura C.

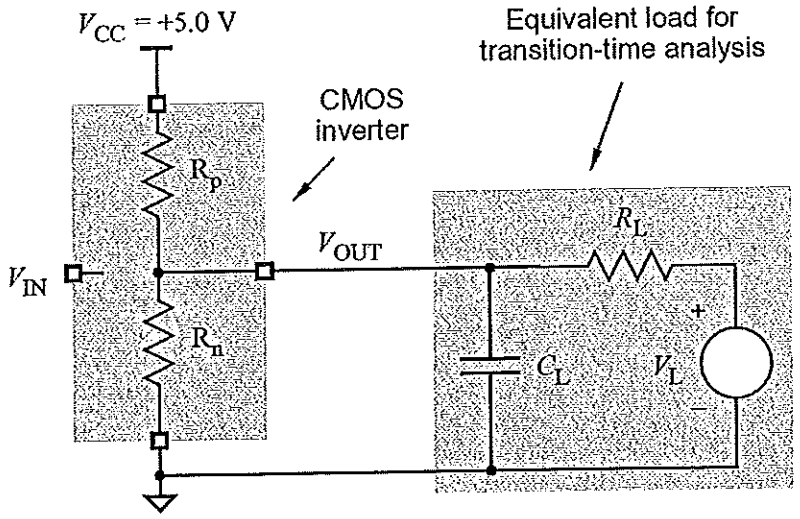
En la figura C, los tiempos de ascenso y caída indican cuánto tiempo tarda un voltaje de salida en pasar a través de la región indefinida entre BAJO y ALTO.

A5.2.2. Análisis de tiempos de ascenso y caída de una salida CMOS

Los tiempos de ascenso y caída de una salida de CMOS dependen principalmente de dos factores, la resistencia del transistor "encendido" y la capacidad de carga. Una capacidad grande incrementa los tiempos de transición, por lo que será raro que conectemos a propósito un condensador en la salida del circuito lógico. Sin embarlo, las capacidades parásitas están presentes en cualquier circuito, provenientes de tres fuentes: los circuitos de salida, el cableado que conecta la salida con otras entradas, y los circuitos de entrada.

La capacidad parásita se conoce en ocasiones como carga capacitiva o carga de CA.

Se pueden analizar los tiempos de acceso y caída de una salida CMOS utilizando el circuito equivalente que se muestra a continuación:



Observa que en el ejemplo estudiamos un inversor CMOS como el que hemos visto en estos apuntes.

- Los transistores de canal-p y canal-n están modelados por las resistencias R_p y R_n, respectivamente. En operación normal, una resistencia es alta y otra baja, dependiendo de estado de salida.

- La carga de salida se modela mediante un circuito de carga equivalente con tres componentes:

  R_L, V_L: Ambos componentes representan la carga de contínua (CD): determinan los voltajes que se establecen cuando la salida se estabiliza en un nivel ALTO o BAJO. La carga de CD no tiene demasiado efecto sobre los tiempos de transición cuando la salida cambia de estado.

  C_L: esta capacidad representa la carga de CA: determina los voltajes y corrientes que están presentes cuando cambia la salida de un estado a otro.

Cuando una salida CMOS controla sólo entradas CMOS, la carga CD es despreciable. Para simplificar las cosas, analizaremos solamente este caso, con $R_L = \infty$ y $V_L = 0$ en el resto de estos apuntes.

Vamos a estudiar el análisis mediante un ejemplo:

Ejemplo:

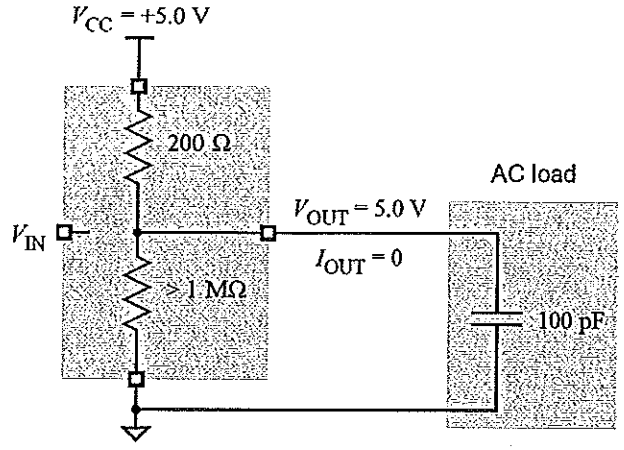
Vamos a analizar los tiempos de transición de una salida CMOS, para el circuito anterior, con $C_L = 100\ pF$ (carga capacitiva) y con $R_p = 200\ \Omega$ y $R_n = 100\ \Omega$ (resistencias "de encendido" de los transistores)

Los tiempos de ascenso y de caída dependen del tiempo que se necesita para cargar y descargar la carga capacitiva C_L

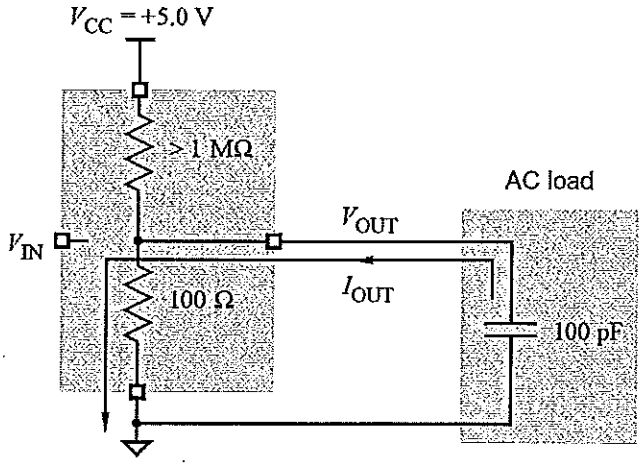
Primero observamos el tiempo de caída:

La figura siguiente muestra las condiciones eléctricas en el circuito cuando la salida se encuentra en un estado estable ALTO:

En este ejemplo suponemos que cuando los transistores CMOS cambian entre "encendido" y "apagado", lo hacen de forma instantánea.



Vamos a suponer que al tiempo $t = 0$ la salida CMOS cambia al estado BAJO, generando así la situación que planteamos a continuación:



Recuerda que el voltaje en un condensador no puede cambiar de forma instantánea

Cuando $t = 0$, V_{OUT} todavía es 5.0 V. En el tiempo $t = \infty$ el condensador tiene que estar completamente descargado, y V_{OUT} debe ser 0 V.

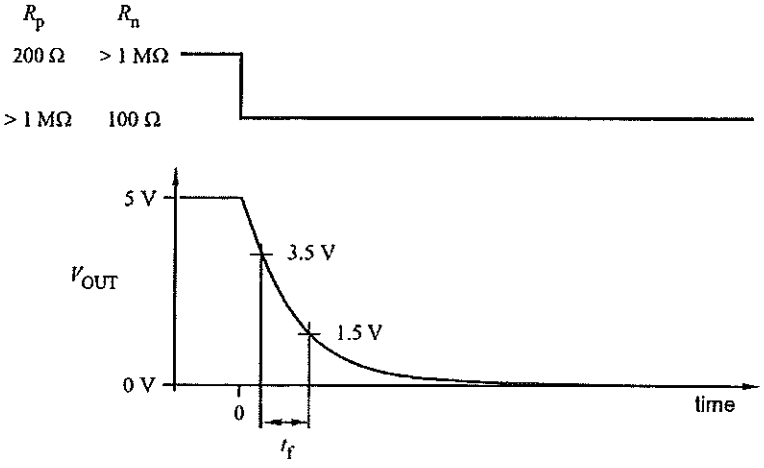
Entre los dos momentos, como sabemos, el valor de V_{OUT} está gobernado por una ley exponencial:

Esta ley exponencial es resultado de resolver una Ecuación Diferencial, tal y como veremos en algún problema de examen.

$$V_{OUT} = V_{DD} \cdot e^{-\frac{t}{R_L C_L}} = 5.0 \cdot e^{-\frac{t}{100 \cdot 100 \cdot 10^{-12}}} = 5.0 \cdot e^{-\frac{t}{10 \cdot 10^{-9}}} \ V$$

La figura siguiente muestra una gráfica de V_{OUT} en función del tiempo:

El factor $R_n C_L$ tiene unidades de tiempo (segundos) y se conoce como constante de tiempo *RC*. Como hemos visto, en este ejemplo la constante de tiempo *RC* para las transiciones ALTO a BAJO es de 10 ns.



Para calcular el tiempo de caída recordamos que 1.5 V y 3.5 V son las fronteras que definimos para los niveles BAJO y ALTO.

Para obtener el tiempo de caída, debemos resolver la ecuación anterior para $V_{OUT} = 3.5$ y $V_{OUT} = 1.5$. Así:

$$t = -R_n C_L \cdot \ln \frac{V_{OUT}}{V_{DD}} = -10 \cdot 10^9 \cdot \ln \frac{V_{OUT}}{5.0}$$

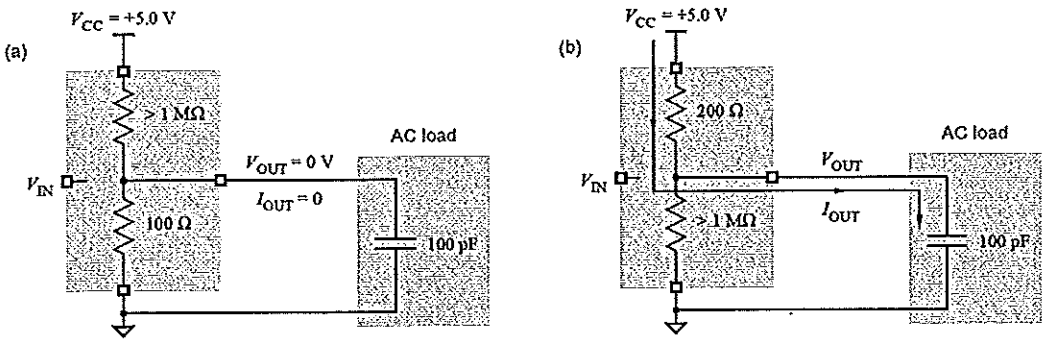
Obteniendo:

- $t_{3.5} = 3.57\ ns$
- $t_{1.5} = 12.04\ ns$

} El tiempo de caída t_f es la diferencia entre ambos números, 8.47 ns.

Tiempo de ascenso:

El tiempo de ascenso puede calcularse con un método semejante. La figura (a) muestra las condiciones en el circuito cuando la salida se encuentra en un estado estable BAJO. Si al tiempo $t = 0$ la salida CMOS cambia al estado ALTO, resulta la situación descrita en (b).

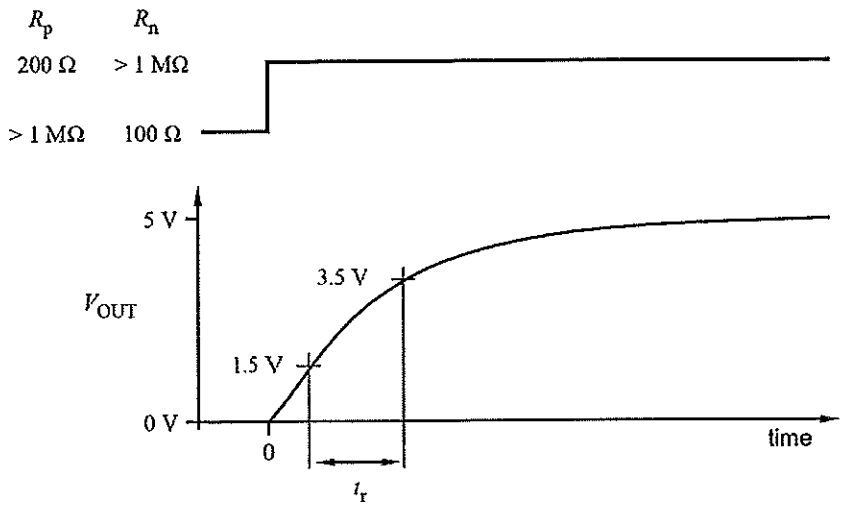


Una vez más, V_{OUT} no puede cambiar de forma instantánea, pero al tiempo $t = \infty$ el capacitor estará completamente cargado y V_{OUT} será 5V.

De nuevo, el valor de V_{OUT} está determinado por una ley exponencial:

También sabemos deducir esta ley a partir de la resolución de una ecuación diferencial.

$$V_{OUT} = V_{DD} \cdot \left(1 - e^{\frac{t}{R_p C_L}}\right) = 5.0 \cdot \left(1 - e^{\frac{t}{200 \cdot 100 \cdot 10^{-12}}}\right) = 5.0 \cdot \left(1 - e^{\frac{t}{20 \cdot 10^{-9}}}\right) V$$



En este caso la constante de tiempo *RC* es 20 ns.

Resolviendo la ecuación anterior (despejamos el tiempo), obtenemos:

$$t = -RC \cdot \ln \frac{V_{DD} - V_{OUT}}{V_{DD}} = -20 \cdot 10^{-9} \cdot \ln \frac{5.0 - V_{OUT}}{5.0}$$

En donde sustituimos los tiempos de frontera, para obtener:

- $t_{1.5} = 7.13\ ns$
- $t_{3.5} = 24.08\ ns$

} El tiempo de ascenso t_r es la diferencia entre ambos números, 16.95 ns

Observaciones:

Se diría aquí que la capacidad de excitación de la salida es "asimétrica".

- En el ejemplo anterior se supone que el transistor de canal-p tiene dos veces la resistencia del transistor de canal-n, y como resultado el tiempo de ascenso es el doble que el de caída.

- Un incremento en la capacidad de carga ocasionará un aumento en la constante de tiempo RC y el correspondiente incremento en los tiempos de transición.

Este truco no es válido para resolver ejercicios de examen, pero sí para verificar, aproximadamente, si nuestros resultados son correctos.

- Una regla sencilla y práctica para estimar los tiempos de transición es que éstos son aproximadamente igual a la constante de tiempo RC del circuito, cuando se está cargando o descargando. Sólo tienes que fijarte como, efectivamente, en el ejemplo resuelto de estos apuntes este "truco" es válido.

Además, en algunas familias lógicas (como TTL) los umbrales no son simétricos en torno a un punto medio de voltaje

- Los tiempos de transición calculados son bastante sensibles a la selección de los niveles lógicos: en el ejemplo resuelto, si usamos 2.0 V y 3.0 V en vez de 1.5 y 3.5 V como los umbrales para BAJO y ALTO, nos saldrían tiempos de transición más cortos. Por otro lado, si empleáramos 0.0 y 5.5, ¡ los tiempos de transición calculados serían infinitos!

A5.2.3. Retardo de propagación

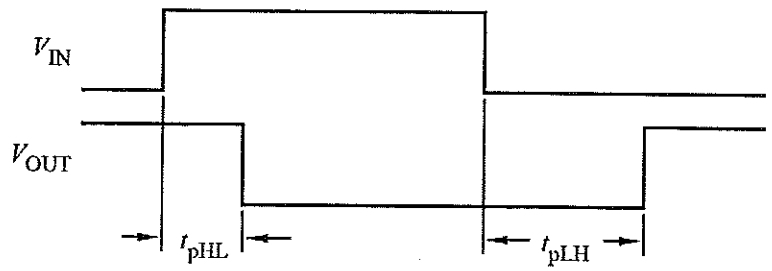
Una trayectoria de señal es la ruta eléctrica de una señal particular de entrada hacia una señal particular de salida, en un elemento lógico.

Los tiempos de ascenso y de caída sólo describen de forma parcial el comportamiento dinámico de un elemento lógico; se necesitan parámetros adicionales para relacionar la temporización de salida con la temporización de entrada.

El retardo de propagación t_p de una trayectoria de señal es la cantidad de tiempo que requiere la señal de entrada para producir un cambio en la señal de salida.

Un elemento lógico complejo con varias entradas y salidas puede especificar un valor diferente de t_p para cada trayectoria diferente de señal.

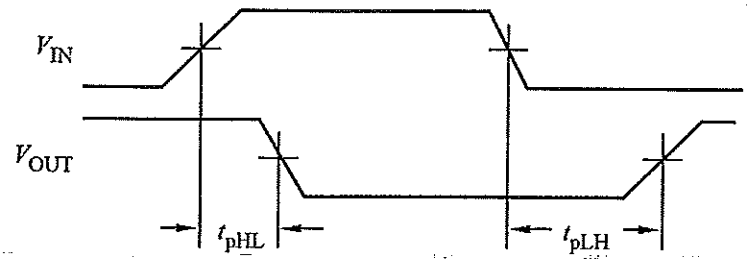
Sin tener en cuenta los tiempos de ascenso y caída, la siguiente figura muestra los diferentes retardos de propagación para la trayectoria de señal de entrada a salida de un inversor CMOS, dependiendo de la dirección del cambio en la salida:



Donde:

- t_{pHL} es el tiempo entre un cambio de entrada y el correspondiente cambio de salida, cuando la salida está cambiando de ALTO a BAJO.
- t_{pLH} es el tiempo entre un cambio de entrada y el correspondiente cambio de salida, cuando la salida está cambiando de BAJO a ALTO.

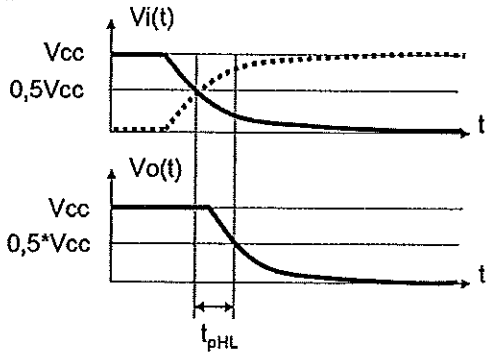
Entre los muchos factores que influyen en el retardo de propagación tenemos que destacar los tiempos de ascenso y caída, lo que hace que especifiquemos los tiempos de retardo de propagación como la distancia temporal entre los puntos medios de las transiciones de entrada y salida, como se ilustra en la figura siguiente:



Esto nos da una definición ya formal de los tiempos de retardo de propagación:

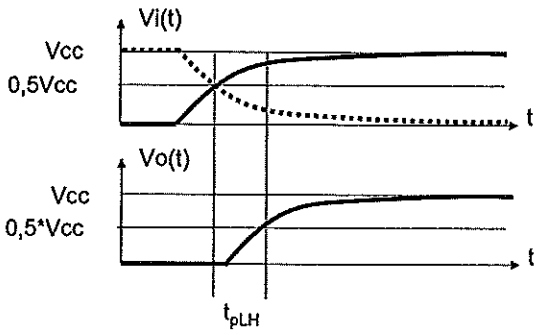
Tiempo de propagación de ALTO a BAJO t_{pHL}

Es el tiempo que transcurre entre que la entrada llega a una tensión del 50% del valor de alimentación (independientemente de si la transición es de H a L o de L a H) y que la salida alcanza dicho valor cuando pasa de H a L.



Tiempo de propagación de BAJO a ALTO t_{pLH}

Es el tiempo que transcurre entre que la entrada llega a una tensión del 50% del valor de alimentación (independientemente de si la transición es de H a L o de L a H) y que la salida alcanza dicho valor cuando pasa de L a H.



EDIG
Ejercicios de clase

TEMA 1: CODIFICACIÓN DE LA INFORMACIÓN

Ejercicio 1

Una función de tres variables F (A, B, C) ha de tomar el valor lógico "0" cuando la variable B se encuentre en estado " 1" y la variable A no esté en el estado "1". En los demás casos posibles, ha de estar en estado "1".

a) Realizar la tabla de verdad de esta función.
b) Obtener las formas canónicas de sumas de productos y de productos de sumas.

a) Tabla de verdad:

Variables → | Función ↓

A	B	C	F	
0	0	0	1	
0	0	1	1	
0	1	0	0	← la variable B se encuentre en estado '1'
0	1	1	0	← y A no este en estado '1'.
1	0	0	1	
1	0	1	1	
1	1	0	1	
1	1	1	1	

b) Formas canónicas: hay 4 formas

1ª) Suma de productos (miniterminos):

- Nos fijamos en las filas de la tabla donde F = 1. La función tendrá un termino sumando por cada una de estas filas.
- Cada uno de esos terminos está formado por el producto de las variables de la función.
- En cada termino negamos ($\bar{A}$) aquellas variables que en esa fila valen '0'.

$$F = \bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}C + A\bar{B}\bar{C} + A\bar{B}C + AB\bar{C} + ABC$$

2ª) Producto de sumas (maxiterminas):

- Nos fijamos en las filas de la tabla donde $F \neq \emptyset$. La función tendrá un termino(factor) por cada una de estas filas.
- Cada término está formado por la suma de las tres variables.
- En cada termino negamos ($\bar{A}$) aquellas variables que en su fila valían '1'.

$$F = (A + \bar{B} + C)(A + \bar{B} + \bar{C})$$

3ª) Producto de productos:

- Solo tenemos que tomar 1ª) forma canonica y la negamos dos veces.
- Aplicamos el teorema de DeMorgan. (página 1.8).
- Esta forma canónica es útil para implementar la función mediante puertas NAND.

$$F = \overline{\bar{A}\bar{B}\bar{C} \oplus \bar{A}\bar{B}C \oplus A\bar{B}\bar{C} \oplus A\bar{B}C \oplus AB\bar{C} \oplus ABC} =$$

$$= \overline{\bar{A}\bar{B}\bar{C}} \cdot \overline{\bar{A}\bar{B}C} \cdot \overline{A\bar{B}\bar{C}} \cdot \overline{A\bar{B}C} \cdot \overline{AB\bar{C}} \cdot \overline{ABC}$$

4ª) Suma de sumas:

- Tomamos la 2ª) forma canónica y la negamos dos veces.
- Aplicamos el teorema de DeMorgan
- Útil para implementar mediante puertas NOR.

$$F = \overline{(A + \bar{B} + C) \odot (A + \bar{B} + \bar{C})} = \overline{A + \bar{B} + C} + \overline{A + \bar{B} + \bar{C}}$$

Ejercicio 2

Supongamos que hay un comité formado por cuatro miembros, de los que uno es presidente, y que las decisiones se toman por mayoría simple, decidiendo el voto del presidente cuando existe empate. Llamaremos A, B, C y D a las variables lógicas que representan el estado del pulsador de cada miembro (donde A es e que corresponde al presidente) y S a la que representa la salida del circuito. Se trata de diseñar una máquina con cuatro entradas (un pulsador para cada miembro) cuya salida dé el resultado de la votación.

Tabla de verdad:

Variables → Función

A	B	C	D	S	
0	0	0	0	0	
0	0	0	1	0	
0	0	1	0	0	
0	0	1	1	0	
0	1	0	0	0	
0	1	0	1	0	
0	1	1	0	0	
0	1	1	1	1	
1	0	0	0	0	
1	0	0	1	1	
1	0	1	0	1	
1	1	1	1	1	
1	1	0	0	1	
1	1	0	1	1	
1	1	1	0	1	
1	1	1	1	1	

Intento de implementación con 1ª forma canonica:

$$S = \bar{A}BCD + A\bar{B}\bar{C}D + \ldots + ABCD$$ (Muy difícil de implementar)

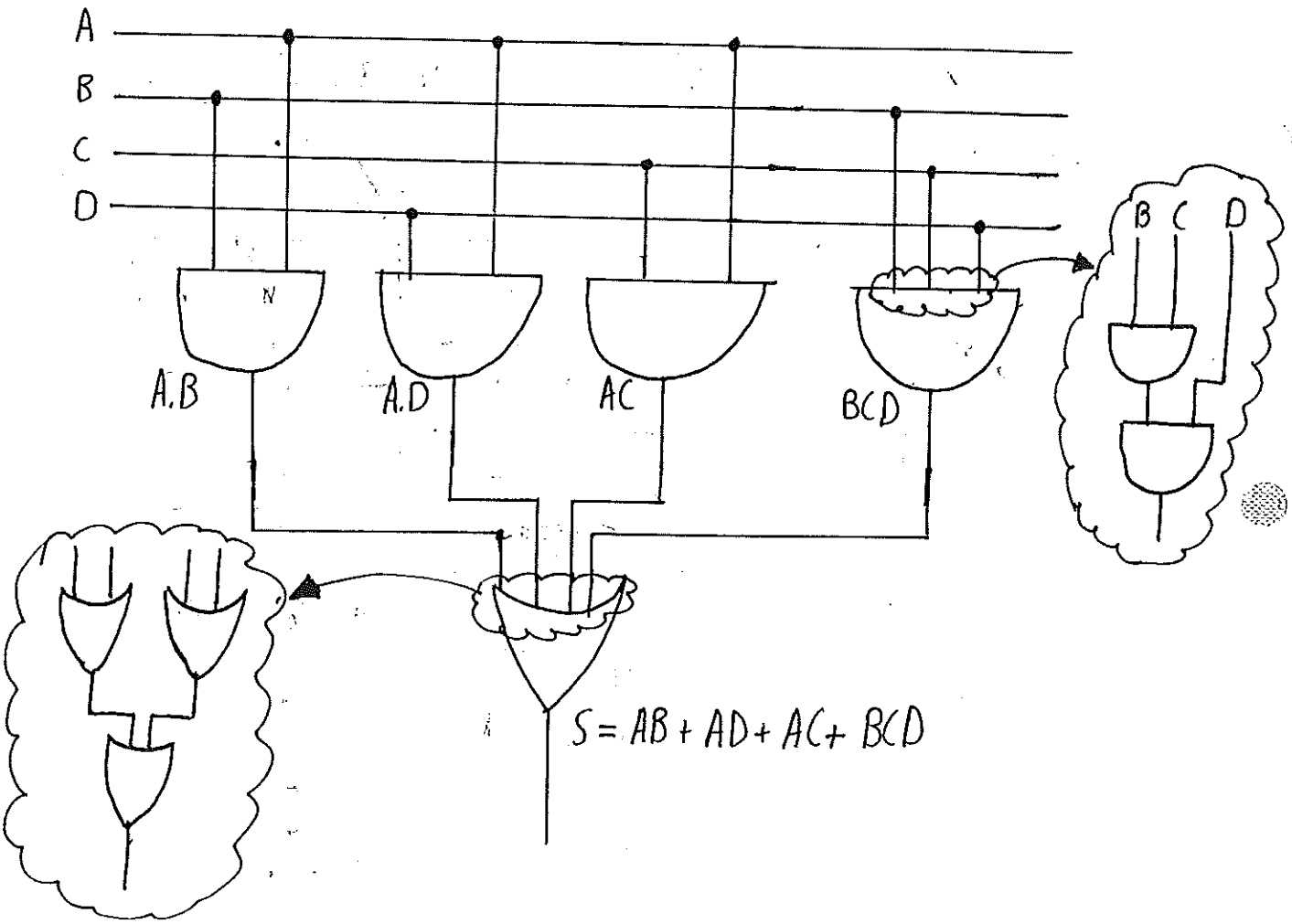
Simplificación mediante el método de Karnaugh:

Mapa de Karnaugh:

CD \ AB	00	01	11	10
00	0	0	1	0
01	0	0	1	1
11	0	1	1	1
10	0	0	1	1

$$S = \underline{A.B} + \underline{AD} + \underline{AC} + \underline{BCD}$$

■ Implementación del circuito mediante puertas lógicas:



Ejercicio 3

Sea la función booleana definida por: $F = \Sigma\, 4,5,10,12,13,14,15$

$$F = \bar{A}B\bar{C}\bar{D} + AB\bar{C}\bar{D} + A\bar{B}C\bar{D} + ABC\bar{D} + \bar{A}B\bar{C}D + AB\bar{C}D + ABCD$$

0100 + 1100 + 1010 + 1110 + 0101 + 1101 + 1111 = 1ª forma canónica

a) Implementar la función con puertas de cualquier número de entradas simplificando por "1".
b) Implementar la función con puertas de cualquier número de entradas simplificando por "0".
c) Implementar la función sólo con puertas NAND de cualquier número de entradas.
d) Implementar la función sólo con puertas NOR de cualquier número de entradas.

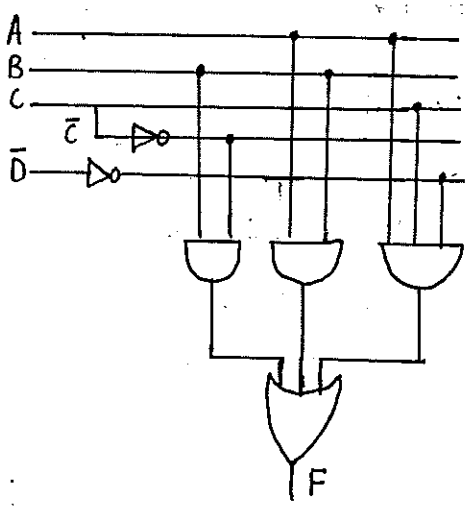
Paso previo:
Tabla de verdad:

	A	B	C	D	F
0	0	0	0	0	0
1	0	0	0	1	0
2	0	0	1	0	0
3	0	0	1	1	0
4	0	1	0	0	1
5	0	1	0	1	1
6	0	1	1	0	0
7	0	1	1	1	0
8	1	0	0	0	0
9	1	0	0	1	0
10	1	0	1	0	1
11	1	0	1	1	0
12	1	1	0	0	1
13	1	1	0	1	1
14	1	1	1	0	1
15	1	1	1	1	1

a) Simplificar "por unos"

CD \ AB	00	01	11	10
00	0	1	1	0
01	0	1	1	0
11	0	0	1	0
10	0	0	1	1

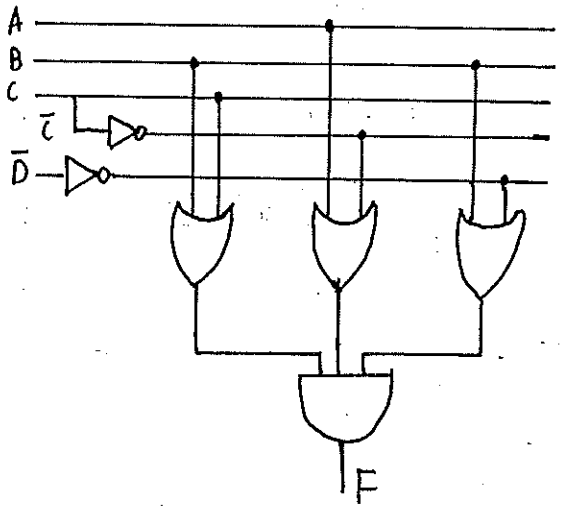
$$F = B\bar{C} + AB + AC\bar{D}$$



b) Simplificar "por ceros"

CD \ AB	00	01	11	10
00	0	1	1	0
01	0	1	1	0
11	0	0	1	0
10	0	0	1	1

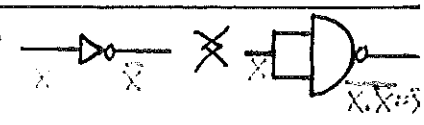
$$F = (B+C).(A+\bar{C}).(B+\bar{D})$$



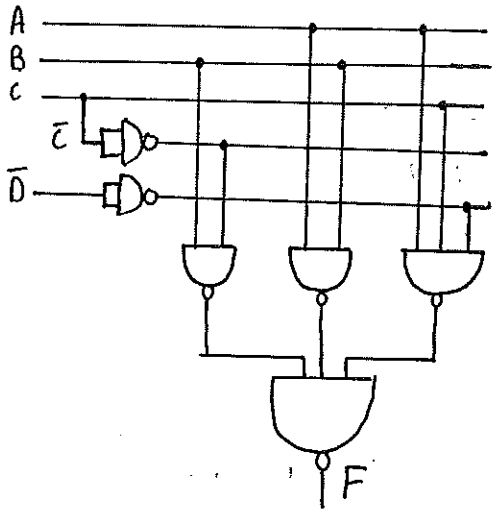
c) Implementación mediante NAND:

$$F = \overline{B\bar{C} + AB + AC\bar{D}} = \overline{B\bar{C}}.\overline{AB}.\overline{AC\bar{D}}$$

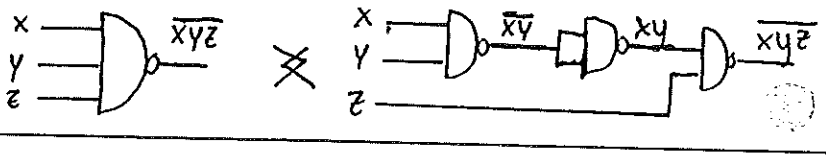
NOT mediante NAND:



$\overline{X.X} = \bar{X}$

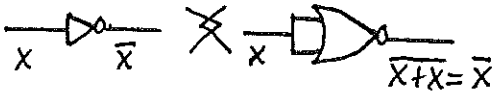


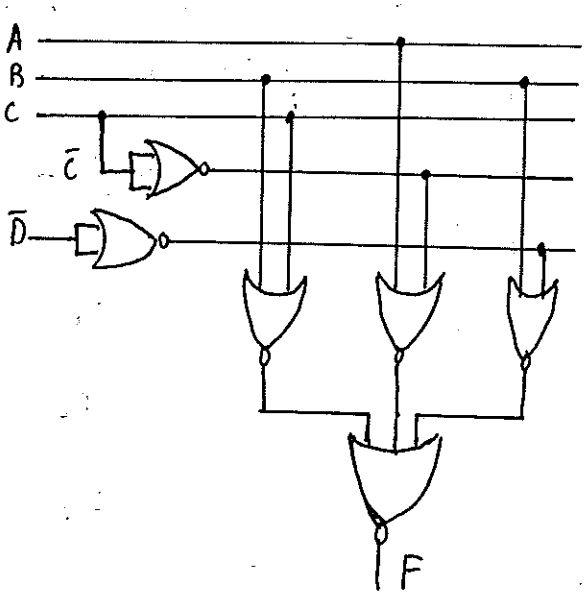
NOTA: implementación de $NAND_3$ mediante $NAND_2$



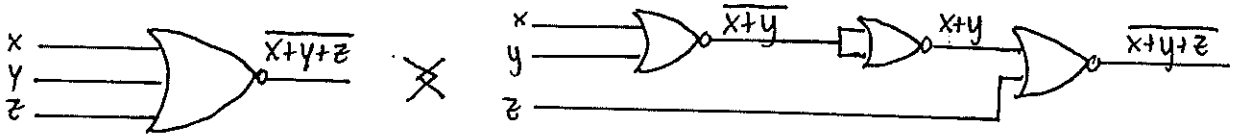
d) Implementación mediante NOR: $\bar{F} = \overline{\overline{(B+C)} \cdot \overline{(A+\bar{C})} \cdot \overline{(B+\bar{D})}} = \overline{\overline{B+C} + \overline{A+\bar{C}} + \overline{B+\bar{D}}}$

Nota: not mediante NOR





NOTA: implementación de NOR_3 mediante NOR_2



Ejercicio 4

Dada la función F(A, B, C, D, E) de cinco variables, por medio de su tabla de verdad, simplificarla por el método de Karnaugh.

E	D	C	B	A	F
0	0	0	0	0	1
0	0	0	0	1	0
0	0	0	1	0	1
0	0	0	1	1	0
0	0	1	0	0	1
0	0	1	0	1	1
0	0	1	1	0	0
0	0	1	1	1	0
0	1	0	0	0	0
0	1	0	0	1	0
0	1	0	1	0	0
0	1	0	1	1	0
0	1	1	0	0	1
0	1	1	0	1	1
0	1	1	1	0	1
0	1	1	1	1	0
1	0	0	0	0	1
1	0	0	0	1	0
1	0	0	1	0	1
1	0	0	1	1	1
1	0	1	0	0	1
1	0	1	0	1	1
1	0	1	1	0	0
1	0	1	1	1	0
1	1	0	0	0	0
1	1	0	0	1	0
1	1	0	1	0	0
1	1	0	1	1	0
1	1	1	0	0	1
1	1	1	0	1	1
1	1	1	1	0	0
1	1	1	1	1	1

E=0

E=1

Necesitamos dos mapas de Karnaugh:

Para E=0

DC \ BA	00	01	11	10
00	1	0	0	1
01	1	1	0	0
11	1	1	0	1
10	0	0	0	0

Para E=1

DC \ BA	00	01	11	10
00	1	0	1	1
01	1	1	0	0
11	1	1	1	0
10	0	0	0	0

$$F = \bar{B}C + \bar{A}.\bar{C}.\bar{D} + \bar{A}CD\bar{E} + ACDE + B\bar{C}\bar{D}E$$

TEMA 3: CIRCUITOS COMBINACIONALES

Ejercicio 1

Utilizando únicamente un multiplexor de cuatro entradas de datos y dos entradas de selección se pide implementar las siguientes funciones lógicas:

a) $F(a,b) = \bar{a}\,b + a\,\bar{b}$

b) $F(a,b,c) = ab + ac + \bar{b}\,c$

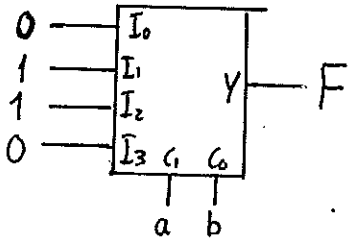
c) $F(a,b,u,w,x,y) = abu + a\bar{b}\,x + \bar{a}\,b\,y + \bar{a}\,\bar{b}\,w$

a) $F(a,b) = \bar{a}b + a\bar{b}$

Caso fácil: mismo nº de variables que entradas de control

$Y = \bar{C}_1\bar{C}_0 I_0 + \bar{C}_1 C_0 I_1 + C_1\bar{C}_0 I_2 + C_1 C_0 I_3$

$F = \bar{a}\bar{b}\,\boxed{0} + \bar{a}\,b\,\boxed{1} + a\bar{b}\,\boxed{1} + ab\,\boxed{0}$



b) $F(a,b,c) = ab + ac + \bar{b}c$ Caso típico: una variable más que entrada de control

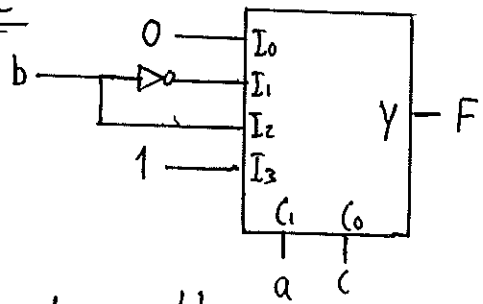
Elemento neutro → $x + \bar{x} = 1$

$F = ab + ac + \bar{b}c = ab(c+\bar{c}) + ac + \bar{b}c(a+\bar{a}) = acb + a\bar{c}b + ac + a\bar{b}c + \bar{a}\bar{b}c =$

$= ac\,(\cancel{b + 1 + \bar{b}}) + a\bar{c}b + \bar{a}c\bar{b} = \underline{\bar{a}c\bar{b} + a\bar{c}b + ac}$

$Y = \bar{C}_1\bar{C}_0 I_0 + \bar{C}_1 C_0 I_1 + C_1\bar{C}_0 I_2 + C_1 C_0 I_3$

$F = \bar{a}\bar{c}\,\boxed{0} + ac\,\boxed{\bar{b}} + a\bar{c}\,\boxed{b} + ac\,\boxed{1}$



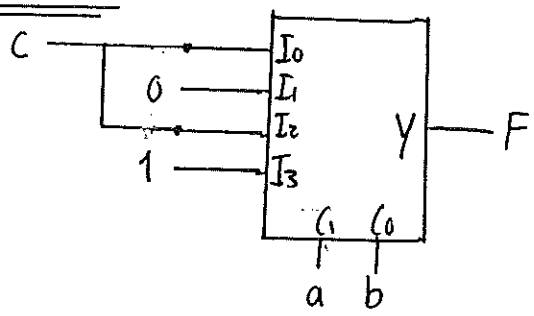
PROBLEMA: hemos necesitado una puerta NOT!! En muchos problemas nos van a pedir explícitamente que sólo usemos multiplexores.

Para no necesitar negadores tenemos que conectar como entradas de control las variables que aparezcan negadores en la expresión de F

Repetimos para no usar puertas NOT: ahora elegimos $a \equiv C_1$ $b \equiv C_0$

$F = ab + ac + \bar{b}c = ab + ac(b+\bar{b}) + \bar{b}c(a+\bar{a}) = ab + abc + a\bar{b}c + a\bar{b}c + \bar{a}\bar{b}c =$

$= ab(\underbrace{1}_{1} + c) + a\bar{b}(\underbrace{c+\bar{c}}_{c}) + \bar{a}\bar{b}c = \underline{\bar{a}\bar{b}c + a\bar{b}c + ab}$



$Y = \bar{C}_1\bar{C}_0 I_0 + \bar{C}_1 C_0 I_1 + C_1\bar{C}_0 I_2 + C_1 C_0 I_3$

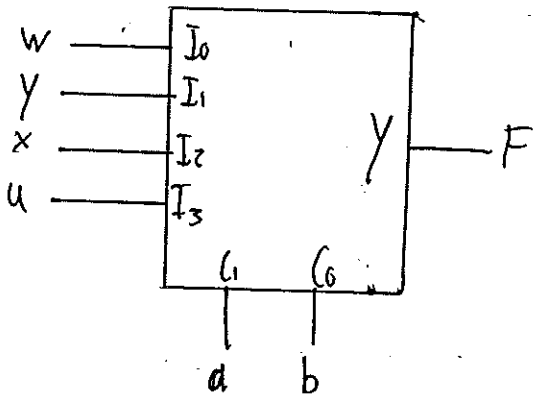
$F = \bar{a}\bar{b}\boxed{c} + \bar{a}b\boxed{0} + a\bar{b}\boxed{c} + ab\boxed{1}$

c) $F(a, b, u, w, x, y) = abu + a\bar{b}x + \bar{a}by + \bar{a}\bar{b}w$

En general, si tenemos dos variables (ó 3 ó 4) más que de entradas de control necesitaremos conectar en cascada varios multiplexores, salvo que la expresión de F "este preparada".

$Y = \bar{C}_1\bar{C}_0 I_0 + \bar{C}_1 C_0 I_1 + C_1\bar{C}_0 I_2 + C_1 C_0 I_3$

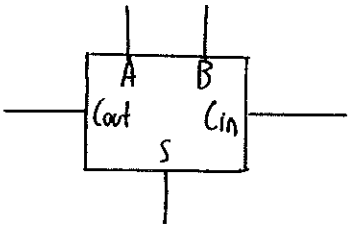
$F = \bar{a}\bar{b}\boxed{w} + \bar{a}b\boxed{y} + a\bar{b}\boxed{x} + ab\boxed{u}$



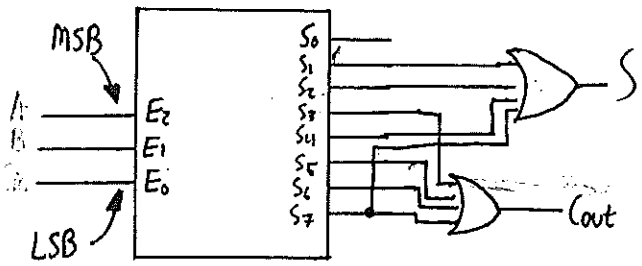
Ejercicio 2

Implementar un sumador de dos bits con acarreo serie (*full adder*) mediante un decodificador de tres entradas de control y ocho salidas y las puertas que necesite.

A	B	Cin	Cout	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

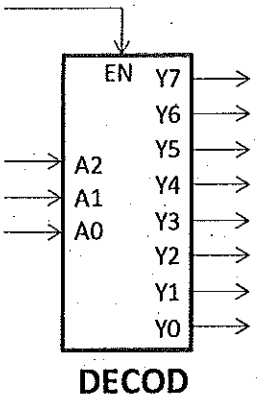


• Implementación mediante un decodificador 3 a 8:



PROBLEMAS TEMA 3

1. Utilizando únicamente 5 decodificadores como el que se muestra en la figura (cuya tabla de verdad se adjunta), construir un demultiplexor de 5 entradas de control.



EN	A2	A1	A0	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0
0	X	X	X	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0	0	0	1	0
1	0	1	0	0	0	0	0	0	1	0	0
1	0	1	1	0	0	0	0	1	0	0	0
1	1	0	0	0	0	0	1	0	0	0	0
1	1	0	1	0	0	1	0	0	0	0	0
1	1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0

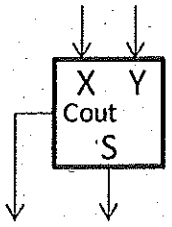
2. Dada la función lógica: $F = \bar{A}DC + B\overline{D} + A\bar{B}$, implementarla con un único multiplexor de 4 entradas de datos. Se dispone de las variables negadas.

3. Se tienen 3 números de 4 bits expresados en complemento a dos. [A3..A0], [B3..B0], [C3..C0]. Utilizando sumadores completos, diseñe un circuito que realice la suma de los tres números de tal manera que la salida tenga el número de bits adecuado para que el resultado tenga el signo correcto.

4. Diseñe un circuito que multiplique dos números de dos bits [A1,A0] y [B1,B0] utilizando sumadores completos y puertas AND.

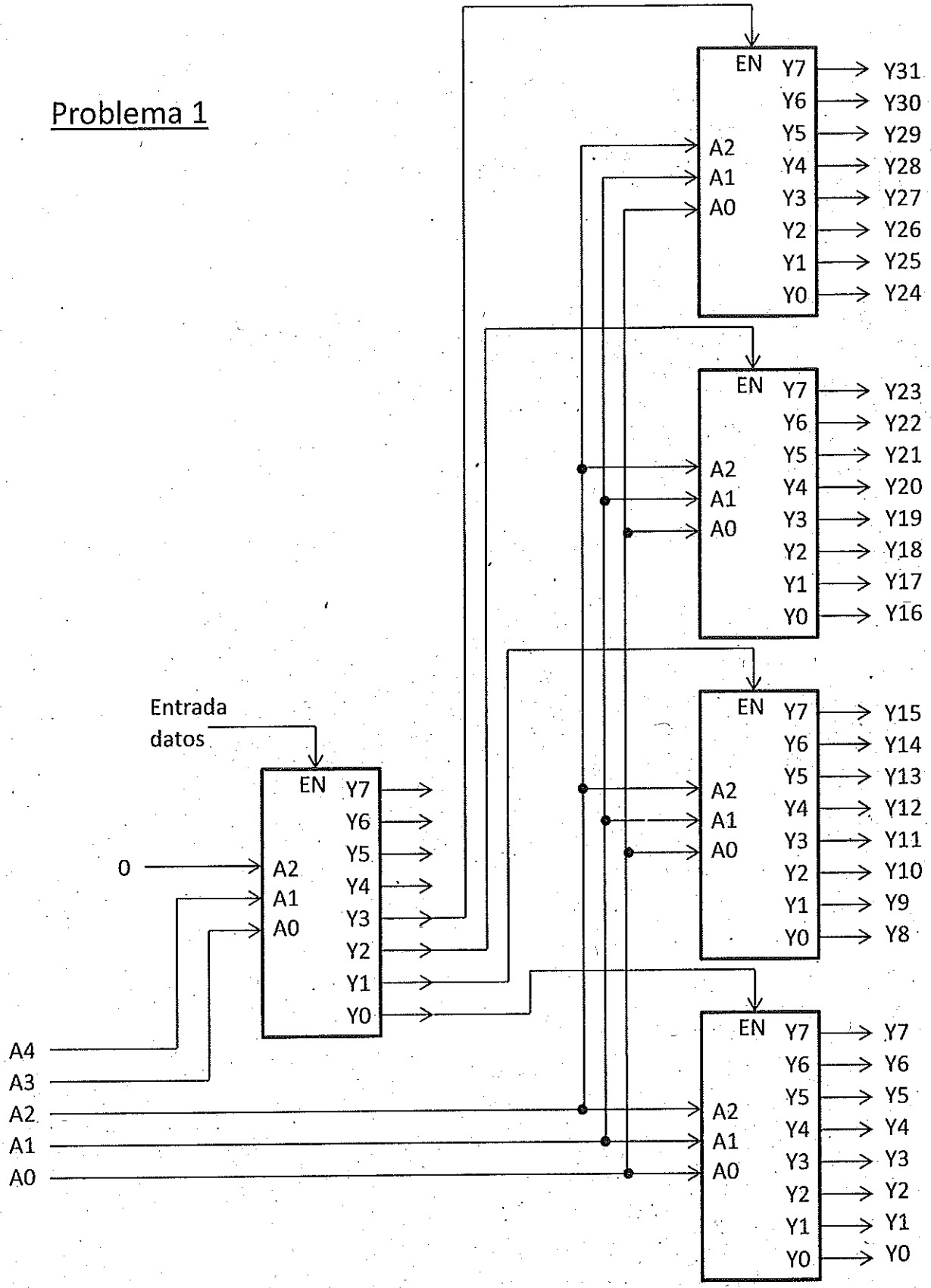
5. Construya un circuito que sume dos números A y B expresados en BCD y entregue el resultado también en BCD. Utilice para ello dos sumadores tipo 74HC283 y las puertas lógicas que considere oportunas.

6. El semisumador (*"half adder"*) mostrado en la figura, es un circuito que suma dos bits pero no posee entrada de acarreo. Utilizando 2 semisumadores y puertas lógicas, construya un sumador completo (*"full adder"*). Detalle los cálculos.



"Half Adder" (HA)

Problema 1


EN
Y7
Y6
Y5
Y4
Y3
Y2
Y1
Y0
A2
A1
A0
Y31
Y30
Y29
Y28
Y27
Y26
Y25
Y24
Y23
Y22
Y21
Y20
Y19
Y18
Y17
Y16
Y15
Y14
Y13
Y12
Y11
Y10
Y9
Y8
Entrada
datos
0
A4
A3
A2
A1
A0

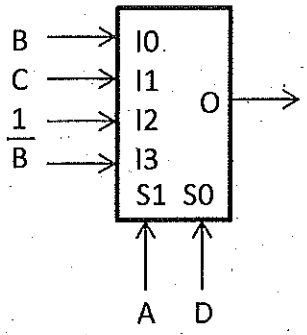
Problema 2

Elegimos A y D como variables de control:

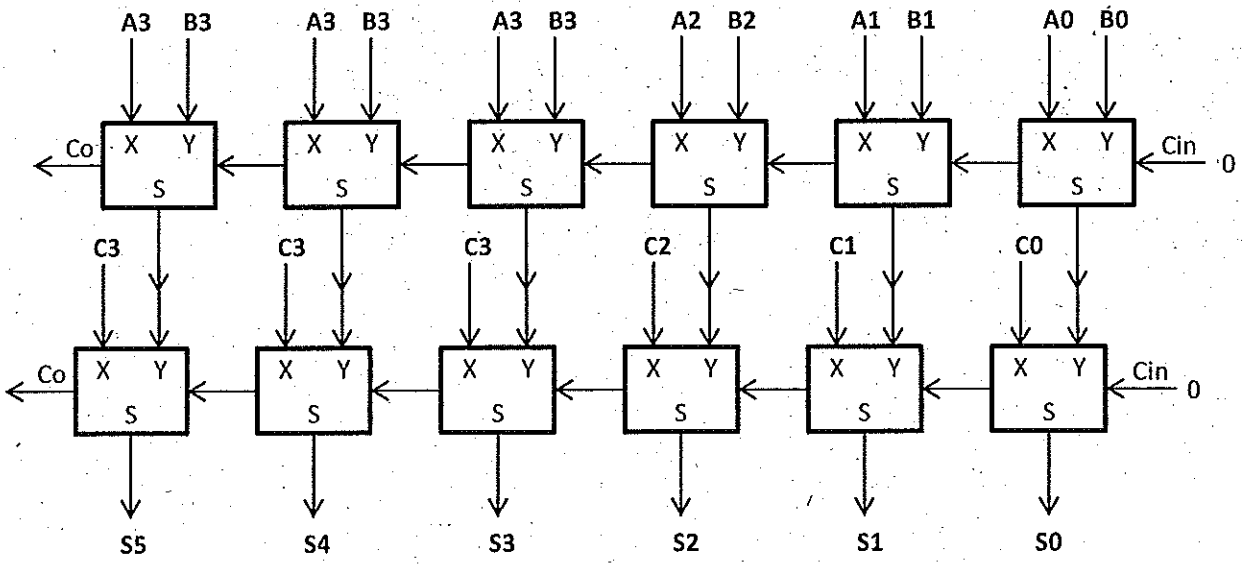
$$F = \overline{A}DC + B\overline{D} + A\overline{B} = \overline{A}DC + (A + \overline{A})B\overline{D} + A\overline{B}(D + \overline{D})$$

$$F = \overline{A}DC + AB\overline{D} + \overline{A}B\overline{D} + A\overline{B}D + A\overline{B}\,\overline{D}$$

$$F = \overline{AD} \cdot B + \overline{A}D \cdot C + A\overline{D} + AD \cdot \overline{B}$$



Problema 3

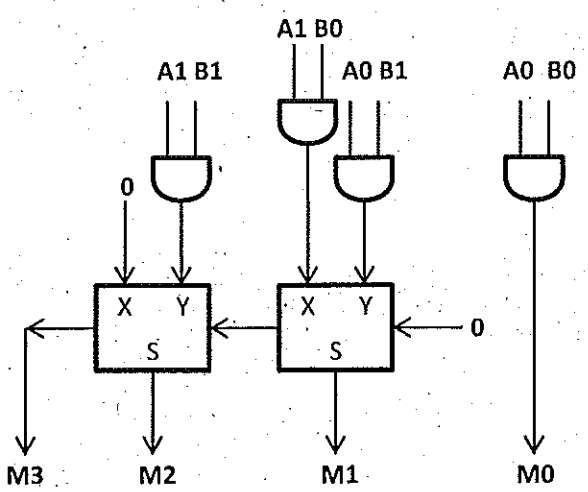


Dado que el resultado final puede tener 6 bits, extendemos el signo de todos los números (A, B y C) hasta los 6 bits.

Problema 4

Se trata de resolver la siguiente multiplicación:

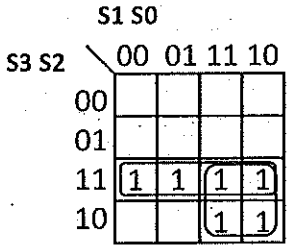
	A1	A0
x	B1	B0
	A1B0	A0B0
A1B1	A0B1	
A1B1	A1B0+ A0B1	A0B0



Problema 5

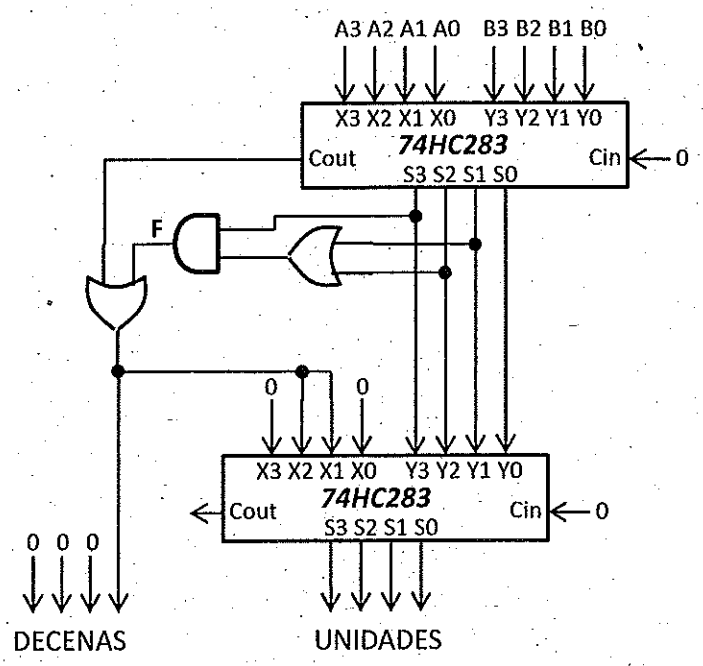
Si sumamos dos dígitos BCD con un sumador binario, el resultado puede exceder el valor 9 (lo cual no tiene sentido en BCD). En estos casos, para obtener el resultado correcto en BCD, debemos restar 10 al valor obtenido (unidades) y añadir un 1 a las decenas. La operación de restar 10, se implementa sumando -10 en complemento a 2 (10110), que para 4 bits resulta ser 0110 = 6 decimal.

Para detectar si el resultado excede el valor de 9 debemos construir una lógica digital que compruebe su hay acarreo de salida (valor mayor de 15) o si la salida del sumador es mayor de 9.

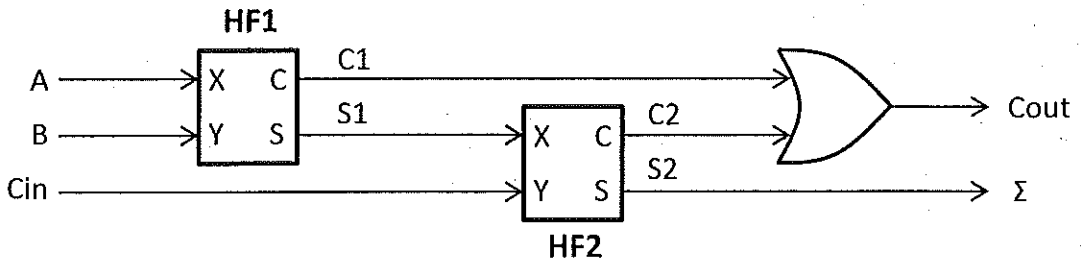


F = S3 S2 + S3 S1
= S3 (S2 + S1)

F indica si S>9



Problema 6



En un sumador completo las salidas deben cumplir:

$$\Sigma = A \oplus B \oplus Cin$$

$$Cout = A \cdot B + A \cdot Cin + B \cdot Cin$$

En el semisumador HF1 tenemos:

$$S1 = A \oplus B$$

$$C1 = A \cdot B$$

En el semisumador HF2 tenemos:

$$\boxed{S2 = S1 \oplus Cin = A \oplus B \oplus Cin = \Sigma}$$

$$C2 = S1 \cdot Cin = (A \oplus B) \cdot Cin = \overline{A}B \cdot Cin + A\overline{B} \cdot Cin$$

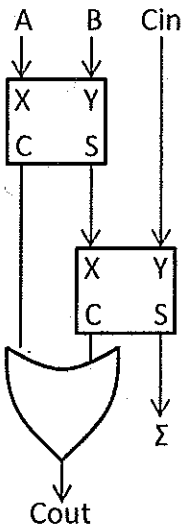
A la salida de la puerta OR tenemos:

$$Cout = C1 + C2 = AB + \overline{A}B \cdot Cin + A\overline{B} \cdot Cin = AB + AB \cdot Cin + AB \cdot Cin + \overline{A}B \cdot Cin + A\overline{B} \cdot Cin =$$

$$= AB + AB \cdot Cin + AB \cdot Cin + \overline{A}B \cdot Cin + A\overline{B} \cdot Cin = AB + A(B + \overline{B})Cin + B(A + \overline{A})Cin \Rightarrow$$

$$\boxed{Cout = AB + ACin + BCin}$$

Visto de otro modo, sumando los dígitos por orden, tenemos:



Los dígitos de menor orden (A,B y Cin, en azul) se suman con sumadores encadenados. Los dígitos de orden superior (acarreos, en rojo) se deben sumar. Para ello sería necesaria una puerta XOR. Pero dado que dichos acarreos no pueden ser 1 simultáneamente, se puede sustituir la XOR por una OR.

Ejercicio 1 | Diseñar un biestable J-K utilizando un biestable T con entrada de reloj.

- Tabla de verdad de un J-K:

Variables

J	K	Q_t	Q_{t+1}	T
0	0	0	0	0
0	0	1	1	0
0	1	0	0	0
0	1	1	0	1
1	0	0	1	1
1	0	1	1	0
1	1	0	1	1
1	1	1	0	1

T	Q_{t+1}
0	
1	

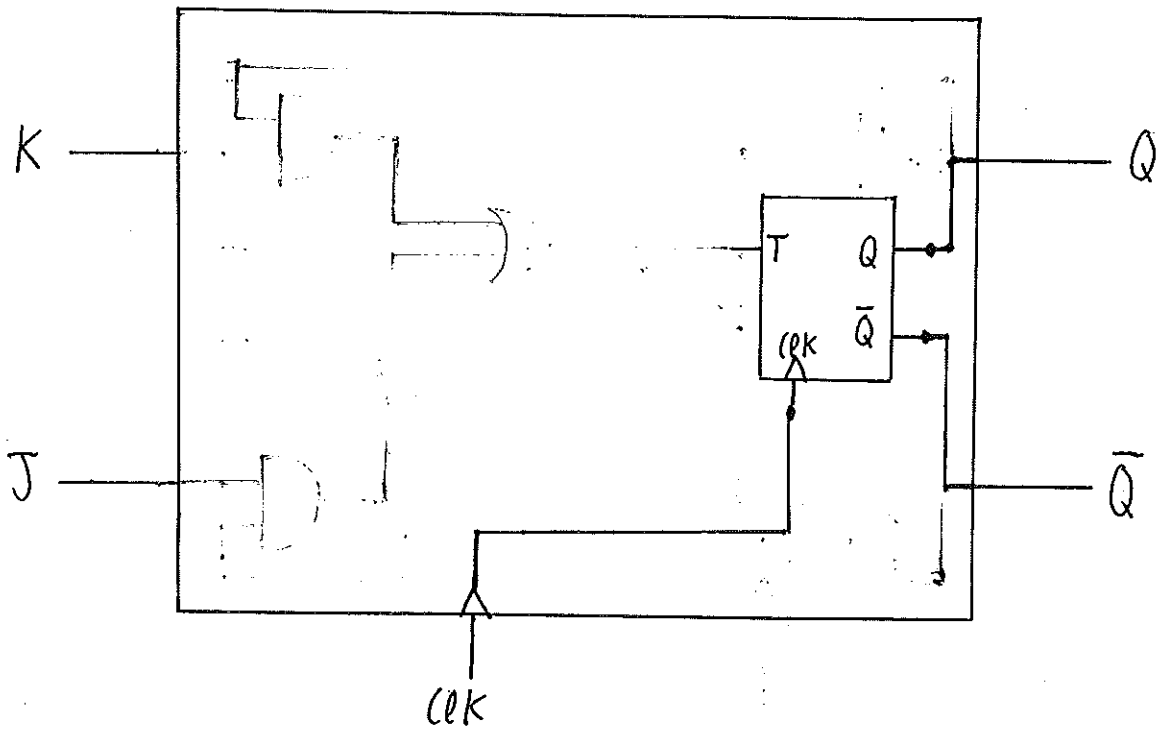
En un biestable T provocamos conmutación (↕) con T = 1 y no provocamos conmutación (↕) con T = 0.

¿Que debemos poner en cada para provocar la transición correspondiente?

- Lo último que tenemos que hacer es implementar la función T con las variables J, K, Q_t (ya no usamos la columna Q_{t+1})

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

$T = \bar{Q}.J + Q.K$



Realizar un biestable J-K síncrono mediante un biestable D síncrono

- Tabla de verdad:

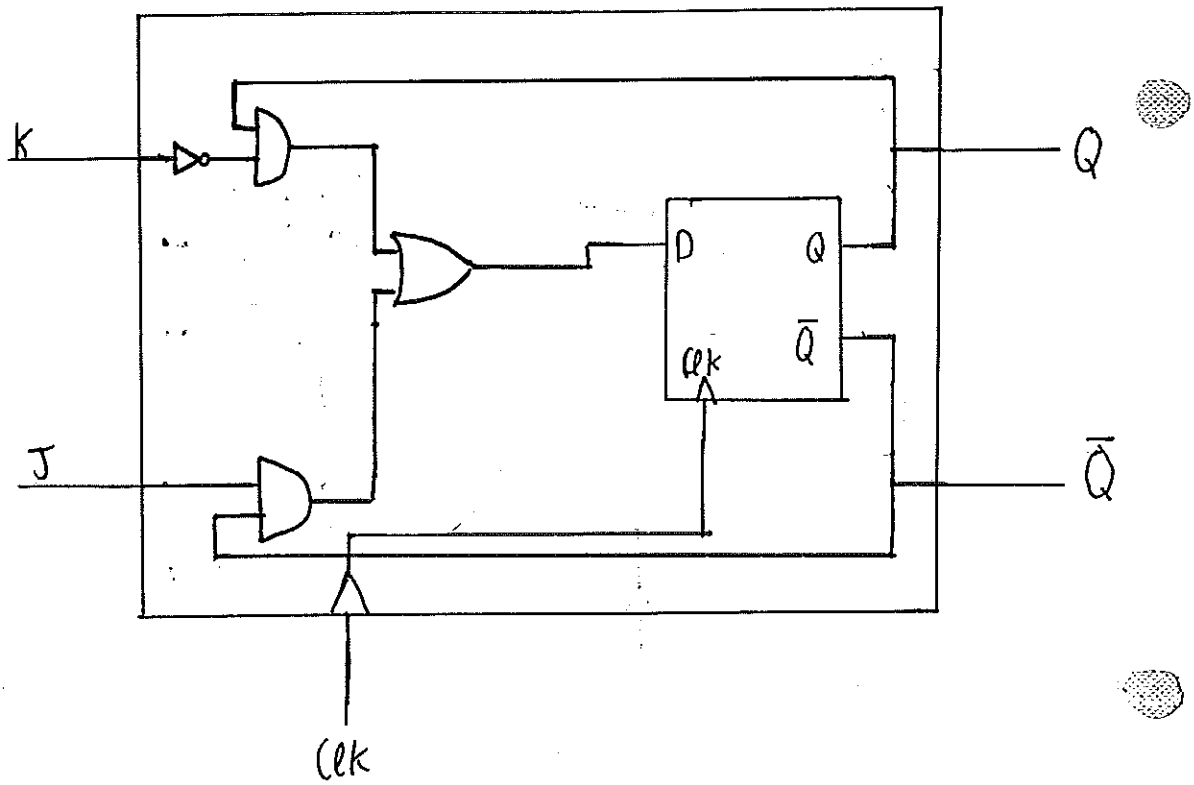
J	K	Q_t	Q_{t+1}	D
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	0	0
1	0	0	1	1
1	0	1	1	1
1	1	0	1	1
1	1	1	0	0

$Q_{t+1} = D$

- Simplificación de $D(J, K, Q)$:

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

$$D = \bar{Q}J + Q.\bar{K}$$



Ejercicio 2: Tema 4 (no está en la carpeta)

a) J-K mediante D (ya está hecho detrás del ejercicio 1))

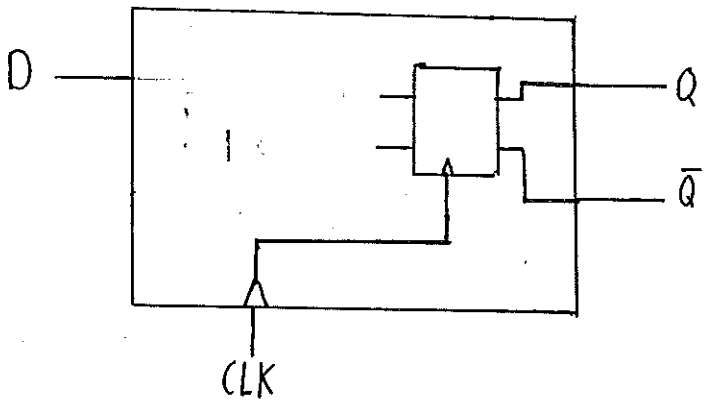
b) D mediante JK

D	Q_t	Q_{t+1}	J	K
0	0	0	0	X
0	1	0	X	1
1	0	1	1	X
1	1	1	X	0

Tabla de excitaciones de un biestable J-K

Q_t	Q_{t+1}	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

$J = D$ $K = \overline{D}$



Páginas 4.11 - 4.12

Ejercicio 3

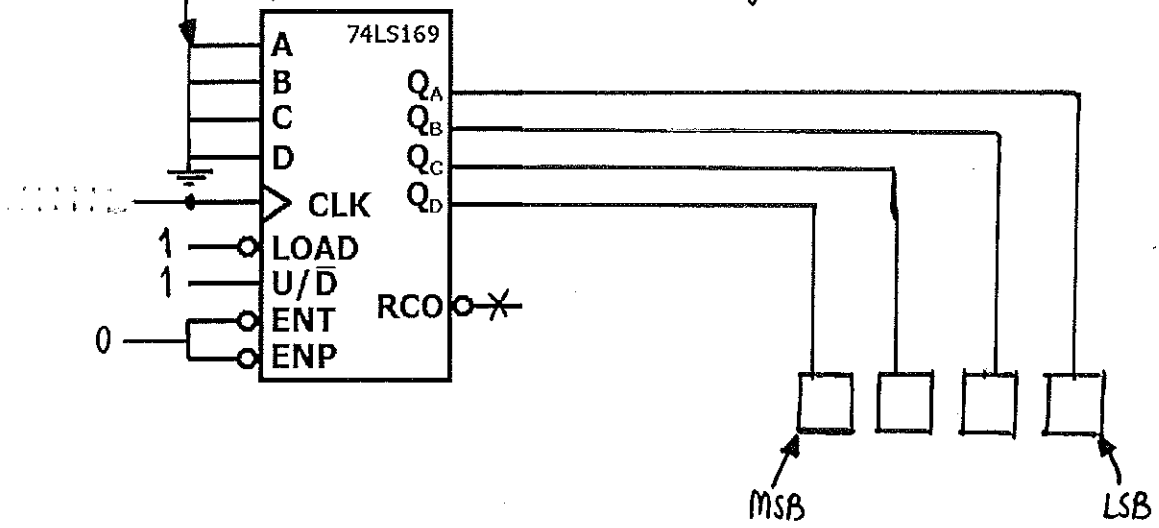
Utilizando contadores 74HC169 y la lógica adicional que precise realice contadores que cumplan las siguientes especificaciones:

1) Contador que cuente de 0 a 15 cíclicamente.
2) Contador que cuente de 6 a 15 cíclicamente.
3) Contador que cuente de 0 a 11 cíclicamente.
4) Contador que cuente de 6 a 11 cíclicamente.
5) Contador que cuente de 15 a 0 cíclicamente.
6) Contador que cuente de 15 a 6 cíclicamente.
7) Contador que cuente de 11 a 0 cíclicamente.
8) Contador que cuente de 11 a 6 cíclicamente.
9) Contador que cuente de 0 a 15 y de 15 a 0 cíclicamente.
10) Contador que cuente de 6 a 15 y de 15 a 6 cíclicamente.
11) Contador que cuente de 0 a 11 y de 11 a 0 cíclicamente.
12) Contador que cuente de 6 a 11 y de 11 a 6 cíclicamente.

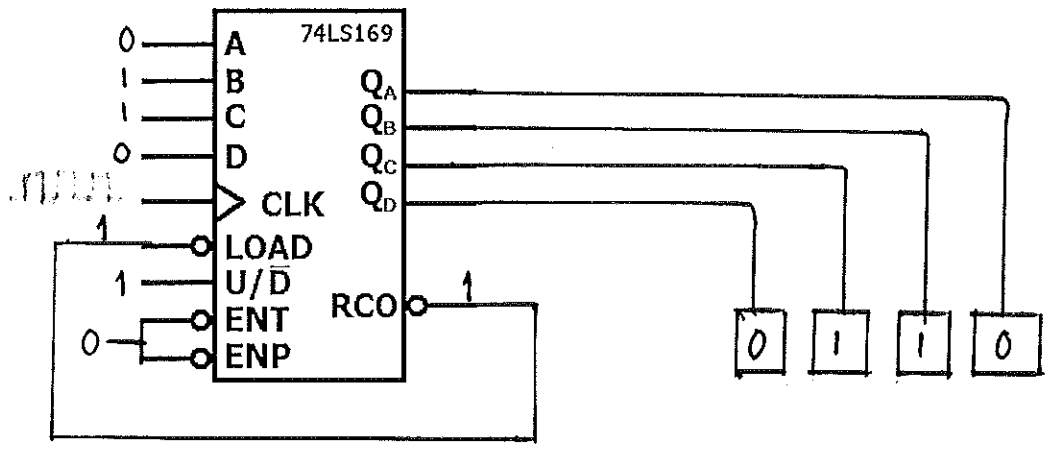
U/$\bar{D}$ = Up & Down

1 ... 13, 14, 15, 0, 1, 2, 3, ... , 14, 15, 0, 1, 2, ...

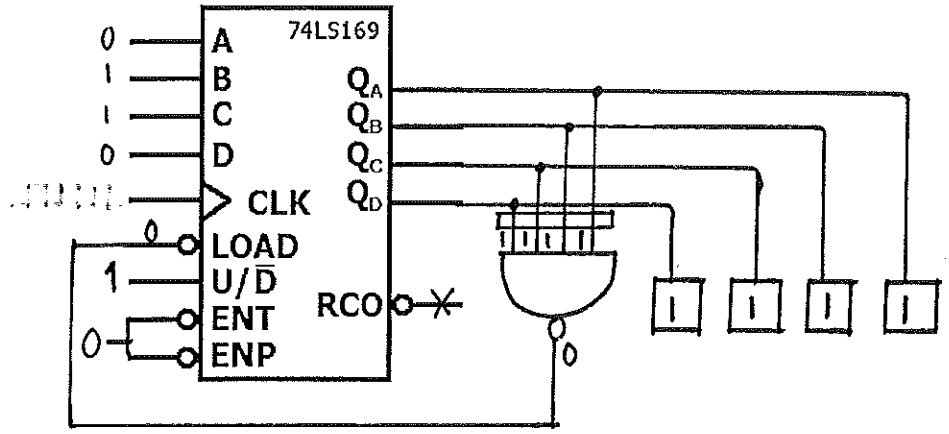
Todas las entradas deben tener un valor conocido!!
Aunque en este caso no se van a cargar nunca los datos



2 ... 13, 14, 15, 6, 7, 8, ... , 14, 15, 6, 7, 8, ...



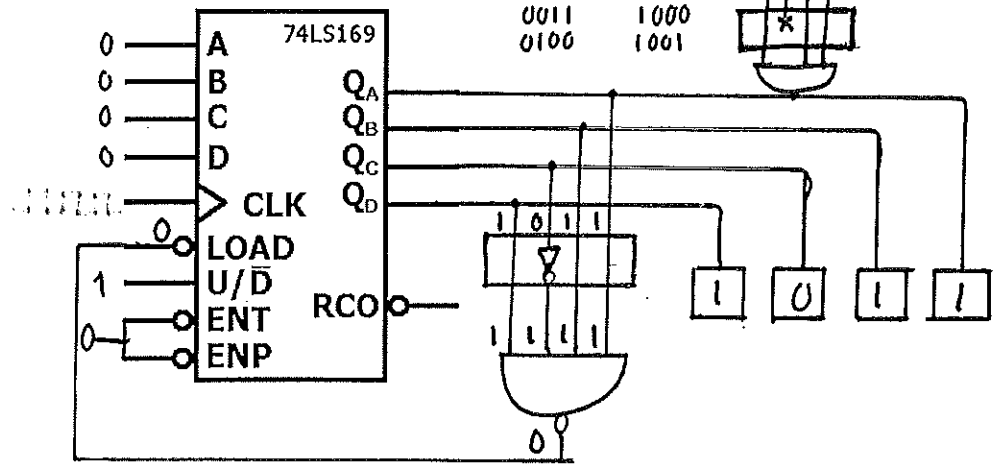
Otra forma: **Sin** utilizar RCO (sólo con puertas lógicas)



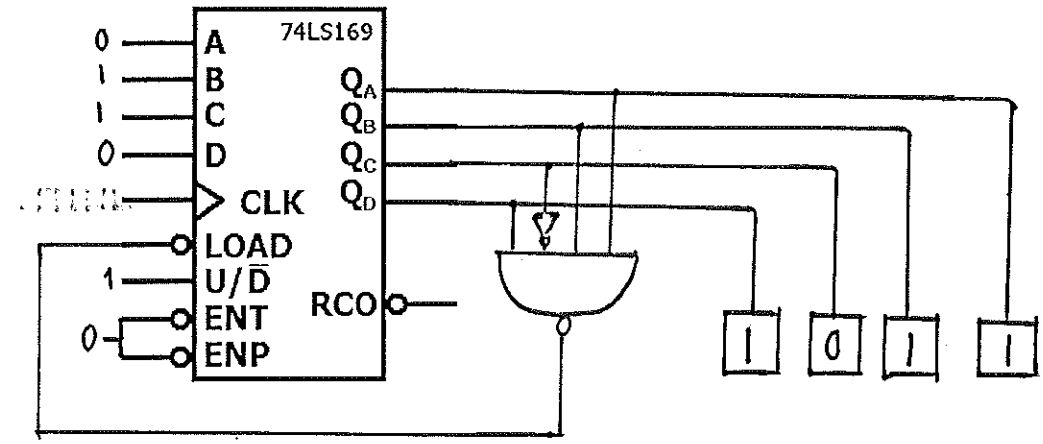
3 ... 9, 10, 11, 0, 1, 2, ... , 9, 10, 11, 0, 1, ...

No hacía falta una NAND de 4 entradas:

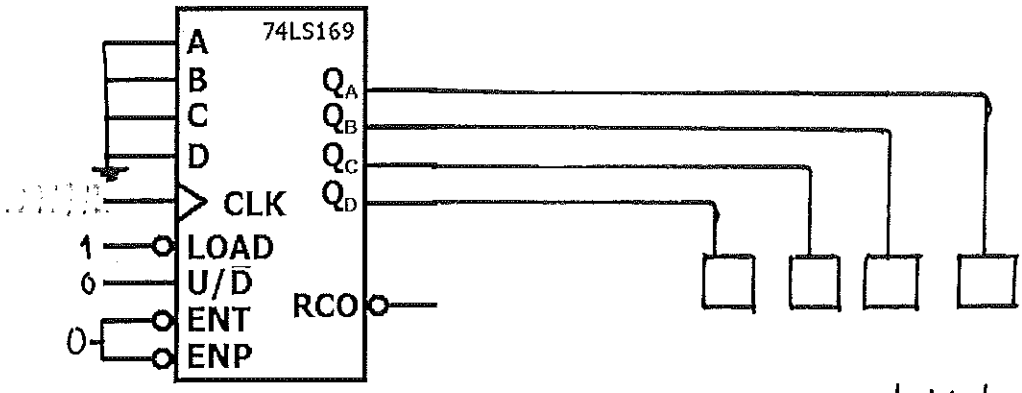
0000 0101 1010
0001 0110 1011
0010 0111
0011 1000
0100 1001



4 ... 9, 10, 11, 6, 7, 8, 9, 10, 11, 6, 7, ...



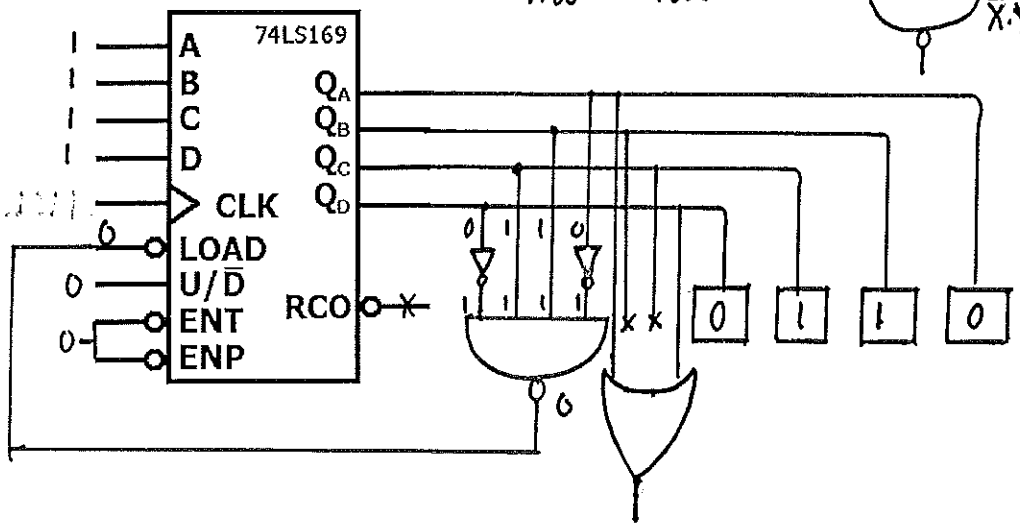
5 ... 3, 2, 1, 0, 15, 14, 13, ... , 2, 1, 0, 15, 14, ...



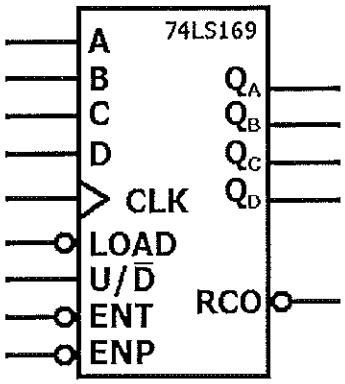
6 ... 8, 7, 6, 15, 14, 13, ... , 8, 7, 6, 15, 14, 13, ...

1111 1011 0111
1110 1010 0110
1101 1001
1100 1000

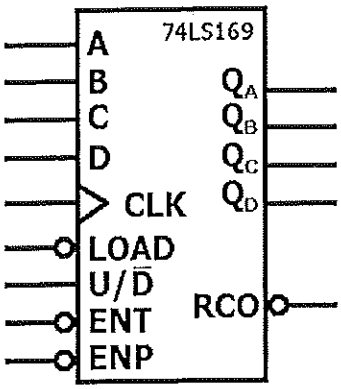
$\overline{X}.\overline{Y} = \overline{\overline{X}+\overline{Y}} = X+Y$



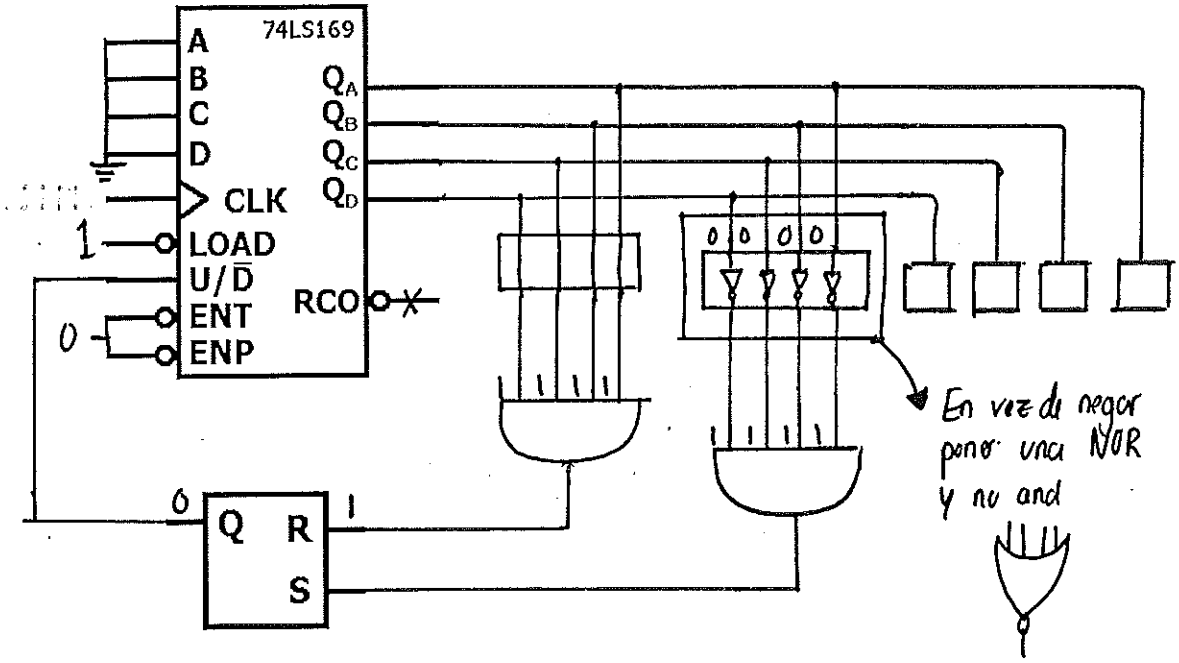
7 ... 3, 2, 1, 0, 11, 10, 9, ... , 3, 2, 1, 0, 11, 10, 9, ...



8 ... 8, 7, 6, 11, 10, 9, 8, 7, 6, 11, 10, 9, 8, ...



9 ... 12, 13, 14, 15, 14, 13, 12, ... , 2, 1, 0, 1, 2, 3, ... "Avisa" cuando llegamos a 15 "Avisa" cuando llegamos a 0



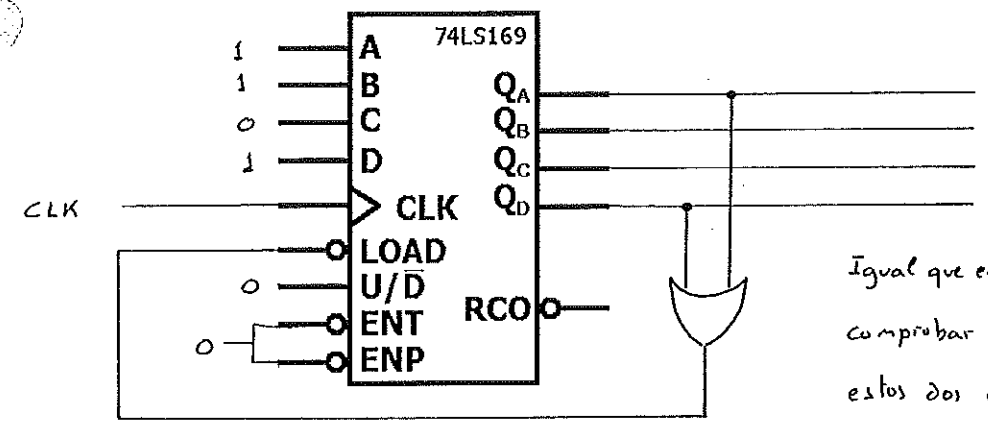
Contadores resueltos

8

... 8, 7, 6, 11, 10, 9, 8, 7, 6, 11, 10, 9, 8, ...

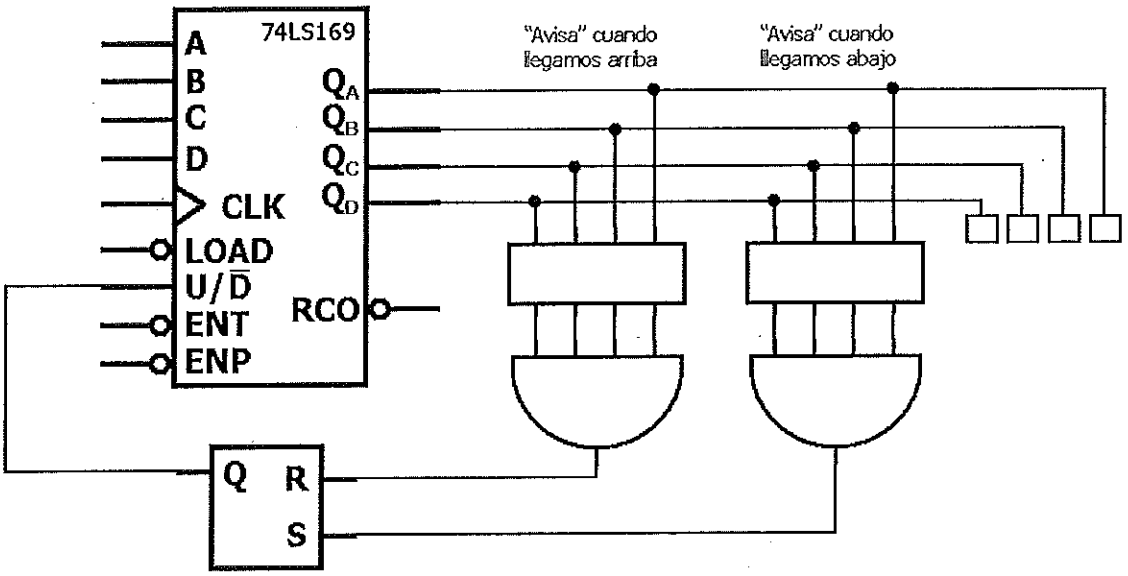
D'6 = B'0110

D'11 = B'1011



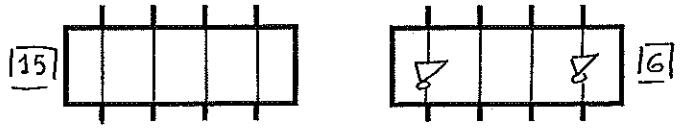
Igual que en el 16 basta con comprobar que, bajando, tengamos estos dos ceros: 0110

10, 11, 12



... 6, 7, 8, 9, ... , 14, 15, 14, 13, 12, ... , 8, 7, 6, 7, 8, 9 ...

Bajando, bastaría con comprobar estos dos ceros: 0110

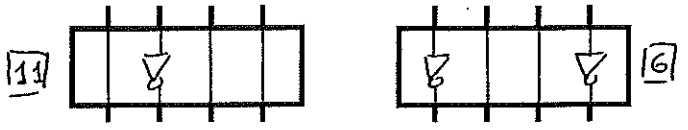


... 0, 1, 2, 3, ... , 9, 10, 11, 10, 9, ... , 2, 1, 0, 1, 2, 3 ...

Subiendo, bastaría con comprobar estos tres unos: 1011. Además, en el otro avisador podemos usar NOR.

... 6, 7, 8, 9, 10, 11, 10, 9, 8, 7, 6, 7, 8, 9, 10, 11, 10, 9 ...

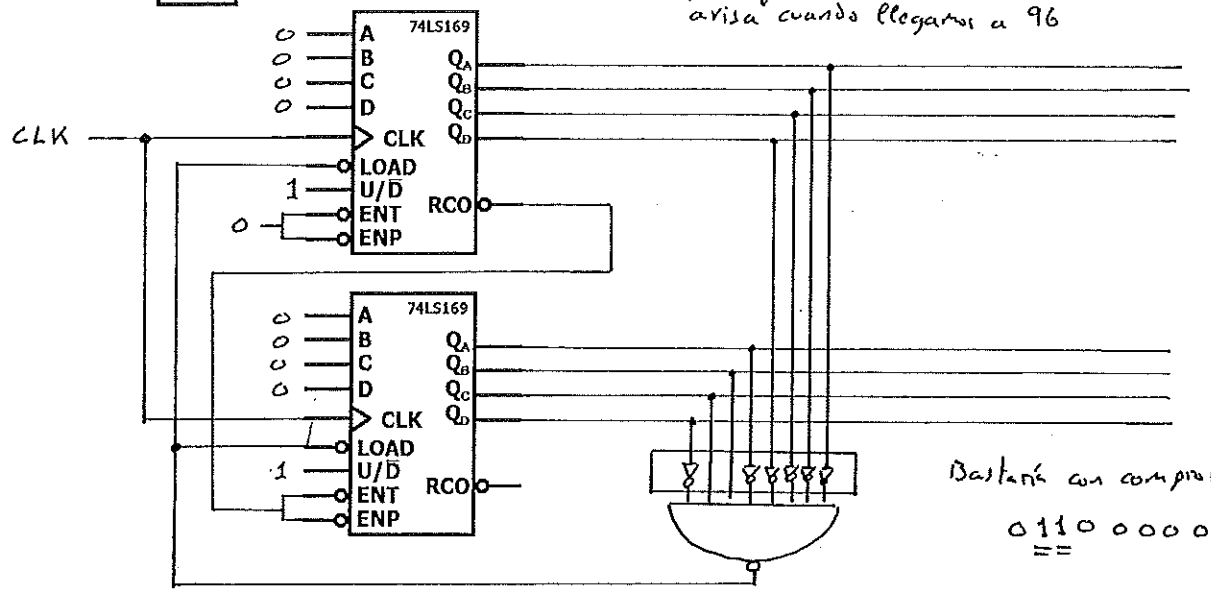
Mismos comentarios respecto a 1011 y 0110



D'96 = B'01100000

15 ... 94, 95, 96, 0, 1, 2, ... , 94, 95, 96, 0, 1, 2, ...

"avisa" cuando llegamos a 96

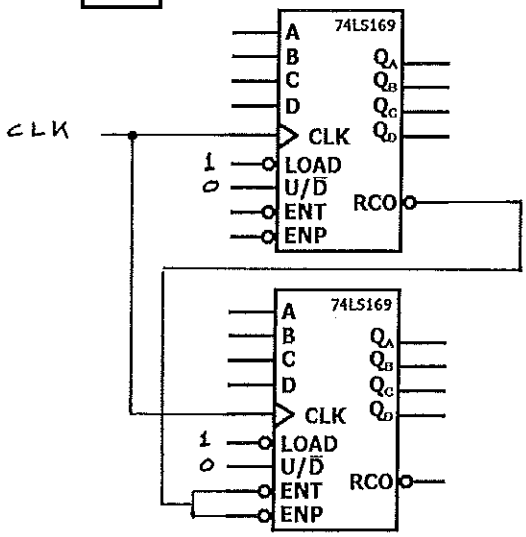


Bastará con comprobar que:

0110 0000

==

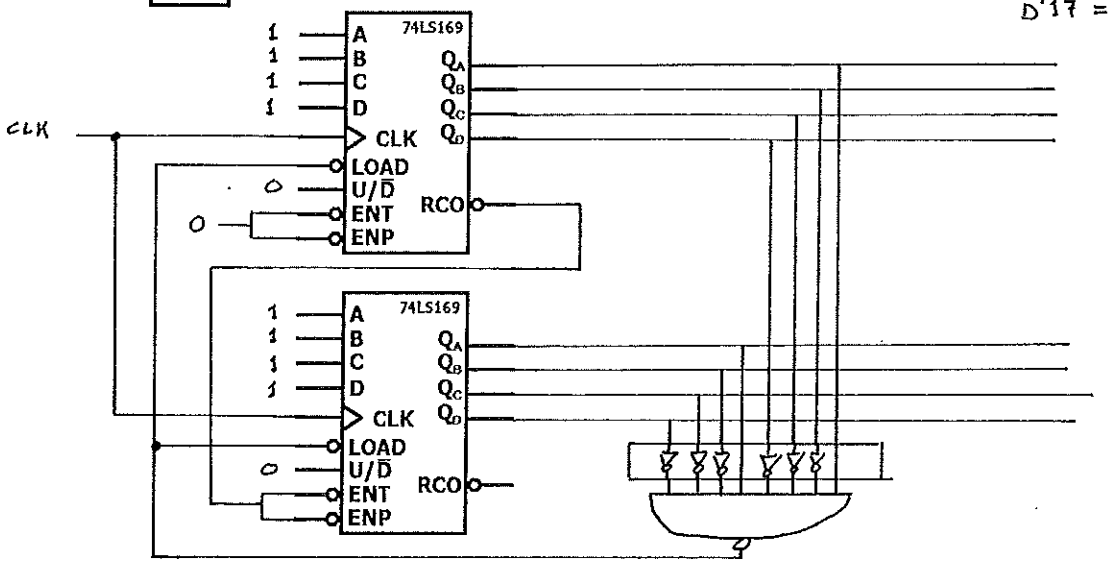
17 ... 3, 2, 1, 0, 255, 254, 253, ... , 2, 1, 0, 255, 254, ...



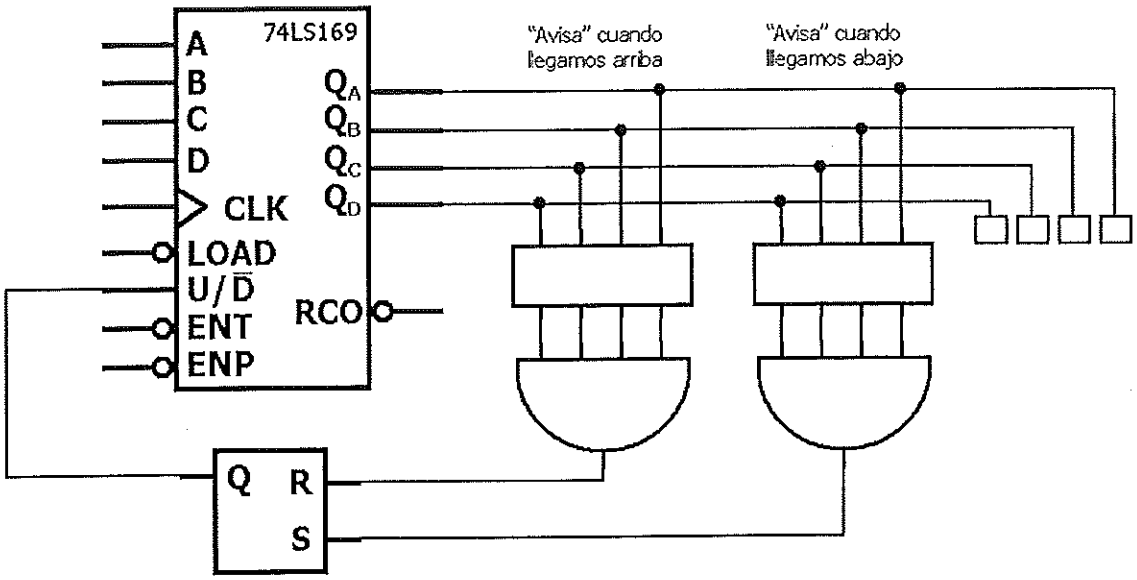
Este es el comportamiento "nativo" de un contador, hacia abajo.

18 ... 19, 18, 17, 255, 254, 253, ... , 19, 18, 17, 255, 254, 253, ...

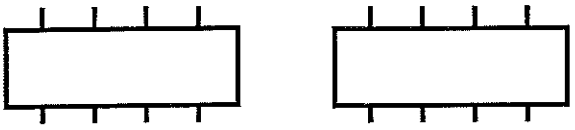
D'17 = B'00010001



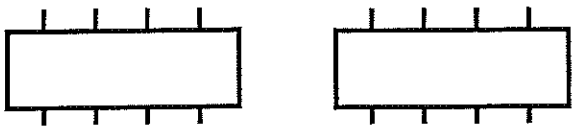
10, 11, 12



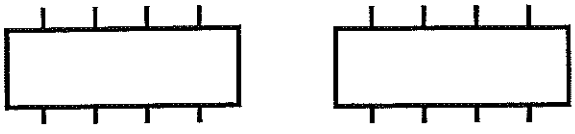
... 6, 7, 8, 9, ... , 14, 15, 14, 13, 12, ... , 8, 7, 6, 7, 8, 9 ...



... 0, 1, 2, 3, ... , 9, 10, 11, 10, 9, ... , 2, 1, 0, 1, 2, 3 ...



... 6, 7, 8, 9, 10, 11, 10, 9, 8, 7, 6, 7, 8, 9, 10, 11, 10, 9 ...

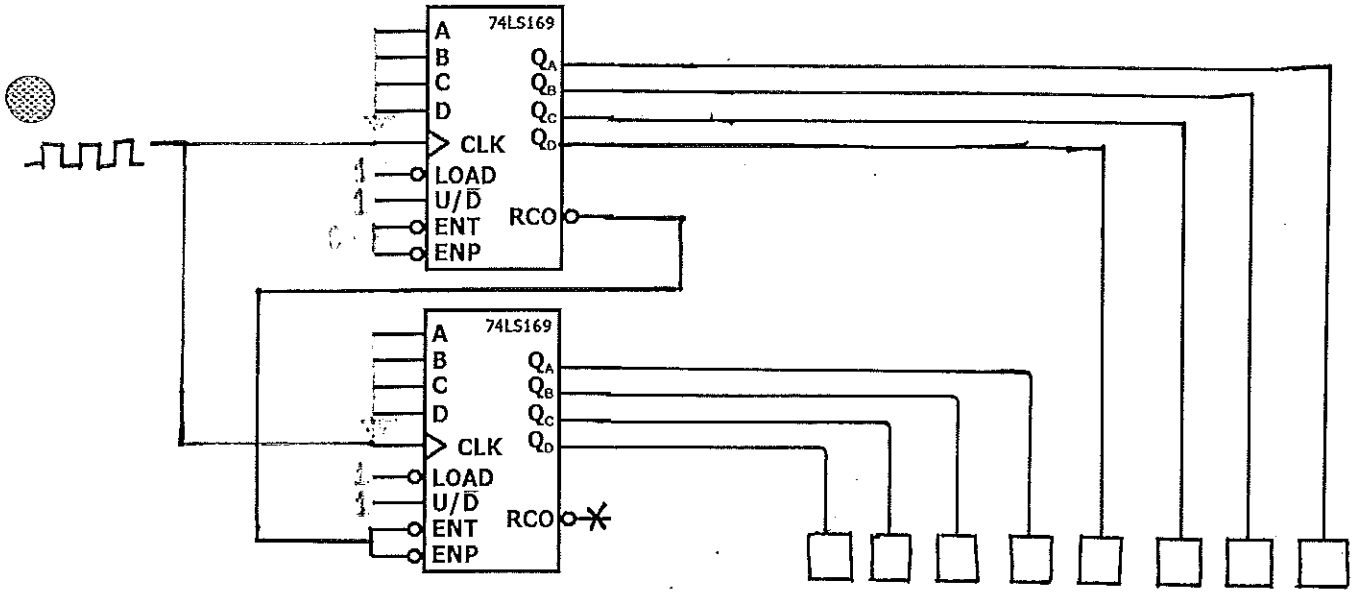


Ejercicio 4

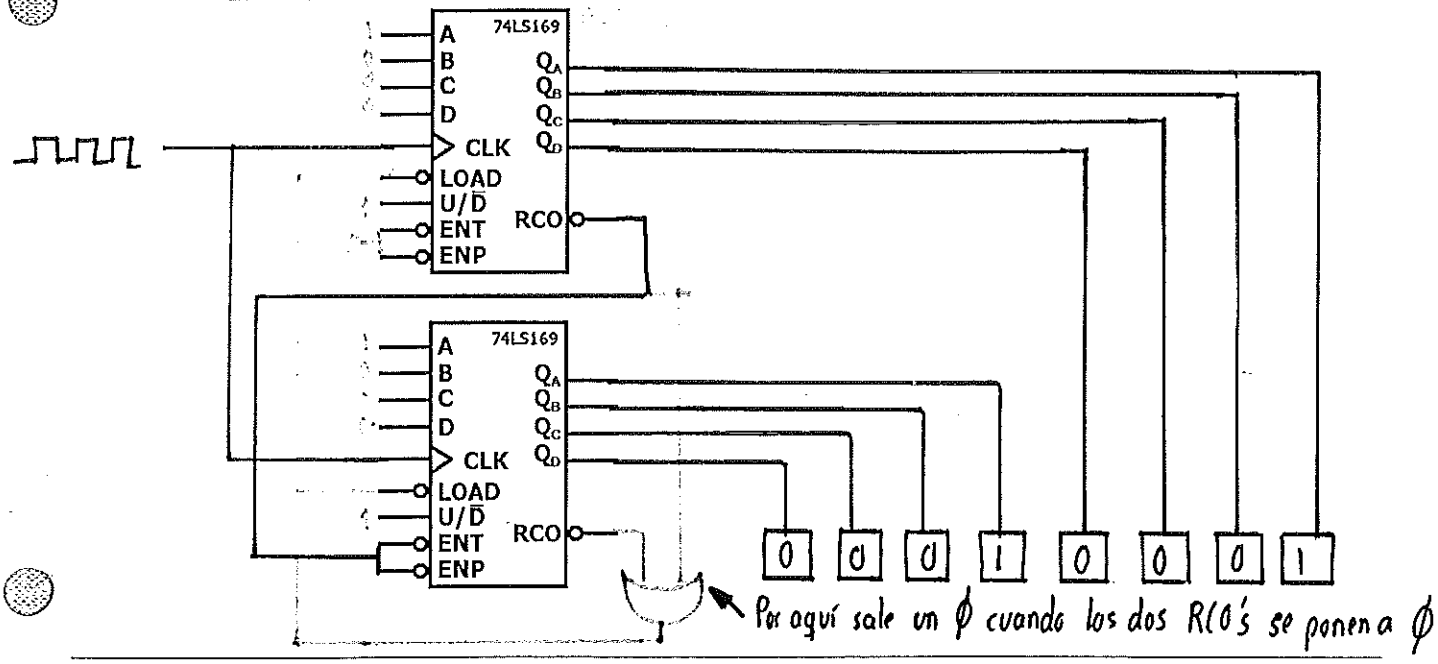
Utilizando dos contadores 74LS169 y la lógica adicional que precise realice circuitos que cumplan las siguientes especificaciones. No tenga en cuenta la inicialización de los mismos.

13) Contador que cuente de 0 a 255 cíclicamente.
14) Contador que cuente de 17 a 255 cíclicamente.
15) Contador que cuente de 0 a 96 cíclicamente.
16) Contador que cuente de 17 a 96 cíclicamente.
17) Contador que cuente de 255 a 0 cíclicamente.
18) Contador que cuente de 255 a 17 cíclicamente.
19) Contador que cuente de 96 a 0 cíclicamente.
20) Contador que cuente de 96 a 17 cíclicamente.
21) Contador que cuente de 0 a 255 y de 255 a 0 cíclicamente.
22) Contador que cuente de 17 a 255 y de 255 a 17 cíclicamente.
23) Contador que cuente de 0 a 96 y de 96 a 0 cíclicamente.
24) Contador que cuente de 17 a 96 y de 17 a 96 cíclicamente.

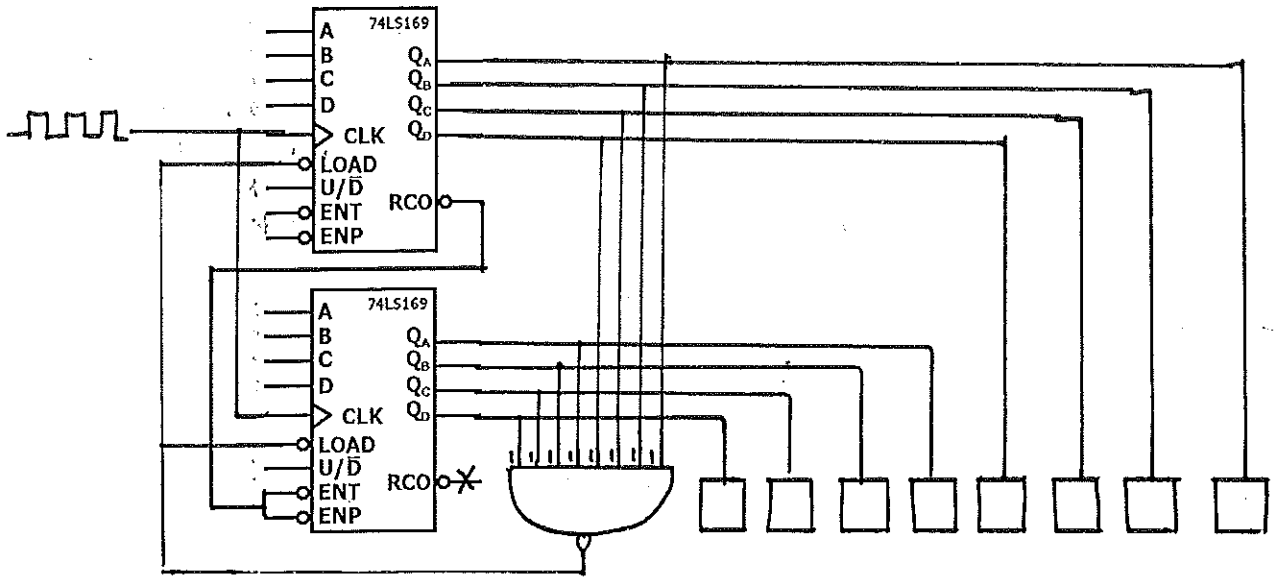
13 ... 253, 254, 255, 0, 1, 2, 3, ... , 253, 254, 255, 0, 1, 2, ...



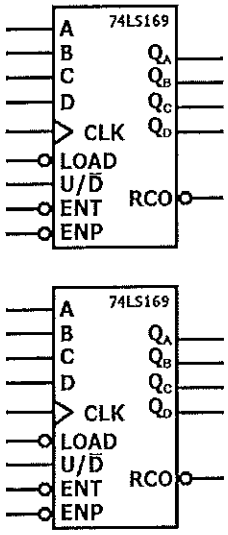
14 ... 253, 254, 255, 17, 18, 19, ... , 253, 254, 255, 17, 18, 19, ...



Otra forma: **Sin** utilizar RCO como LOAD (sólo con puertas lógicas)

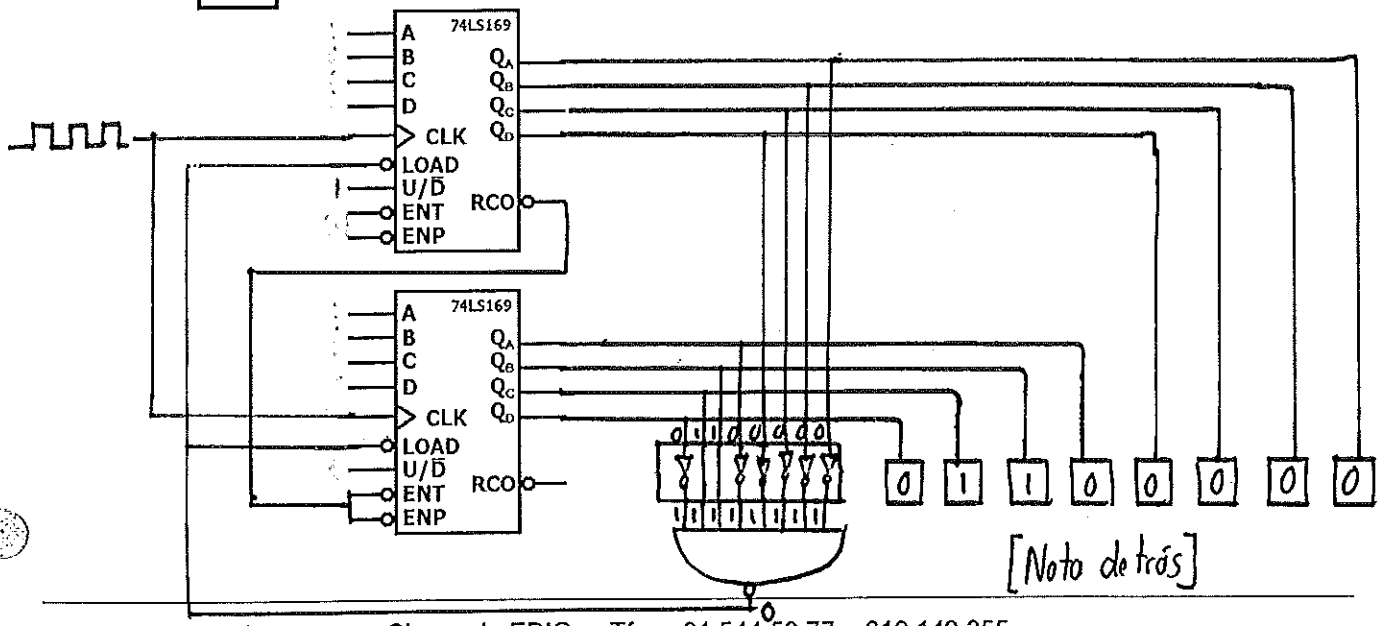


15 ... 94, 95, 96, 0, 1, 2, ... , 94, 95, 96, 0, 1, 2, ...



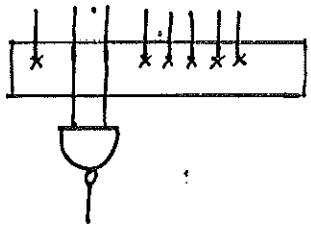
01100000 → 00010001

16 ... 94, 95, 96, 17, 18, 19, ... , 94, 95, 96, 17, 18, 19, ...



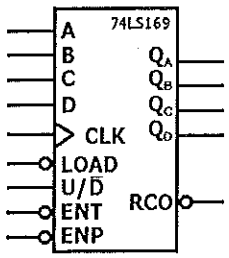
No hacía falta una puerta de 8 entradas ni los 6 inversores!!!

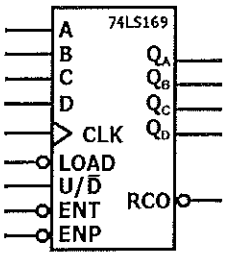
0	0	0	1	0 0 0 1
0	0	0	1	0 0 1 0
0	0	0	1	0 0 1 1
				...
0	1	0	1	1 1 1 0
0	1	0	1	1 1 1 1
0	1	1	0	0 0 0 0



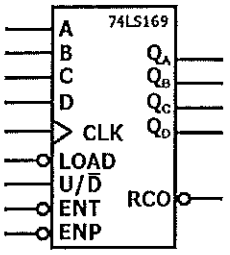
Solo tenemos que "avisar" cuando estos dos bits valgan '1'.

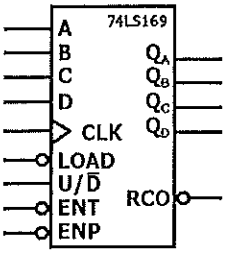
17 ... 3, 2, 1, 0, 255, 254, 253, ... , 2, 1, 0, 255, 254, ...



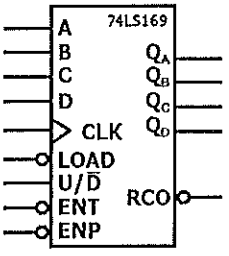


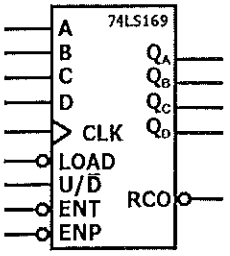
18 ... 19, 18, 17, 255, 254, 253, ... , 19, 18, 17, 255, 254, 253, ...



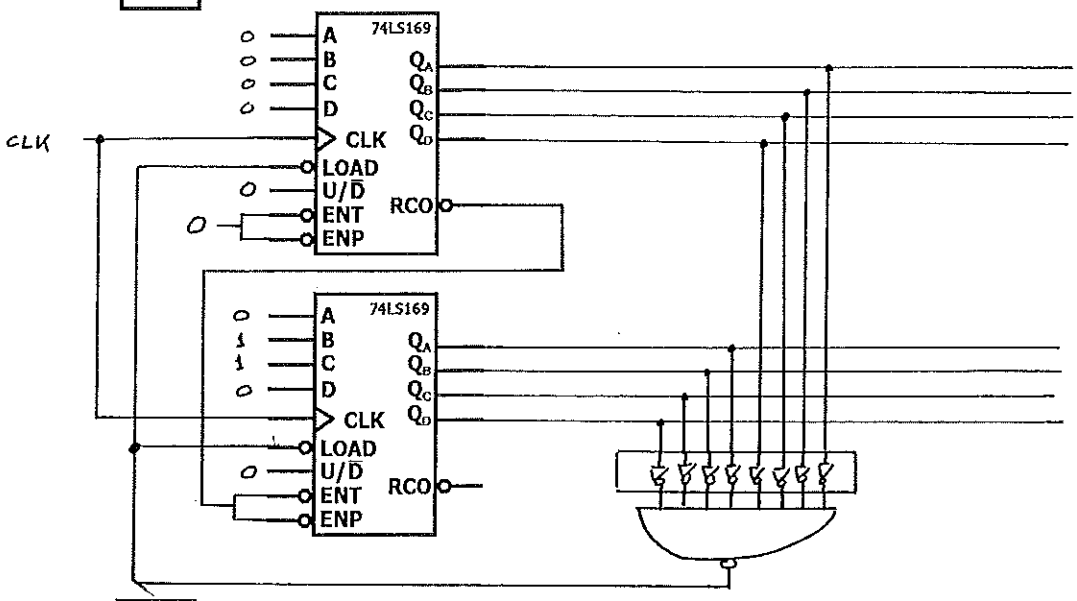


19 ... , 2, 1, 0, 96, 95, 94, ... , 2, 1, 0, 96, 95, ...

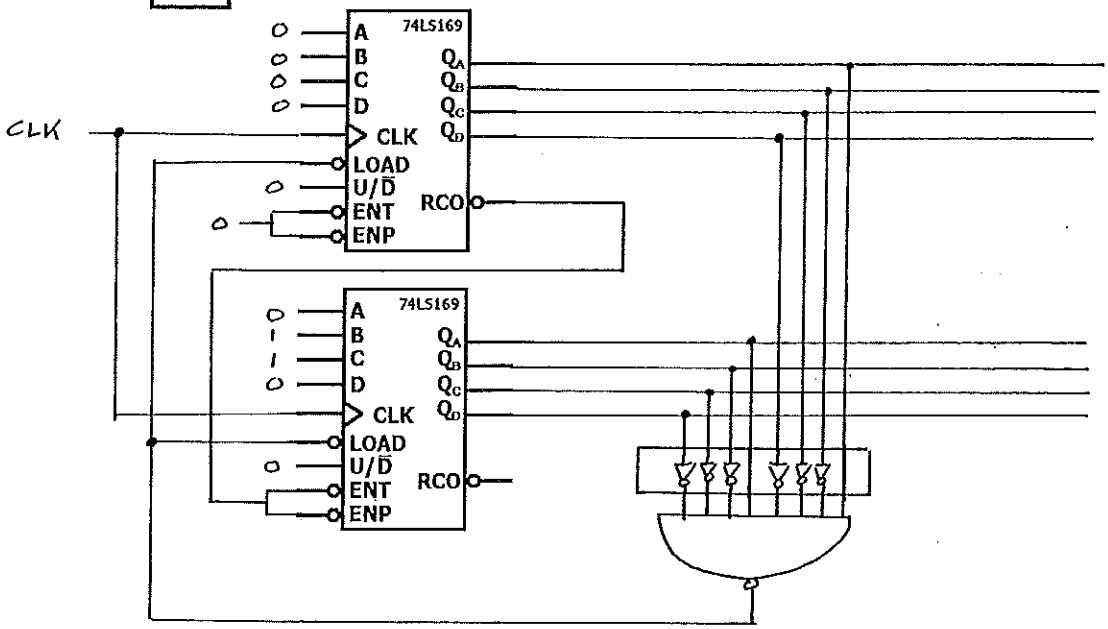




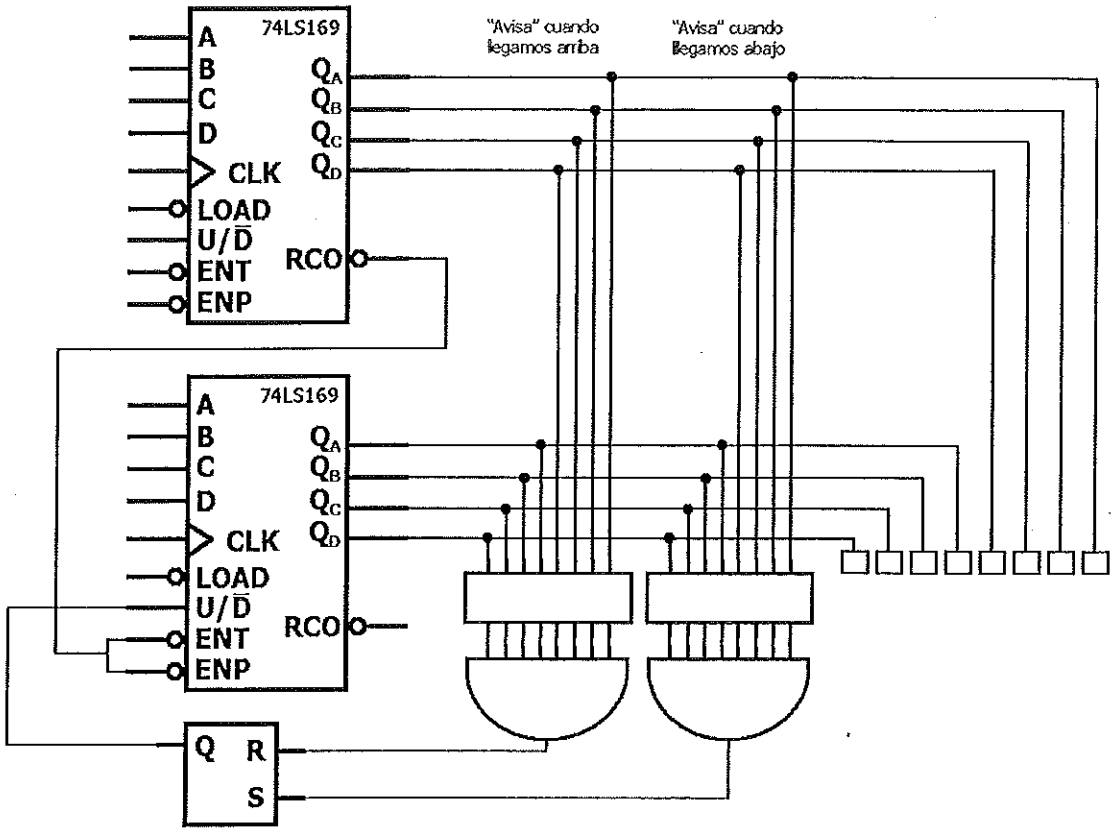
19 ... , 2, 1, 0, 96, 95, 94, ... , 2, 1, 0, 96, 95, ...



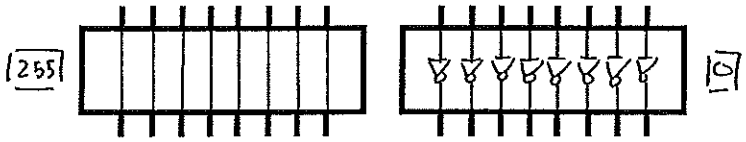
20 ... 19, 18, 17, 96, 95, 94, ... , 19, 18, 17, 96, 95, 94, ...



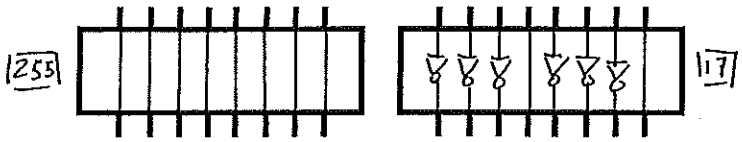
21, 22, 23, 24



... 253, 254, 255, 254, 253, 252 ... , 3, 2, 1, 0, 1, 2, 3, ... , 253, 254, 255, 254, 253, 252 ...

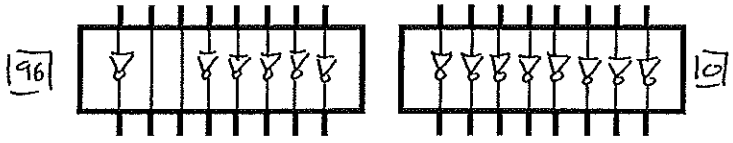


... 253, 254, 255, 254, 253, 252 ... , 19, 18, 17, 18, 19, ... , 253, 254, 255, 254, 253, 252 ...

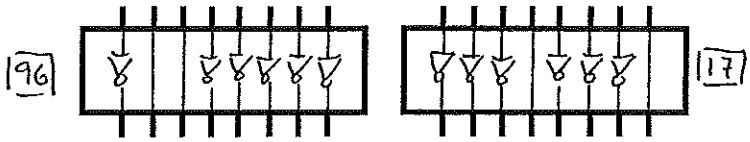


hay opciones más sencillas...

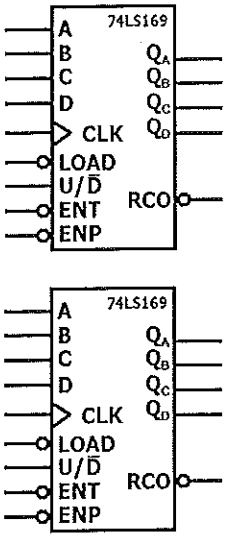
... 94, 95, 96, 95, 94, ... , 3, 2, 1, 0, 1, 2, 3, ... , 94, 95, 96, 95, 94, ...



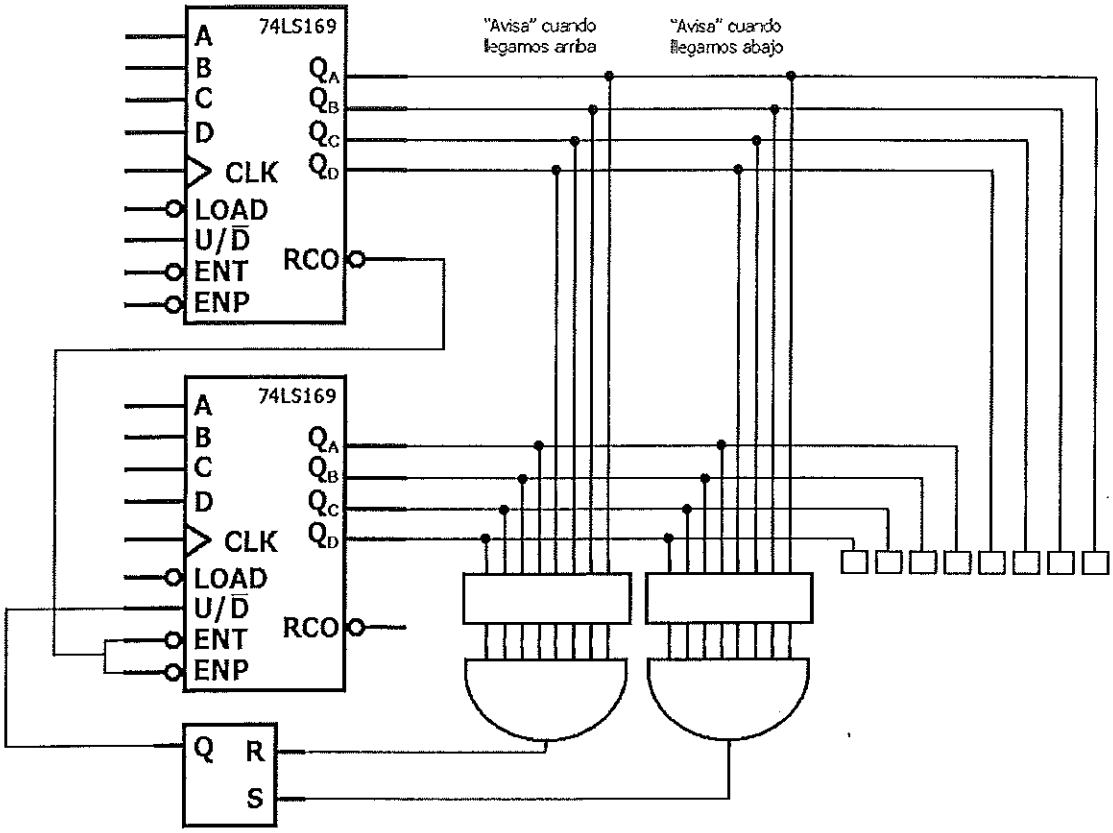
... 94, 95, 96, 95, 94, ... , 19, 18, 17, 18, 19, ... , 94, 95, 96, 95, 94, ...



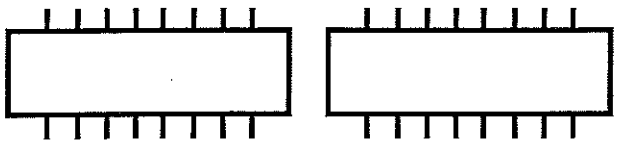
20 ... 19, 18, 17, 96, 95, 94, ... , 19, 18, 17, 96, 95, 94, ...



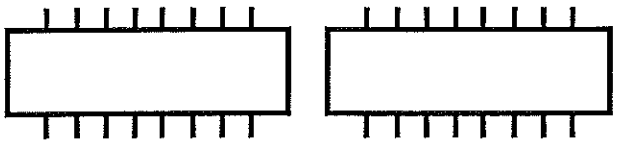
21, 22, 23, 24



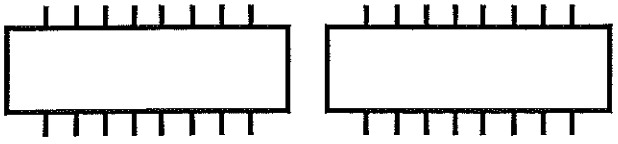
... 253, 254, 255, 254, 253, 252 ... , 3, 2, 1, 0, 1, 2, 3, ... , 253, 254, 255, 254, 253, 252 ...



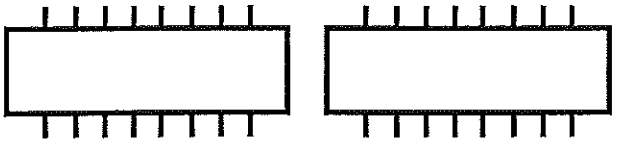
... 253, 254, 255, 254, 253, 252 ... , 19, 18, 17, 18, 19, ... , 253, 254, 255, 254, 253, 252 ...



... 94, 95, 96, 95, 94, ... , 3, 2, 1, 0, 1, 2, 3, ... , 94, 95, 96, 95, 94, ...



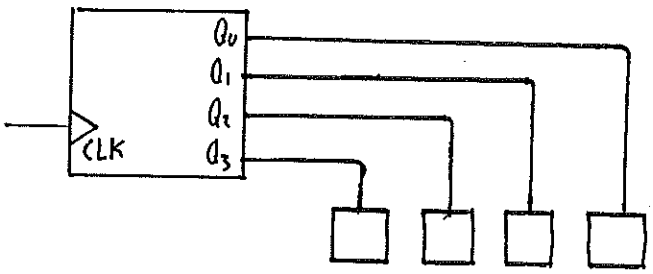
... 94, 95, 96, 95, 94, ... , 19, 18, 17, 18, 19, ... , 94, 95, 96, 95, 94, ...



Ejercicio 5

Diseñe un contador que cuente cíclicamente de 0 a 15 utilizando cuatro biestables J-K y la lógica adicional que precise.

Q_t estado act.				Q_{t+1} estado sig.				JK_3		JK_2		JK_1		JK_0	
Q_3	Q_2	Q_1	Q_0	Q_3	Q_2	Q_1	Q_0	J3	K3	J2	K2	J1	K1	J0	K0
0	0	0	0	0	0	0	1	0	X	0	X	0	X	1	X
0	0	0	1	0	0	1	0	0	X	0	X	1	X	X	1
0	0	1	0	0	0	1	1	0	X	0	X	X	0	1	X
0	0	1	1	0	1	0	0	0	X	1	X	X	1	X	1
0	1	0	0	0	1	0	1	0	X	X	0	0	X	1	X
0	1	0	1	0	1	1	0	0	X	X	0	1	X	X	1
0	1	1	0	0	1	1	1	0	X	X	0	X	0	1	X
0	1	1	1	1	0	0	0	1	X	X	1	X	1	X	1
1	0	0	0	1	0	0	1	X	0	0	X	0	X	1	X
1	0	0	1	1	0	1	0	X	0	0	X	1	X	X	1
1	0	1	0	1	0	1	1	X	0	0	X	X	0	1	X
1	0	1	1	1	1	0	0	X	0	1	X	X	1	X	1
1	1	0	0	1	1	0	1	X	0	X	0	0	X	1	X
1	1	0	1	1	1	1	0	X	0	X	0	1	X	X	1
1	1	1	0	0	0	0	0	X	0	X	0	X	0	1	X
1	1	1	1	1	1	1	0	X	1	X	1	X	1	X	1



Como hay 4 cifras necesitamos 4 biestables

- Tabla de excitaciones de un biestable JK:

Q_t	Q_{t+1}	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

- Implementación de J_i, K_i:

J_0, K_0 → $J_0 = K_0 = 1$

J_1, K_1 → $J_1 = K_1 = Q_0$

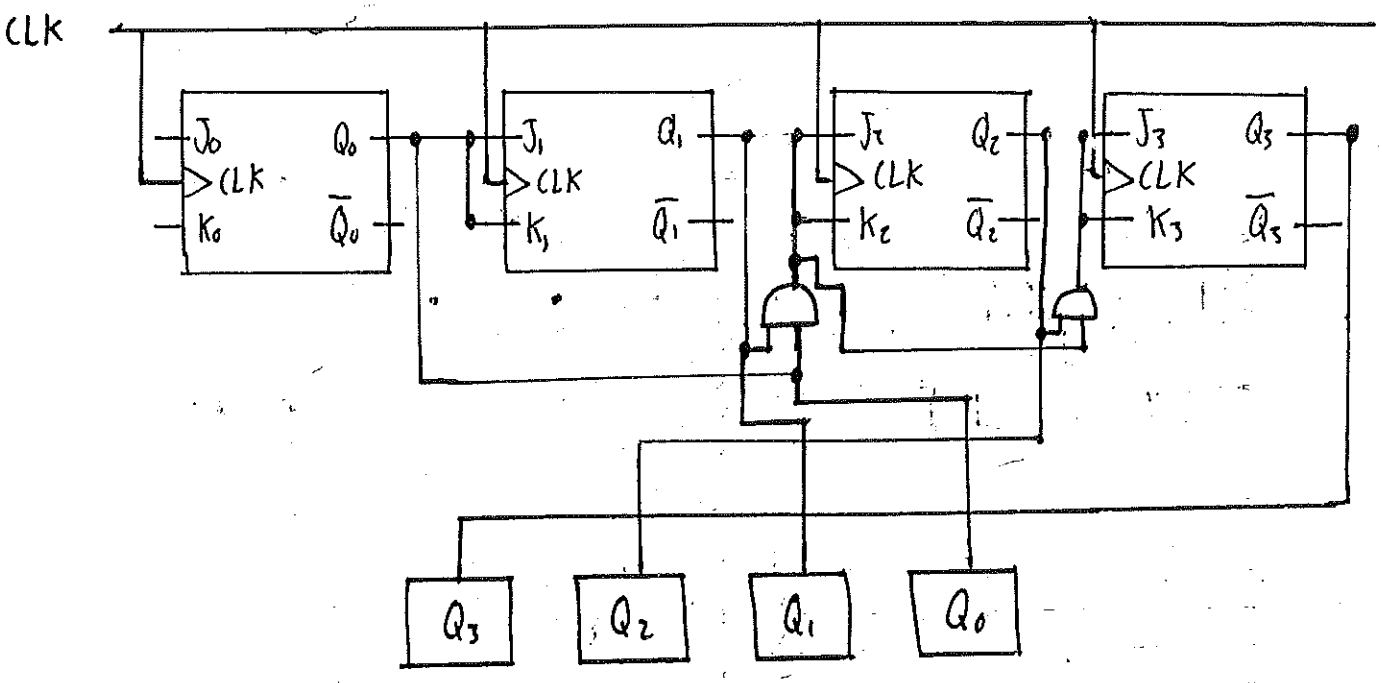
J_2, K_2

Q_3Q_2 \ Q_1Q_0	00	01	11	10
00	0	0	1	0
01	X	X	X	X
11	X	X	X	X
10	0	0	1	0

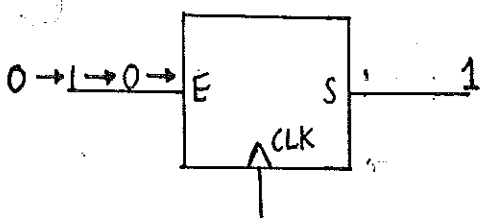
$J_2 = Q_1Q_0$

$K_2 = J_2 = Q_1Q_0$

J_3 K_3 .... $J_3 = Q_2Q_1Q_0$ $K_3 = Q_2Q_1Q_0$

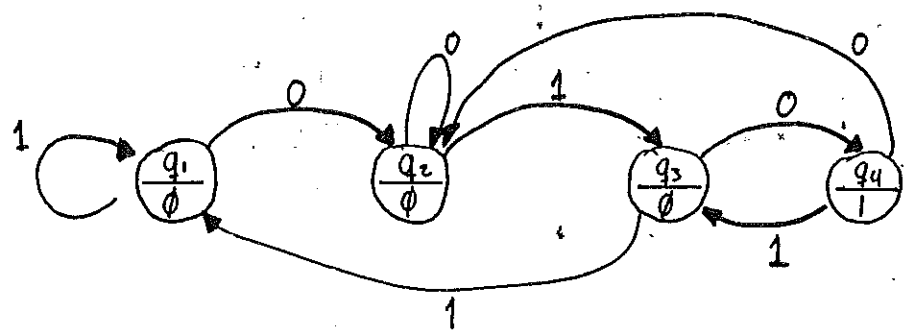

CLK
J0
CLK
K0
Q0
Q̄0
J1
CLK
K1
Q1
Q̄1
J2
CLK
K2
Q2
Q̄2
J3
CLK
K3
Q3
Q̄3
Q3
Q2
Q1
Q0

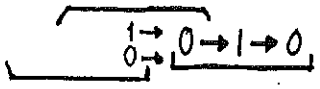
Ejercicio 2 | Diseñar un circuito secuencial con una entrada y una salida que reconozca la cadena "010".



Estados:

$q_1 \equiv$ "El último bit recibido ha sido un 1"
$q_2 \equiv$ "El último bit recibido ha sido un 0"
$q_3 \equiv$ "Los dos últimos bits recibidos han sido ϕ y 1"
$q_4 \equiv$ "Los tres últimos bits recibidos han sido ϕ, 1, ϕ"





- Como tenemos 4 estados necesitamos: dos dígitos binarios para numerarlos $\begin{cases} q_1 \equiv \phi\phi \\ q_2 \equiv \phi 1 \\ q_3 \equiv 1\phi \\ q_4 \equiv 11 \end{cases}$

- Rellenamos la tabla de transiciones y excitaciones:

E	estado actual Q_1	estado actual Q_0	estado siguiente Q_1^*	estado siguiente Q_0^*	S	D1	Dϕ
0	0	0	0	1	0	0	1
0	0	1	0	1	0	0	1
0	1	0	1	0	0	1	1
0	1	1	0	0	1	0	1
1	0	0	0	0	0	0	0
1	0	1	1	0	0	1	0
1	1	0	0	0	0	0	0
1	1	1	1	0	1	1	0

Variables

Tabla de transiciones

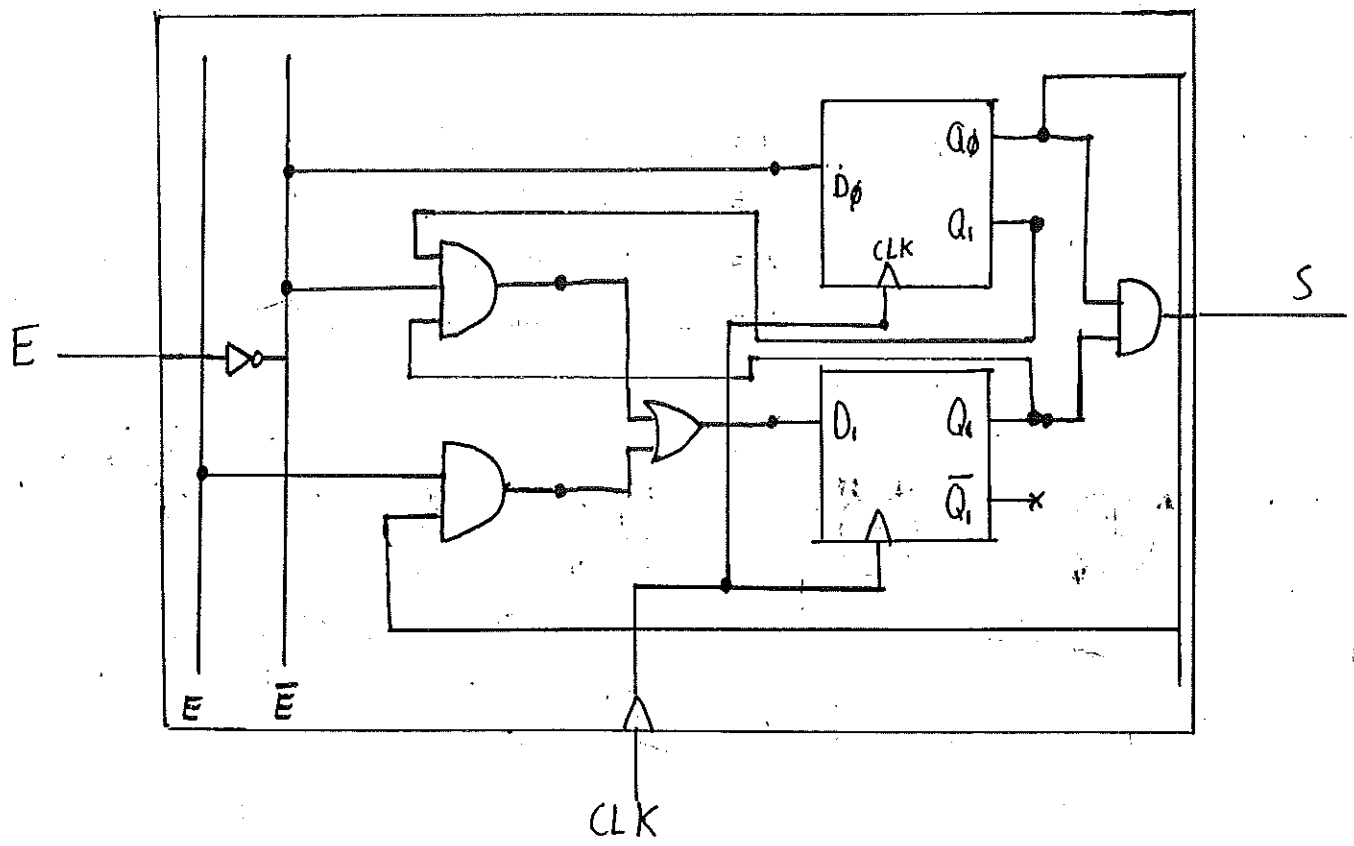
Es una maquina de Moore, donde la salida solo depende del estado actual.

Podemos obtener una tabla de $S = f(Q_1, Q_0)$

Q_1	Q_0	S
0	0	0
0	1	0
1	0	0
1	1	1

Utilizamos un biestable por cada dígito de estado. Vamos a usar biestables D ($Q_{t+1} = D$)

- Simplificamos todas las funciones ($S(Q_1, Q_0)$, $D_1(E, Q_1, Q_0)$, $D_0(E, Q_1, Q_0)$).
  - $S = Q_1$ AND $Q_0 = Q_1 \cdot Q_0$
  - $D_1 = E Q_0 + \bar{E} Q_1 \bar{Q}_0$ (Simplificando por Karnaugh)
  - $D_0 = \bar{E}$


E
S
D_ϕ
Q_ϕ
Q_1
CLK
D_1
Q_1
$\overline{Q_1}$
E
$\overline{E}$
CLK

EDIG
Problemas de examen

SEPTIEMBRE 2003

2. Se pretende diseñar un sistema de llenado de un depósito de agua, cuyo esquema es el de la *Figura 3,* y donde:

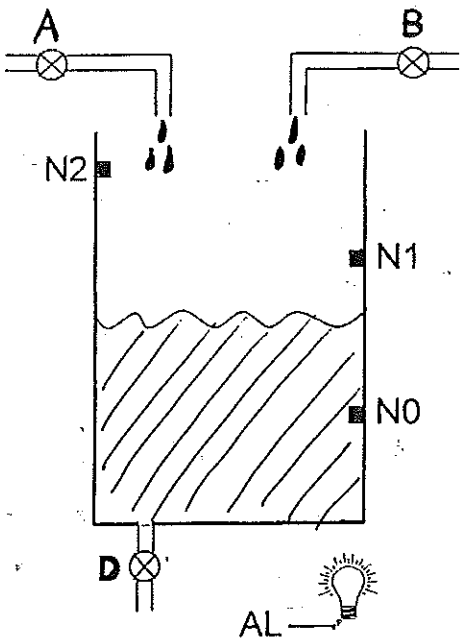


Figura 3

- **A** y **B** son dos señales, activas a nivel alto, que controlan dos electroválvulas que permiten el llenado del depósito, abriendo las entradas de agua correspondientes.
- **D** es una señal que controla una electroválvula que permite el vaciado del depósito, a través del desagüe correspondiente.
- **N0, N1, N2** son detectores de nivel de agua que se activan generando un nivel alto de tensión "1" al entrar en contacto con el agua.
- **AL** es una señal de alarma que se activa en ciertos casos (ver más abajo).

El funcionamiento del circuito a diseñar es el siguiente:

- Si el agua desciende por debajo de **N0**, se deben activar las dos electroválvulas simultáneamente **A** y **B**, permitiendo el llenado del tanque a través de las dos entradas.
- Si el agua alcanza **N0**, solo se activará **A**, desactivándose **B**.
- Si el agua alcanza **N0** y **N1**, se desactivarán las dos señales **A** y **B**.
- Si el agua alcanza **N0, N1** y **N2**, se desactivarán **A** y **B** y se activará **D**.
- Cualquier otra situación anómala de **N[0,1,2]** activará la alarma y provocará la parada la parada de **A** y **B**, así como la desactivación de **D**.

NOTA: Se considera una situación anómala cualquiera de los casos no contemplados anteriormente, por ejemplo, el que se activen **N2** y **N1** y no active **N0**.

2.1 Realice el mapa de Karnaugh para las salidas **D** y **AL** respecto a las entradas **N0, N1** y **N2** e implemente el circuito completo de ambas señales tras la simplificación. (*10 puntos*)

2.2 Implemente el circuito completo correspondiente a las salidas **A** y **B** utilizando exclusivamente los siguientes multiplexores: (*15 puntos*)

- Señal **A** – **Un** multiplexor de **cuatro** entradas
- Señal **B** – **Tres** multiplexores de **dos** entradas

2.1) Implementación de D, AL

■ Tabla de verdad:

Sólo se activa el desagüe cuando N2 = N1 = NØ = 1

variables / funciones

N2	N1	NØ	D	AL
0	0	0	0	0
0	0	1	0	0
0	1	0	0	1
0	1	1	0	0
1	0	0	0	1
1	0	1	0	1
1	1	0	0	1
1	1	1	1	0

■ Simplificación por Karnaugh:

(D)

N2 \ N1Nø	00	01	11	10
0	0	0	0	0
1	0	0	(1)	0

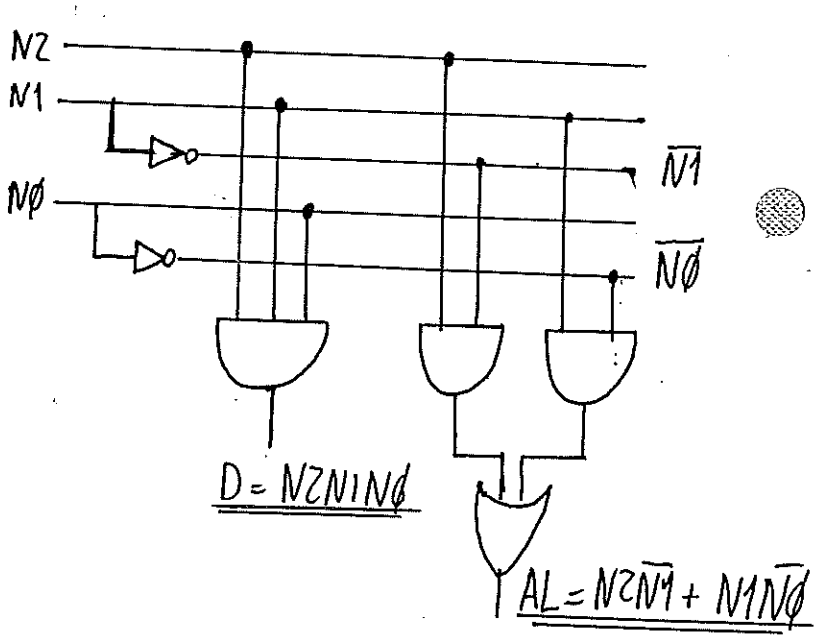
$D = N2N1N\emptyset$

(AL)

N2 \ N1Nø	00	01	11	10
0	0	0	0	(1)
1	(1	1)	0	(1)

$AL = \bar{N}1N2 + N1\bar{N}\emptyset$

■ Implementación con puertas lógicas:

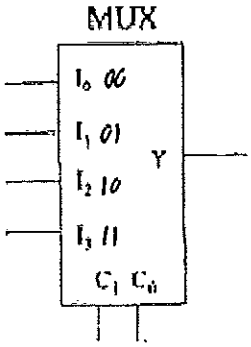


FEBRERO 2002

2. La función *mayoría*, MAY(x_0, x_1, ..., x_{n-1}), recibe un número impar de bits x_0, x_1, ..., x_{n-1} e indica si existe una mayoría de unos o no. Por ejemplo, dos casos concretos de función mayoría de 3 bits serían:

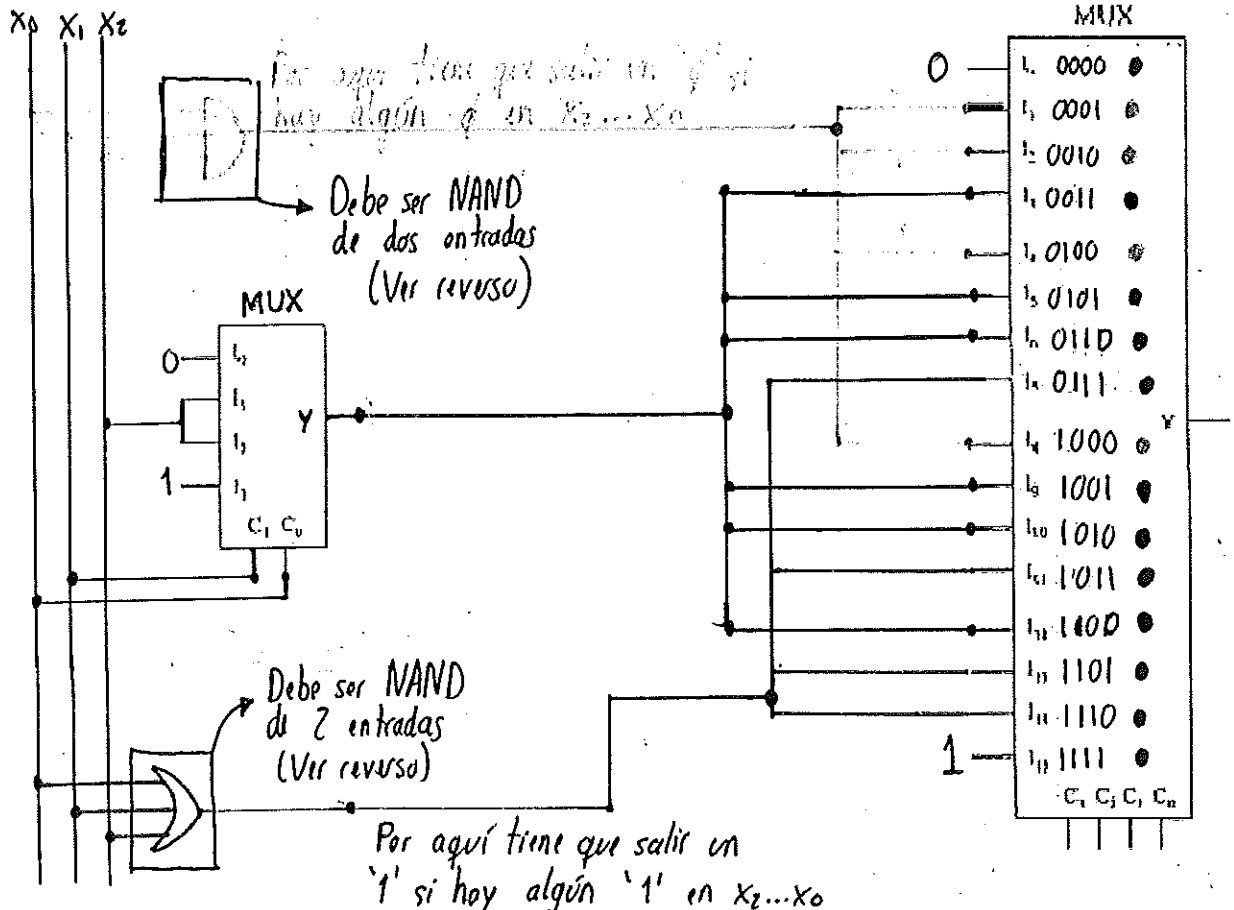
- MAY(1,1,0) = 1 (pues hay más unos que ceros)
- MAY(1,0,0) = 0 (pues no hay más unos que ceros)

2.1. Construya una función mayoría de tres bits, x_0, x_1 y x_2 utilizando únicamente el multiplexor de 4 entradas de datos de la figura.



2.2. Complete la siguiente figura para construir una función mayoría de 7 bits, x_0, ..., x_6, utilizando:

- El diseño del apartado 2.1. para generar una función *mayoría* de 3 bits por medio de un multiplexor de 4 entradas de datos. **NOTA:** En caso de no haber resuelto el apartado 2.1, suponga que dispone de una *caja negra* que implemente la función *mayoría* de 3 bits y utilícela donde lo precise.
- Un multiplexor de 16 entradas de datos.
- Todas las puertas, NAND de 2 entradas que necesite. Se valorará utilizar el menor número de puertas NAND posible.



2.3. Se desea implementar una función *mayoría* de 3 bits empleando esta vez una memoria ROM. Dimensione el tamaño mínimo que habrían de tener los buses de dirección y de datos e indique el contenido de cada una de las palabras de la memoria. Justifique su respuesta.

2.1) ■ Tabla de verdad:

X_2	X_1	X_0	MAY
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

■ Simplificación por unos:

X_2 \ X_1X_0	00	01	11	10
0	0	0	1	0
1	0	1	1	1

$$MAY_3 = X_1X_0 + X_2X_1 + X_2X_0$$

■ Implementación con MUX 4x2:

Elegimos dos variables como entradas de control $\begin{cases} X_1 \equiv C_1 \\ X_0 \equiv C_0 \end{cases}$

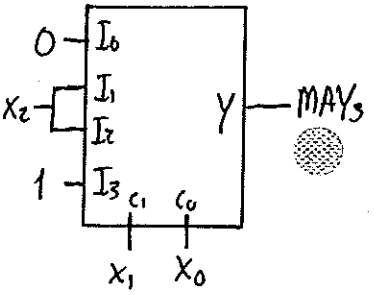
$$MAY_3 = X_1X_0 + X_2X_0(X_1+\bar{X}_1) + X_1X_2(X_0+\bar{X}_0) = X_1X_0 + X_1X_0X_2 + \bar{X}_1X_0X_2 + X_1X_2X_0 + X_1\bar{X}_0X_2 =$$

$$= X_1X_0(1 + X_2) + \bar{X}_1X_0X_2 + X_1\bar{X}_0X_2 \quad (1 + X_2 = 1)$$

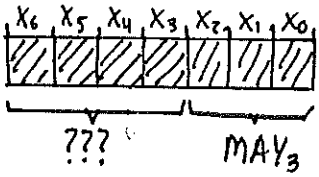
$$Y = \bar{C}_1\bar{C}_0\boxed{I_0} + \bar{C}_1C_0\boxed{I_1} + C_1\bar{C}_0\boxed{I_2} + C_1C_0\boxed{I_3}$$

$$MAY_3 = \bar{X}_1\bar{X}_0\boxed{0} + \bar{X}_1X_0\boxed{X_2} + X_1\bar{X}_0\boxed{X_2} + X_1X_0\boxed{1}$$

$I_0 \equiv 0$
$I_1 \equiv X_2$
$I_2 \equiv X_2$
$I_3 = 1$



2.2)



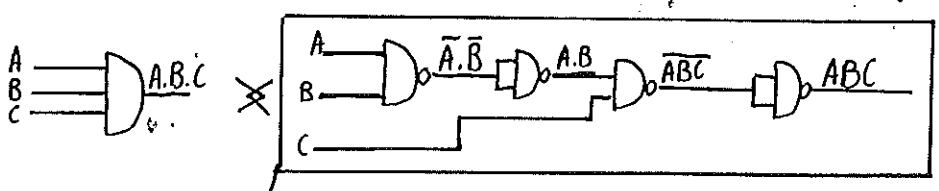
Vamos a estudiar el comportamiento del cto. en función de $X_6 \ldots X_3$:

- Cuando $X_6 \ldots X_3 = 1111$ directamente el resultado es '1' ■
- Cuando $X_6 \ldots X_3 = 0000$ directamente el resultado es '0' ■
- Cuando tengamos tres '1s' y un 'ϕ' basta con que haya un '1' más en $X_2 \ldots X_0$ para que el resultado sea '1' ■

[Ver dibujo Atrás]

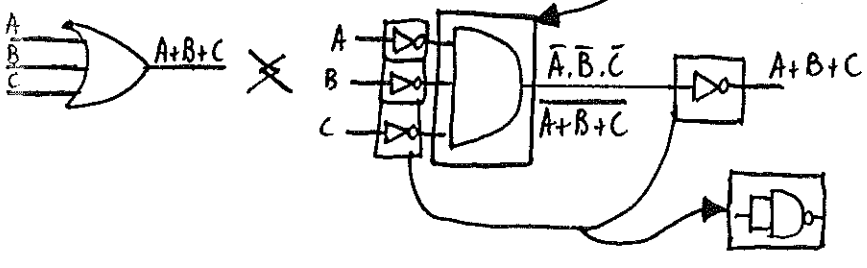
- Cuando tengamos tres 'ϕs' y un '1' basta con que haya un '0' más en $X_2 \ldots X_0$ para que el resultado sea 'ϕ' ■
- Cuando haya empate de 'ϕs' y '1s' en $X_6 \ldots X_3$, decidirá el resultado la función MAY_3 ■

- AND_3 mediante $NAND_2$:



- OR_3 mediante $NAND_2$:

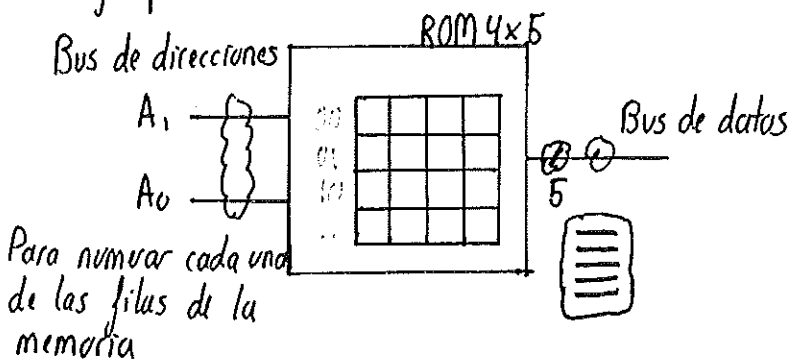
paso previo: OR_3 mediante AND y NOT



2.3) Implementar MAY_3 mediante una memoria ROM

NOTA: páginas 3.9, 3.10 : memoria ROM

Ejemplo ROM 4x5

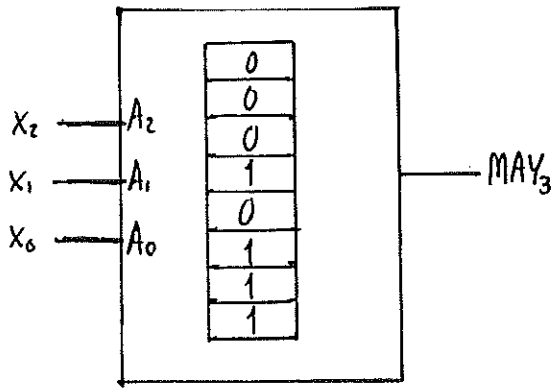


Necesito 5 bits para entregar la palabra completa que hay en la posición A_1, A_0

X_2	X_1	X_0	MAY_3
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Para implementar esta función necesitamos:

- 8 palabras: bus de direcciones de 3 bits
- 1 bit de bus de datos: cada fila de la tabla está formada por un sólo valor de la función.



SEPTIEMBRE 2002

1. El circuito de la figura 1 utiliza un multiplexor 74x151 (tabla de verdad en la figura 1) y un inversor para realizar la función lógica combinacional F de cuatro variables (A, B, C y D).

Inputs				*Outputs*	
EN_L	C	B	A	Y	Y_L
1	x	x	x	0	1
0	0	0	0	D0	D0'
0	0	0	1	D1	D1'
0	0	1	0	D2	D2'
0	0	1	1	D3	D3'
0	1	0	0	D4	D4'
0	1	0	1	D5	D5'
0	1	1	0	D6	D6'
0	1	1	1	D7	D7'

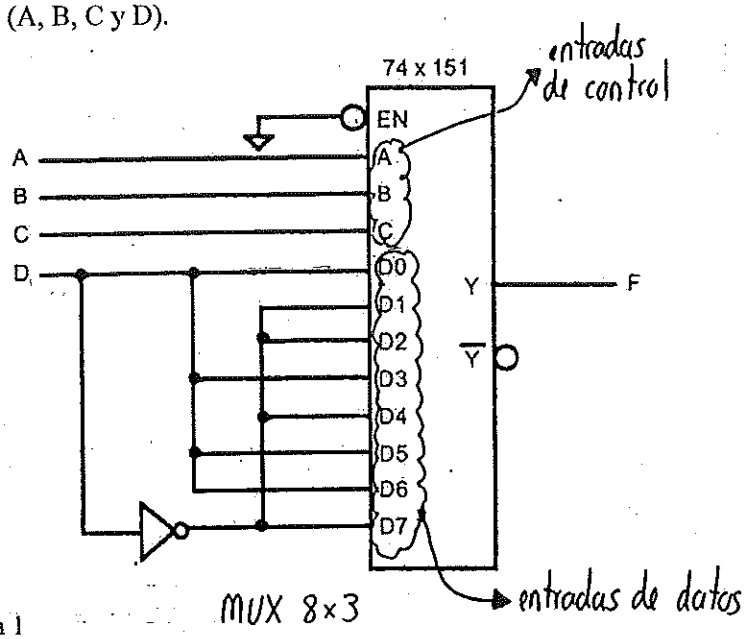


Figura 1

1.1. Función lógica
Rellene el mapa de Karnaugh de dicha función lógica F. No olvide etiquetar convenientemente las filas y columnas de la tabla (valores de las variables de entrada.

1.2. Implementación
Implemente la función F utilizando única y exclusivamente tres puertas lógicas y justifique la solución que ha empleado.

1.1) $Y = \bar{C}\bar{B}\bar{A}\cdot D_0 + \bar{C}\bar{B}A D_1 + \bar{C}B\bar{A}D_2 + \bar{C}BAD_3 + C\bar{B}\bar{A}D_4 + C\bar{B}AD_5 + CB\bar{A}D_6 + CBAD_7$

$F = \bar{C}\bar{B}\bar{A}.D + \bar{C}\bar{B}A\bar{D} + \bar{C}B\bar{A}\bar{D} + \bar{C}BAD + C\bar{B}\bar{A}\bar{D} + C\bar{B}AD + CB\bar{A}D + CBA\bar{D}$ | Ver ejercicio 4 Tema 1

Mapa de Karnaugh:

AD \ CB	00	01	11	10
00	0	1	0	1
01	1	0	1	0
11	0	1	0	1
10	1	0	1	0

Cuando tengamos un mapa de Karnaugh como este ("tablero de ajedrez") se implementará se hace usando puertas XOR y NOT.

Apendice 1

XOR

X	Y	X⊕Y
0	0	0
0	1	1
1	0	1
1	1	0

$X \oplus Y = X\bar{Y} + \bar{X}Y$

XNOR

X	Y	$\overline{X\oplus Y}$
0	0	1
0	1	0
1	0	0
1	1	1

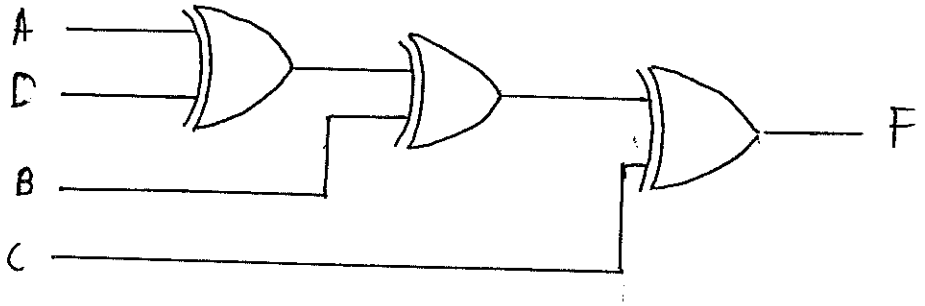
$\overline{X \oplus Y} = \bar{X}\bar{Y} + XY$

$F = \bar{C}(\bar{B}\bar{A}D + \bar{B}A\bar{D} + B\bar{A}\bar{D} + BAD) + C(\bar{B}\bar{A}\bar{D} + \bar{B}AD + B\bar{A}D + BA\bar{D}) =$

$= \bar{C}\left[\bar{B}\overbrace{(\bar{A}D + A\bar{D})}^{A\oplus D} + B\overbrace{(\bar{A}\bar{D} + AD)}^{\overline{A\oplus D}}\right] + C\left[\bar{B}\overbrace{(\bar{A}\bar{D} + AD)}^{\overline{A\oplus D}} + B\overbrace{(\bar{A}D + A\bar{D})}^{A\oplus D}\right] =$

$= \bar{C}\left[\overline{\bar{B}(A\oplus D) + B(\overline{A\oplus D})}\right] + C\left[\overline{\bar{B}(\overline{A\oplus D}) + B(A\oplus D)}\right] =$

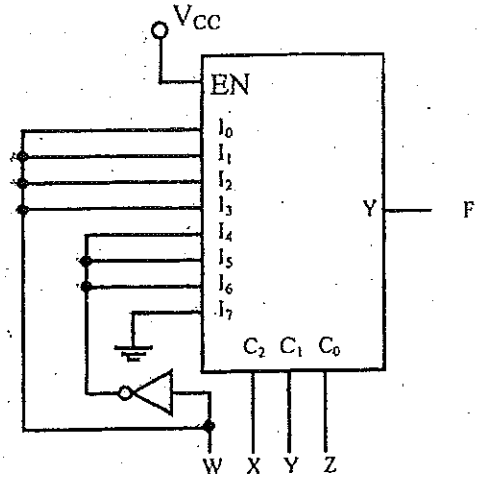
$= \left[\bar{C}\left[B\oplus(A\oplus D)\right] + C\left[\overline{B\oplus(A\oplus D)}\right]\right] = C\oplus\left[B\oplus(A\oplus D)\right]$


A
D
B
C
F

JUNIO 2003

MUX (análisis)

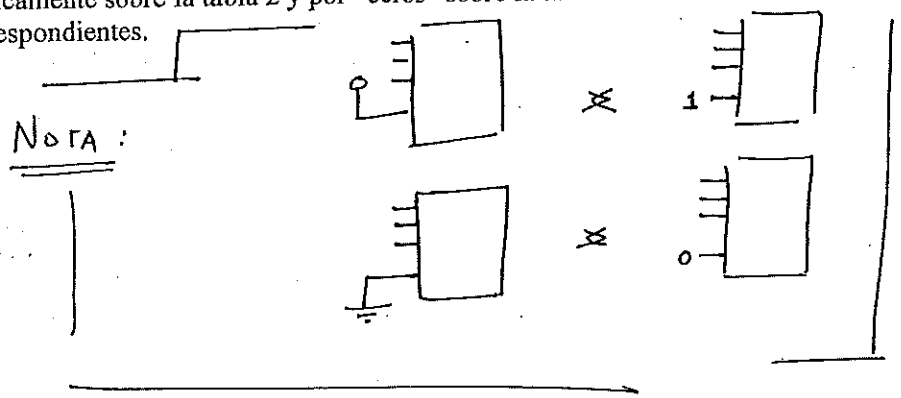
3. El circuito del multiplexor de la figura 4, cuya tabla de verdad encontrará junto a la figura 4, implementa una función lógica F de cuatro variables:



EN	C2	C1	C0	Y
0	X	X	X	0
1	0	0	0	I0
1	0	0	1	I1
1	0	1	0	I2
1	0	1	1	I3
1	1	0	0	I4
1	1	0	1	I5
1	1	1	0	I6
1	1	1	1	I7

Figura 4

3.1. Escriba la expresión lógica de la función F.
3.2. Obtenga la expresión simplificada de F usando el método de Karnaugh. Realice la simplificación por "unos" gráficamente sobre la tabla 2 y por "ceros" sobre la tabla 3. Escriba a continuación las expresiones correspondientes.



3.1

Salida de un multiplexor 8x3

$$Y = \bar{C}_2\bar{C}_1\bar{C}_0 I_0 + \bar{C}_2\bar{C}_1 C_0 I_1 + \bar{C}_2 C_1\bar{C}_0 I_2 + \bar{C}_2 C_1 C_0 I_3 + C_2\bar{C}_1\bar{C}_0 I_4 + C_2\bar{C}_1 C_0 I_5 + C_2 C_1\bar{C}_0 I_6 + C_2 C_1 C_0 \cdot$$

$$F = \bar{X}\bar{Y}\bar{Z}W + \bar{X}\bar{Y}Z\,W + \bar{X}Y\bar{Z}W + \bar{X}YZW + X\bar{Y}\bar{Z}\bar{W} + X\bar{Y}Z\bar{W} + XY\bar{Z}\bar{W} + XYZ \cdot$$

Identificamos → Lo que hemos obtenido son minterms !!

3.2

Tengo que pasarlos a una tabla de verdad, de la función $F(W, X, Y, Z)$. Igual lo vemos mejor así:

$$F = W\bar{X}\bar{Y}\bar{Z} + W\bar{X}\bar{Y}Z + W\bar{X}Y\bar{Z} + W\bar{X}YZ + \bar{W}X\bar{Y}\bar{Z} + \bar{W}X\bar{Y}Z + \bar{W}XY\bar{Z} =$$

$$= W\left(\bar{X}\bar{Y}\bar{Z} + \bar{X}\bar{Y}Z + \bar{X}Y\bar{Z} + \bar{X}YZ\right) + \bar{W}\left(X\bar{Y}\bar{Z} + X\bar{Y}Z + XY\bar{Z}\right)$$

w	x	y	z	F
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

Simplificamos por Karnaugh (por '1's)

yz \ wx	00	01	11	10
00	0	1	0	1
01	0	1	0	1
11	0	0	0	1
10	0	1	0	1

$F = \bar{w}x\bar{y} + \bar{w}x\bar{z} + w\bar{x}$

Simplificamos por Karnaugh (por '0's)

yz \ wx	00	01	11	10
00	0	1	0	1
01	0	1	0	1
11	0	0	0	1
10	0	1	0	1

$F = (w + x') \cdot (\bar{w} + \bar{x}) \cdot (w + \bar{y} + \bar{z})$

PROBLEMA 2 (20 *PUNTOS*)

La apertura y cierre de una puerta deslizante vertical de un almacén (*Figura 4*) se controla mediante 2 conmutadores: uno situado en el interior del almacén (y que determina el valor de la señal lógica A) y el otro en el exterior (y que determina el valor de la señal lógica B). Se considera que inicialmente (en reposo) la puerta está cerrada y ambos conmutadores en la posición 0 (ie. A=0, B=0). La inversión de uno de los conmutadores provoca la apertura de la puerta y otra inversión de cualquiera de ellos el cierre. Además se requiere un sistema de seguridad, de manera que la puerta no se cierre cuando exista un objeto debajo de ella. La presencia de objetos se detecta mediante una célula fotoeléctrica que genera una señal C, de manera que cuando NO hay ningún objeto entre la puerta y el suelo dicha señal vale 0, valiendo 1 cuando existe algún objeto..

El movimiento de ascenso o descenso de la puerta se realiza mediante un cilindro neumático según el valor de una señal lógica de control M, de manera que cuando M vale 0 provoca el descenso de la puerta y cuando vale 1 provoca el ascenso de la misma.

Se trata de diseñar el circuito lógico que regula el valor de M.

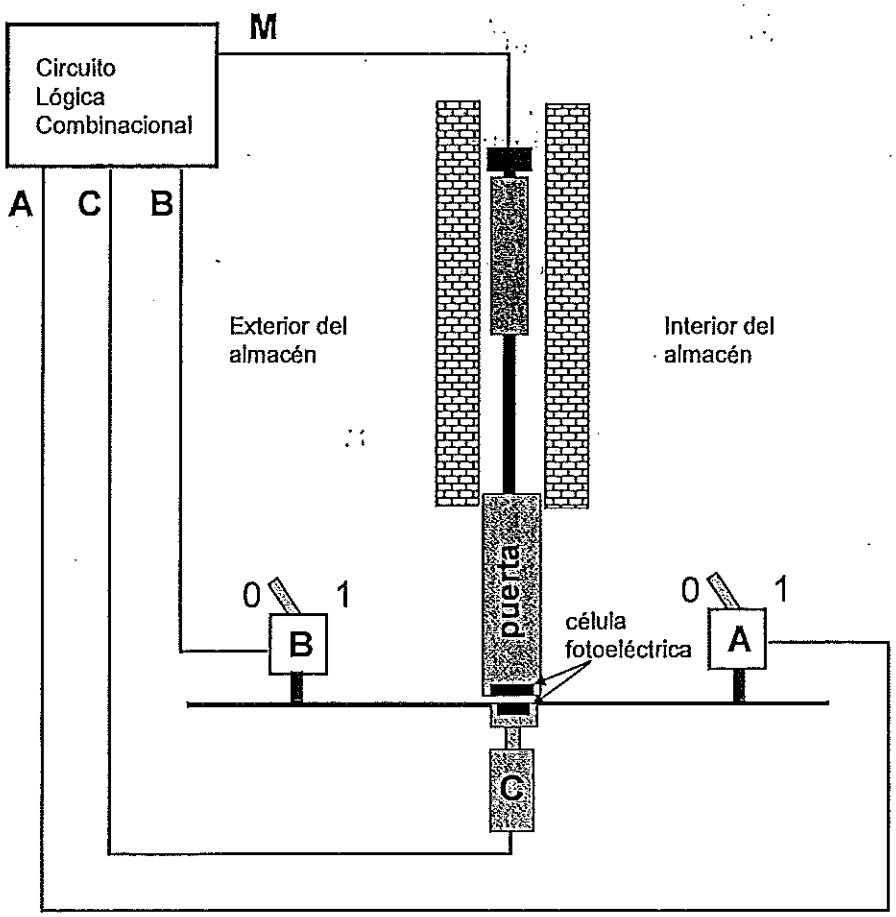


Figura 4

2.2

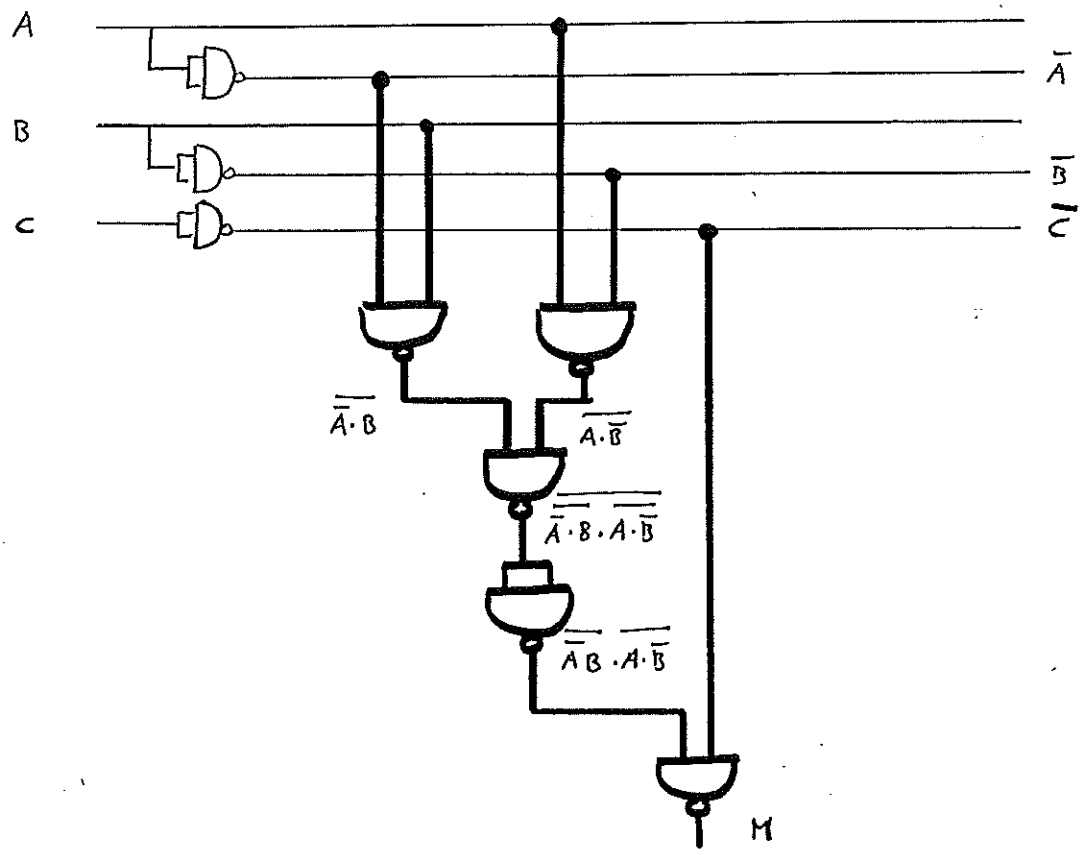
M

C \ AB	00	01	11	10
0	0	1	0	1
1	1	1	1	1

$$M = C + \bar{A}B + A\bar{B} = C + (A \oplus B)$$

2.3

$$M = \overline{\overline{C + \bar{A}B + A\bar{B}}} = \overline{\bar{C} \cdot \overline{\bar{A}B} \cdot \overline{A\bar{B}}}$$



2.4

Tenemos 3 variables, y 2 entradas de control. Vamos a elegir A y B como variables de control, y a hacer, por tanto, que aparezcan en los tres sumandos:

$$M = C \cdot (A + \bar{A}) \cdot (B + \bar{B}) + \bar{A}B + A\bar{B} = (CA + C\bar{A}) \cdot (B + \bar{B}) + \bar{A}B + A\bar{B} =$$

$$= ABC + A\bar{B}C + \bar{A}BC + \bar{A}\bar{B}C + \bar{A}B + A\bar{B} = \bar{A}\bar{B}C + \bar{A}B(C+1) + A\bar{B}(C+1) + ABC =$$

$$= \bar{A}\bar{B}C + \bar{A}B + A\bar{B} + ABC$$

Comparando con $Y = \bar{S}_1\bar{S}_0 I_0 + \bar{S}_1 S_0 I_1 + S_1\bar{S}_0 I_2 + S_1 S_0 I_3$

Y sabiendo que hemos elegido $A \equiv S_1$, $B \equiv S_0$

Tenemos: $I_0 \equiv C$, $I_1 \equiv 1$, $I_2 \equiv 1$, $I_3 \equiv C$

Hemos elegido C como entrada de datos porque no está negada.

2.1 Obtenga la Tabla de verdad de la señal M. *(5 puntos)*

variables

A	B	C	M	Z
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	0
1	0	0	1	1
1	0	1	1	1
1	1	0	0	1
1	1	1	1	1

Cuando C = 1 ⇒ M = 1 (con objetos en la célula, puerta abierta)

Con todo desactivado, y nada en la célula, M = 0

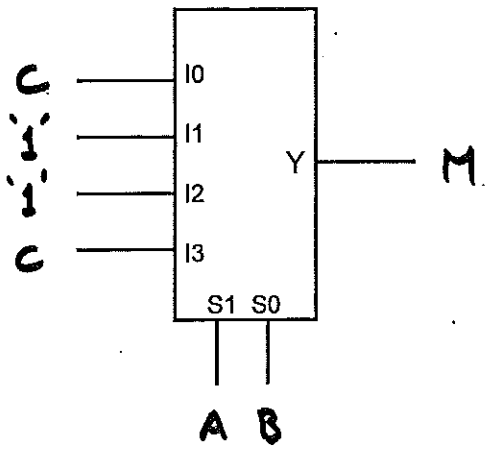
Si uno de al pulsador y el otro no, abre, M = 1

" " " " " " " " tb, cierra, M = 0

2.2 Utilice un mapa de Karnaugh para obtener la función lógica simplificada de M. En caso de no haber respondido el primer apartado, simplifique la función Z. *(5 puntos)*

2.3 Implemente la función M utilizando exclusivamente puertas NAND de dos entradas. En caso de no haber respondido el primer apartado, implemente la función Z. *(5 puntos)*

2.4 Implemente la función M utilizando un multiplexor de 4 entradas de datos. En caso de no haber respondido el primer apartado, implemente la función Z. Justifique su respuesta. *(5 puntos)*



NOVIEMBRE 2011

PROBLEMA 1 (40 PUNTOS)

Se desea implementar las funciones lógicas K, F y H, que dependen de X, Y y Z, según la siguiente Tabla de Verdad:

X	Y	Z	K	F	H
0	0	0	0	1	1
0	0	1	1	1	0
0	1	0	1	0	1
0	1	1	1	1	0
1	0	0	1	0	0
1	0	1	1	0	1
1	1	0	0	1	0
1	1	1	0	0	1

1.1 Obtenga la expresión lógica simplificada de K utilizando un mapa de Karnaugh, e impleméntela con el menor número de puertas lógicas posibles de 2 entradas. *(10 puntos)*

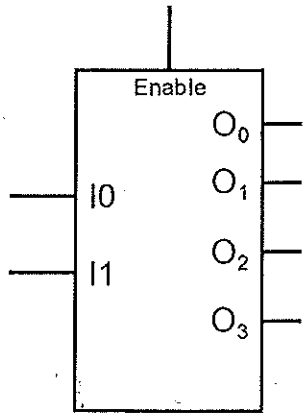
Simplificando por UNOS Simplificando por CEROS

1.2. Implemente la función K utilizando exclusivamente un multiplexor de 4 entradas de datos, como el de la Figura. Para ello, **NO se dispone** de las señales negadas. **Justifique** adecuadamente su solución. *(15 puntos)*

1.3 Finalmente, implemente las dos funciones, F y H, utilizando **exclusivamente** un decodificador 2 a 4 como el de la Figura (con enable y salidas activas a nivel alto) y las siguientes puertas lógicas:

- 3 puertas OR de 2 entradas.
- 2 puertas XOR de 2 entradas.

Importante Note que los valores de F y H para una combinación dada de Y y Z tiene valores complementarios dependiendo del valor de X. *(15 puntos)*



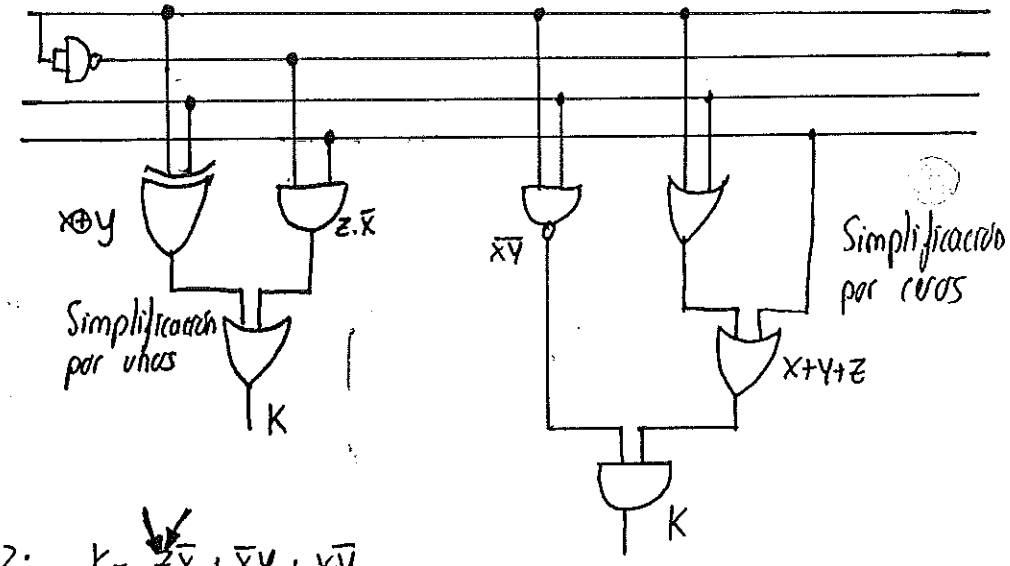
1.1) Karnaugh por 1's y por 0's

Z \ XY	00	01	11	10
0	0	1	0	1
1	1	1	0	1

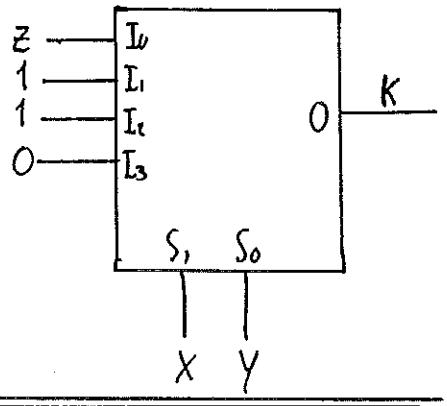
$K = \bar{Z}\bar{X} + \bar{X}Y + X\bar{Y} = Z.\bar{X} + (X \oplus Y)$ ⟹ Simplificando por unos

$K = (\bar{X} + \bar{Y}).(X + Y + Z) = \overline{(XY)}.(X + Y + Z)$ ⟹ Simplificado por ceros

Implementación de la función:



1.2) Implementación con MUX 4x2: $K = z\bar{x} + \bar{x}y + x\bar{y}$



Como no podemos usar negadores vamos a elegir la variable z para conectarla a las entradas de datos, es decir, x e y a las entradas de control.

$K = z\bar{x}(y+\bar{y}) + \bar{x}y + x\bar{y} = \bar{x}yz + \bar{x}\bar{y}z + \bar{x}y + x\bar{y} =$

$\bar{x}y(\underbrace{z+1}_{=1}) + \bar{x}\bar{y}z + x\bar{y}$

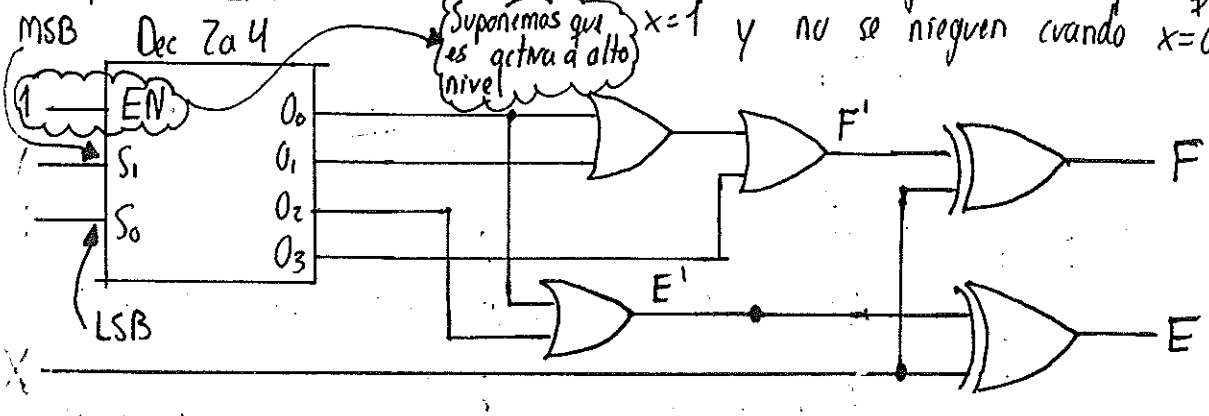
$O = \bar{S}_1\bar{S}_0 I_0 + \bar{S}_1 S_0 I_1 + S_1\bar{S}_0 I_2 + S_1 S_0 I_3$

$K = \bar{x}\bar{y}\boxed{z} + \bar{x}y\boxed{1} + x\bar{y}\boxed{1} + xy\boxed{0}$

1.3) NOTA: Las puertas XOR se pueden usar para negar selectivamente una función dependiendo del valor de otra variable:

X	T	X⊕T
0	0	0
0	1	1
1	0	1
1	1	0

Vamos a implementar las funciones F y E para x=0 (dependen solo de Y, Z) mediante un codificador 2a4 y puertas OR. Después completamos las funciones conectándolas a puertas XOR junto con X para que se nieguen cuando x=1 y no se nieguen cuando x=0.



FEBRERO 2005

Problema 2

Se desea diseñar un circuito de sumar y restar dos números de un bit **A0** y **B0** con acarreo de entrada C_{i-1} mediante una señal de control **M**. El resultado se compone de tres bits **[S2,S0]** expresado en complemento a dos. La operación, en función de **M** es la siguiente:

M = "0" **[S2,S0] = A0 + B0 + C_{i-1}**

M = "1" **[S2,S0] = A0 – (B0 + C_{i-1})**

2.1 Rellene la tabla de verdad, *Tabla 2*, y obtenga **justificadamente** la función lógica simplificada para **S0**.

Pasamos a:

M	A0	B0	C_{i-1}	S2	S1	S0	AUX	Decimal
0	0	0	0	0	0	0	1	0
0	0	0	1	0	0	1	0	1
0	0	1	0	0	0	1	1	1
0	0	1	1	0	1	0	0	2
0	1	0	0	0	0	1	1	1
0	1	0	1	0	1	0	0	2
0	1	1	0	0	1	0	1	2
0	1	1	1	0	1	1	0	3
1	0	0	0	0	0	0	0	0
1	0	0	1	1	1	1	0	-1
1	0	1	0	1	1	1	0	-1
1	0	1	1	1	1	0	0	-2
1	1	0	0	1	1	0	1	1
1	1	0	1	1	0	1	0	0
1	1	1	0	1	0	0	1	0
1	1	1	1	1	0	1	0	-1

A0+B0+C_{i-1}

A0 – (B0+C_{i-1})

-1 a complemento a 2:
1 → 001 → 110 → 111

-2 a complemento a 2
10 → 010 → 101 → 110

2.2 Implemente la función **AUX** de la *Tabla 2* con el mínimo número de puertas **NOR** de dos entradas. Nota: Sólo se dispone de las entradas **M, A0, B0** y C_{i-1} y no de sus complementarias.

2.3 Implemente la función $F = \overline{M} \cdot \overline{C} + A \cdot \overline{C} + A \cdot B \cdot \overline{M}$ utilizando únicamente dos multiplexores de 4 entradas de datos. Nota: Sólo se dispone de las entradas **M, A, B** y **C** y no de sus complementarias.

2.4 Implemente la función $F = \overline{M} \cdot \overline{C} + A \cdot \overline{C} + A \cdot B \cdot \overline{M}$ con un único decodificador y una única puerta lógica (ambos a su elección). Nota: Sólo se dispone de las entradas **M, A, B** y **C** y no de sus complementarias.

FEB 2005 - 2

En clase sólo resuelvo la tabla de verdad.

2.1 FUNCIÓN LÓGICA SIMPLIFICADA PARA $S\phi$

$B\phi C_{i-1}$ \ $M A\phi$	00	01	11	10
00	0	1	1	0
01	1	0	0	1
11	0	1	1	0
10	1	0	0	1

$$S\phi = A\phi\,\overline{B\phi}\,\overline{C_{i-1}} + \overline{A\phi}\,\overline{B\phi}\,C_{i-1} + A\phi B\phi\,C_{i-1} + \overline{A\phi}\,B\phi\,\overline{C_{i-1}}$$

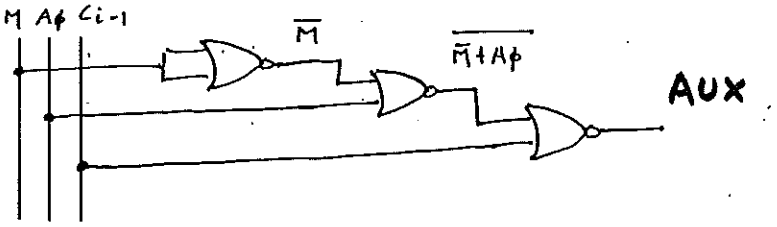
2.2 IMPLEMENTACIÓN DE AUX MEDIANTE NOR_2

Primero vamos a simplificar dicha función mediante el método de Karnaugh:

$B\phi C_{i-1}$ \ $M A\phi$	00	01	11	10
00	1	1	1	0
01	0	0	0	0
11	0	0	0	0
10	1	1	1	0

$$AUX = \overline{M}\cdot\overline{C_{i-1}} + A\phi\cdot\overline{C_{i-1}} = \overline{C_{i-1}}\,(\overline{M} + A\phi) =$$

$$= \overline{\overline{C_{i-1}}\,(\overline{M} + A\phi)} = \overline{C_{i-1} + \overline{\overline{M} + A\phi}}$$



2.3 IMPLEMENTACIÓN DE F MEDIANTE DOS MUX 4X2 (¡OJO!! No tenemos las entradas negadas)

vamos a elegir M y C como entradas de control de un MUX

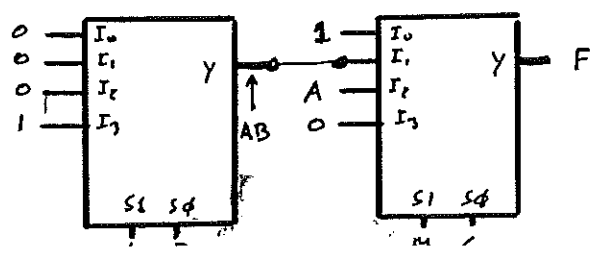
$$F = \overline{M}\,\overline{C} + A\overline{C} + AB\overline{M} = \overline{M}\,\overline{C} + A\overline{C}\,(M+\overline{M}) + AB\overline{M}\,(C+\overline{C}) =$$

$$= \overline{M}\,\overline{C} + M\overline{C}A + \overline{M}\,\overline{C}A + \overline{M}CAB + \overline{M}\,\overline{C}AB = \overline{M}\,\overline{C}\,(1+A+AB) + \overline{M}CAB +$$

$$+ M\overline{C}A = \underbrace{\overline{M}\,\overline{C}\,\boxed{1}}_{I_0} + \underbrace{\overline{M}C\,\boxed{AB}}_{I_1} + \underbrace{M\overline{C}\,\boxed{A}}_{I_2} \quad (I_3 = 0)$$

Implementamos AB con otro MUX, eligiendo A y B como entradas de control

$$AB = \underbrace{\overline{A}\,\overline{B}\cdot\boxed{0}}_{I_0} + \underbrace{\overline{A}B\cdot\boxed{0}}_{I_1} + \underbrace{A\overline{B}\cdot\boxed{0}}_{I_2} + \underbrace{AB\cdot\boxed{1}}_{I_3}$$



2.4 IMPLEMENTACIÓN DE F MEDIANTE UN DECODIFICADOR Y UNA PUERTA LÓGICA

Vamos a completar la expresión de F para que en cada término aparezcan todas las variables. Conseguiremos así que F esté en su forma canónica (1ª), representando cada término a los '1' de la tabla de verdad de F. Luego sólo tenemos que unir en una OR todas las salidas de un DEC que corresponden con dichas filas de la tabla.

$$F = \bar{M}\bar{C} + A\bar{C} + AB\bar{M} = \bar{M}\bar{C}(A+\bar{A})(B+\bar{B}) + A\bar{C}(B+\bar{B})(M+\bar{M}) +$$

$$+ AB\bar{M}(C+\bar{C}) = (\bar{M}\bar{C}A + \bar{M}\bar{C}\bar{A})(B+\bar{B}) + (A\bar{C}B + A\bar{C}\bar{B})(M+\bar{M}) +$$

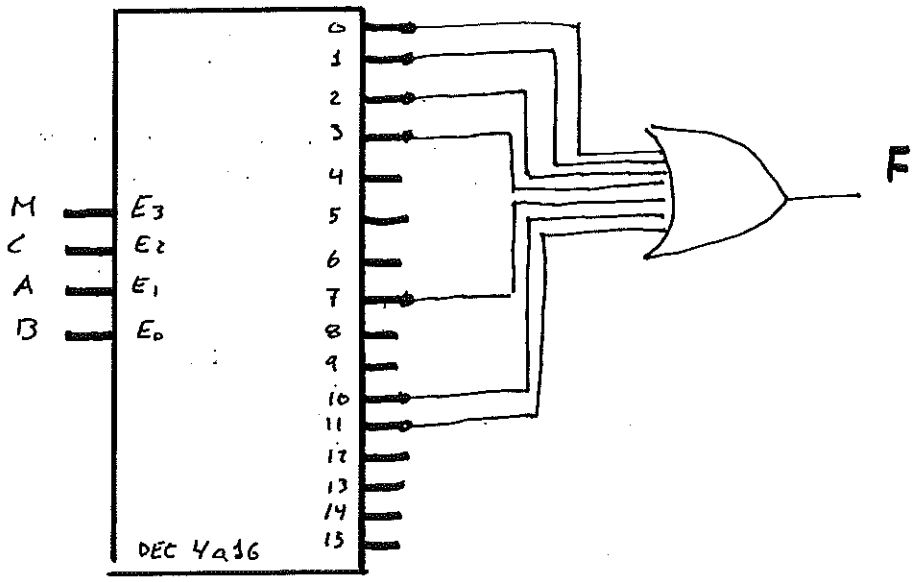
$$+ AB\bar{M}C + AB\bar{M}\bar{C} = \underline{\bar{M}\bar{C}AB} + \underline{\bar{M}\bar{C}A\bar{B}} + \underline{\bar{M}\bar{C}\bar{A}B} + \underline{\bar{M}\bar{C}\bar{A}\bar{B}} +$$

$$+ A\bar{C}BM + \underline{A\bar{C}B\bar{M}} + A\bar{C}\bar{B}M + \underline{A\bar{C}\bar{B}\bar{M}} + \underline{AB\bar{M}C} + \underline{AB\bar{M}\bar{C}} =$$

$$= \bar{M}\bar{C}\bar{A}\bar{B} + \bar{M}\bar{C}\bar{A}B + \bar{M}\bar{C}A\bar{B} + \bar{M}\bar{C}AB + \bar{M}CAB + M\bar{C}A\bar{B} + M\bar{C}AB$$

0000 0001 0010 0011 0111 1010 1011

0 1 2 3 7 10 11



SEPTIEMBRE 2006

PROBLEMA 1

La lectura de la temperatura de una cámara frigorífica, en grados centígrados, se obtiene con cuatro bits (T3,T2,T1,T0) codificada en complemento a 2, siendo T0 el bit menos significativo. Se necesitan dos salidas (LR y LV) que activen un led rojo (LR) y un led verde (LV) respectivamente (ambos activos a nivel alto). Para ello, se quiere implementar un circuito que active el led verde cuando la temperatura de la cámara esté entre -3 y +4 °C, ambas incluidas, y el led rojo en el resto de los casos.

1.1 Complete la tabla de verdad siguiente (Tabla 1). *(5 puntos)*

Lo pasamos a: Complemento a 2

Decima	T3	T2	T1	T0	LR	LV
0	0	0	0	0	0	1
1	0	0	0	1	0	1
2	0	0	1	0	0	1
3	0	0	1	1	0	1
4	0	1	0	0	0	1
5	0	1	0	1	1	0
6	0	1	1	0	1	0
7	0	1	1	1	1	0
-8	1	0	0	0	1	0
-7	1	0	0	1	1	0
-6	1	0	1	0	1	0
-5	1	0	1	1	1	0
-4	1	1	0	0	1	0
-3	1	1	0	1	0	1
-2	1	1	1	0	0	1
-1	1	1	1	1	0	1

LV=1 (4 primeras filas y últimas 3 filas)

LR=1 (filas -8 a -4)

1.2 Obtenga la función lógica simplificada para la salida LV. Para ello utilice un mapa de Karnaugh. *(5 puntos)*

1.3 Implemente la salida LV utilizando el mínimo número necesario de multiplexores de cuatro y de dos entradas de datos. *(10 puntos)*

1.2) $LV = \bar{T}_3\bar{T}_2 + \bar{T}_3\bar{T}_1\bar{T}_0 + T_3T_2T_0 + T_3T_2T_1$ (El mapa de Karnaugh no lo hacemos, nos lo dan hecho)

1.3) Implementación de LV mediante MUX 4x2, MUX 2x1:

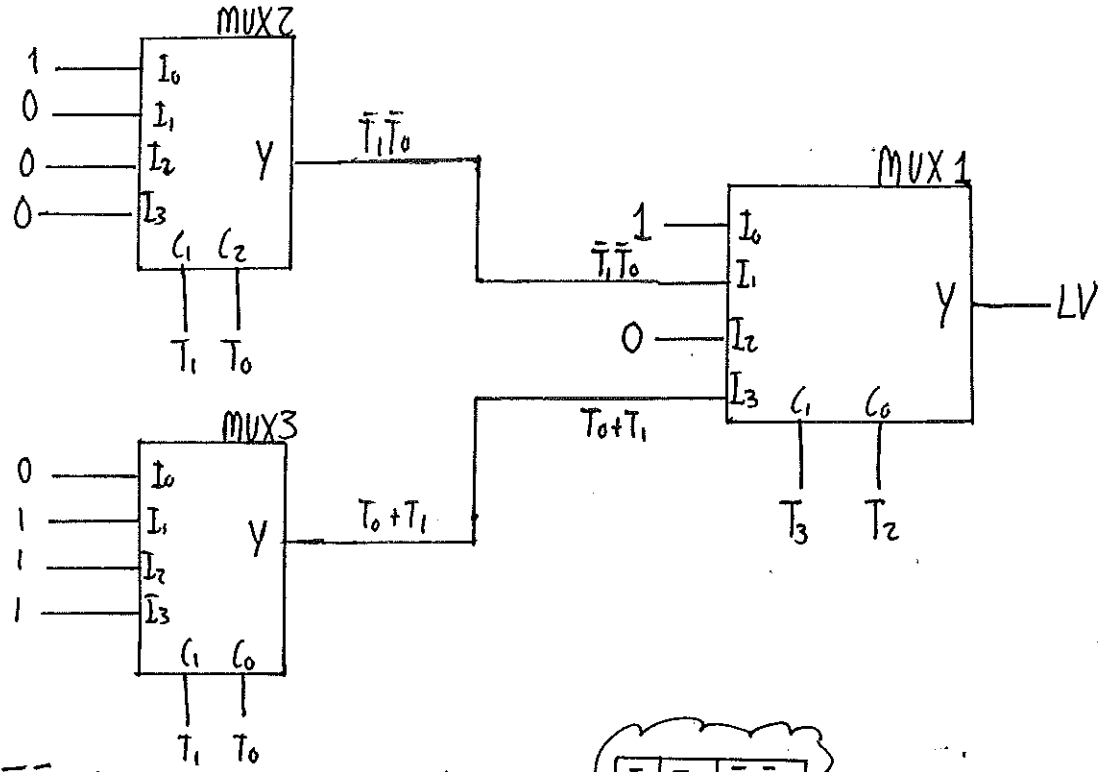
Vamos a elegir como variables de control T_3, T_2:

$$LV = \bar{T}_3\bar{T}_2 + \bar{T}_3\bar{T}_1\bar{T}_0(T_2+\bar{T}_2) + T_3T_2T_0 + T_3T_2T_1 = \bar{T}_3\bar{T}_2 + \bar{T}_3T_2\bar{T}_1\bar{T}_0 + \bar{T}_3\bar{T}_2\bar{T}_1\bar{T}_0 + T_3T_2T_0 + T_3T_2T_1 =$$

$$= \bar{T}_3\bar{T}_2(\underbrace{1}_{1} + \bar{T}_1\bar{T}_0) + \bar{T}_3T_2\bar{T}_1\bar{T}_0 + T_3T_2(T_0+T_1)$$

$$Y = \bar{C}_1\bar{C}_0I_0 + \bar{C}_1C_0I_1 + C_1\bar{C}_0I_2 + C_1C_0I_3$$

$$LV = \bar{T}_3\bar{T}_2\boxed{1} + \bar{T}_3T_2\boxed{\bar{T}_1\bar{T}_0} + T_3\bar{T}_2\boxed{0} + T_3T_2\boxed{T_0+T_1}$$



■ Implementación de $\bar{T}_1\bar{T}_0$ (2 variables, 2 cnt. control):

$\bar{T}_1\bar{T}_0 = \bar{T}_1\bar{T}_0\boxed{1} + \bar{T}_1 T_0\boxed{0} + T_1\bar{T}_0\boxed{0} + T_1 T_0\boxed{0}$

T_1	T_0	$\bar{T}_1\bar{T}_0$
0	0	1
0	1	0
1	0	0
1	1	0

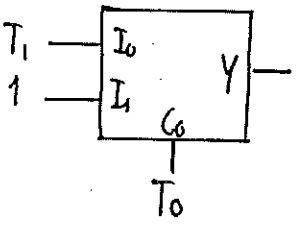
■ Implementación de T_1+T_0: Lo hacemos con la tabla de verdad:

T_1	T_0	T_1+T_0
0	0	0
0	1	1
1	0	1
1	1	1

■ Otra opción de hacerlo:

■ Implementamos T_0+T_1 mediante MUX 2x1: $Y = \bar{C}_0 I_0 + C_0 I_1$. Elegimos T_0 como variable de control

$T_0+T_1 = T_0 + T_1(T_0+\bar{T}_0) = T_0 + T_1T_0 + T_1\bar{T}_0 = \bar{T}_0(T_1) + T_0(1+T_1)$



■ Sin embargo con $\bar{T}_1\bar{T}_0$ no podemos hacer lo mismo:

$\bar{T}_1 \cdot \bar{T}_0 = \bar{T}_1\boxed{\bar{T}_0} + T_1\boxed{0}$

FEBRERO 2007

PROBLEMA 2

Se quiere construir un circuito que proporcione los bits (S7...S0) de la función aritmética **s** =2·**n**+14, siendo **n** un número de 4 bits expresado en complemento a 1 [**N3 N2 N1 N0**]. El resultado debe expresarse mediante dos dígitos en código BCD (cada uno de ellos de 4 bits).

2.1 Ignore la columna **F** y complete la tabla de verdad siguiente correspondiente a dicho circuito *(5 puntos)*

2.1)

	n				s								
	N3	N2	N1	N0	Dígito BCD más significativo				Dígito BCD menos significativo				F
0	0	0	0	0	0	0	0	1	0	1	0	0	1
1	0	0	0	1	0	0	0	1	0	1	1	0	1
2	0	0	1	0	0	0	0	1	1	0	0	0	0
3	0	0	1	1	0	0	1	0	0	0	0	0	0
4	0	1	0	0	0	0	1	0	0	0	1	0	1
5	0	1	0	1	0	0	1	0	0	1	0	0	1
6	0	1	1	0	0	0	1	0	0	1	1	0	0
7	0	1	1	1	0	0	1	0	1	0	0	0	0
-7	1	0	0	0	0	0	0	0	0	0	0	0	1
-6	1	0	0	1	0	0	0	0	0	0	1	0	1
-5	1	0	1	0	0	0	0	0	0	1	0	0	X
-4	1	0	1	1	0	0	0	0	0	1	1	0	X
-3	1	1	0	0	0	0	0	0	1	0	0	0	0
-2	1	1	0	1	0	0	0	1	0	0	0	0	0
-1	1	1	1	0	0	0	0	1	0	0	1	0	0
0	1	1	1	1	0	0	0	1	0	1	0	0	0

2.2 Simplifique la señal **F** mediante Karnaugh en función de las variables **N0, N1, N2** y **N3** de tal modo que pueda realizarse con el menor número de puertas lógicas. Implemente dicha función utilizando solamente una puerta AND y otra OR de dos entradas cada una. Nota: Se dispone tanto de las variables como de sus negadas. *(5 puntos)*

2.3 Implemente la salida **F** utilizando el mínimo número necesario de multiplexores de cuatro entradas de datos. Nota: En este caso NO se dispone de las variables negadas. *(5 puntos)*

2.4 Se quiere ahora construir un circuito que implemente la función **s** = 2·**n** + 14 del apartado 2.1 utilizando sumadores. Para ello se dispone de dos cadenas de sumadores (Figura 4). En este caso queremos que la salida **s** esté codificada en binario (**S4 S3 S2 S1 S0**) en lugar de en BCD. Tenga en cuenta lo siguiente:

- En primer lugar, siendo **n** la representación de un número en complemento a 1, si éste fuera negativo es necesario convertirlo primero a complemento a dos. Utilice para ello la primera cadena de sumadores.
- En segundo lugar tenga en cuenta que multiplicar por 2 un número binario implica añadir un "0" a la derecha del bit menos significativo.
- Por último utilice la segunda cadena de sumadores para implementar 2·**n** + 14. *(10 puntos)*

NOTA: Complemento a 1: ejemplos

- 0111 → +7
- 1000 → 0111 → -7
- 0000 → 0
- 1111 → 0000 → 0

NOTA: se codifican, por separado, cada una de las cifras decimales, en binario.

1	3

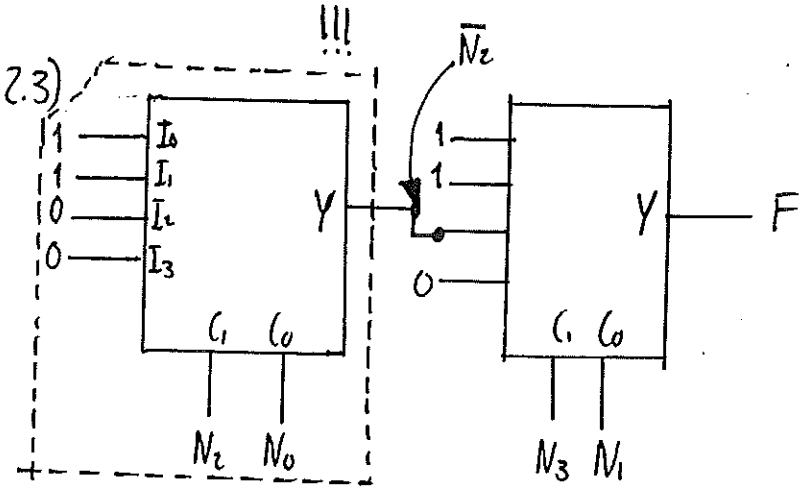
↑ Decimal

1 3

2.2) $F = \overline{N}_1 \cdot (\overline{N}_3 + \overline{N}_2)$, simplificando por ceros

N1N0 / N3N2	00	01	11	10
00	1	1	0	0
01	1	1	0	0
11	0	0	0	0
10	1	1	X	X

2.3)



$F = \overline{N}_3 \overline{N}_1 \cdot 1 + N_3 \overline{N}_1 \cdot \overline{N}_2$

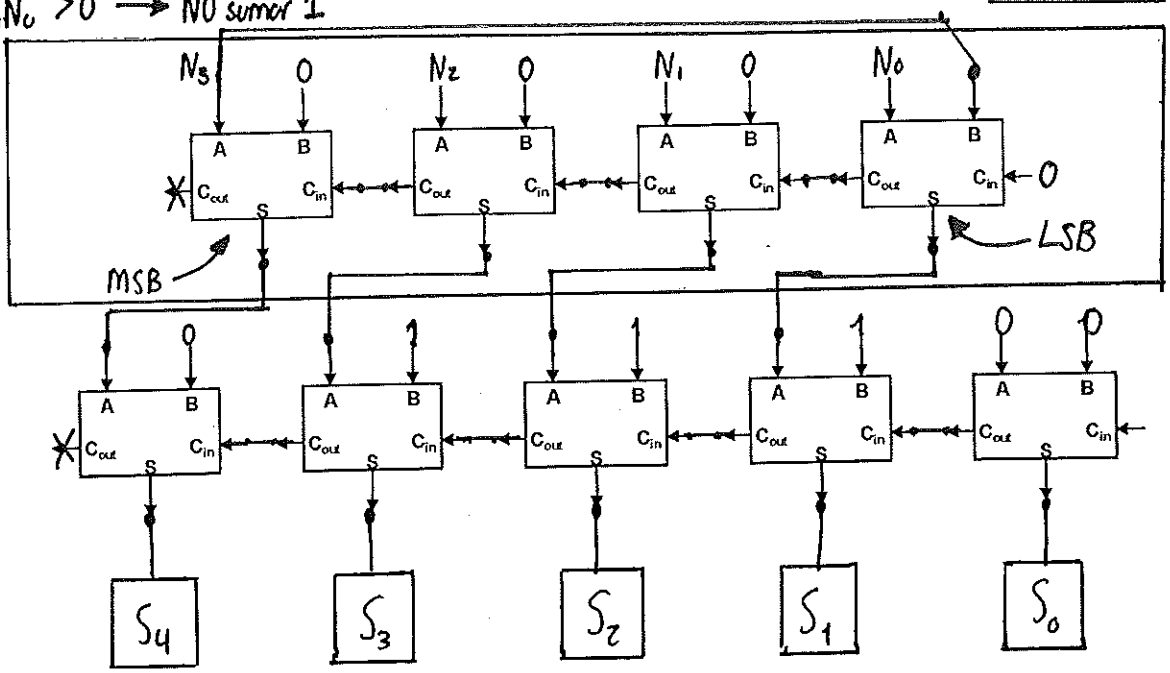
$N_3 N_2 N_1 N_0 < 0 \rightarrow$ sumar 1

$N_3 N_2 N_1 N_0 > 0 \rightarrow$ NO sumar 1

Si $N_3 = 1$ se suma 1. Si $N_3 = 0$ sumamos 0.

0 0 0 N_3

$N_3 N_2 N_1 N_0$ +



Este bloque tiene que sumar N_3

SEPTIEMBRE 2008 (Resuelto o extra)

PROBLEMA 2 (18 *PUNTOS*)

En binario natural (base 2) el complemento a 1 se genera simplemente invirtiendo todos los bits. Lo equivalente en el sistema decimal (base 10) es el complemento a 9 y se genera obteniendo la diferencia a 9 de cada cifra decimal. Así, el complemento a 9 del número 248 será 751. En este problema se trata de diseñar un circuito complementador a 9 de un número expresado en código BCD (**A3 A2 A1 A0**, siendo **A0** el bit menos significativo).

2.1 Escriba la tabla de verdad de las funciones **C3, C2, C1** y **C0** (bits que representan el complemento a 9 de **A3 A2 A1 A0**, ambos expresados en código BCD). (*3 puntos*)

Si esto es BCD, solo llega al 9

$C = 9 - A$

	A3	A2	A1	A0	C3	C2	C1	C0	
0	0	0	0	0	1	0	0	1	9-0 = 9
	0	0	0	1	1	0	0	0	9-1 = 8
	0	0	1	0	0	1	1	1	7 ...
	0	0	1	1	0	1	1	0	6
	0	1	0	0	0	1	0	1	5
	0	1	0	1	0	1	0	0	4
	0	1	1	0	0	0	1	1	3
	0	1	1	1	0	0	1	0	2
	1	0	0	0	0	0	0	1	1
	1	0	0	1	0	0	0	0	0
esto no es BCD	10	1	0	1	0	X	X	X	X	X
	11	1	0	1	1	X	X	X	X
	12	1	1	0	0	X	X	X	X
	13	1	1	0	1	X	X	X	X
	14	1	1	1	0	X	X	X	X
	15	1	1	1	1	X	X	X	X

ponemos X porque no son casos de interés, es decir, para implementar las funciones no "nos importan" estas filas.

2.2 Determine las funciones lógicas simplificadas de las cuatro funciones **C3-C0**. Para ello haga uso de las herramientas necesarias (mapas de Karnaugh, etc.). Justifique claramente sus respuestas. (*5 puntos*).

2.3 Implemente dichas funciones, **C3-C0**, utilizando únicamente puertas NAND de 2 y 3 entradas. (*5 puntos*).

2.4 Implemente las funciones **C3-C0** utilizando multiplexores de 4 entradas de datos y los inversores necesarios. (*5 puntos*).

2.2 Observando la tabla, directamente:

$C\phi = \overline{A\phi}$ $\quad$ $C1 = A1$

C2

A1Aφ \ A3A2	00	01	11	10
00	0	1	X	0
01	0	1	X	0
11	1	0	X	X
10	1	0	X	X

$C2 = A2 \cdot \overline{A1} + \overline{A2} \cdot A1 = A1 \oplus A2$

C3

A1Aφ \ A3A2	00	01	11	10
00	1	0	X	0
01	1	0	X	0
11	0	0	X	X
10	0	0	X	X

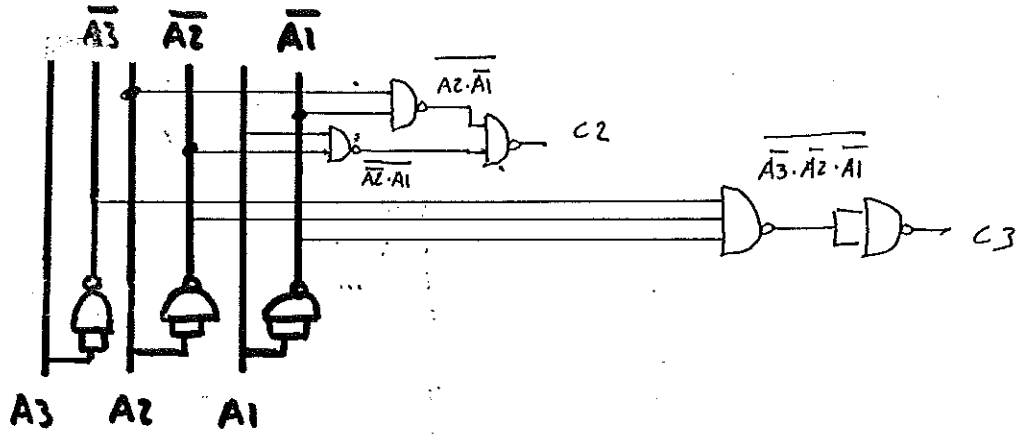
$C3 = \overline{A3} \cdot \overline{A2} \cdot \overline{A1} = \overline{A3 + A2 + A1}$

2.3 Implementación con puertas $NAND_2$ y $NAND_3$:

Hombre... $C1$ y $C\phi$ son mierda pura: $C1 = A1$, $C\phi = \overline{A\phi}$

☑ $C2 = \overline{A2 \cdot \overline{A1} + \overline{A2} \cdot A1} = \overline{\overline{A2 \cdot \overline{A1}} \cdot \overline{\overline{A2} \cdot A1}}$

☑ $C3 = \overline{A3 + A2 + A1} = \overline{\overline{A3} \cdot \overline{A2} \cdot \overline{A1}}$

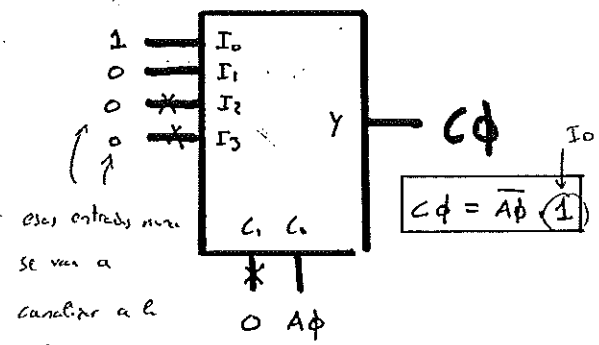


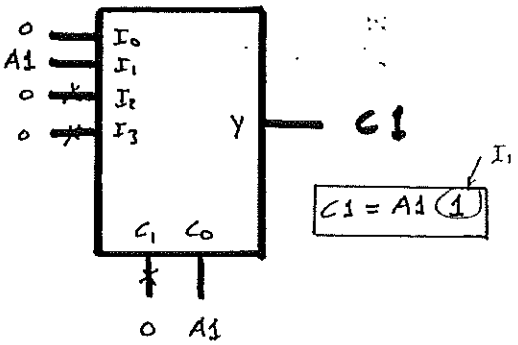
2.4 Implementación con MUX 4X2

Salida de un MUX 4X2

$$Y = \overline{C_1}\,\overline{C_0}\,I_0 + \overline{C_1}\,C_0\,I_1 + C_1\,\overline{C_0}\,I_2 + C_1\,C_0\,I_3$$

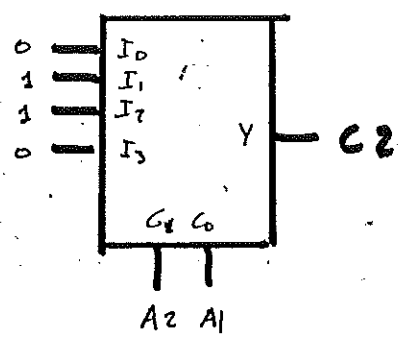
☑ $C1$ y $C\phi$ solo tienen una variable: CABLEAMOS la entrada más significativa de control

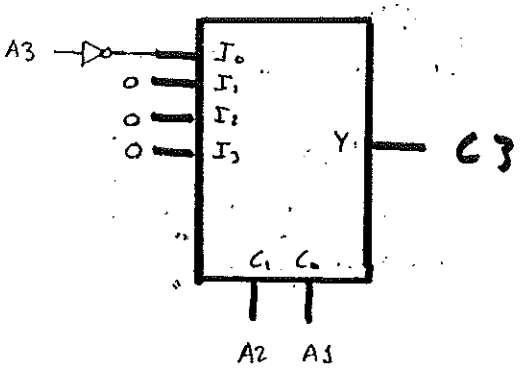




☑ $C2 = A2 \cdot \overline{A1} \cdot (1) + \overline{A2} \cdot A1 \,(1)$ (I2, I1)

☑ $C3 = \overline{A2} \cdot \overline{A1} \cdot (\overline{A3})$ (I0)





FEBRERO 2010

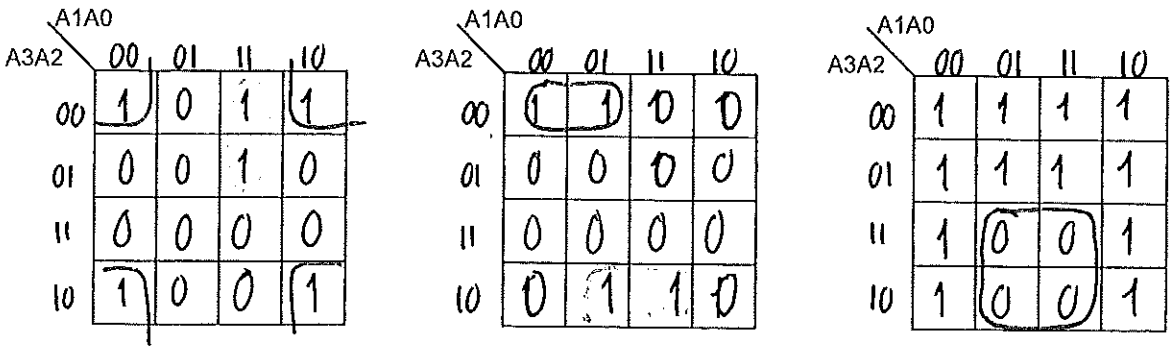
PROBLEMA 2 (15 *PUNTOS*)

Queremos diseñar un circuito que convierta un número en código binario sin signo de 4 bits (A3 A2 A1 A0, siendo A3 el bit más significativo) en su equivalente en código BCD representado con 2 cifras (unidades [U3 U2 U1 U0] y decenas [D3 D2 D1 D0], siendo D3 y U3 los bits más significativos).

2.1 Rellene la siguiente tabla de verdad para que el circuito realice la citada conversión. Para este apartado ignore las funciones F,G y H. *(3 puntos)*

	ENTRADAS				SALIDAS										
	Número binario				Decenas BCD				Unidades BCD						
	A3	A2	A1	A0	D3	D2	D1	D0	U3	U2	U1	U0	F	G	H
0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1
1	0	0	0	1	0	0	0	0	0	0	0	1	0	1	1
2	0	0	1	0	0	0	0	0	0	0	1	0	1	0	1
3	0	0	1	1	0	0	0	0	0	0	1	1	1	0	1
4	0	1	0	0	0	0	0	0	0	1	0	0	0	0	1
5	0	1	0	1	0	0	0	0	0	1	0	1	0	0	1
6	0	1	1	0	0	0	0	0	0	1	1	0	0	0	1
7	0	1	1	1	0	0	0	0	0	1	1	1	1	0	1
8	1	0	0	0	0	0	0	0	1	0	0	0	1	0	1
9	1	0	0	1	0	0	0	0	1	0	0	1	0	1	0
10	1	0	1	0	0	0	0	1	0	0	0	0	1	0	1
11	1	0	1	1	0	0	0	1	0	0	0	1	0	1	0
12	1	1	0	0	0	0	0	1	0	0	1	0	0	0	1
13	1	1	0	1	0	0	0	1	0	0	1	1	0	0	0
14	1	1	1	0	0	0	0	1	0	1	0	0	0	0	1
15	1	1	1	1	0	0	0	1	0	1	0	1	0	0	0

2.2 Simplifique por Karnaugh las funciones F, G y H, en función de A3 A2 A1 y A0 (emplee las tablas que se adjuntan). *(7 puntos)*



$F = \bar{A0}\bar{A2} + \bar{A3}A1A0$

$G = \bar{A3}\bar{A2}\bar{A1} + A3\bar{A2}A0$

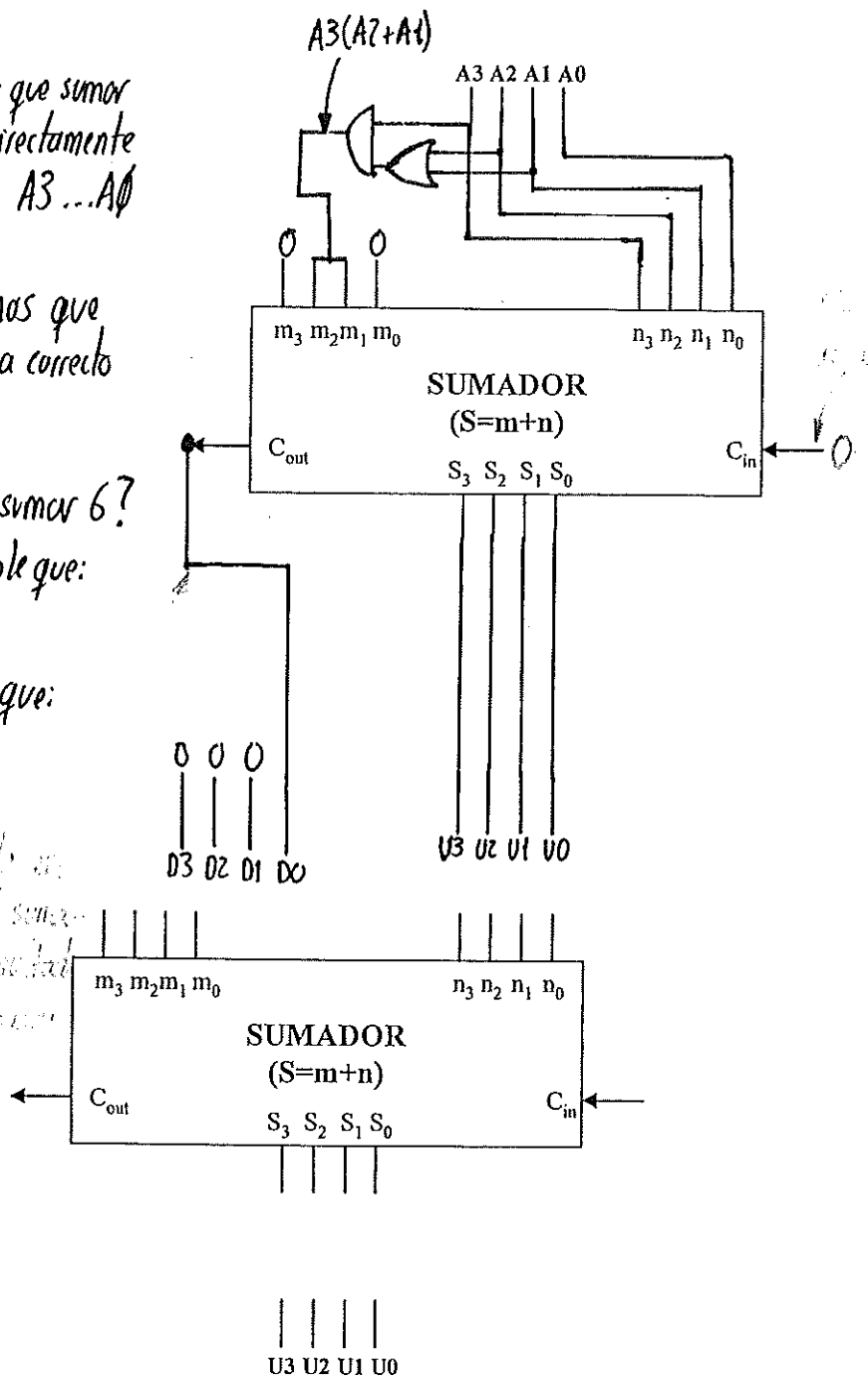
$H = \bar{A3} + \bar{A0}$

2.3 Realice ahora la misma conversión binario-BCD utilizando para ello 2 sumadores de 4 bits y las puertas lógicas que necesite. Dibuje las conexiones y puertas necesarias sobre el esquema siguiente. *(5 puntos)*

■ En los números del 0 al 9 no hay que sumar nada para obtener el resultado: directamente el resultado es igual que los bits A3...A0

■ En los números del 10 al 15 tenemos que sumar 6 para que el resultado sea correcto al interpretarlo como BCD

■ ¿Cómo detectamos cuando hay que sumar 6?
En las filas 10-15 siempre se cumple que:
$(A3 = 1) \wedge (A2 = 1 \vee A1 = 1)$
Tendremos que sumar 6 si se cumple que:
$A3 \cdot (A2 + A1) = 1$

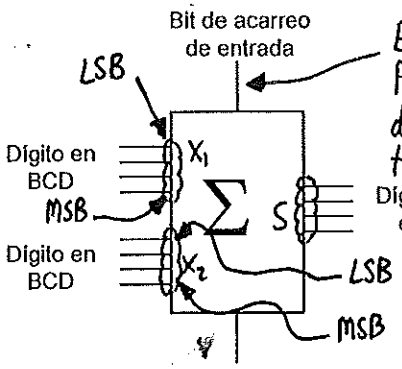


FEBRERO 2008

PROBLEMA 1

1.1 Sea un número binario de 6 bits sin signo, **A5 A4 A3 A2 A1 A0**, siendo **A0** el bit menos significativo. Implementar un conversor de este número a formato BCD. Para ello se dispone de bloques sumadores de 2 dígitos BCD, con acarreo de entrada y acarreo de salida tal como se presenta en el anexo 1, y el mínimo número de puertas lógicas adicionales que necesite. *(15 puntos)*

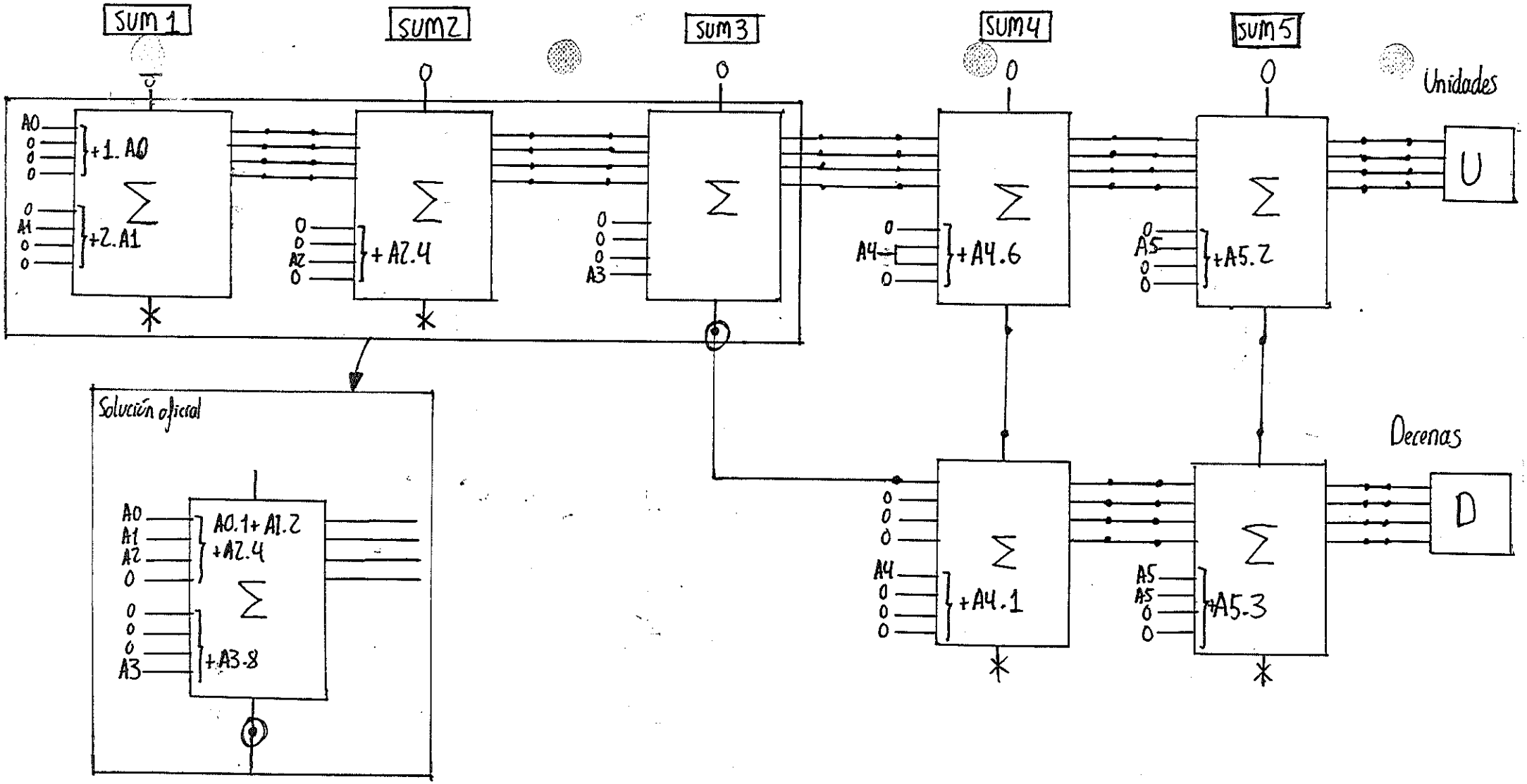
Bloque sumador de dos dígitos BCD con acarreos de entrada y de salida.



dos números x_1 y x_2 y S están en BCD, es decir, son números binarios de 0 a 9. Si la suma es mayor que nueve se usa el bit de acarreo de salida como decenas.

Este acarreo sirve para concatenar sumadores. Por ejemplo: "la que te llevas" en un sumador de unidades la conectamos al acarreo de entrada de un sumador de decenas.

- Porque sumadores BCD? $A5A4A3A2A1A0 = \underbrace{A0.1 + A1.2}_{sum1} + \overbrace{A2.4}^{sum2} + \underbrace{A3.8}_{sum3} + \overbrace{A4.16}^{sum4} + \underbrace{A5.32}_{sum5}$
- Cuantos dígitos decimales necesitamos: En el caso peor: $111111_B = \boxed{63_D}$ necesitamos dos cadenas de sumadores!!


SUM 1
SUM2
SUM 3
SUM 4
SUM 5
Unidades
U
Decenas
D
A0
0
0
0
+1. A0
0
A1
0
0
+2. A1
0
0
A2
0
+ A2.4
0
0
0
A3
A4
0
+A4.6
0
A5
0
0
+A5.2
0
0
0
A4
0
0
0
+A4.1
A5
A5
0
0
+A5.3
Solución oficial
A0
A1
A2
0
A0.1+ A1.2
+A2.4
0
0
0
A3
+A3.8

NOVIEMBRE 2011

PROBLEMA 2 (30 PUNTOS)

Implemente un conversor de código BCD de dos dígitos (B para las decenas y A para las unidades) a código binario de 7 bits (S6…S0) utilizando exclusivamente dos sumadores de 4 bits con acarreos de entrada y salida como los de la Figura. **Justifique** adecuadamente su solución.

Conversor BCD - Binario:

■ Dado un número en BCD ($\underbrace{B_3 B_2 B_1 B_0}_{\text{"decenas"}}$ $\underbrace{A_3 A_2 A_1 A_0}_{\text{"unidades"}}$) son cifras ya están en binario pero "separadas"

Sólo tenemos que mezclarlas y para ello:

1) Multiplicamos por 10 (en binario 1010) la cifra de decenas.
2) Al resultado le sumamos la cifra de las unidades.

■ 1º) Multiplico las decenas por 10

	B_3	B_2	B_1	B_0
×	1	0	1	0
	0	0	0	0
	B_3	B_2	B_1	B_0
0	0	0	0	
+ B_3	B_2	B_1	B_0	

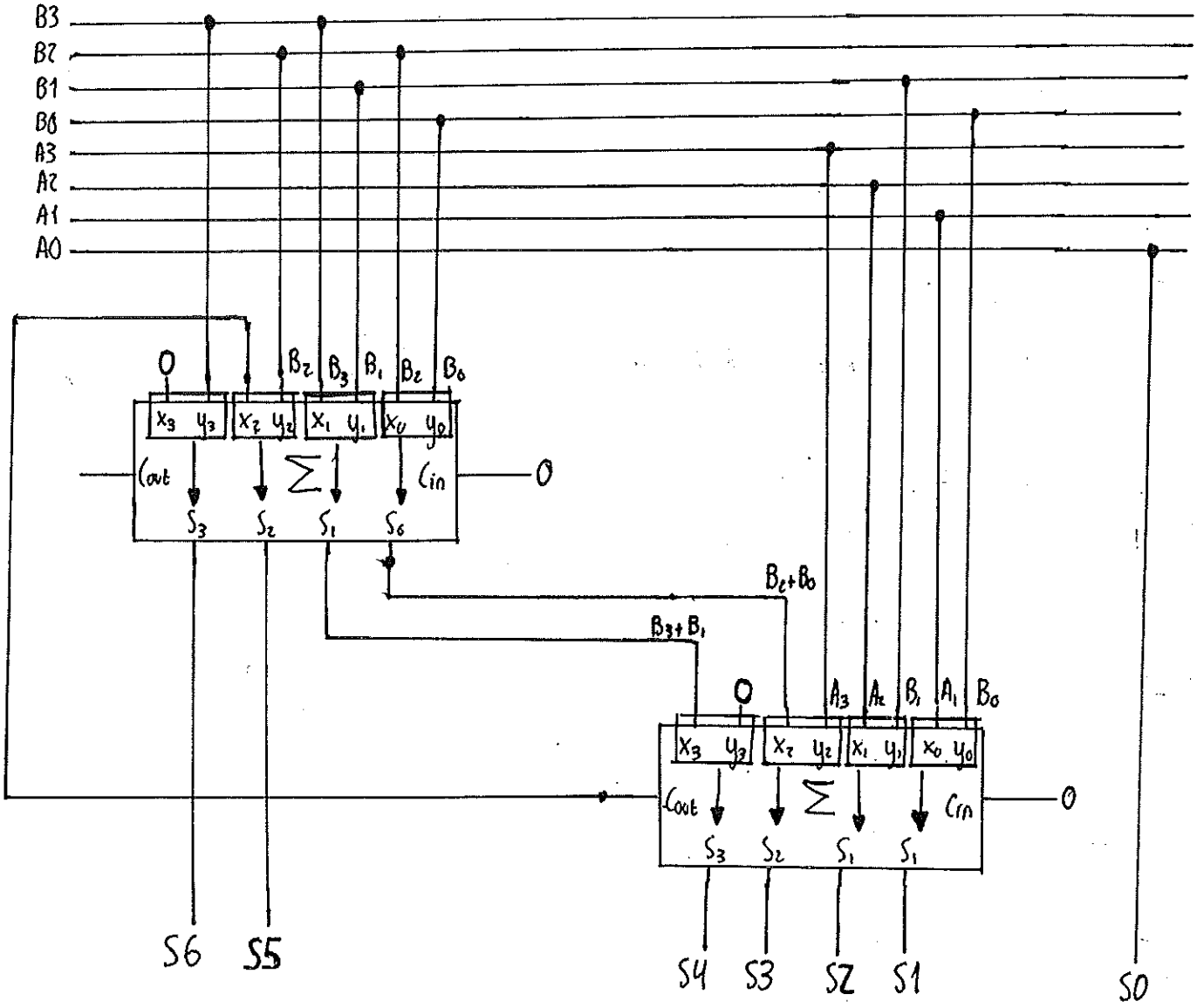
2º) Sumo la cifra de las unidades

B_3	B_2	B_3+B_1	B_2+B_0	B_1	B_0	0	
+			A_3	A_2	A_1	A_0	
B_3	B_2	B_3+B_1	$B_2+B_0+A_3$	B_1+A_2	B_0+A_1	A_0	S_0
S_6	S_5	S_4	S_3	S_2	S_1		

■ ¿Cuántas cifras necesitamos para el resultado (en binario)?

Caso peor: $99_D = 1100011_B$

S6 S0

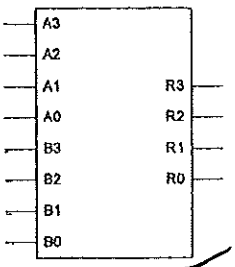

B3
B2
B1
B0
A3
A2
A1
A0
0
B2
B3
B1
B2
B0
X3 Y3
X2 Y2
X1 Y1
X0 Y0
Cout
Σ
Cin
0
S3
S2
S1
S0
B2+B0
B3+B1
0
A3
A2
B1
A1
B0
X3 Y2
X2 Y2
X1 Y1
X0 Y0
Cout
Σ
Cin
0
S3
S2
S1
S1
S6
S5
S4
S3
S2
S1
S0

SEPTIEMBRE 2004

PROBLEMA 2

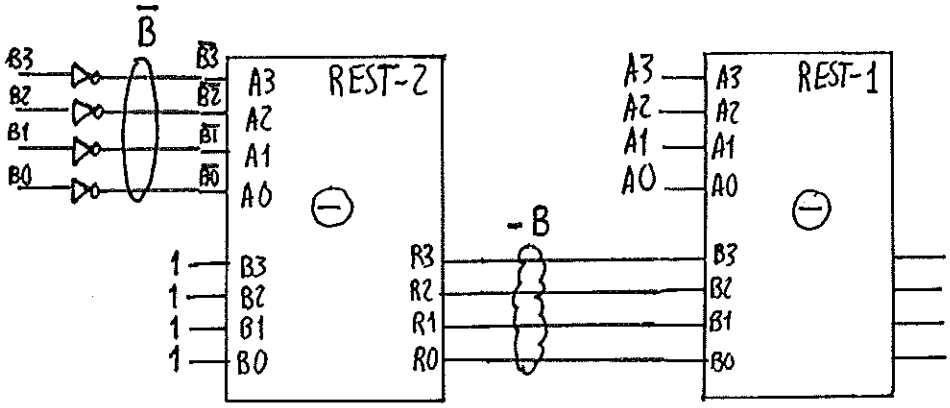
Diseñe un sumador de dos números **A** y **B** (**S = A + B**) de cuatro bits en complemento a dos **utilizando el mínimo número de componentes**. Dibuje el esquema resultante. Para ello dispone de restadores de dos números **A** y **B** (**R = A - B**) de cuatro bits en complemento a dos (*Figura 2*) y de inversores: (*15 puntos*)

$S = A - (-B) = A - (0 - B)$



Rest-1 Rest-1 Rest-2

$S = A + B = A \ominus (-B) = A \ominus [\overline{B} \ominus (-1)]$



También podríamos haber obtenido -B como $\boxed{0 - B}$

PROBLEMA 5 (10 PUNTOS)

Consideremos el siguiente fragmento de un módulo hardware descrito en VHDL.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
-------------------------------------------------------
entity XXXXX is

port(    AA:   in std_logic_vector(5 downto 0);
         BB:   in std_logic_vector(5 downto 0);
         CC:   out std_logic;
         DD:   out std_logic_vector(5 downto 0)
       );

end XXXXX;
-------------------------------------------------------
architecture behv of XXXXX is

signal result: std_logic_vector(N downto 0);

begin

result <= ('0' & AA)+('0' & BB);
DD <= result(5 downto 0);
CC <= result(6);

end behv;
-------------------------------------------------------
```

Declaramos una señal auxiliar de N+1 bits.

Necesitamos 7 bits para manejar el resultado: N+1=7

N=6

Sumamos AA+BB añadiendo a cada sumando un bit 0.

Del resultado, los 6 bits menos significativos son la suma.

Del resultado, el bit más significativo es el acarreo de salida.

Tener en cuenta que el operador '&' permite la concatenación de cadenas de bits. (por ejemplo, '1' & '0011' = '10011').

5.1 Dibujar en el bloque de la Figura 6 la interfaz del módulo hardware (puertos de entrada y salida), colocando las entradas a la izquierda y las salidas a la derecha. (*1 punto*)

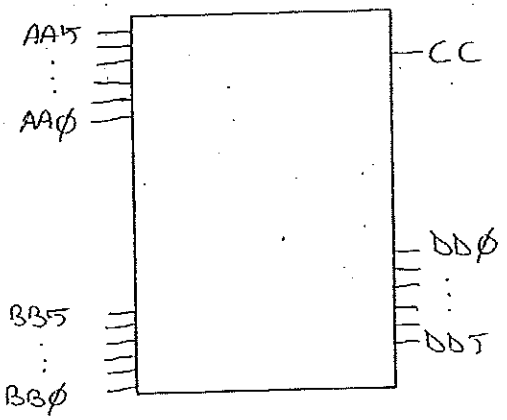


Figura 6

5.2 ¿Cuál es la función que realiza el circuito? Es decir, ¿qué módulo hardware estamos definiendo? *(1 punto)*

Sumador de 6 bits con acarreo de salida.

5.3 ¿Cuál debiera ser el valor de 'N' en el tamaño de la señal **result**? (Justificar la respuesta). *(2 puntos)*

N = 6

- CC <= result(6) ; implica que el séptimo bit (6+1) de la señal result debe existir.
- result contiene el resultado de la suma (6 bits) más el acarreo (1 bit): 6+1 = 7 bits => => (6 downto 0)

Después de la fase de síntesis, la implementación con puertas lógicas de otro módulo hardware es el representado en la Figura 7.

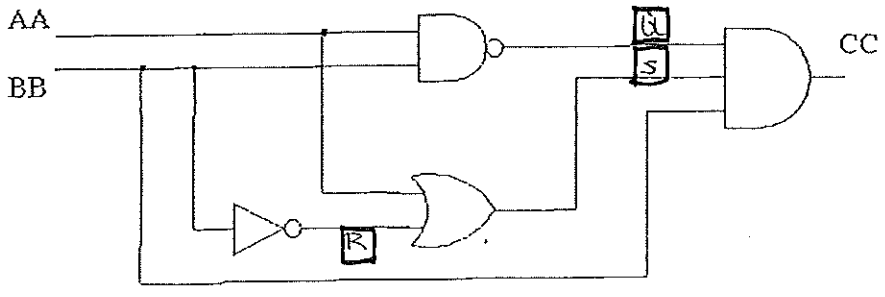


Figura 7

5.4 Complete, rellenando las casillas necesarias que aparecen en trazo discontinuo, el siguiente fragmento de código VHDL de forma que defina el esquemático de la figura 7. *(3 puntos)*

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
-------------------------------------------------------
entity XXXXX is

port(  AA:   in std_logic;
       BB:   in std_logic;
       CC:   out std_logic
     );

end XXXXX;
-------------------------------------------------------
architecture syn of XXXXX is

   component NAND2
      port( A, B : in std_logic;  Z : out std_logic);
   end component;

   component INV
      port( A : in std_logic;  Z : out std_logic);
   end component;

   component OR2
      port( A, B : in std_logic;  Z : out std_logic);
   end component;

   component NAND3
      port( A, B, C : in std_logic;  Z : out std_logic);
   end component;

-- declaración de señales
      signal Q : std_logic;
      signal R : std_logic;
      signal S : std_logic;
begin
   U1: NAND2 port map  (A=> AA , B=> BB , Z=> Q );
   U2: INV port map    (A=> BB , Z=> R );
   U3: OR2 port map    (A=> R , B=> AA , Z=> S );
   U4: AND3 port map   (A=> Q , B=> S , C=> BB , Z=> CC );
end syn;
```

5.5 Indicar qué modificaciones deberían realizarse en el código anterior para obtener la versión negada de la salida en lugar de la versión implementada. *(3 puntos)*

Sol. 1

component NAND3 ...

U4 : NAND3 port map (A => Q, B => S, C => BB, Z => CC);

Sol. 2

```
component INV ...
signal T : std_logic;
U4 : AND3 port map (A => Q, B => S, C => BB, Z => T);
U5 : INV port map (A => T, Z => CC);
```

En ambos casos, se podría cambiar CC por CC_L en la ENTITY.

PROBLEMA 2

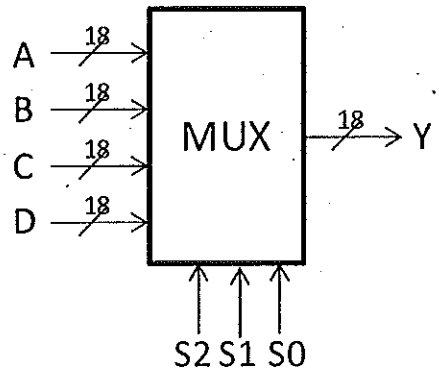
El siguiente módulo está descrito en VHDL. Complete las etiquetas (**XXX_XXX, YYY_YYY, ZZZ_ZZZ**) indique en el esquema de bloques las entradas y salidas y explique la funcionalidad de dicho módulo..

```
library IEEE;
use IEEE.std_logic_1164.all;

entity componente is
  port (
      S:          in STD_LOGIC_VECTOR (2 downto 0);
      XXX_XXX:    in STD_LOGIC_VECTOR (1 to 18);
      Y:          out STD_LOGIC_VECTOR (1 to 18)  );
end componente;

architecture arqComponente of YYY_YYY is
  begin
  case ZZZ_ZZZ is
    when "000" | "010" | "100" | "110" => Y <= A;
    when "001" | "111" => Y <= B;
    when "011" => Y <= C;
    when "101" => Y <= D;
    when others => Y <= (others => '0');
  end case;
end arqComponente;
```

XXX_XXX:	**A, B, C, D**
YYY_YYY:	componente
ZZZ_ZZZ:	**S**



El código describe un **multiplexor** con 4 entradas de datos (**A,B,C,D**) de 18 bits cada una, tres bits de selección (**S2,S1,S0**) y una salida de 18 bits (**Y**). La selección del dato se realiza según la siguiente asignación:

- Se selecciona la entrada **A** cuando **S** vale 0,2,4 ó 6 (en valor decimal)
- Se selecciona la entrada **B** cuando **S** vale 1 ó 7 (en valor decimal)
- Se selecciona la entrada **C** cuando **S** vale 3 (en valor decimal)
- Se selecciona la entrada **D** cuando **S** vale 5 (en valor decimal)

PROBLEMA 2 (25 *PUNTOS*)

Queremos diseñar un conversor de un número de 4 bits en binario sin signo, **(A3 A2 A1 A0)**, a un número representado en código BCD.

2.1 Obtener la representación del número binario 1100 en código BCD. **Nota**: En lógica digital se trabaja exclusivamente con bits. (5 puntos)

$1100_2 = 12_{10} \rightarrow$ 12 en decimal corresponde a **0001 0010** en BCD

2.2 Para realizar el conversor del número binario sin signo de 4 bits a código BCD represente las funciones que los relacionan como suma canónica. A continuación, y SIN SIMPLIFICAR, implemente todas las funciones únicamente con cuatro multiplexores de tres entradas de control cada uno, y los inversores que sean necesarios.
Nota: Sólo se considerará en la corrección los multiplexores en los cuales estén CLARAMENTE indicadas y etiquetadas las entradas, salidas y señales de control. (10 puntos)

	A3	A2	A1	A0	D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	0	0	0	0	1
2	0	0	1	0	0	0	0	0	0	0	1	0
3	0	0	1	1	0	0	0	0	0	0	1	1
4	0	1	0	0	0	0	0	0	0	1	0	0
5	0	1	0	1	0	0	0	0	0	1	0	1
6	0	1	1	0	0	0	0	0	0	1	1	0
7	0	1	1	1	0	0	0	0	0	1	1	1
8	1	0	0	0	0	0	0	0	1	0	0	0
9	1	0	0	1	0	0	0	0	1	0	0	1
10	1	0	1	0	0	0	0	1	0	0	0	0
11	1	0	1	1	0	0	0	1	0	0	0	1
12	1	1	0	0	0	0	0	1	0	0	1	0
13	1	1	0	1	0	0	0	1	0	0	1	1
14	1	1	1	0	0	0	0	1	0	1	0	0
15	1	1	1	1	0	0	0	1	0	1	0	1

$D7 = D6 = D5 = 0$

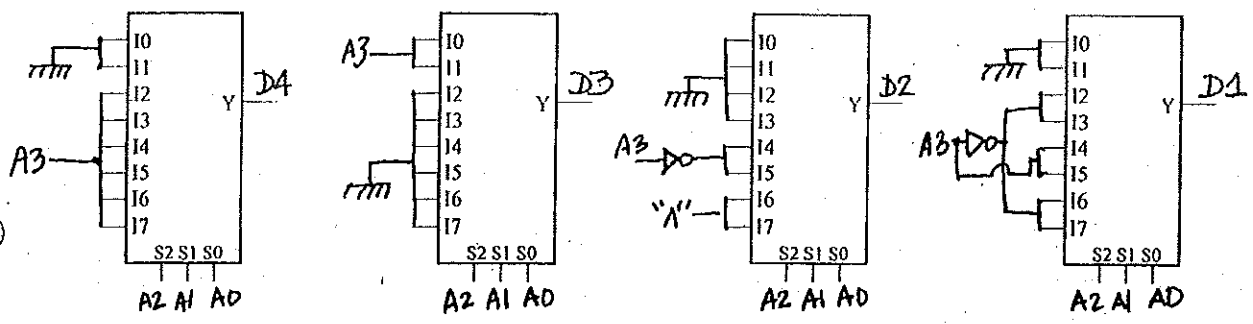
$D4 = \sum_{A3A2A1A0}(10,11,12,13,14,15) = A3\overline{A2}A1\overline{A0} + A3\overline{A2}A1A0 + A3A2\overline{A1}\overline{A0} + A3A2\overline{A1}A0 + A3A2A1\overline{A0} + A3A2A1A0$

$D3 = \sum_{A3A2A1A0}(8,9) = A3\overline{A2}\overline{A1}\overline{A0} + A3\overline{A2}\overline{A1}A0$

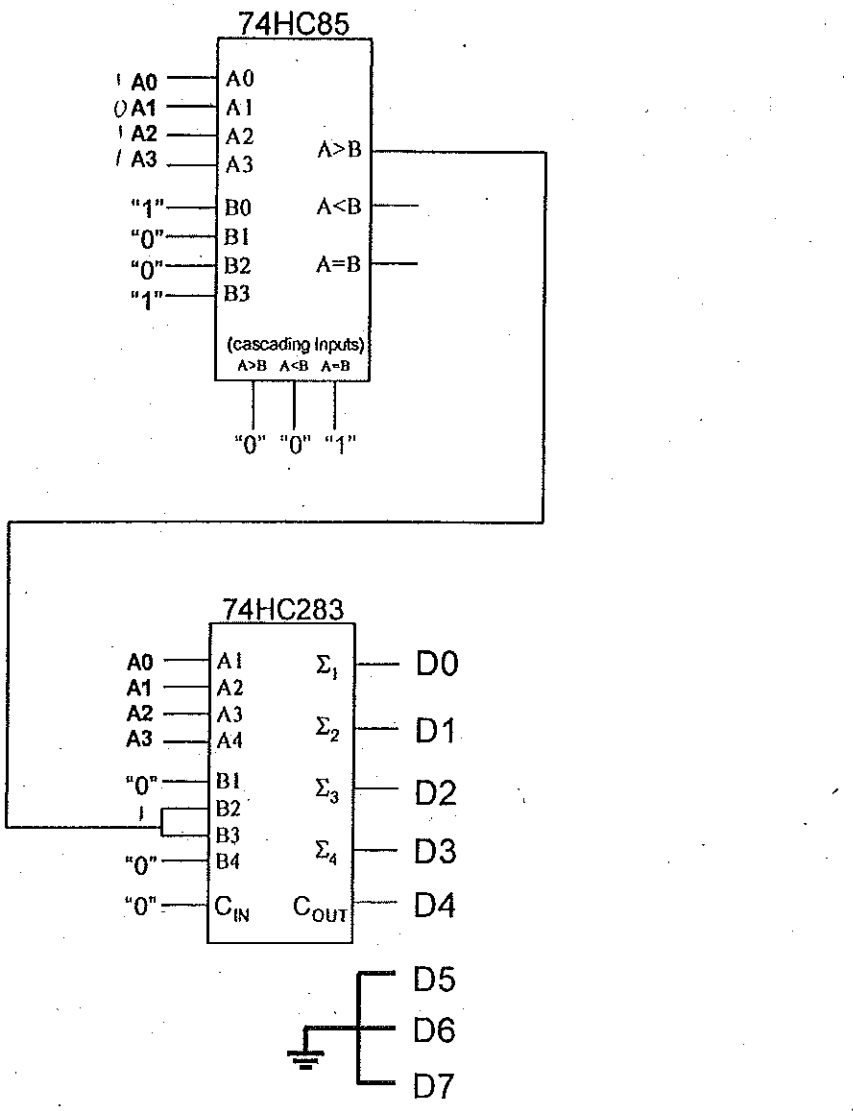
$D2 = \sum_{A3A2A1A0}(4,5,6,7,14,15) = \overline{A3}A2\overline{A1}\overline{A0} + \overline{A3}A2\overline{A1}A0 + \overline{A3}A2A1\overline{A0} + \overline{A3}A2A1A0 + A3A2A1\overline{A0} + A3A2A1A0$

$D1 = \sum_{A3A2A1A0}(2,3,6,7,12,13) = \overline{A3}\overline{A2}A1\overline{A0} + \overline{A3}\overline{A2}A1A0 + \overline{A3}A2A1\overline{A0} + \overline{A3}A2A1A0 + A3A2\overline{A1}\overline{A0} + A3A2\overline{A1}A0$

$D0 = \sum_{A3A2A1A0}(1,3,5,7,9,11,13,15) = A0$



2.3 Implemente ahora el conversor de número binario sin signo de 4 bits a código BCD utilizando exclusivamente un comparador de dos números de 4 bits (74HC85, hoja 10 del examen), y un sumador de dos números de 4 bits (74HC283). (10 puntos)



PROBLEMA 3 (*25 PUNTOS*)

Se desea diseñar un contador binario libre de 3 bits, (**B2 B1 B0**), que permita contar hacia arriba cuando una señal de control **U/D** es '1', o hacia abajo cuando vale '0'.

3.1 Dibuje el diagrama de estados del contador utilizando una máquina de Moore con OCHO estados, cada uno de ellos CODIFICADO con su valor en binario. Indique CLARAMENTE las entradas, las salidas, los estados y su codificación. (5 puntos)
Ejemplo: Número 3 = estado (011) = salida (011).

— SOLUCION —

Departamento de Ingeniería Electrónica
E.T.S.I. Telecomunicación. U.P.M.
EXAMEN DE CIRCUITOS ELECTRÓNICOS DIGITALES
6 de Febrero de 2004 **Duración: 2:30 horas**

Apellidos _____

Nombre _____ DNI/PAS: _____

Fecha publicación de calificaciones: **23 de Febrero de 2004**
Fecha límite solicitud de revisión (en el B-042): **27 de Febrero de 2004**
Fecha de revisión: **1 de Marzo de 2004**

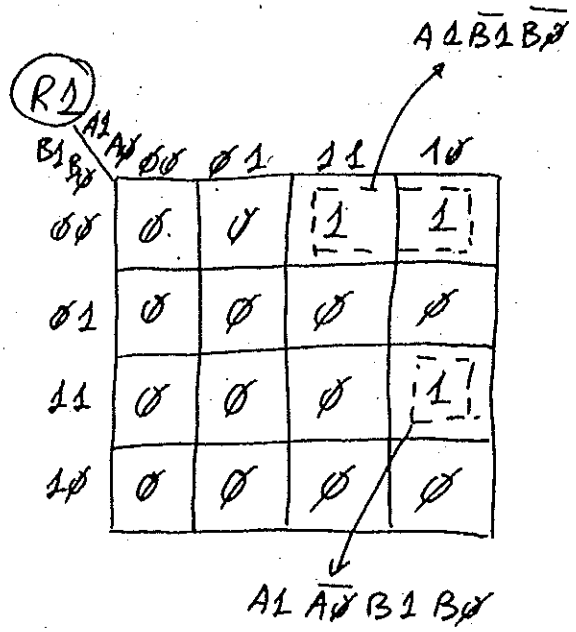
PROBLEMA 1

Se desea diseñar un circuito capaz de dividir dos números naturales de dos bits cada uno **[A1,A0]** entre **[B1,B0]** para obtener el cociente **[C1,C0]** y el resto **[R1,R0]**. En el caso en los que el divisor **[B1,B0]** sea cero, el cociente y el resto serán iguales al dividendo **[A1,A0]** y se activará una señal de error **E**.

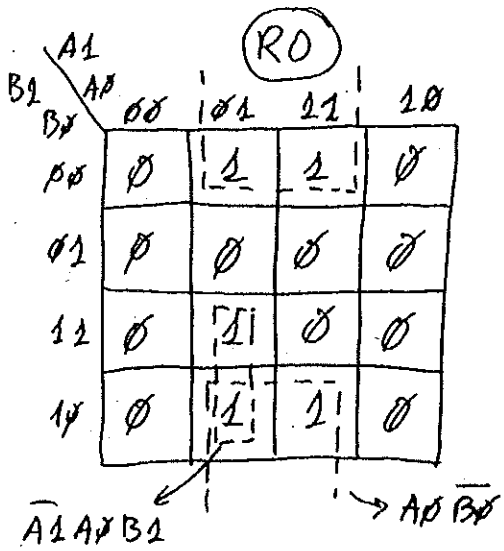
1.1 Rellene la tabla de verdad *Tabla 1* y obtenga justificadamente las funciones lógicas simplificadas para **R1**, **R0** y **E**. *(5 puntos)*

A1	A0	B1	B0	C1	C0	R1	R0	E	AUX
0	0	0	0	Ø	Ø	Ø	Ø	1	0
0	0	0	1	Ø	Ø	Ø	Ø	Ø	0
0	0	1	0	Ø	Ø	Ø	Ø	Ø	0
0	0	1	1	Ø	Ø	Ø	Ø	Ø	0
0	1	0	0	Ø	1	Ø	1	1	1
0	1	0	1	Ø	1	Ø	Ø	Ø	0
0	1	1	0	Ø	Ø	Ø	1	Ø	1
0	1	1	1	Ø	Ø	Ø	1	Ø	1
1	0	0	0	1	Ø	1	Ø	1	0
1	0	0	1	1	Ø	Ø	Ø	Ø	0
1	0	1	0	Ø	1	Ø	Ø	Ø	0
1	0	1	1	Ø	Ø	1	Ø	Ø	0
1	1	0	0	1	1	1	1	1	1
1	1	0	1	1	1	Ø	Ø	Ø	0
1	1	1	0	Ø	1	Ø	1	Ø	1
1	1	1	1	Ø	1	Ø	Ø	Ø	0

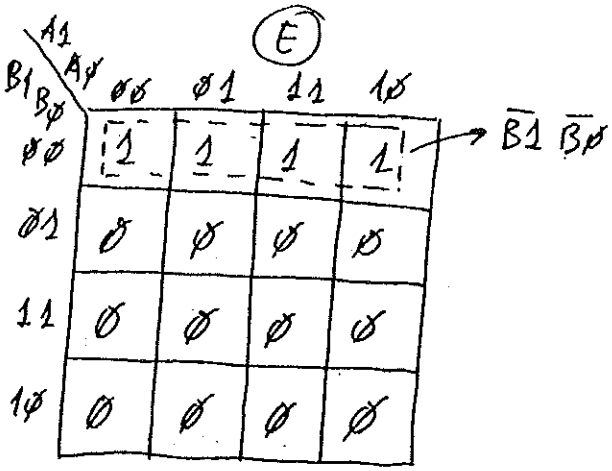
Tabla 1



$$R1 = A1\,\bar{B}1\,\bar{B}\emptyset + A1\,\bar{A}\emptyset\,B1\,B\emptyset$$



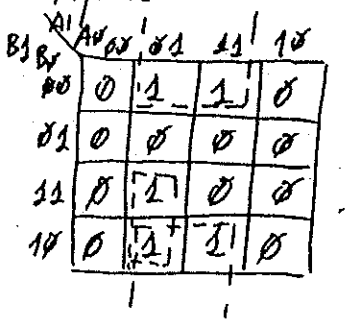
$$RO = A\emptyset\,\bar{B}\emptyset + \bar{A}1\,A\emptyset\,B1$$



$$E = \bar{B}1\,\bar{B}\emptyset$$

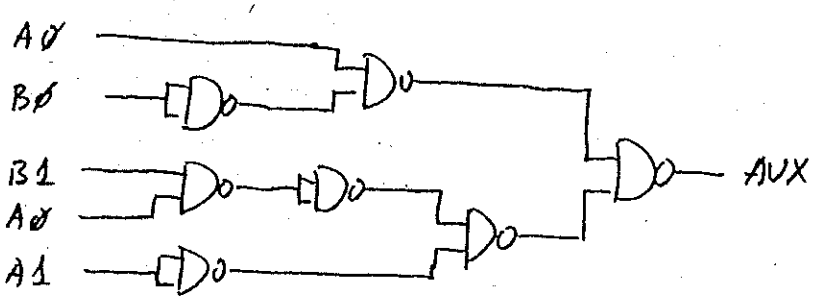
1.2 Implemente la función **AUX** de la *Tabla 1* con el mínimo número de puertas **NAND** de dos entradas. **Nota:** Sólo se dispone de las entradas **A1, A0, B1** y **B0** y no de sus complementarios. *(5 puntos)*

Aplicando Karnaugh directamente:



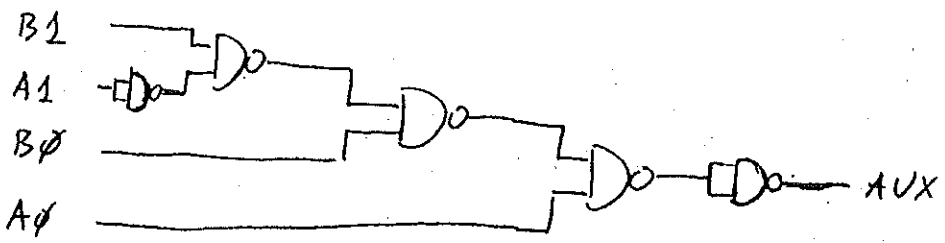
$AUX = A\emptyset\,\bar{B}\emptyset + \bar{A}1\,A\emptyset\,B1$, negando dos veces:

$$AUX = \overline{\overline{A\emptyset\,\bar{B}\emptyset} \cdot \overline{\bar{A}1 \cdot A\emptyset \cdot B1}}$$

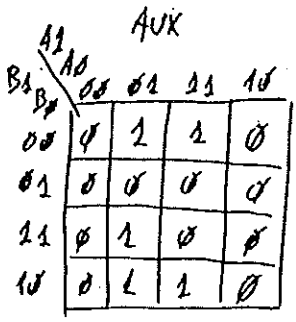


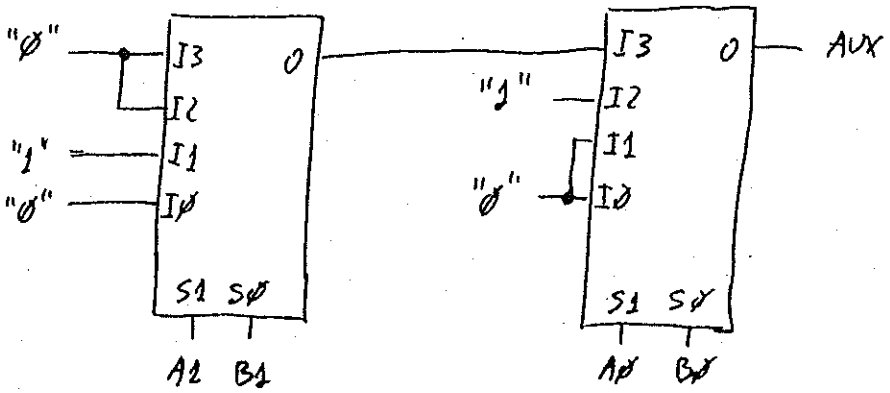
Si realizamos la simplificación: $AUX = A\emptyset\,(\bar{B}\emptyset + \bar{A}1\,B1)$

$$AUX = A\emptyset \cdot (\overline{B\emptyset \cdot \overline{\bar{A}1\,B1}})$$

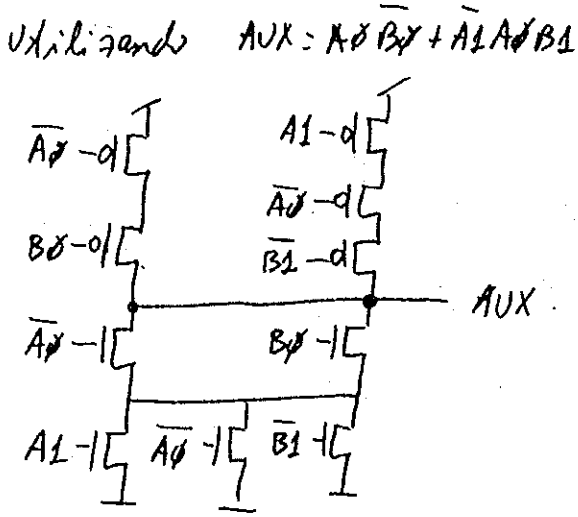


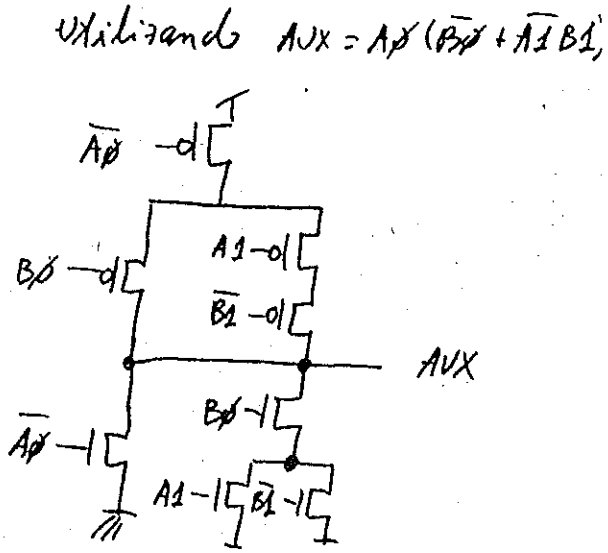
1.3 Implemente la función **AUX** de la *Tabla 1* con dos multiplexores de 4 entradas de datos. **Nota:** Sólo se dispone de las entradas **A1, A0, B1** y **B0** y no de sus complementarios. *(10 puntos)*





1.4 Implemente la función **AUX** de la *Tabla 1* en tecnología CMOS utilizando el mínimo número de transistores. En este caso supondremos que se dispone de las entradas **A1, A0, B1** y **B0** negadas y sin negar. *(10 puntos)*





PROBLEMA 2

Se pretende diseñar un dispositivo luminoso consistente en una hilera de luces (*Figura 1*), tipo LED, de las cuales sólo una se encuentra encendida en cada momento dado.

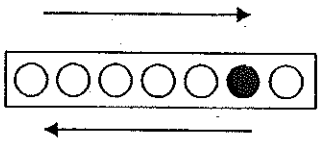


Figura 1: Hilera de luces

PROBLEMA 3 (35 PUNTOS)

Queremos realizar un visualizador de números romanos (encontrará la tabla de números romanos al final de este examen) comprendidos entre el 1 y el 15 a partir de una palabra binaria de cuatro bits ($D_3D_2D_1D_0$), siendo D_3 el más significativo y entendiendo que el valor de cero tiene como equivalente el visualizador apagado.

Para ello dispondremos de cuatro elementos C_4, C_3, C_2 y C_1 como los de la **figura 4** que formarán la cifra romana. En cada uno de ellos colocaremos tres neones, uno para cada símbolo romano "I", "V" y "X" que se controlarán con tres señales activas a nivel alto I_i, V_i y X_i, respectivamente (por ejemplo, si I_2= '1', se encenderá el neón correspondiente a la cifra romana "I" del elemento C_2, y así sucesivamente). La cifra romana completa se alineará **a la derecha**, dejando en blanco los elementos no usados en la parte izquierda del visualizador.

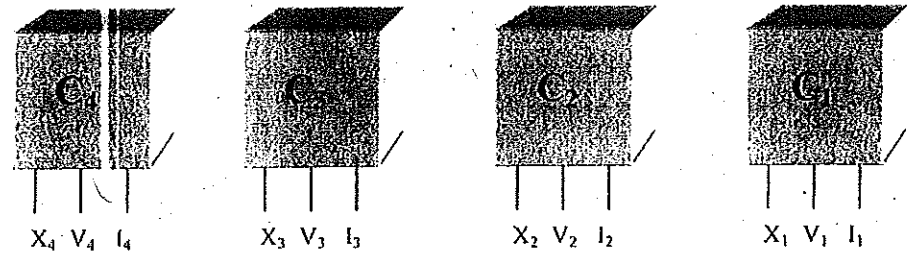


Figura 4

Para gobernar el visualizador deberemos realizar un decodificador de binario ($D_3D_2D_1D_0$) a las señales que atacan a dicho visualizador (X_4, V_4, I_4, X_3, V_3, I_3, X_2, V_2, I_2, X_1, V_1, I_1).

1. Obtenga la tabla de verdad correspondiente a I_2, rellenando la **tabla 2** (5 puntos)

D_3	D_2	D_1	D_0	I_2
0	0	0	0	0
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

Tabla 2

D_1D_0 \ D_3D_2	00	01	11	10
00	0	1	1	1
01	0	0	1	1
11	1	1	0	0
10	1	0	1	0

Tabla 3

D_1D_0 \ D_3D_2	00	01	11	10
00	0	1	1	1
01	0	0	1	1
11	1	1	0	0
10	1	0	1	0

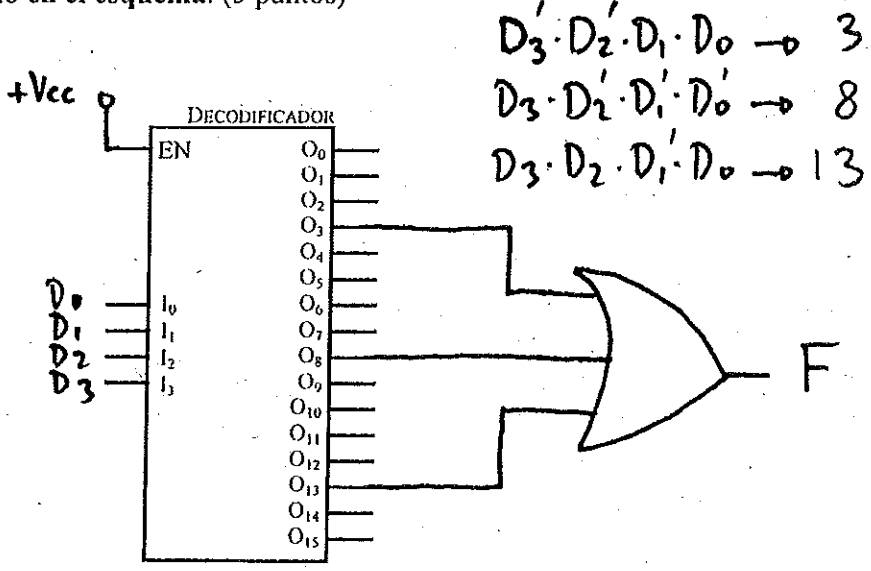
Tabla 4

2. Obtenga la expresión simplificada de I_2 usando el método de *Karnaugh*. Realice la simplificación por "unos" gráficamente sobre la **tabla 3** y por "ceros" sobre la **tabla 4**. Escriba a continuación las expresiones correspondientes. (10 puntos)

$F_{por\ unos} = (D_3 \cdot D_1') + (D_3 \cdot D_2 \cdot D_0') + (D_3' \cdot D_2' \cdot D_1) + (D_3' \cdot D_1 \cdot D_0) + (D_2 \cdot D_1' \cdot D_0')$

$F_{por\ ceros} = (D_3 + D_1 + D_0') \cdot (D_3' + D_1' + D_0') \cdot (D_3' + D_2 + D_1') \cdot (D_3 + D_2' + D_1' + D_0) \cdot (D_3 + D_2 + D_1)$

3. Utilizando el decodificador de cuatro entradas de la **figura 5** (cuya tabla de verdad encontrará al final de este examen), la lógica adicional que crea conveniente y las conexiones necesarias, obtenga razonadamente la función $F = (D_3' \cdot D_2' \cdot D_1 \cdot D_0) + (D_3 \cdot D_2' \cdot D_1' \cdot D_0') + (D_3 \cdot D_2 \cdot D_1' \cdot D_0)$. Considere que representamos una señal S negada como **S'. Por favor, sea limpio en el esquema.** (5 puntos)



$D_3' \cdot D_2' \cdot D_1 \cdot D_0 \rightarrow 3$
$D_3 \cdot D_2' \cdot D_1' \cdot D_0' \rightarrow 8$
$D_3 \cdot D_2 \cdot D_1' \cdot D_0 \rightarrow 13$

Figura 5

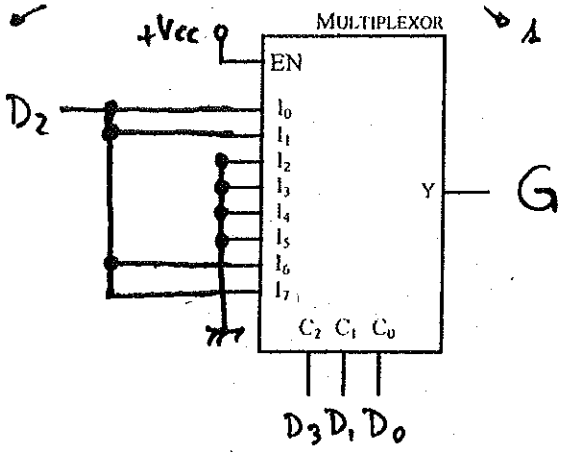
Los minterms asociados a F son el 3, el 8 y el 13, con lo que bastará unir las salidas O_3, O_8, O_{13} con una puerta OR

4. Utilizando el multiplexor de 8 a 1 de la **figura 6** (cuya tabla de verdad encontrará al final de este examen) y las conexiones necesarias, obtenga razonadamente la función $G = (D_3' \cdot D_2 \cdot D_1' \cdot D_0') + (D_3' \cdot D_2 \cdot D_1 \cdot D_0') + (D_3 \cdot D_2 \cdot D_1 \cdot D_0') + (D_3 \cdot D_2 \cdot D_1 \cdot D_0)$. Para ello **sólo** dispone de las señales D_3 a D_0 afirmadas (**no las negadas**) y no se puede utilizar **ninguna** puerta lógica adicional. Considere que representamos una señal S negada como S'. **Por favor, sea limpio en el esquema.** (10 puntos)

Al no poder usar puertas lógicas, usaremos D_2 para atacar las entradas del MUX y D_3, D_1, D_0 para las de control

$\Rightarrow G = D_2 \cdot (D_3' \cdot D_1' \cdot D_0') + D_2 \cdot (D_3' \cdot D_1' \cdot D_0) + D_2(D_3 \cdot D_1 \cdot D_0') + D_2 (D_3 \cdot D_1 \cdot [$

$\emptyset$ $\quad$ $\searrow 1$ $\quad$ $\searrow 6$ $\quad$ $\searrow 7$



Así, D_2 actuará en las entradas $\emptyset$, 1, 6 y 7. El resto deben conectarse a masa

Figura 6

También podría simplificarse G, llegando a métodos de implementación más eficientes.

5. Indique qué señales de entre las que atacan al visualizador de números romanos (X_4, V_4, I_4, X_3, V_3, I_3, X_2, V_2, I_2, X_1, V_1, I_1) son implementadas por las funciones F y G de los apartados 3 y 4 respectivamente. (5 puntos)

F = I_3 $\qquad$ G = V_1

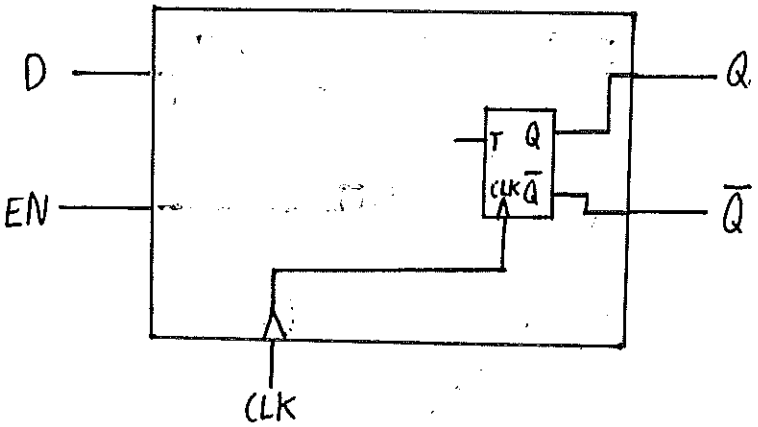
PROBLEMA 3

3.1 Implemente un biestable tipo **D** con "**ENABLE**" usando un biestable tipo **T** y la lógica combinacional que crea necesaria. Se valorará el grado de simplicidad. *(5 puntos)*

Ver ejercicio 1 de clase Tema 4.

EN	D	Q_t	Q_{t+1}	T
0	0	0	0	0
0	0	1	1	0
0	1	0	0	0
0	1	1	1	0
1	0	0	0	0
1	0	1	0	1
1	1	0	1	1
1	1	1	1	0

Suponemos Enable activo a nivel alto.



Q \ EN,D	00	01	11	10
0	0	0	1	0
1	0	0	0	1

$T = EN.\bar{Q}.D + Q.EN.\bar{D} = EN(\bar{Q}D + Q\bar{D}) = \underline{EN(Q \oplus D)}$

3.2 Obtenga justificadamente la máxima frecuencia del reloj **CLK** del circuito de la *Figura 8* que asegure un correcto funcionamiento del mismo. Suponga que el ciclo de trabajo del reloj **CLK** es del 50%. *(10 puntos)*

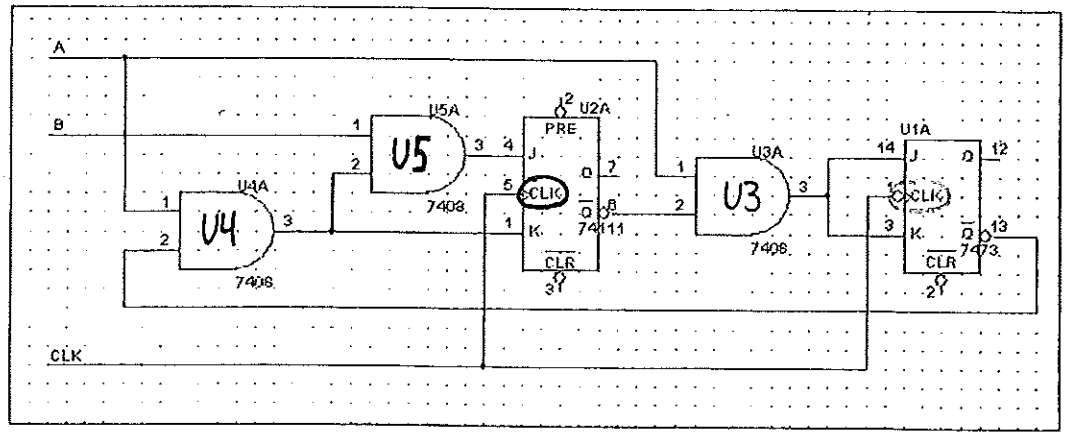


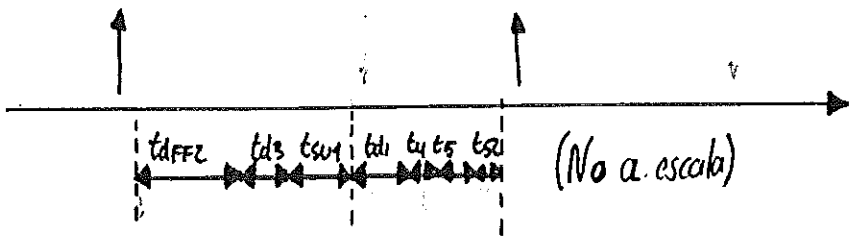
Figura 8

Nota: Considere que las señales **CLR*** y **PRE*** están desactivadas.

Parámetros: $t_{dpuertas} = 1$ ns
$t_{dFF} = 3$ ns
$t_{setup} = 2$ ns

DC = 50%

3.2)



• $t_{1min} = t_{dFF2} + t_{d3} + t_{su1} = \underline{6ns}$

• El camino más largo desde U1 hasta U2 es: U1-U4-U5-U2

$$\underline{t_{0min}} = t_{dFF1} + t_{d4} + t_{d5} + t_{su2} = \underline{7ns}$$

Como hay que garantizar un DC = 50%:

$$\frac{T_{min}}{2} = \max\{t_{0min}, t_{1min}\} = 7ns \Rightarrow T_{min} = 14ns \Rightarrow \underline{f_{max} = 71'4\ MHz}$$

PROBLEMA 5

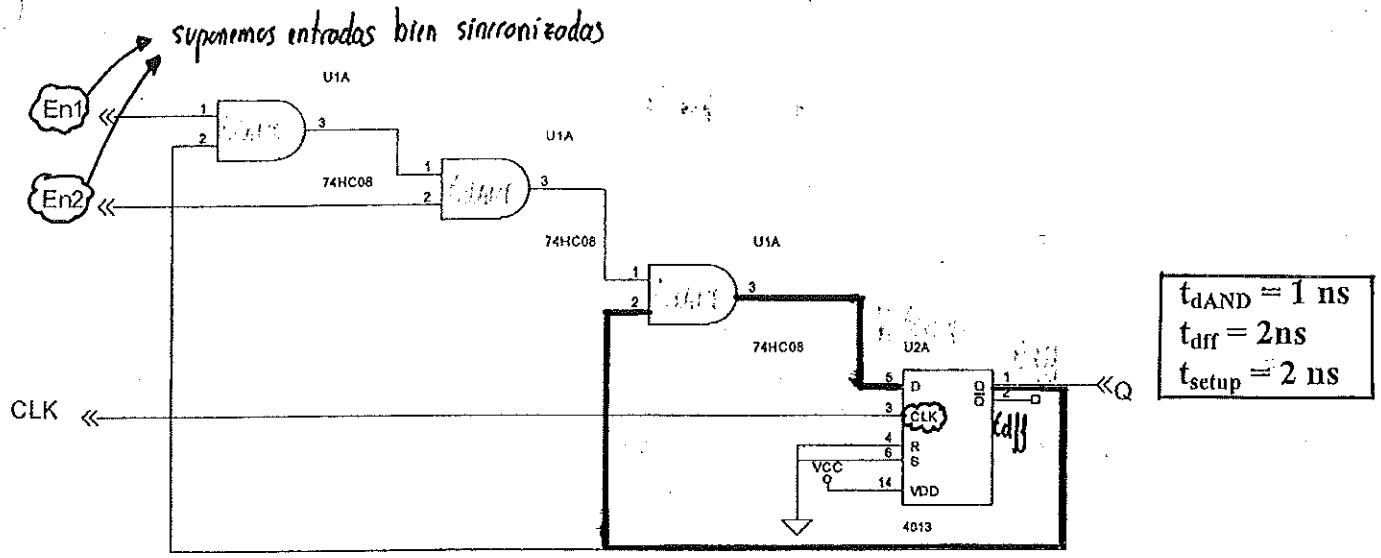


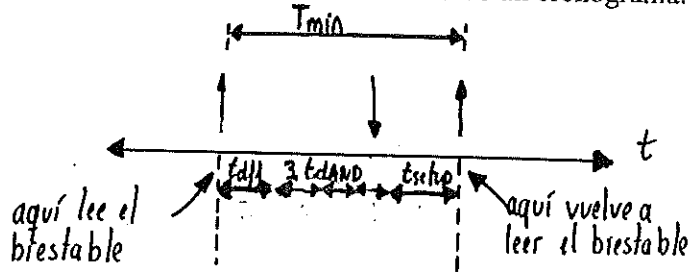
Figura 8

5.1 Obtenga la máxima frecuencia de funcionamiento el circuito de la *Figura 8* suponiendo los valores de temporización que aparecen en el lateral de dicha figura. (*5 puntos*)

5.2 Obtenga justificadamente el máximo tiempo de hold ($t_{holdmax}$) del biestable para un correcto funcionamiento del circuito. (*5 puntos*)

5.3 Obtenga justificadamente la probabilidad de que el circuito de la *Figura 8* no funcione correctamente si utilizamos una frecuencia de reloj **CLK** de 125 MHz, suponiendo que las entradas **En1** y **En2** pueden cambiar aleatoriamente en el tiempo con una distribución uniforme. Indique **claramente** los motivos de fallo sobre un cronograma. (*15 puntos*)

5.1) $f_{CLK_{max}}$? (T_{min})



$T_{min} = t_{dff} + 3t_{dAND} + t_{setup} = 7ns$

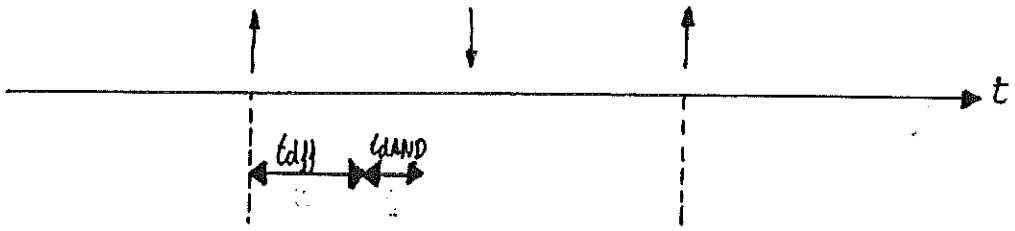
$f_{max} = \frac{1}{T_{min}} = 143\,MHz$

■ El periodo mínimo: tiempo del camino más largo, incluyendo el t_{setup} del biestable de destino.

$T_{min} = t_{prop\_biestable\_partida} + t_{combinacional\_max} + t_{setup\_biestable\_llegada}$

T_{min} = es el mínimo periodo que puede tener el reloj del sistema para darle tiempo a llegar a su destino incluso a la señal "mas lenta"

5.2) ¿t_{hold_max}?



$t_{hold\_max} = t_{dff} + t_{dAND} = 3ns$

t_{hold_max} ≡ es el máximo tiempo que puede necesitar el biestable para leer las entradas porque en ese tiempo ya llega a "molestar" una señal por el camino más corto. Como el biestable necesite más tiempo de hold necesitaríamos sustituirlo por uno mejor (rápido)

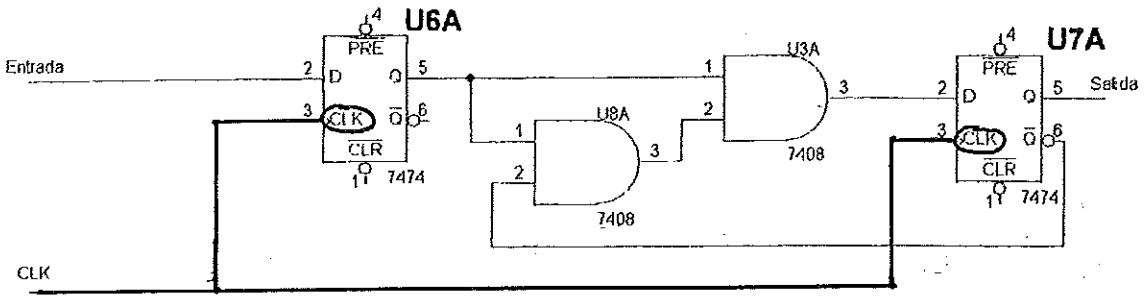
t_{hold_max} = es el tiempo del camino más corto, sin contar el t_{setup} del destino.

$$t_{hold\_max} = t_{prop\_biestable\_partida} + t_{combinacional\_mínimo}$$

FEBRERO 2005

Problema 1

En el circuito de la figura los biestables tipo D, **U6A** y **U7A**, tienen distintos tiempos de propagación (t_p) y de setup (t_{setup}). Considere así mismo que las entradas de PRESET (**PRE**) y CLEAR (**CLR**) están desactivadas en todo momento.

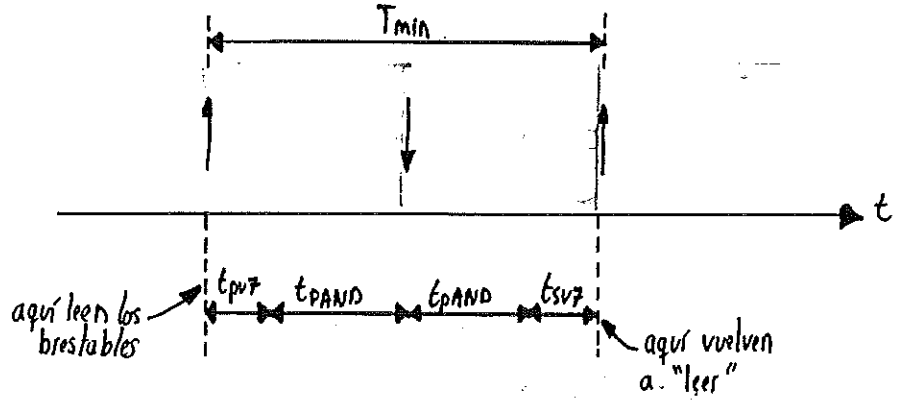


	t_p	t_{setup}
Biestable U6A	1 ns	1 ns
Biestable U7A	2 ns	2 ns
Puerta AND	5 ns	

1.1 Utilizando los valores de la tabla obtenga **justificadamente,** dibujando un cronograma, la máxima frecuencia del reloj (f_{CLKmax}) del circuito que asegure un correcto funcionamiento del mismo.

1.2 Determinar **justificadamente,** también sobre un cronograma, el máximo tiempo de *hold* ($t_{holdmax}$) que el biestable **U7A** puede tener para que el circuito funcione correctamente.

1.1) ¿f_{max}? (T_{min})



*Comparación de caminos:

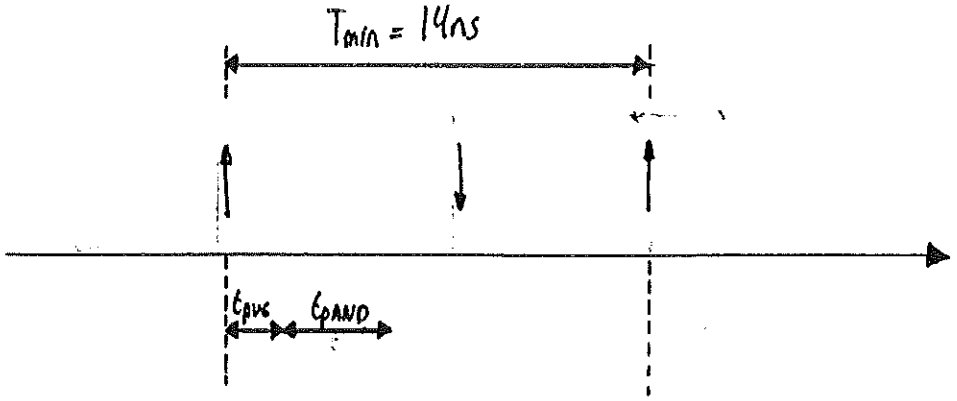
U6 - U3 - U7 · $t_1 = t_{pU6} + t_{pAND} + t_{sU7} = 8\ ns$

U6 - U8 - U3 - U7: $t_2 = t_{pU6} + 2t_{pAND} + t_{sU7} = 13\ ns$

U7 - U8 - U3 - U7: $t_3 = t_{pU7} + 2t_{pAND} + t_{sU7} = 14\ ns$

$T_{min} \equiv \max\{t_1, t_2, t_3\} = 14\ ns \Rightarrow f_{max} = \frac{1}{T_{min}} = 71'4\ MHz$

1.2) ¿t_{hold_max} del U7?



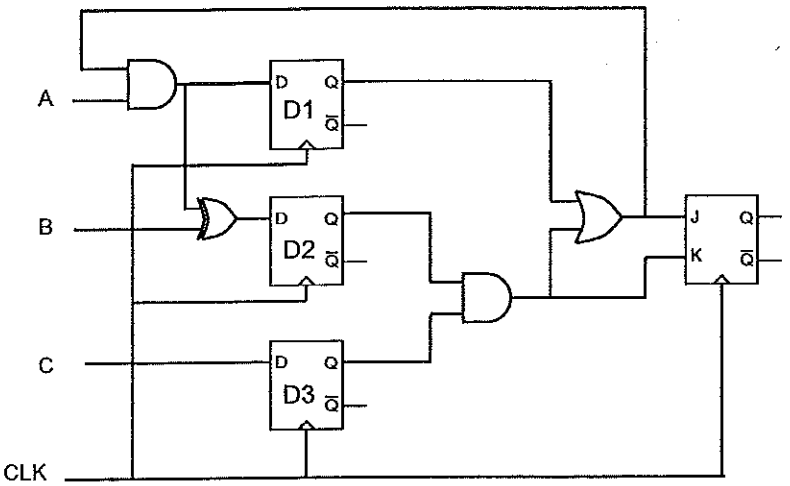
Comparación de tiempos: $t_i' = t_i - t_{setup}$:

$$\left.\begin{aligned} t_1' &= 6ns \\ t_2' &= 11ns \\ t_3' &= 12ns \end{aligned}\right\} \quad t_{hold\_max} = \min\{t_1', t_2', t_3'\} = 6ns$$

FEBRERO 2008

PROBLEMA 5

Considere el siguiente circuito y los datos que se indican al lado.



	t_p	t_{setup}
Biestable D1	1 ns	1 ns
Biestable D2	2 ns	1.5 ns
Biestable D3	2 ns	0.5 ns
Biestable J-K	2 ns	1 ns
Puerta AND	1.5 ns	
Puerta OR	1 ns	
Puerta XOR	1.5 ns	

Figura 5

5.1 Utilizando los valores de la tabla obtenga **justificadamente**, dibujando un cronograma, la máxima frecuencia del reloj (f_{CLKmax}) del circuito de la figura 5 que asegure un correcto funcionamiento del mismo. Exprese claramente los tiempos de todos los caminos que considere. *(5 puntos)*

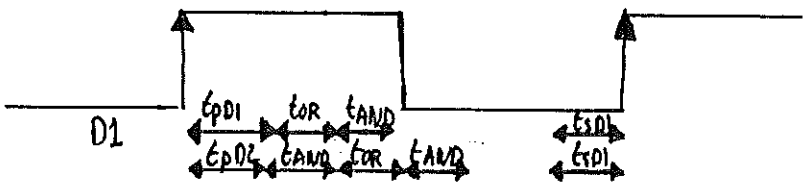
5.2. Determinar **justificadamente**, sobre un cronograma, el máximo tiempo de *hold* ($t_{holdmax}$) que puede tener el biestable D1 y el J-K para que el circuito de la figura 5 funcione correctamente (*5 puntos*)

5.1)
D1 { D1 - OR - AND - D1
D2 - AND - OR - AND - D1
D3 - AND - OR - AND - D1

D2 { D1 - OR - AND - XOR - D2
D2 - AND - OR - AND - XOR - D2
D3 - AND - OR - AND - XOR - D2

J-K { D1 - OR - JK
D2 - AND - OR - JK
D3 - AND - OR - JK

NOTA IMPORTANTE: En clase sobre un cronograma debemos poner todos los caminos no solo vale con ponerlo como en la izquierda sino sobre el cronograma poner dichos caminos:



T_{min} = D3 - AND - OR - AND - XOR - D2 = 9 ns

$f_{max} = \frac{1}{T_{min}} = 111'1$ MHz

5.2) Una vez hecho el apartado 5.1) sobre el mismo cronograma podemos ver directamente desde ahí los caminos que nos interesan para calcular el tiempo de hold, es decir, el menor de los tiempos que llegan a un biestable sin contar el tiempo de setup.

- Tiempo hold D1: $T_{hold_{D1}} \leq t_{pD1} + t_{OR} + t_{AND}$

$$t_{hold_{D1}} \leq 3'5\,ns$$

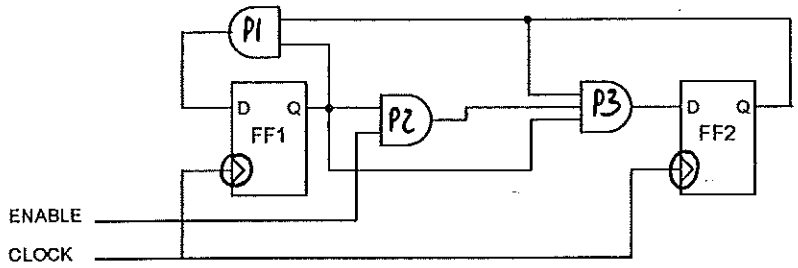
- Tiempo hold J-K:: $T_{hold_{J-K}} \leq t_{pD1} + t_{OR}$

$$t_{hold_{J-K}} \leq 2\,ns$$

SEPTIEMBRE 2007

PROBLEMA 3

Considere el siguiente circuito y los datos que se indican debajo:



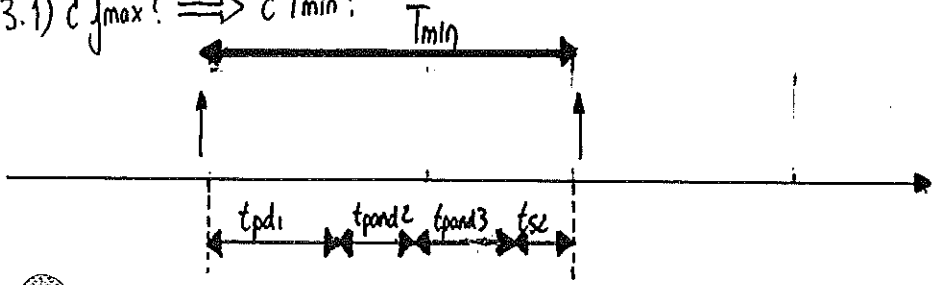
	t_p	t_{setup}
Biestable tipo-D	2 ns	1 ns
Puerta AND 2 entradas	1 ns	
Puerta AND 3 entradas	1,5 ns	

3.1 Utilizando los valores de la tabla obtenga **justificadamente**, dibujando un cronograma, la máxima frecuencia del reloj (f_{CLKmax}) del circuito de la figura que asegure un correcto funcionamiento del mismo. *(5 puntos)*

3.2 Determinar **justificadamente**, sobre un cronograma, el máximo tiempo de *hold* ($t_{holdmax}$) que puede tener el biestable FF1 para que el circuito de la figura funcione correctamente. *(5 puntos)*

3.3 Determinar **justificadamente**, sobre un cronograma, el máximo tiempo de *hold* ($t_{holdmax}$) que puede tener el biestable FF2 para que el circuito de la figura funcione correctamente. *(5 puntos)*

3.1) ¿ f_{max} ? ⟹ ¿ T_{min} ?



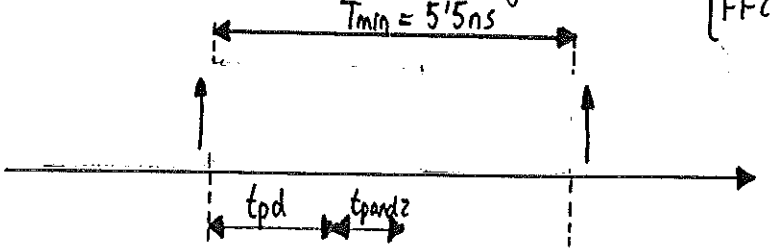
El camino más largo desde un biestable hasta un biestable es el que pasa por FF1-P2-P3-FF2.
El periodo mínimo es:

$T_{min} = t_{pd1} + t_{pond2} + t_{pond3} + t_{sz} = 5'5ns$

$f_{max} = \frac{1}{T_{min}} = 181'8\ MHz$

3.2) ¿ $t_{holdmax\ FF1}$? Caminos que llegan a FF1 { FF1-P1-FF1 / FF2-P1-FF1

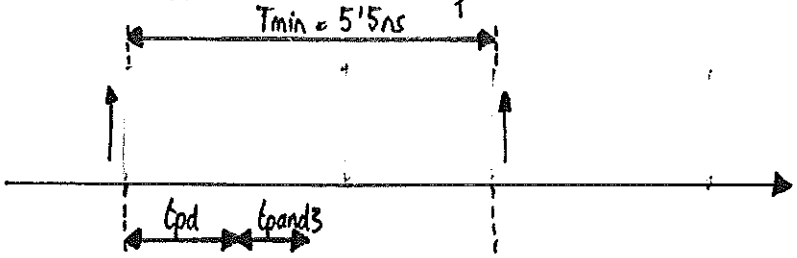
Elegiríamos el tiempo más corto pero en este caso son iguales.



El 2 es porque P1 tiene 2 entradas

$t_{holdmax\ FF1} = t_{pd} + t_{pond2} = 3ns$

3.3) ¿$t_{holdmax\ FF2}$? caminos que terminan en FF2: { FF1 - P3 - FF2 ; FF1 - P2 - P3 - FF2 }



$t_{holdmaxFF2} = t_{pd} + t_{pand3} = 3'5ns$

Apartado extra:

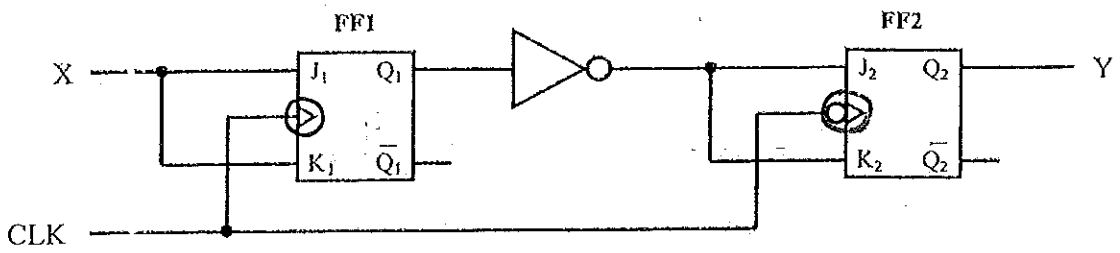
3.4) ¿$t_{holdmáximodelcircuito}$?

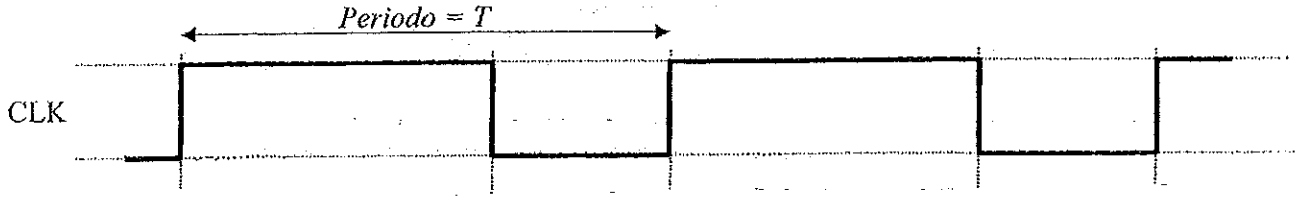
$t_{holdmaxdelcircuito} = \min\{t_{holdmaxFF1}, t_{holdmaxFF2}\} = 3ns$

FEBRERO 2003

PROBLEMA 5 (15 PUNTOS)

1. Obtenga razonadamente la frecuencia máxima de reloj CLK a la que puede funcionar el circuito de la figura suponiendo que el inversor tiene un retardo t_{dinv} = 1ns y los biestables (FF1 Y FF2) un tiempo de propagación t_{dff} = 2ns y un tiempo *setup* t_{setup} = 3ns. Justifique la respuesta dibujando en un cronograma los tiempos pertinentes involucrados y razónela en función de dicha figura. NOTA: Como retricción adicional, suponga que la anchura mínima de los pulsos de la señal de reloj (tanto el de nivel alto como el de nivel bajo) es de 1 ns, y tenga en cuenta que la señal de reloj no tiene por qué tener un ciclo de trabajo del 50%.

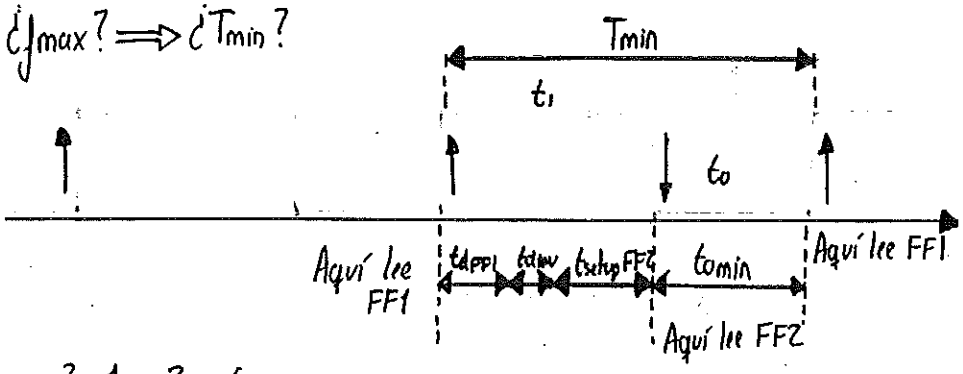




2. Calcule razonadamente el tiempo de hold máximo tholdmax que podrá tener el biestable FF2, para que el circuito funcione correctamente. Asuma las mismas restricciones que las descritas en la nota del apartado anterior y justifique su respuesta a partir del cronograma que ha realizado en el apartado anterior.

t_{min} = 1ns
DC no tiene que ser el 50%

5.1) ¿f_{max}? ⟹ ¿T_{min}?



$t_{1min} = t_{dFF1} + t_{dinv} + t_{setupFF2} = 2 + 1 + 3 = 6ns$

$t_{0min} = 1ns$ (dato) (Porque no hay otra restricción)

$T_{min} = t_{1min} + t_{0min} = 7ns \Longrightarrow f_{max} = \frac{1}{T_{min}} = 143\,MHz$

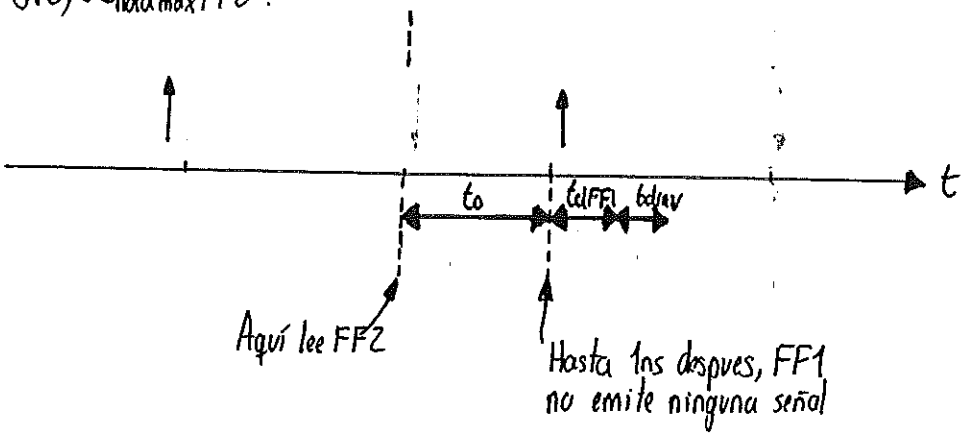
NOTA: Si nos obligasen tener un DC = 50%:

$\frac{T_{min}}{2} = \max\{t_{1min}, t_{0min}\} = \max\{6, 1\} = 6ns$

$T_{min} = 12ns \Longrightarrow f_{max} = \frac{1}{12ns} = 83'3\,MHz$

5.2) ¿$t_{holdmaxFF2}$?

{El origen de tiempos en el flanco activo de el biestable en cuestión !! ⊛⊛



Aquí lee FF2

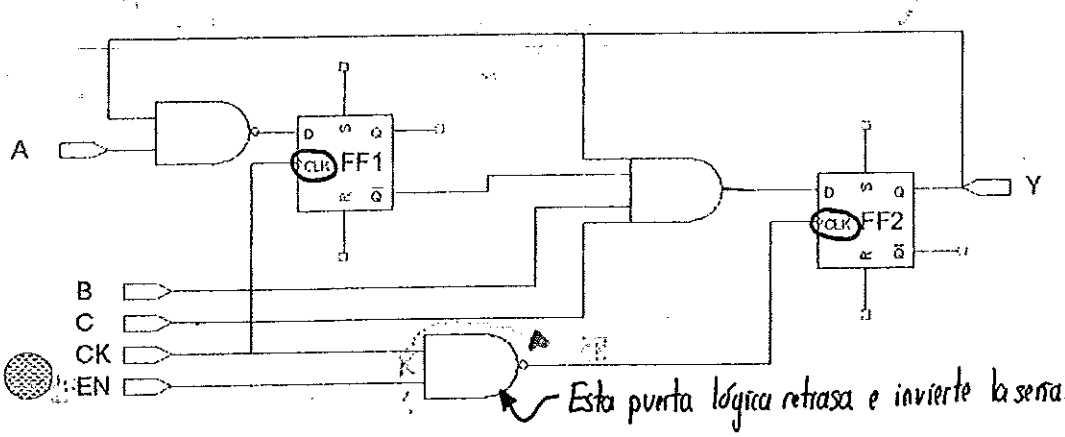
Hasta 1ns despues, FF1 no emite ninguna señal

$$\underline{t_{holdmaxFF2}} = t_o + t_{dFF1} + t_{dinv} = \underline{4ns}$$

SEPTIEMBRE 2005

Problema 1

1 Sobre el circuito de la figura adjunta y suponiendo:
- Que la señal **EN** siempre vale un '1' lógico.
- Que la señal **A** sólo puede cambiar su valor en el flanco de bajada de **CK**
- Que las señales **B** y **C** sólo pueden cambiar su valor en el flanco de subida de **CK**
- Que las señales **S** y **R** de los biestables no están activadas.



PARÁMETROS

$t_{dNAND2} = 1$ ns

$t_{dAND4} = 3$ ns

$t_{dFF1} = t_{dFF2} = 3$ ns

$t_{setup-FF1} = t_{setup-FF2} = 2$ ns

1.1 Obtenga justificadamente, sobre un cronograma, la máxima frecuencia de funcionamiento del mismo.

1.2 Obtenga justificadamente, sobre un cronograma, cuál será el t_{hold} máximo que podrán tener cada biestable (**FF1** y **FF2**) para que el circuito funcione correctamente.

1.3 Suponiendo que la puerta **AND4** del circuito está realizada de forma discreta con el mínimo número de puertas lógicas del tipo **AND2** (de dos entradas) que tienen un $t_{dAND2} = 1$ ns, dibuje justificadamente el circuito que implementa esta función.

1.4 Si ahora la misma puerta **AND4** tuviese un retardo $t_{dAND4} = 2$ ns, dibuje justificadamente el circuito que le correspondería.

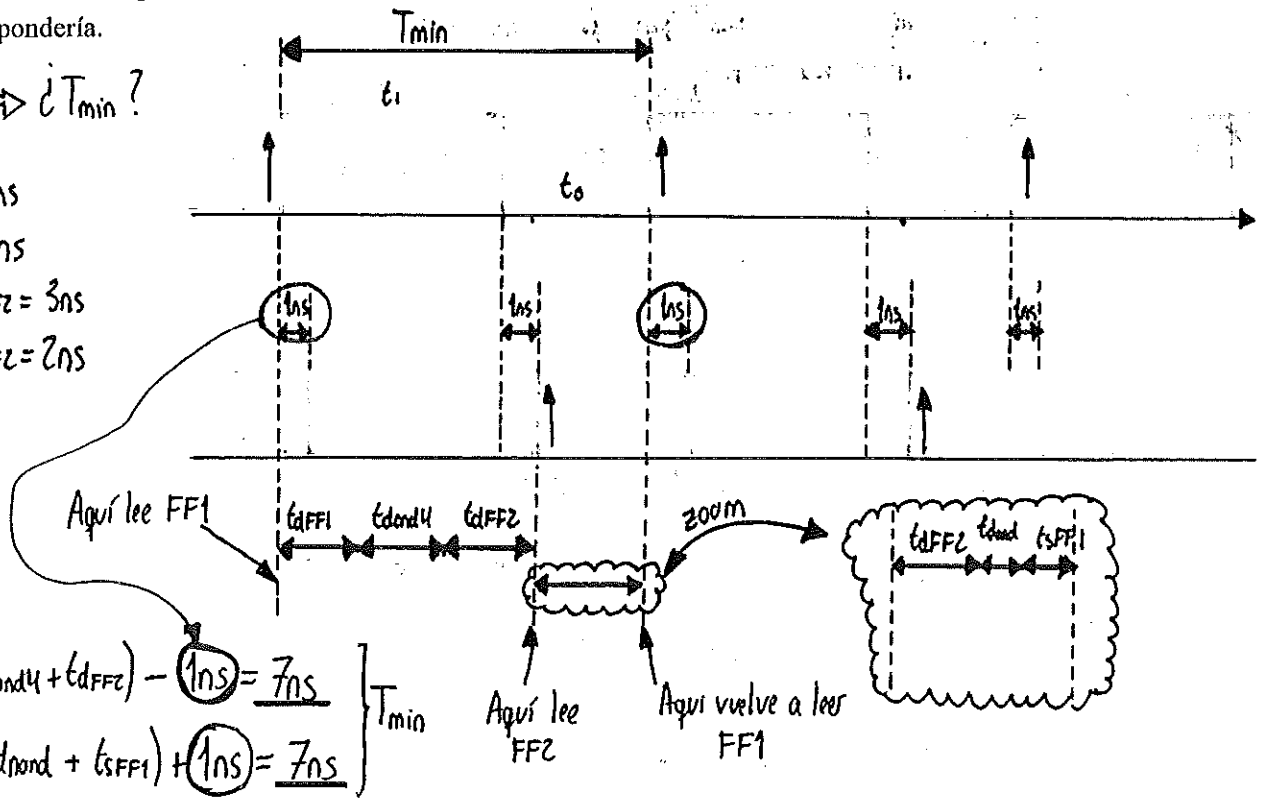
1.1) ¿f_{max}? ⟹ ¿T_{min}?

$t_{dand2} = 1$ns

$t_{dand4} = 3$ns

$t_{dFF1} = t_{dFF2} = 3$ns

$t_{sFF1} = t_{sFF2} = 2$ns



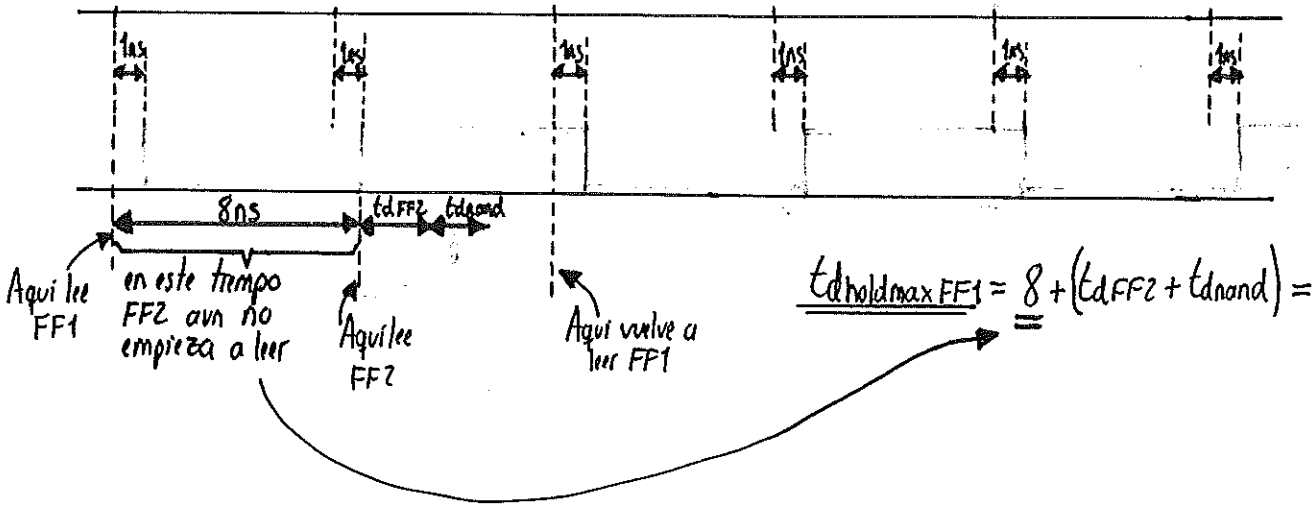
$t_1 = (t_{dFF1} + t_{dand4} + t_{dFF2}) - 1ns = 7ns$

$t_0 = (t_{dFF2} + t_{dand} + t_{sFF1}) + 1ns = 7ns$

T_{min}

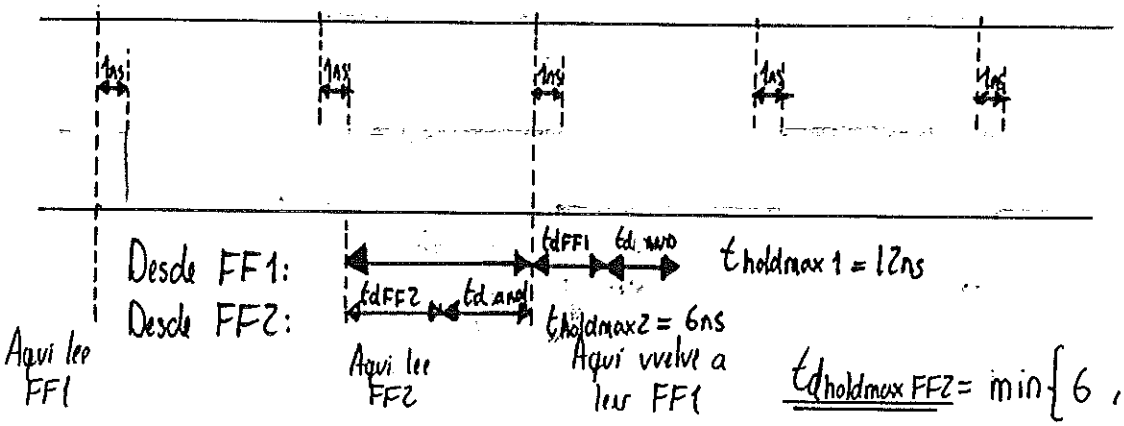
$T_{min} = t_1 + t_0 = 14ns \Longrightarrow f_{max} = \frac{1}{14} = 71'4$ MHz

1.2) ¿$t_{holdmax\ FF1}$?



$$t_{dholdmax\ FF1} = 8 + (t_{dFF2} + t_{dnand}) = 12ns$$

- ¿$t_{holdmax\ FF2}$?



$$t_{dholdmax\ FF2} = \min\{6, 12\} = 6ns$$

FEBRERO 2007

Resuelto

PROBLEMA 3

El circuito de la figura 5 funciona con una señal de reloj **CLK** con ciclo de trabajo del 50 %.

Los dos relojes son activos en flanco de subida

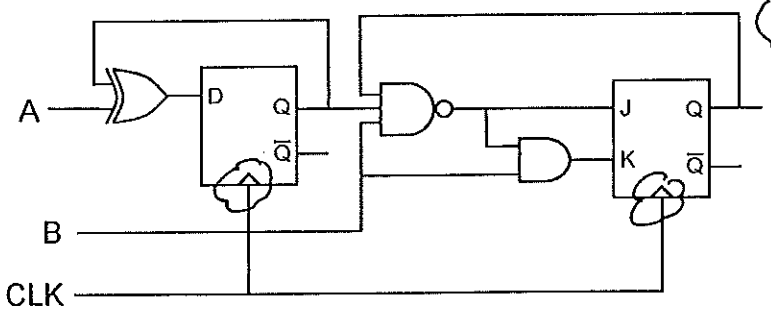


Figura 5

	t_p	t_{setup}
Biestable D	7 ns	1 ns
Biestable J-K	8 ns	2 ns
Puerta AND	3 ns	
Puerta NAND	2 ns	
Puerta XOR	3 ns	

Parámetros de temporización de los componentes del circuito

3.1 Utilizando los valores de la tabla obtenga **justificadamente**, dibujando un cronograma, la máxima frecuencia del reloj (f_{CLKmax}) del circuito de la Figura 5 que asegure un correcto funcionamiento del mismo. *(7 puntos)*

3.2 Determinar **justificadamente**, sobre un cronograma, el máximo tiempo de *hold* ($t_{holdmax}$) que pueden tener cada uno de los biestables (el tipo D y el tipo J-K) para que el circuito de la Figura 5 funcione correctamente. *(8 puntos)*

3.1 ¿ $f_{CLK-max}$?

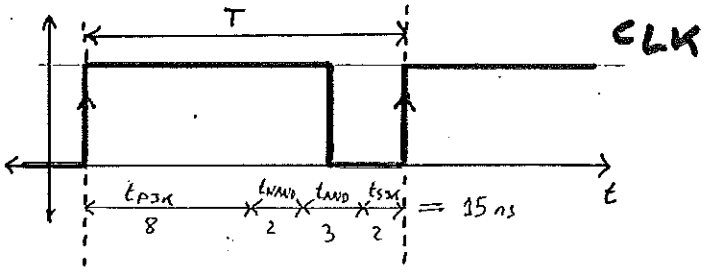
El camino más largo, desde la salida de un biestable, hasta la entrada de otro, es:

JK - NAND - AND - JK Tenéis que comprobarlo!!! Hay hasta 5 caminos distintos!!!

$T_{min} = t_{pJK} + t_{pNAND} + t_{pAND} + t_{sJK} =$

$= 8 + 2 + 3 + 2 = 15\,ns$

$$f_{CLK-max} = \frac{1}{T_{min}} = \frac{1}{15\,ns} = 66'6\,MHz$$



3.2 ¿ $t_{HOLD-max-d}$? ¿ $t_{HOLD-max-JK}$?

☑ BIESTABLE D

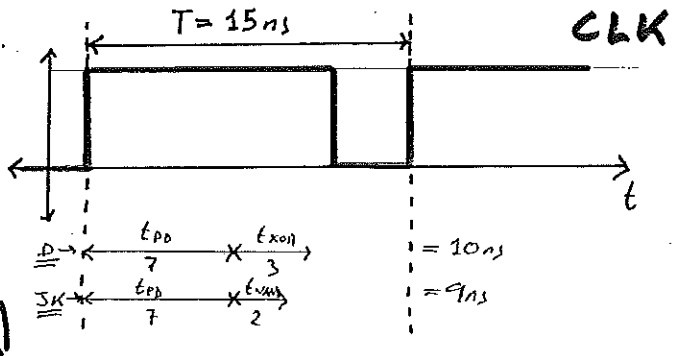
El único camino que ataca a D procede del propio D: $t_{hold-max-d} = t_{pD} + t_{xor} =$

$= 7 + 3 = 10\,ns$

☑ BIESTABLE JK

El camino más corto que ataca al JK es: D - NAND - JK

$t_{HOLD-max-JK} = t_{pd} + t_{nand} = 7 + 2 = 9\,ns$



SEPTIEMBRE 2006

Resuelto

PROBLEMA 3

El circuito de la figura 2 funciona con una señal de reloj **CLK** con periodo T = 30 ns y ciclo de trabajo variable.

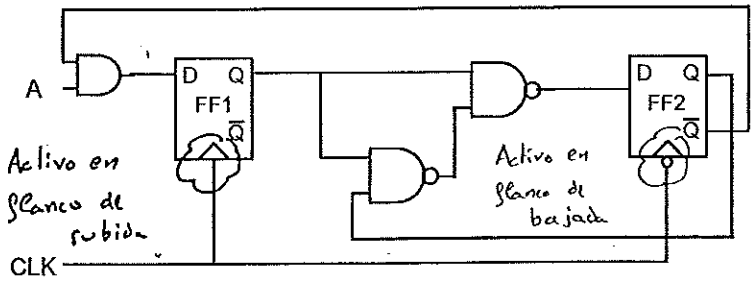
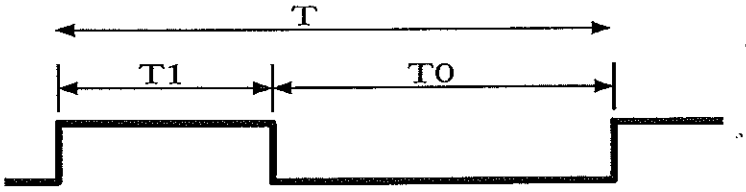


Figura 2

	t_p	t_{setup}
Biestable FF1	7 ns	1 ns
Biestable FF2	8 ns	2 ns
Puerta AND	3 ns	
Puerta NAND	2 ns	

Parámetros de temporización de los componentes del circuito

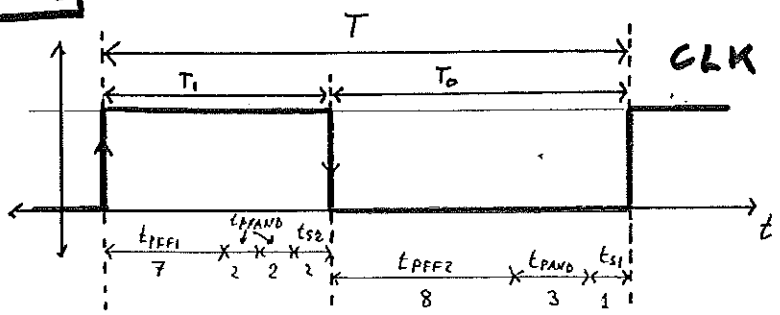
3.1 Si definimos como T1 y T0 los tiempos durante los cuales las señal del reloj **CLK** toma los valores lógicos de 1 y 0 respectivamente, dentro de un periodo T, determinar **justificadamente**, sobre el dibujo de este apartado, los mínimos valores que pueden adquirir T1 y T0 para un correcto funcionamiento del circuito. Obtenga también la máxima frecuencia del reloj **CLK** que garantiza el correcto funcionamiento del circuito para los casos: (a) ciclo de trabajo del reloj del 50 %; (b) ciclo de trabajo del reloj diferente del 50%. *(10 puntos)*



3.2 Con los valores obtenidos en el primer apartado, determine el rango del ciclo de trabajo de la señal de reloj **CLK** que garantiza un correcto funcionamiento del circuito. Si no ha realizado el primer apartado considere los valores de $T1_{min}$= 14 ns y $T0_{min}$= 12 ns. *(5 puntos)*

3.3 Determine **justificadamente**, sobre un cronograma, el máximo tiempo de *hold* que pueden tener los biestables FF1 y FF2 del circuito de la Figura 2 para garantizar un correcto funcionamiento. Considere para este apartado un ciclo de trabajo para la señal de reloj **CLK** del 50 %. *(5 puntos)*

3.1 ¡¡Ojo!! Aquí no usamos el dato de que T = 30 ns.



$T_{1min} = t_{PFF1} + 2 \cdot t_{PNAND} + t_{setup2} =$
$= 7 + 2 \cdot 2 + 2 = 13$ ns

$T_{0min} = t_{PFF2} + t_{PAND} + t_{setup1} =$
$= 8 + 3 + 1 = 12$ ns

[a] DC = 50% $\quad \frac{T_{min}}{2} = \max\{13, 12\} = 13 \Rightarrow T_{min} = 2 \cdot 13 = 26$ ns $\Rightarrow f_{max} = 38'4$ MHz

[b] DC ≠ 50% $\quad T_{min} = 13 + 12 = 25$ ns $\Rightarrow f_{max} = 40$ MHz

3.2 ¡¡OJO!! Aquí ya usamos el dato de que $T = 30\,ns$

Vamos a partir de los valores hallados en el ejercicio 3.1: $T_{0min} = 12\,ns$, $T_{1min} = 13\,ns$

Forzando T_{0min}: $T_0 = T_{0min} = 12\,ns$, $T_1 = 30 - 12 = 18\,ns$ $\Rightarrow$ $DC = \frac{18}{30} = 60\%$

Forzando T_{1min}: $T_1 = T_{1min} = 13\,ns$, $T_0 = 30 - 13 = 17\,ns$ $\Rightarrow$ $DC = \frac{13}{30} = 43'3\%$

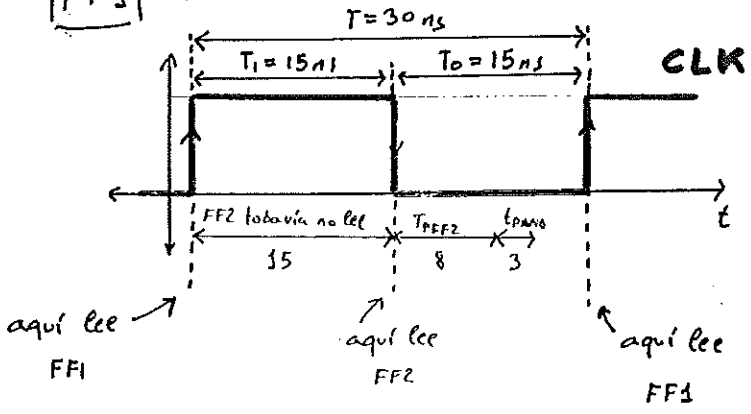
$DC = \frac{T_1}{T}$

Finalmente: $43'3\% \leq$ CICLO DE TRABAJO $\leq 60\%$

3.3 ¿ $t_{HOLD-max-FF1}$? ¿ $t_{HOLD-max\,FF2}$?

Suponemos $DC = 50\%$ y $T = 30\,ns$

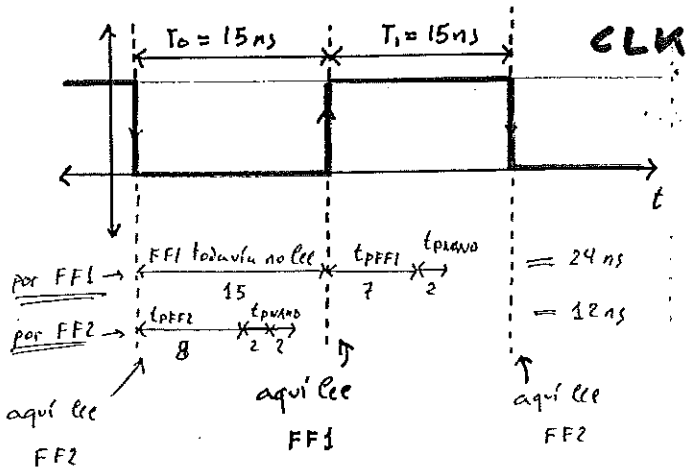
FF1 A FF1 le ataca directamente FF2:



$t_{HOLD-max\,FF1} = T_1 + t_{PFF2} + t_{PAND} =$

$= 15 + 8 + 3 = 26\,ns$

FF2 A FF2 le ataca FF1, pero también el propio FF2:



Nos quedamos con la señal más rápida:

$t_{hold-max\,FF2} = t_{PFF2} + 2 \cdot t_{PNAND} =$

$= 8 + 2 \cdot 2 = 12\,ns$

SEPTIEMBRE 2010 [Lo subirá al twitter]

PROBLEMA 2 (20 *PUNTOS*)

En la Figura 2a, los bloques M1 y M2 representan sumadores completos de dos bits, con acarreo de entrada y salida (según Figura 2b).

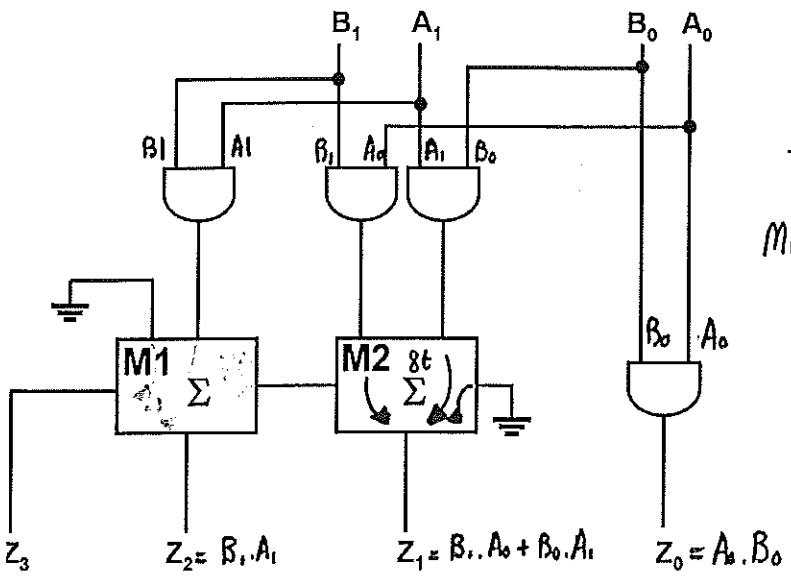


Figura 2a

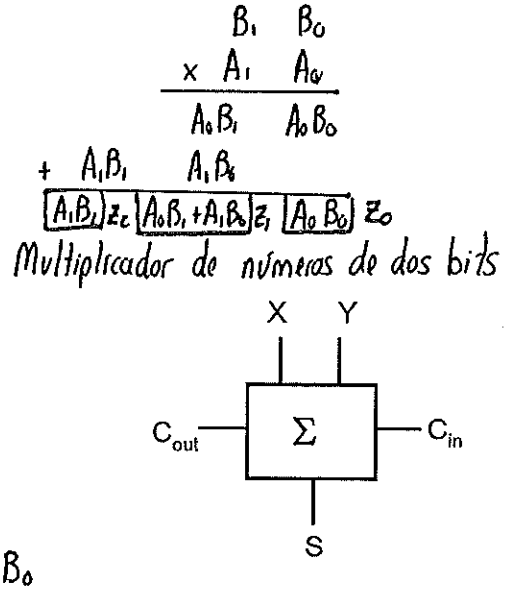


Figura 2b

2.1 Explique razonadamente la función que realiza el circuito de la Figura 2a. *(5 puntos)* Es un multiplicador

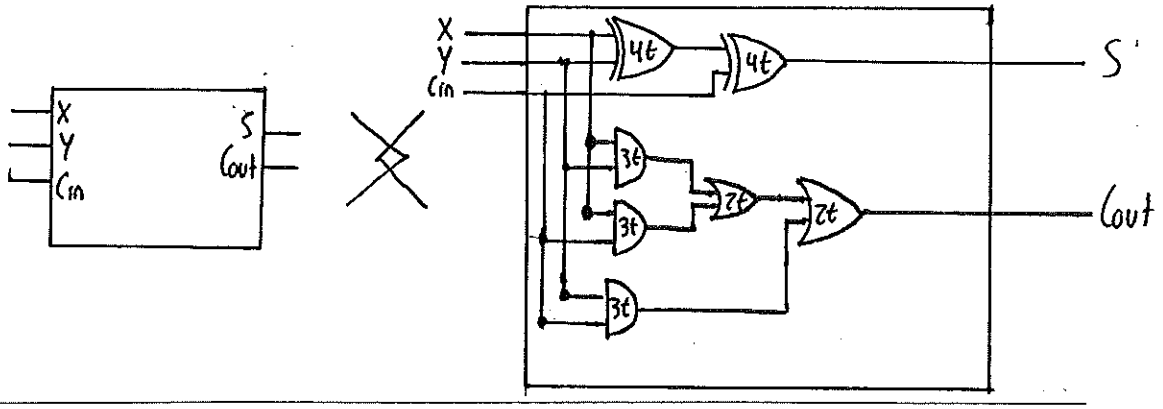
2.2 Suponiendo que las señales de entrada al sumador aparecen sincronizadas, calcule el retardo de las señales S y C_{out} respecto a las señales X, Y y C_{in} del sumador de la Figura 2b, en función de la unidad temporal t. Puede implementar el sumador de la manera que usted desee, valorándose el que el retardo sea el mínimo. Para ello se conocen los siguientes retardos de las diferentes puertas lógicas:

- las puertas AND de 2 entradas tienen un retardo de 3 unidades temporales (3t)
- las puertas OR de 2 entradas tienen un retardo de 2 unidades temporales (2t) y
- las puertas OR exclusivo de 2 entradas tienen un retardo de 4 unidades temporales (4t).

(10 puntos)

2.3 Suponiendo que las señales A_0, A_1 ,B_0 y B_1 de la Figura 2a aparecen a la vez, calcule de manera justificada el retardo de las señales de salida Z_0, Z_1, Z_2, y Z_3 respecto a las anteriores. *(5 puntos)*

2.2) $t_{AND} = 3t$
$t_{OR} = 2t$
$t_{XOR} = 4t$



Buscamos el camino más largo desde las entradas hasta cada salida

- $t_S = t_{XOR} + t_{XOR} = 8t$
- $t_{Cout} = t_{AND} + t_{OR} + t_{OR} = 3t + 2t + 2t = 7t$

PROBLEMA 4 (15 PUNTOS)

4.1 Utilizando lo valores de la *Tabla 1*, y teniendo en cuenta el circuito interno del sumador (Figura 8), obtenga **justificadamente**, dibujando un cronograma, la máxima frecuencia del reloj (f_{CLKmax}) del circuito de la Figura 7 que asegure un correcto funcionamiento del mismo. *(10 puntos)*

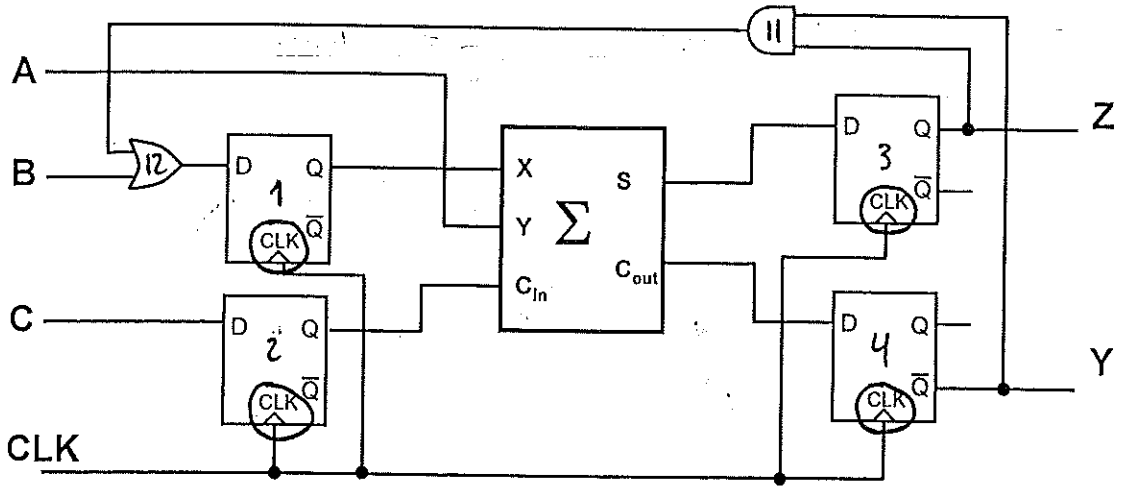
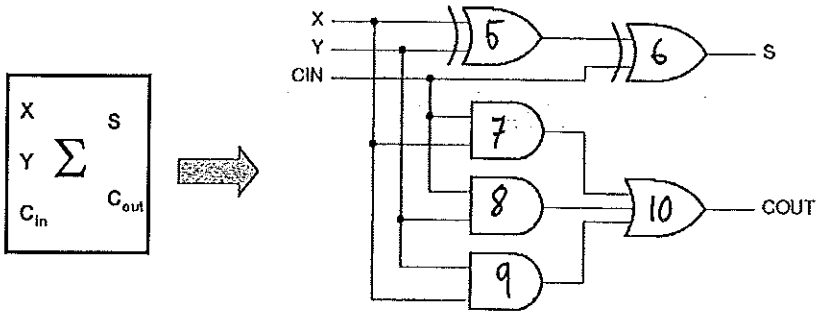


Figura 7



	t_{prop}	t_{setup}
Biestables	1ns	2ns
XOR	3ns	
AND	2ns	
OR	2ns	

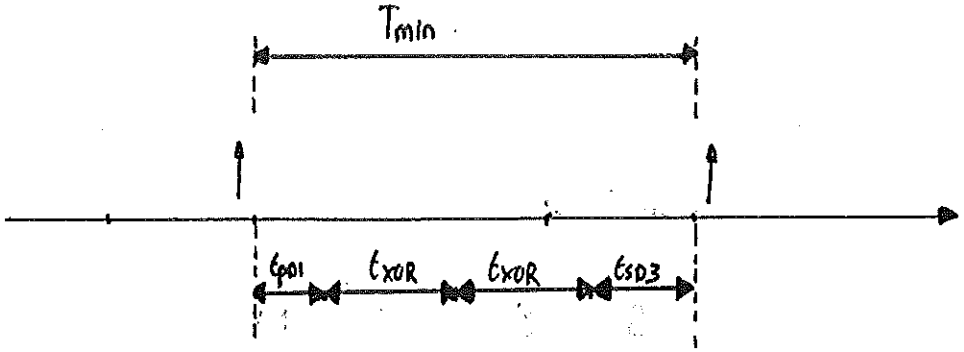
Tabla 1

Figura 8

4.2 Determinar **justificadamente**, también sobre un cronograma, el máximo tiempo de *hold* ($t_{holdmax}$) que los biestables pueden tener para que el circuito funcione correctamente. *(5 puntos)*

4.1) · Caminos de biestable a biestable incluyendo $t_{dFF} + t_{comb} + t_{setup}$:

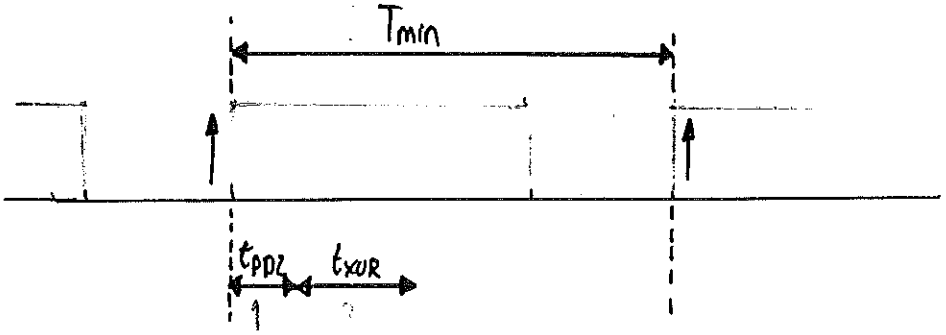
- D1 - XOR5 - XOR6 - D3 : $t_1 = t_{propD1} + 2\,t_{XOR} + t_{SD3} = 9ns$
- D1 - AND967 - OR10 - D4 : $t_2 = t_{PD1} + t_{AND} + t_{OR} + t_{SD4} = 7ns$
- D2 - XOR6 - D3 : $t_3 = t_{PD2} + t_{XOR} + t_{SD3} = 6ns$
- D2 - AND768 - OR10 - D4 : $t_4 = t_{PD2} + t_{AND} + t_{OR} + t_{SD4} = 7ns$
- D3 - AND11 - OR12 - D1 : $t_5 = t_{PD3} + t_{AND} + t_{OR} + t_{SD1} = 7ns$
- D4 - AND11 - OR12 - D1 : $t_6 = t_{PD4} + t_{AND} + t_{OR} + t_{SD1} = 7ns$



$T_{min} = \max\{t_i\} = 9ns \Rightarrow \underline{f_{max} = \frac{1}{T_{min}} = 111\,MHz}$

4.2) ¿$t_{holdmax}$? Caminos t_i' incluyendo $t_{pFF} + t_{comb} + t_{sxor}$ → $\boxed{t_i' = t_i - 2ns}$

$t_1' = 7ns$
$t_2' = 5ns$
$t_3' = 4ns$
$t_4' = 5ns$
$t_5' = 5ns$
$t_6' = 5ns$



$\underline{t_{holdmax} = \min\{t_i'\} = 4ns}$

FEBRERO 2009

PROBLEMA 3 (15 *PUNTOS*)

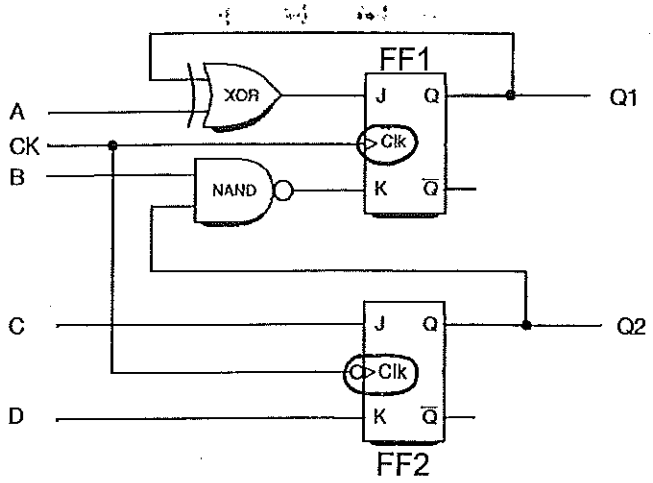


Figura 5 .

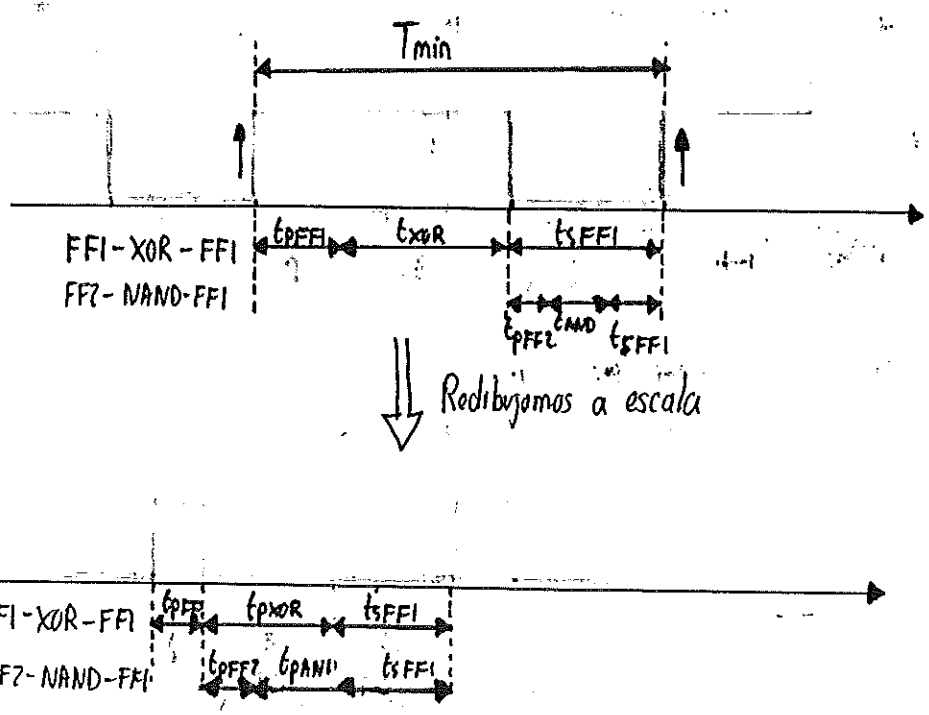
	t_{prop}	t_{setup}
Biestables FF1 y FF2	1ns	2ns
XOR	3ns	
NAND	2ns	

Tabla 1

3.1 Utilizando lo valores de la *Tabla 1* obtenga **justificadamente**, dibujando un cronograma, la máxima frecuencia del reloj (f_{CKmax}) del circuito de la *Figura 5* que asegure un correcto funcionamiento del mismo si: (a) se utiliza un reloj con ciclo de trabajo igual a 50%, y (b) se utiliza un reloj con ciclo de trabajo diferente al 50%, indicándose en este caso el ciclo de trabajo necesario. *(10 puntos)*

3.2 Determinar **justificadamente**, también sobre un cronograma, el máximo tiempo de *hold* ($t_{holdmax}$) que el biestable FF1 puede tener para que el circuito funcione correctamente. *(5 puntos)*

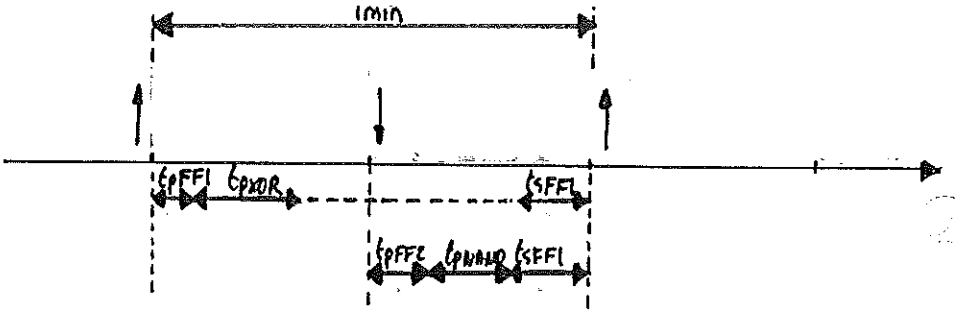
3.1.b) ¿f_{max}? DC ≠ 50%



$T_{min} = t_{pFF1} + t_{pXOR} + t_{sFF1} = 6ns \Rightarrow f_{max} = 167\,MHz$

$t_1 = 1ns$, $t_0 = 5ns \Rightarrow DC = \frac{1\,ns}{6\,ns} = 17\%$

3.1. a) ¿f_{max}? DC = 50%

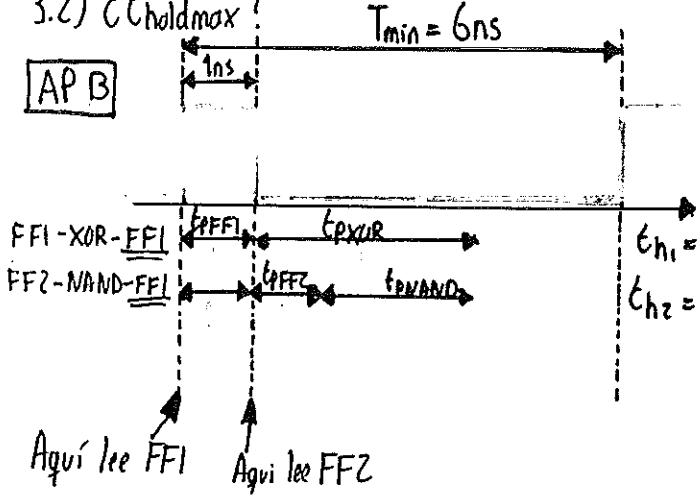


$\frac{T_{min}}{2} = t_{pFF2} + t_{pNAND} + t_{SFF1} = 5ns$

$T_{min} = 5.2 = 10ns \Rightarrow f_{max} = 100\,MHz$

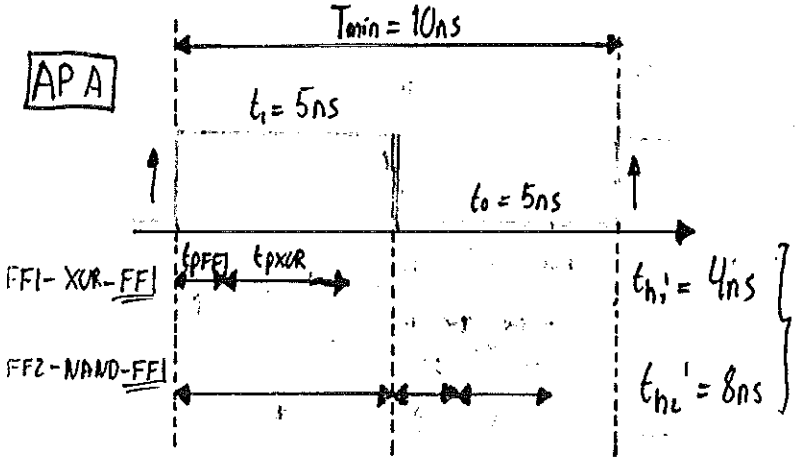
3.2) ¿$t_{holdmax}$?

AP B



$t_{holdmax} = 4ns$

AP A



$t_{holdmax} = \min\{th_i\} = 4ns$

→ Da igual la opción sale lo mismo!!

Problema 4

Se desea que el circuito de la figura 8 funcione a una frecuencia de reloj de 25MHz. Teniendo en cuenta los siguientes parámetros temporales:

t_{su}= 8ns

t_{hold}= 7ns

t_{propFF}=10ns

$t_{propNOR}$=6ns

$t_{propNAND}$=4ns

$T = \frac{1}{f} = 40ns$

determine los instantes de tiempo máximo (t_{max}) y mínimo (t_{min}) relativos al flanco de subida del reloj entre los cuales la señal E_2 podría variar, permitiendo un correcto funcionamiento del circuito. Justifique su respuesta mediante un cronograma. **(10 puntos)**.

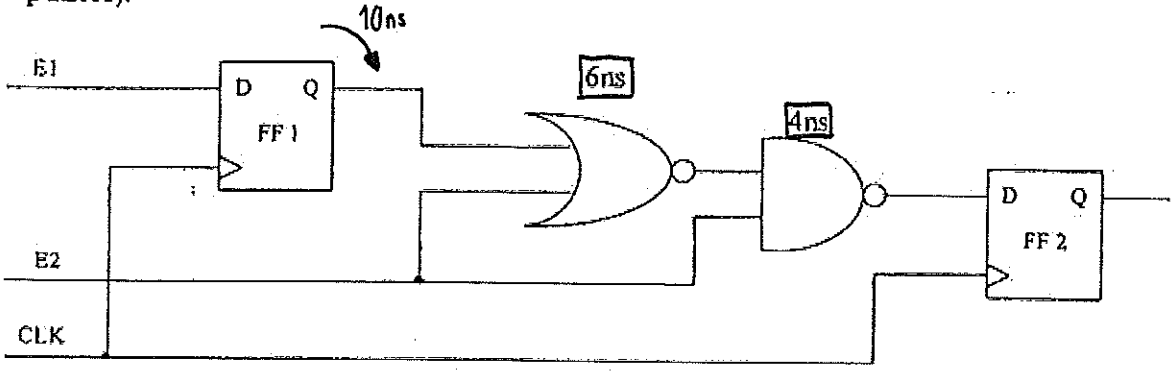
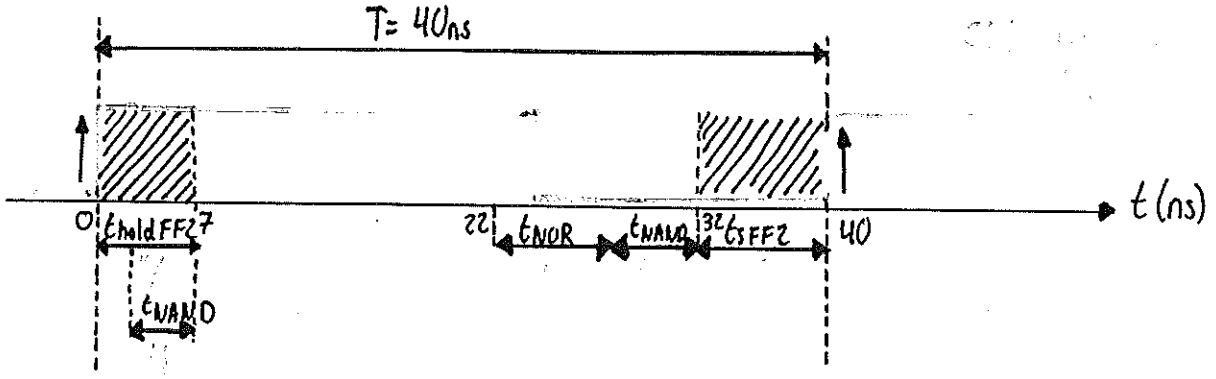


Figura 8



$t_{min} = t_{holdFF2} - t_{NAND} = 7 - 4 = 3ns$

$t_{max} = 40 - (t_{SFF2} - t_{NAND} - t_{NOR}) = 22ns$

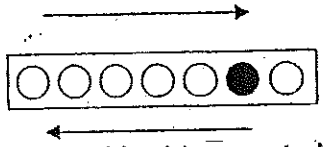
La señal E2 puede variar $t \in [3, 22]$ ns

FEBRERO 2004

PROBLEMA 2

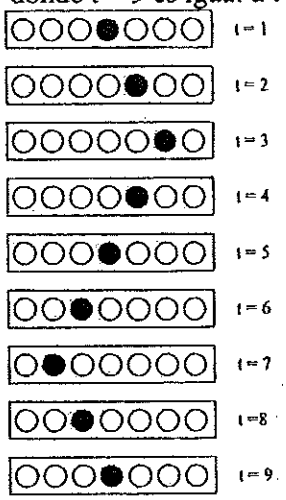
Se pretende diseñar un dispositivo luminoso consistente en una hilera de luces (*Figura 1*), tipo LED, de las cuales sólo una se encuentra encendida en cada momento dado.

Figura 1: Hilera de luces



El objetivo es realizar un dibujo de efecto hipnótico con las luces para atraer la atención de las personas que lo contemplan. Para ello se realizará un movimiento oscilatorio cuya secuencia periódica representamos en la *Figura 2*, y donde t = 9 es igual a t = 1.

Figura 2: Muestra del Dibujo con los LED



La estructura del dispositivo luminoso es la de la *Figura 3*:

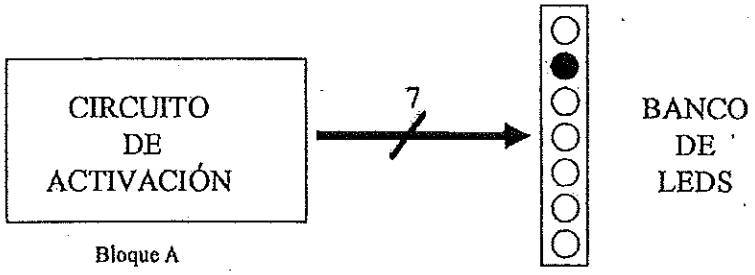
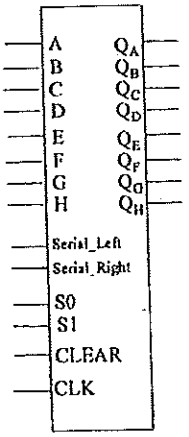
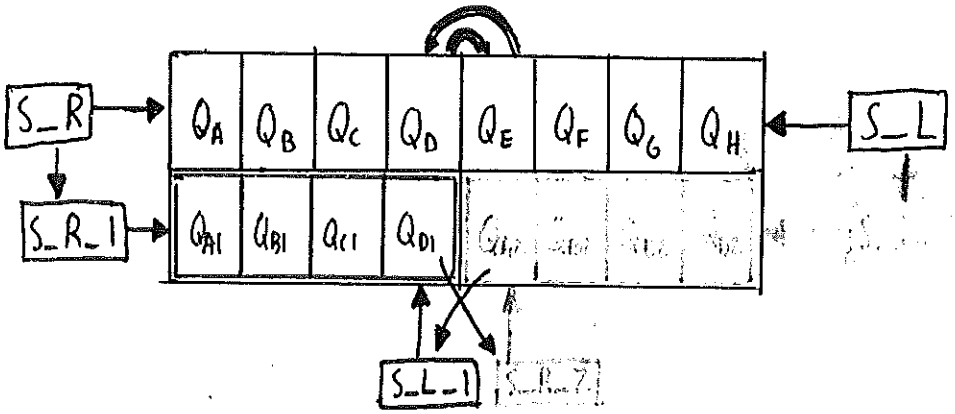


Figura 3: Diagrama de Bloques del Dispositivo Luminoso

El banco de LED tiene siete entradas, una por diodo **[I6..I0]** activas a nivel alto, de forma que si tienen un "1" el LED correspondiente se ilumina. El circuito de activación (*bloque A*) se encarga de generar las señales **[I6..I0]**. Para realizar el diseño se utilizará como núcleo un registro de desplazamiento bidireccional de 8 bits (*Figura 4*).

Figura 4: Registro de desplazamiento




S_R
S_R_1
Q_A Q_B Q_C Q_D Q_E Q_F Q_G Q_H
Q_{A1} Q_{B1} Q_{C1} Q_{D1}
S_L
S_L_1

La salida **QA** del registro irá conectada al LED ubicado en el extremo izquierdo del dispositivo, mientras que **QG** se conectará al LED del extremo derecho (*Figura 5*).

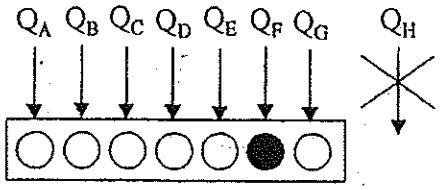


Figura 5: Conexión de las salidas del Registro de Desplazamiento a los LED (en el instante representado, $Q_F = 1$, con lo que su LED asociado está encendido)

2.1 Construya el registro de desplazamiento de la *Figura 4* a partir de dos registros de desplazamiento bidireccionales *74LS194*, a los que llamaremos **REG_1** y **REG_2** y cuya hoja de características puede verse en el apéndice. Para ello complete el circuito de la *Figura 6*. *(10 puntos)*

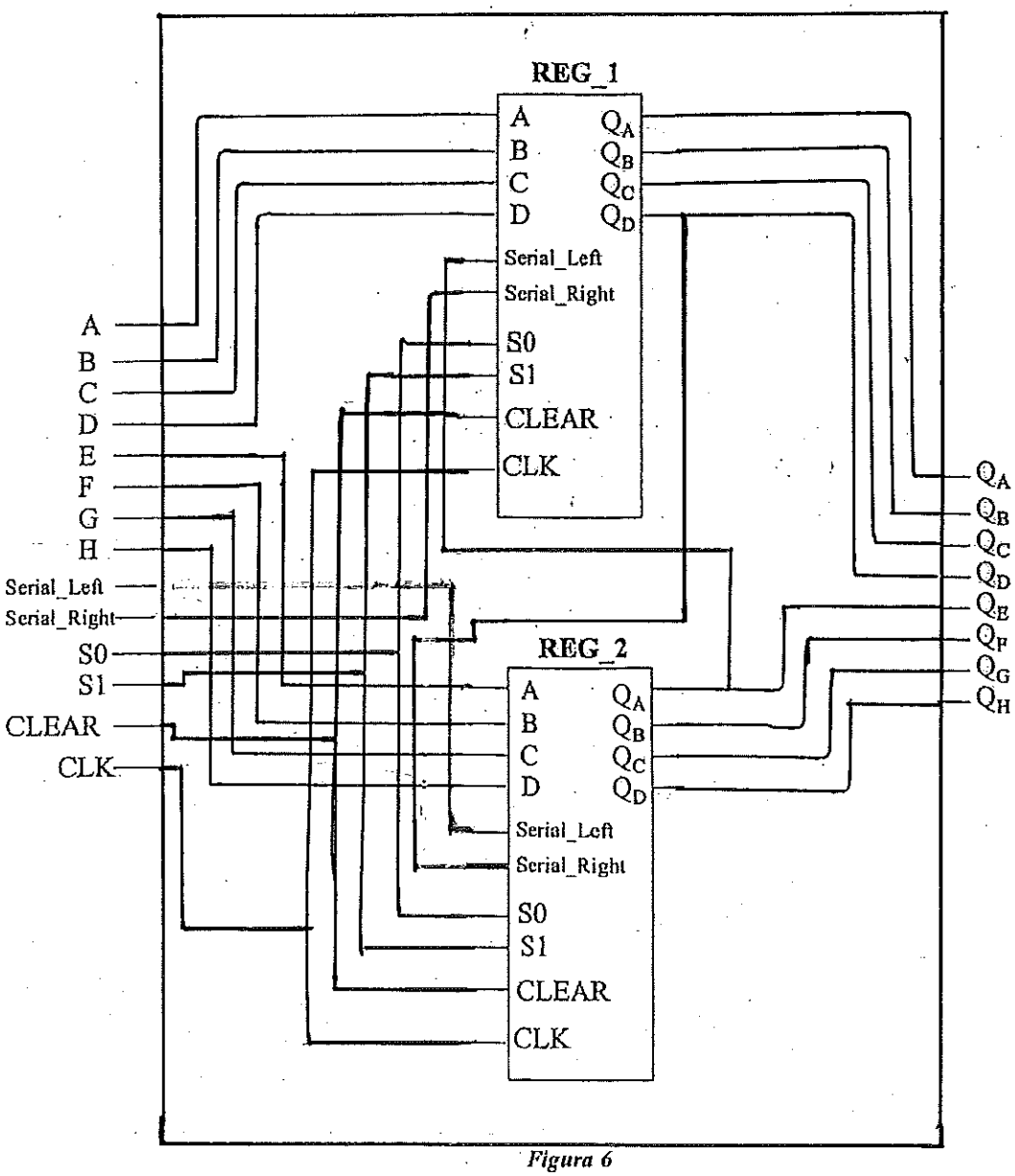


Figura 6

A continuación se conectará el registro de desplazamiento de la *Figura 4* al resto de componentes para obtener el circuito de activación completo, cuyo funcionamiento es el siguiente:

- Antes de iniciarse el movimiento, se activa la señal **LOAD** conectada al biestable tipo D. Como consecuencia, se deberá cargar en el registro el valor "01000000" (el último bit no se usa, pero es necesario cargarlo). **Nota:** La señal **LOAD** es activa a nivel alto y se mantiene activa como mínimo un ciclo de reloj **CK**.
- Cuando la señal **LOAD** se desactiva, el LED iluminado se desplaza con cada pulso de reloj en un sentido u otro (izquierda o derecha) hasta que la luz llega a uno de los extremos del movimiento (**QB** ó **QF** encendidos).
- Cada vez que se llega a un extremo (**QB** ó **QF** encendidos), se cambia el sentido del movimiento.

2.2 Complete las conexiones **necesarias** que faltan en el esquema de la *Figura 7. (20 puntos)*

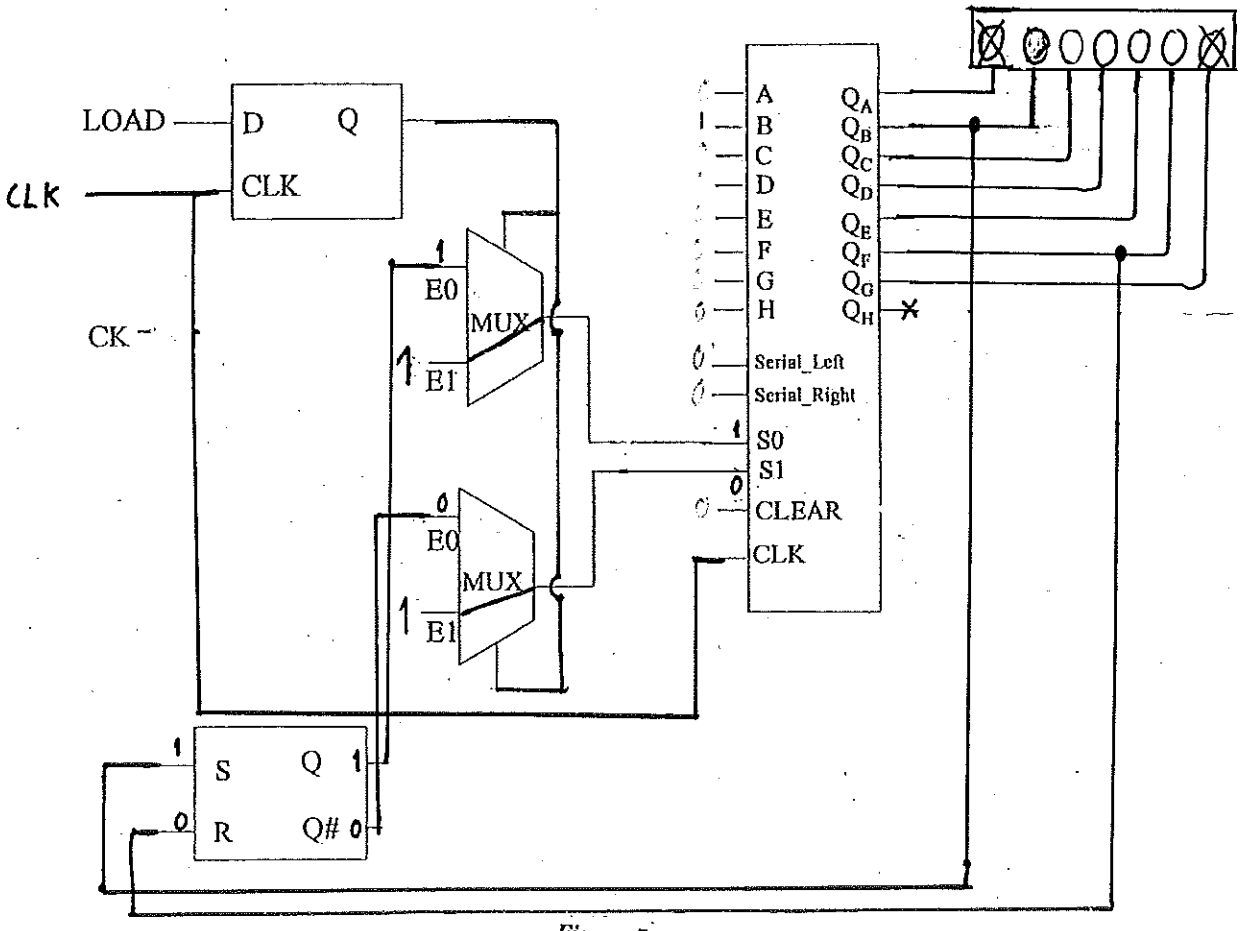
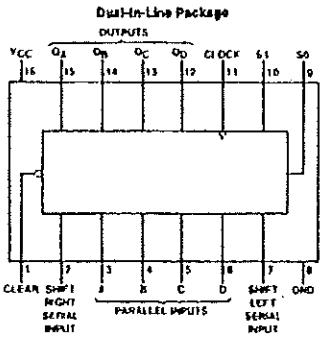


Figura 7

Apéndice

74LS194 *Bidirectional Universal Shift Register*
(Extracto de la Hoja de Características)



Function Table

Inputs										Outputs			
Clear	Mode		Clock	Serial		Parallel				Q_A	Q_B	Q_C	Q_D
	S1	S0		Left	Right	A	B	C	D				
L	X	X	X	X	X	X	X	X	X	L	L	L	L
H	X	X	L	X	X	X	X	X	X	Q_{A0}	Q_{B0}	Q_{C0}	Q_{D0}
H	H	H	↑	X	X	a	b	c	d	a	b	c	d
H	L	H	↑	X	H	X	X	X	X	H	Q_{An}	Q_{Bn}	Q_{Cn}
H	L	H	↑	X	L	X	X	X	X	L	Q_{An}	Q_{Bn}	Q_{Cn}
H	H	L	↑	H	X	X	X	X	X	Q_{Bn}	Q_{Cn}	Q_{Dn}	H
H	H	L	↑	L	X	X	X	X	X	Q_{Bn}	Q_{Cn}	Q_{Dn}	L
H	L	L	X	X	X	X	X	X	X	Q_{A0}	Q_{B0}	Q_{C0}	Q_{D0}

H = High Level (steady state), L = Low Level (steady state), X = Don't Care (any input, including transitions)

↑ = Transition from low to high level

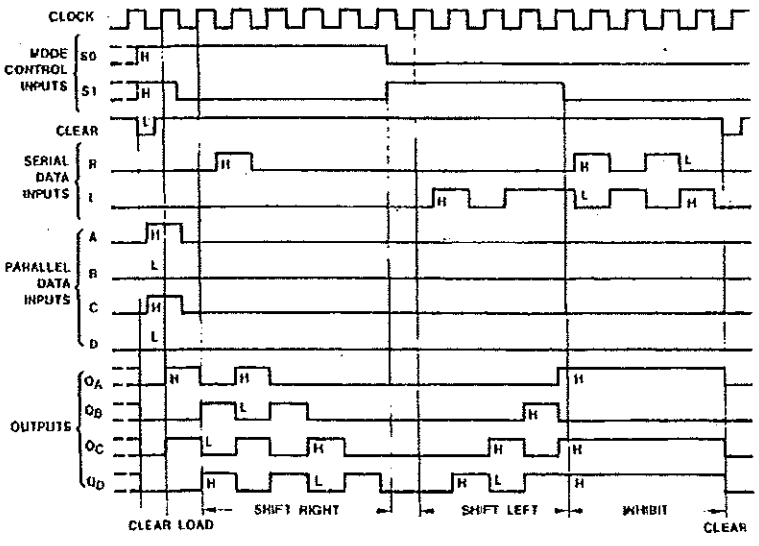
a, b, c, d = The level of steady state input at inputs A, B, C or D, respectively.

Q_{A0}, Q_{B0}, Q_{C0}, Q_{D0} = The level of Q_A, Q_B, Q_C, or Q_D, respectively, before the indicated steady state input conditions were established.

Q_{An}, Q_{Bn}, Q_{Cn}, Q_{Dn} = The level of Q_A, Q_B, Q_C, respectively, before the most-recent ↑ transition of the clock.

Timing Diagram

Typical Clear, Load, Right-Shift, Left-Shift, Inhibit, and Clear Sequences



PROBLEMA 4

Se dispone de una línea serie de datos por la cual se transmiten tramas de 8 bits de información [D7..D0] con una cabecera de 4 bits. El funcionamiento del sistema es el siguiente:

- La línea de datos en reposo se encuentra estado alto ("1").
- Cada vez que se quiere transmitir una nueva trama, primero se transmite una cabecera de 4 bits predeterminados, seguidos de los 8 bits de datos.
- Los 4 bits de cabecera tienen el código **0110**.
- Al detectarse la cabecera se generará una señal **INICIO** de duración un ciclo de reloj.
- Una vez registrado el octavo bit de información (**D7**), se generará una señal **FIN** de duración un ciclo de reloj.

Un ejemplo de cronograma de transmisión se puede ver en la *Figura 2*.

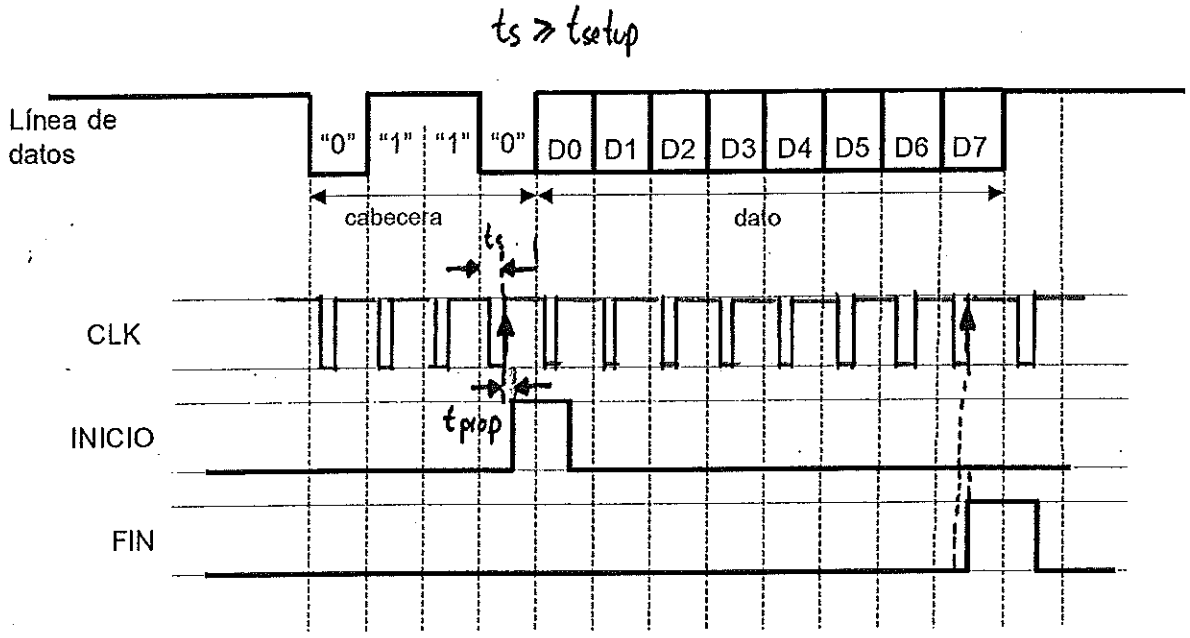


Figura 2

4.1 Utilizando un único registro de desplazamiento 74HC164, un comparador de 8 bits 74HC682 y las puertas inversoras necesarias, implemente el circuito que genere la señal **INICIO** sobre el esquema de la *Figura 3*. Dibuje todas las conexiones necesarias para un funcionamiento correcto. Dibuje también la señal de reloj (CLK) en la *Figura 2*. *(10 puntos)*

Nota. El registro de desplazamiento 74HC164 está formado por biestables tipo D con tiempos de set-up no despreciables. En el anexo del examen se encuentran las características del 74HC164 y del 74HC682. Utilice la información de dichas hojas.