CEDG

Carpeta de Montero Espinosa (resuelta en 2008)

CEDG
Teoría

PROGRAMA DE LA ASIGNATURA CEDG

1. **Introducción** (0,1 créditos)
   Información administrativa. Descripción del temario.

2. **Principios básicos** (0,4 créditos)
   Niveles lógicos en lógica binaria. Carga y descarga de capacidades en CMOS. Temporización básica.

3. **Codificación de la información y álgebra de conmutación** (0,7 créditos)
   Principios de numeración. Lógica booleana. Axiomas y ecuaciones. Representación de circuitos. Simplificación por Karnaugh.

4. **Circuitos Combinacionales** (1,0 crédito)
   Puertas lógicas simples y complejas. Multiplexores. Elementos varios: codificadores y decodificadores, comparadores y operadores. Memorias ROM, RAM y EPROM.

5. **Circuitos Secuenciales** (1,0 crédito)
   Báscula R-S. Latches y biestables. Registros. Contadores. Registros de desplazamiento.

6. **Teoría de Autómatas** (0,6 créditos)
   Máquinas de estados finitos (Mealy y Moore). Diseño de máquinas de estados.

7. **Descripciones funcionales y estructurales** (0,7 créditos)
   Descripciones funcionales y estructurales. Introducción al lenguaje VHDL. Descripciones en VHDL. Componentes sobre los que realizar la síntesis: CPLD y FPGA. Ejemplos adicionales y ejercicios.

TEMA 2: PRINCIPIOS BÁSICOS

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.

2.1 Lógica CMOS

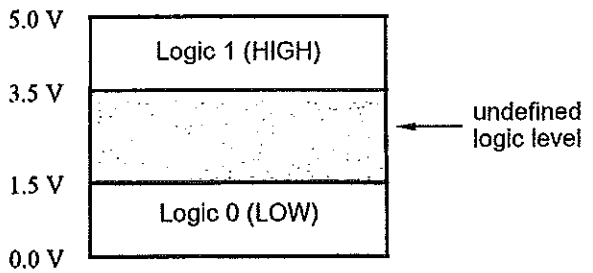
El comportamiento funcional de un circuito lógico CMOS es muy fácil de comprender, ya que los únicos bloques básicos de construcción van a ser transistores CMOS.

2.1.1 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.

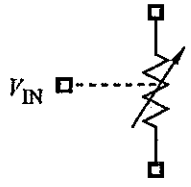
Un circuito lógico CMOS típico funciona a partir de una fuente de alimentación de 5 voltios. Un circuito de esta clase interpretará cualquier voltaje que esté entre 0 y 1.5 voltios como un 0 lógico, e interpretará a cualquier voltaje que esté entre 3.5 y 5 voltios como un 1 lógico.

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



2.1.2 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.



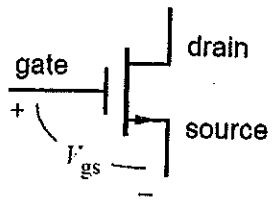
En aplicaciones de lógica digital, un transistor MOS opera de modo que su resistencia siempre sea muy elevada (el transistor se encuentra apagado "OFF"), ó muy baja (el transistor se encuentra encendido "ON)".

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.

2.1.2.1 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.

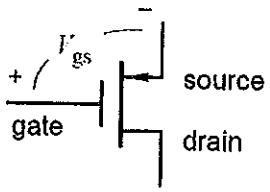
Las terminales se denominan *compuerta* (gate), *fuente* (source) y *drenador* (drain).

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").

2.1.2.2 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:

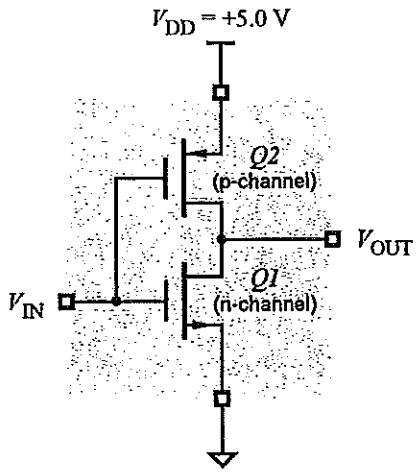
El voltaje de compuerta crea un campo eléctrico que refuerza o retarda el flujo de corriente entre las terminales de fuente y drenador. Este es el "efecto de campo" en el término MOSFET.

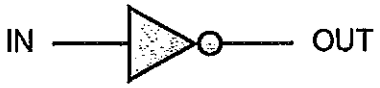
- La compuerta de un transistor MOS tiene una impedancia muy elevada. Es decir, esta compuerta está separada de la fuente y el drenador por un material aislante que tiene una resistencia muy elevada, con lo que la intensidad que corre entre la compuerta y las otras terminales del transistor es despreciable.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

2.1.3 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:

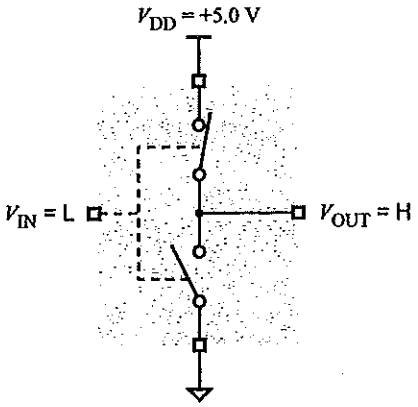
V_{IN}	$Q1$	$Q2$	V_{OUT}
0.0 (L)	off	on	5.0 (H)
5.0 (H)	on	off	0.0 (L)

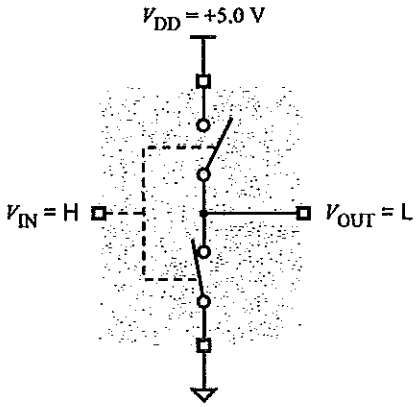
- Cuando $V_{ENT} = 0$, Q1 está apagado porque su V_{gs} es 0, pero Q2 está encendido porque su V_{gs} es un valor negativo muy grande (-5.0 V). En estas condiciones $V_{OUT} = 5V$.
- Cuando $V_{ENT} = 1$, Q1 está encendido porque su V_{gs} es un valor positivo grande (5.0 V), y Q2 está apagado porque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

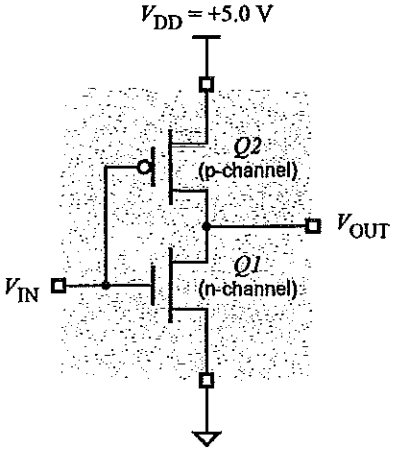
El modelo del transistor de canal-n corresponde a un interruptor normalmente abierto (cuando le ponemos un 1 lógico en la puerta), y el transistor de canal-p corresponde a un interruptor normalmente cerrado.

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**.

2.1.4 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:

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

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.

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:

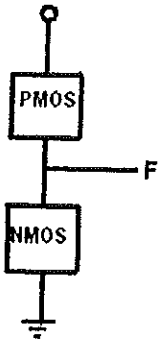
   - En transistores NMOS: ON → tensión de puerta = '1'
     OFF → tensión de puerta = '0'
   - En transistores PMOS: ON → tensión de puerta = '0'
     OFF → tensión de puerta = '1'

   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:

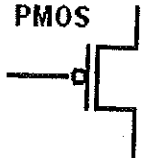
   $\wedge$ → · (AND)
   $\vee$ → + (OR)
   variable igualada a '1' → variable sin negar.
   variable igualada a '0' → variable negada.

2.1.5 Implementación de funciones mediante circuitos CMOS

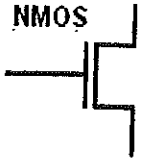
Implementaremos nuestra función siempre formando dos bloques, en cuya unión obtendremos la función booleana deseada.



- 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.



TEMA 7 EBAS

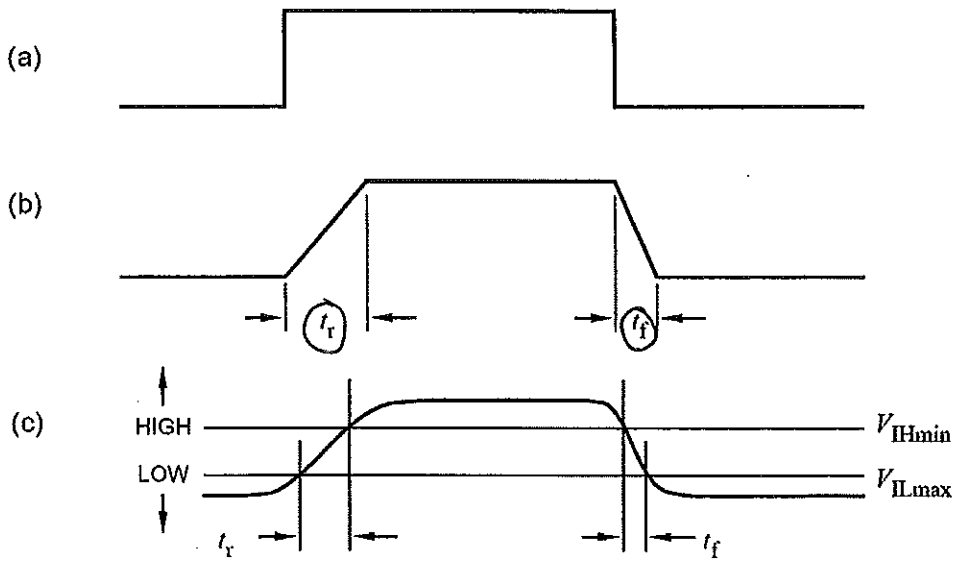
2.2 Comportamiento eléctrico dinámico de los dispositivos CMOS

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.

2.2.1 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.

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.

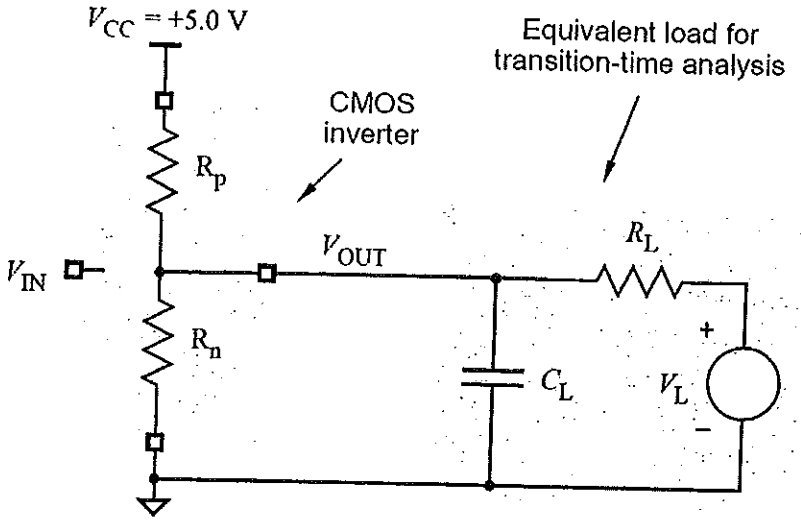
Las salidas no pueden cambiar simultáneamente, porque necesitan tiempo para cargar la capacidad parásita del cableado y otros componentes que controlan.

2.2.2 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. (es decir, solamente existe como carga en C_L)

Vamos a estudiar el análisis mediante un ejemplo:

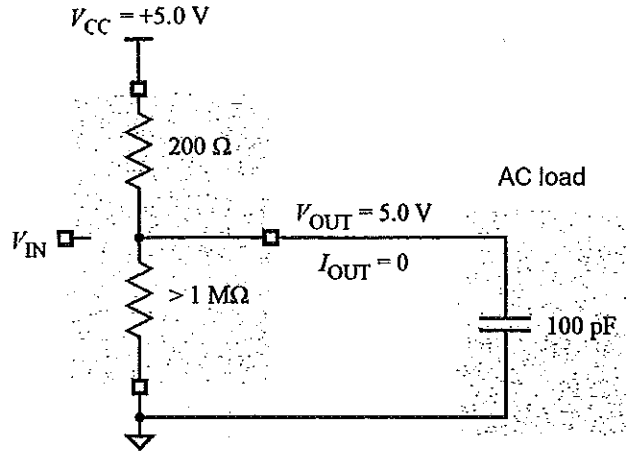
Ejemplo:

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

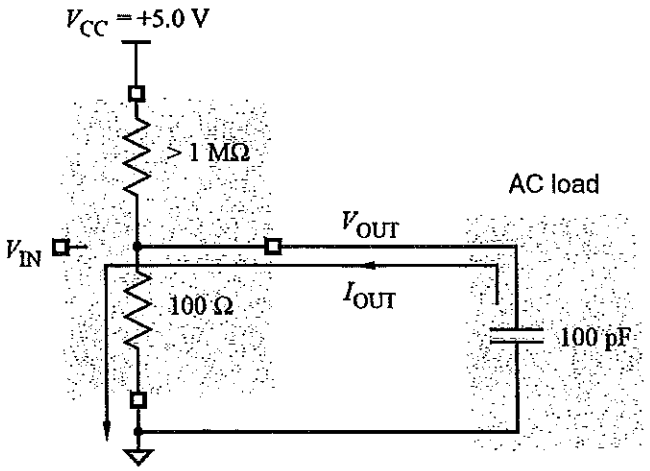
Primero observamos el tiempo de caída: ($t_f = t_{FALL}$)

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:



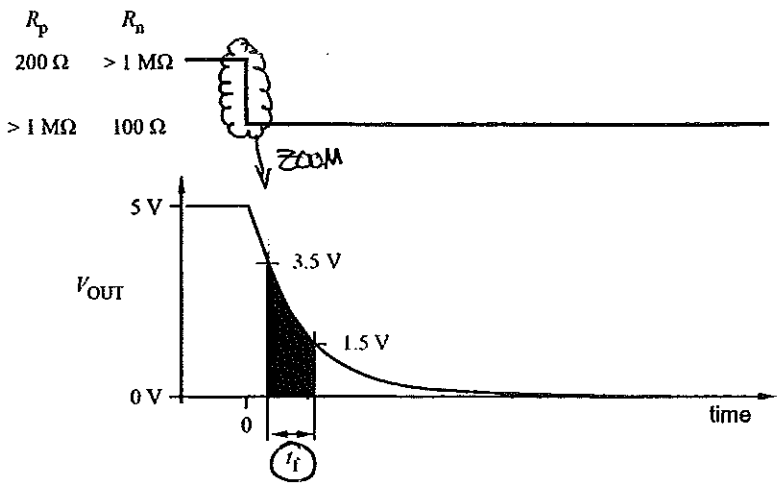
Cuando $t = 0$, V_{OUT} todavía es 5.0 V. En el tiempo $t = \infty$ el condensador tiene que estar completamente descargado, y V_{OUT} debe ser 0 V.

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

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

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

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



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

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 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:

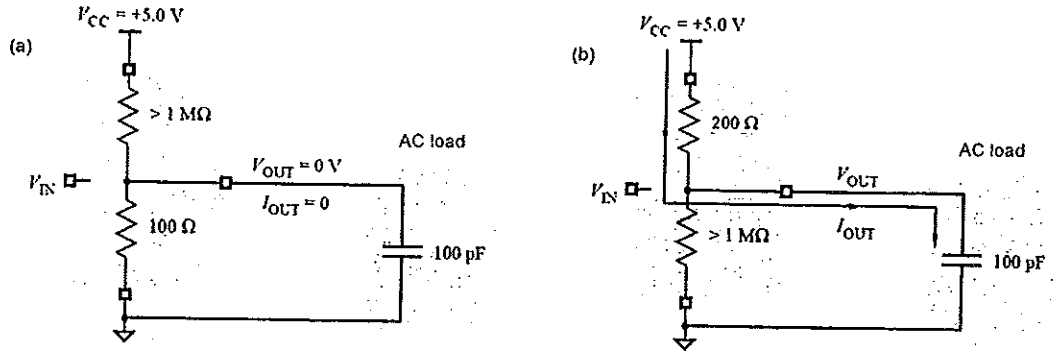
- $t_{3.5} = 3.57\ ns$
- $t_{1.5} = 12.04\ ns$

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

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.

Tiempo de ascenso: (t_r = trise)

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).

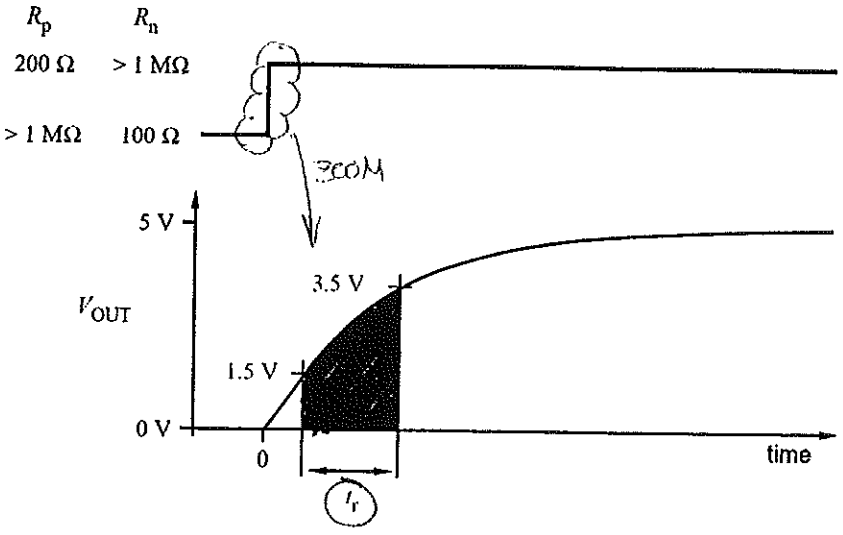


Una vez más, V_{OUT} no puede cambiar de forma instantánea, pero al tiempo $t = \infty$ el capacitor estará completamente cargado y V_{OUT} será 5V.

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

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

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



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:

- 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.
- 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.

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

- 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!

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

2.2.3 Retardo de propagación ó tiempo de propagación t_p < t_{pHL} / t_{pLH}

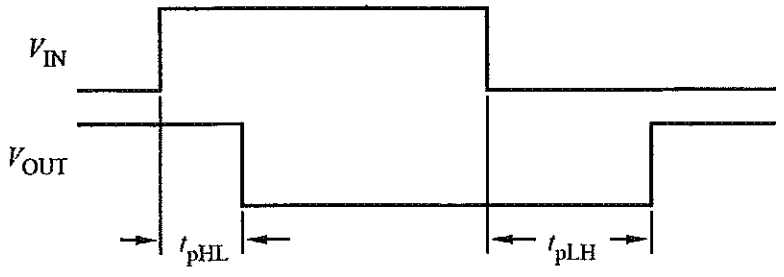
Una trayectoria de señal es la ruta eléctrica de una señal particular de entrada hacia una señal particular de salida, en un elemento lógico.

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.

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

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:



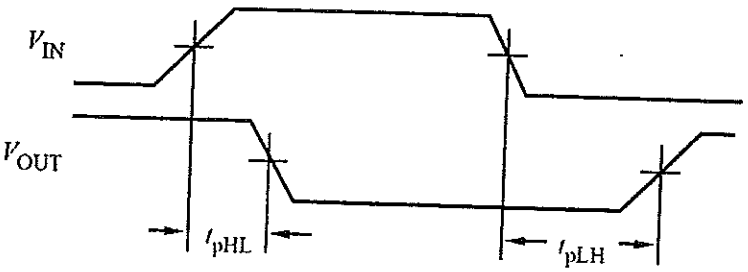
Los tiempos de ascenso y de caída sólo describen de forma parcial el comportamiento dinámico de un elemento lógico; se necesitan parámetros adicionales para relacionar la temporización de salida con la temporización de entrada.

Donde:

ambos ves que se definen para la salida

- 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.

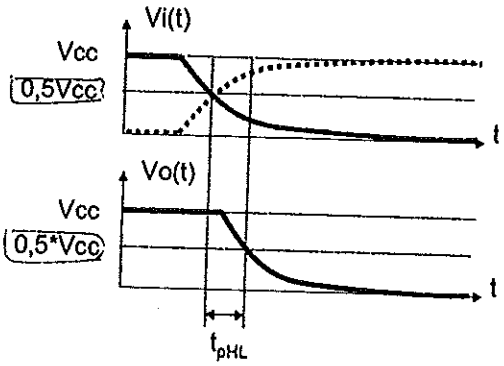
Entre los muchos factores que influyen en el retardo de propagación tenemos que destacar los tiempos de ascenso y caída, lo que hace que especifiquemos los tiempos de retardo de propagación como la distancia temporal entre los puntos medios de las transiciones de entrada y salida, como se ilustra en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



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

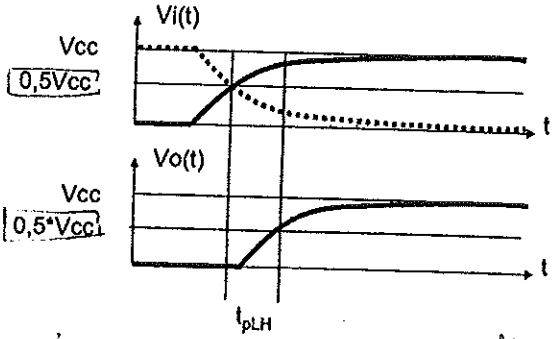
Tiempo de propagación de ALTO a BAJO t_{pHL}

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.



* NOTA: no tiene nada que ver con t_{RISE} y t_{FALL}, los retardos o tiempos de propagación t_{pLH} o t_{pHL} es una relación temporal ENTRE DOS SEÑALES. la de entrada V_i y la de salida V_o, los otros son en la misma señal.

TEMA 3: ÁLGEBRA DE CONMUTACIÓN

3.1 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 "$\ldots d_7 d_6 d_5 d_4 d_3 d_2 d_1$" donde "$d_i$" son los dígitos.

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

- 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

3.1.1 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.

3.2 Conversión entre las bases más comunes

De binario a:

El subíndice indica en qué base está el número

→ **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.

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

Es igual que el octal pero agrupando de cuatro en cuatro

→ **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.

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)

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

3.3 Representación de números binarios negativos

Para representar número negativos en binario existen básicamente dos sistemas, **complemento a uno y complemento a dos.**

3.3.1 Complemento a uno

El bit más significativo indica el signo y el resto el módulo si el número es positivo. En caso de ser negativo el módulo es el resultante de cambiar los "1" por "0" y viceversa. El rango de valores que se pueden representar con n bits es $-(2^{n-1}-1) \leq$ número $\leq (2^{n-1}-1)$.

Usando el complemento a 1, el "0" se puede representar tanto 00000000 como 11111111 (en este ejemplo, trabajando con 8 bits).

El problema del complemento a uno es que el cero no queda unívocamente definido.

Ejemplo: $01010101_2 = 85_{10}$
$10101010_2 = -85_{10}$

3.3.2 Complemento a dos

Igual que complemento a uno, pero si el número es negativo se suma "1" al resultado. Es la representación más utilizada.

Para saber qué cifra representa un número negativo en complemento a 2, basta con volver a complementarlo a 2, y obtendremos su valor absoluto.

El rango de valores que se pueden representar con n bits es $-2^{n-1} \leq$ número $\leq (2^{n-1}-1)$.

Ejemplo: $01010101_2 = 85_{10}$
$10101011_2 = -85_{10}$

En la tabla siguiente tienes ejemplos de representación con 4 bits.

Decimal	Complemento a dos	Complemento a uno
–8	1000	---
–7	1001	1000
–6	1010	1001
–5	1011	1010
–4	1100	1011
–3	1101	1100
–2	1110	1101
–1	1111	1110
0	0000	1111 ó 0000
1	0001	0001
2	0010	0010
3	0011	0011
4	0100	0100
5	0101	0101
6	0110	0110
7	0111	0111

Recuerda que el bit más significativo nos da la información relativa al signo del número.

3.4 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.

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

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.

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:

$$N_{10} = \underbrace{0011}_{3}\underbrace{1000}_{8}\underbrace{0100}_{4}\underbrace{1001}_{9}\underbrace{0000}_{0}\underbrace{0101}_{5} = 384905_{10}$$

3.4.2 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.

- **Complemento a dos** (EJEMPLO)

Paso de decimal (D) a complemento a dos (C2)

+10: 1º] representamos (+10) en binario: 1010

2º] añadimos un bit de signo positivo: **01010** //

-10: 1º] representamos (+10) en binario: 1010

2º] añadimos un bit de signo positivo: 01010

3º] complemento a dos:

3.1 - complemento a 1: 10101

3.2 - sumamos 1: **10110** //

Paso de complemento a 2 (C2) a decimal (D)

01010 ⇒ simplemente pasamos a decimal: **+10** //

↑ es positivo !!

10110 ⇒ sabiendo que es negativo, hallaremos su valor absoluto:

↑ es negativo !!

1º] complementamos a 2: 10110 → 01001 $\xrightarrow{+1}$ 01010

2º] pasamos a decimal: 10

- El valor que estoy buscando es **-10** //

Este apartado te será útil en el laboratorio LCEL, de 3º.

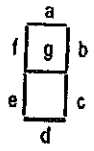
3.4.3 Ejemplo de uso de BCD: excitación de los LEDs de "displays" de 7 segmentos

LS247
FUNCTION TABLE

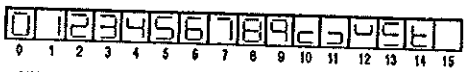
INPUTS				OUTPUTS						
D	C	B	A	a	b	c	d	e	f	g
L	L	L	L	ON	ON	ON	ON	ON	ON	OFF
L	L	L	H	OFF	ON	ON	OFF	OFF	OFF	OFF
L	L	H	L	ON	ON	OFF	ON	ON	OFF	ON
L	L	H	H	ON	ON	ON	ON	OFF	OFF	ON
L	H	L	L	OFF	ON	ON	OFF	OFF	ON	ON
L	H	L	H	ON	OFF	ON	ON	OFF	ON	ON
L	H	H	L	ON	OFF	ON	ON	ON	ON	ON
L	H	H	H	ON	ON	ON	OFF	OFF	OFF	OFF
H	L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

3.5 Álgebra de Boole

Es un tipo de álgebra que, basándose en la teoría de conjuntos, se aplica a sistemas matemáticos en los que sólo existen dos elementos posibles: el 0 y el 1.

De la anterior definición se deduce su posibilidad de ser aplicada al análisis y diseño de **circuitos digitales**, simplemente precisando el siguiente convenio:

Este convenio responde al nombre de "Lógica positiva", que es la más utilizada.

- Presencia de tensión = 1.
- Ausencia de tensión = 0.

3.6 Operaciones y propiedades básicas

En el álgebra de Boole sólo existen tres operaciones:

- Suma.
- Multiplicación.
- Complementación o inversión.

En la tabla se indican las forma de representación, así como sus postulados básicos. Las operaciones del álgebra de Boole cumplen las siguiente propiedades:

a) Conmutativa: $a+b=b+a$
$a\cdot b=b\cdot a$

b) Asociativa: $a+b+c=a+(b+c)$
$a\cdot b\cdot c=(a\cdot b)\cdot c$

c) Distributiva: $a\cdot(b+c)=a\cdot b+a\cdot c$
$a+(b\cdot c)=(a+b)\cdot(a+c)$

Operación	Forma de representarla	Postulados básicos
Suma	$F = a + b$	$0 + 0 = 0$ $\quad a + 0 = a$ $0 + 1 = 1$ $\quad a + 1 = 1$ $1 + 1 = 1$ $\quad a + a = a$ $a + \bar{a} = 1$
Multiplicación	$F = a \cdot b$ $F = ab$ $F = a * b$	$0 \cdot 0 = 0$ $\quad a \cdot 0 = 0$ $0 \cdot 1 = 0$ $\quad a \cdot 1 = a$ $1 \cdot 1 = 1$ $\quad a \cdot a = a$ $a \cdot \bar{a} = 0$
Complementación o inversión	$F = \bar{a}$ $F = \overline{a \cdot b}$	$\bar{0} = 1$ $\quad \bar{a} = a$ $\bar{1} = 0$

3.7 Teoremas y leyes booleanas principales

Los teoremas de álgebra de Boole son demostrables, a diferencia de los del álgebra convencional, por el **método de inducción completa**. Este método consiste en comprobar que la relación entre los elementos que el teorema define se cumplen en todos los casos posibles. Para poder realizar esto se emplean las llamadas **tablas de verdad**, que no son otra cosa que representaciones gráficas de todos los casos que pueden darse en una relación y de sus respectivos resultados.

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	$\overline{(a + b + c + \cdots)} = \bar{a} \cdot \bar{b} \cdot \bar{c} \cdot \ldots$	$\overline{(a \cdot b \cdot c \cdot \cdots)} = \bar{a} + \bar{b} + \bar{c} + \cdots$	(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)

3.8 Formas canónicas de una función booleana

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:

$$z = \overline{\overline{abc} \cdot \overline{abc} \cdot \overline{abc}}$$

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:

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

3.9 Obtención de la ecuación de una función lógica partiendo de su tabla de verdad.

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

3.10 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.

3.10.1 Simplificación utilizando el Método de Karnaugh

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.

TEMA 4: CIRCUITOS COMBINACIONALES

4.1 Definiciones básicas

4.1.1 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.

4.1.2 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.

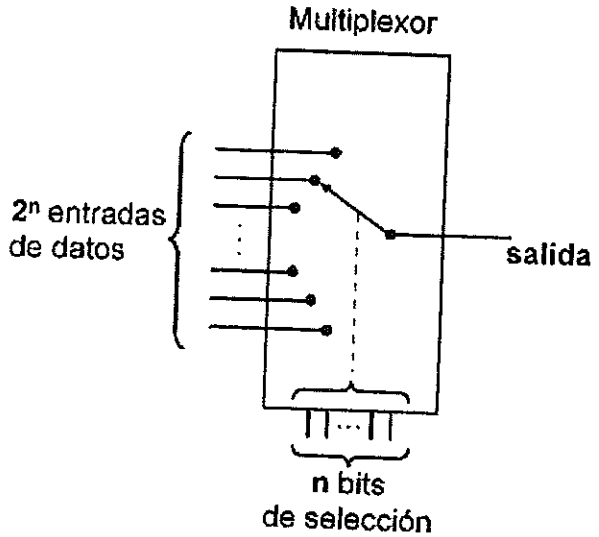
4.2 Multiplexores

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:

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

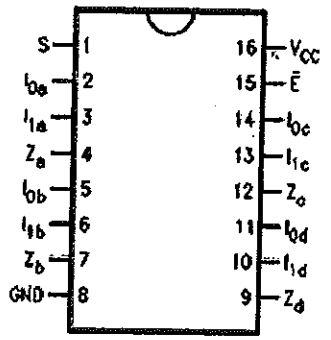


En los esquema representativos de estos circuitos se suele denominar a dichas entradas y salidas con los símbolos que se exponen a continuación:

- 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

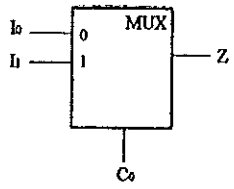
4.2.1 Funcionamiento de un multiplexor

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.

4.2.2 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

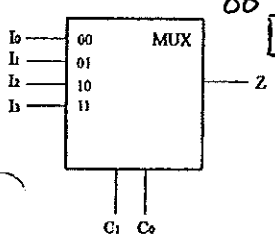
Y esta es la función que implementa:

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

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

4.2.3 Multiplexor 4 x 2

(ver PLANING)

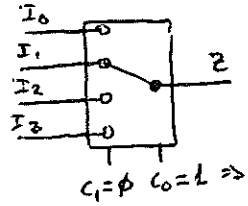


La tabla de verdad de un multiplexor de 4 canales de entrada (I_0, I_1, I_2 e I_3) con 2 entradas de control (C_0 y C_1) es la siguiente:

$Z = \overline{C}_1\overline{C}_0 I_0 + \overline{C}_1 C_0 I_1 + C_1\overline{C}_0 I_2 + C_1 C_0 I_3$

1ª forma canónica

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



$C_1 = 0$ $C_0 = 1$ ⇒

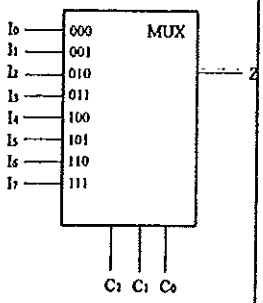
⇒ seleccionamos entrada I_1

Y esta es la función que implementa:

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

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

4.1.4 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:

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

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

4.2.4 Realización de funciones lógicas con multiplexores

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.

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

- 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:

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)

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

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

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

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

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

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

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.

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

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

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.

4.3 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.

4.3.1 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.

4.3.2 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:

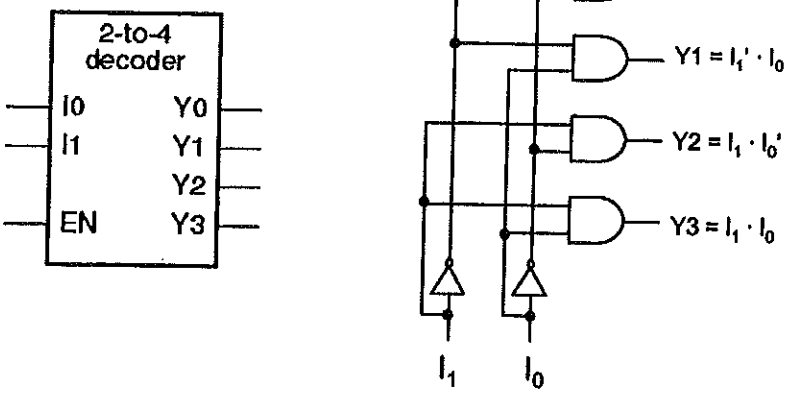
a tabla de verdad del decodificador binario introduce una notación "sin importancia" para combinaciones de entrada. Si uno o más valores de entrada no afectan a los valores de salida para alguna combinación de las entradas restantes, se marcan con una X para esa combinación de entrada.

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

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

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



4.3.3 Decodificador 3 a 8

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

El que presentamos tiene las siguientes entradas y salidas:

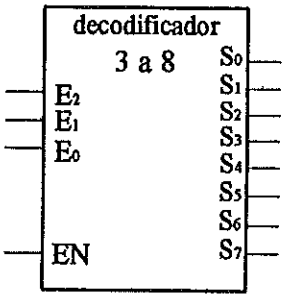
- E_0 a E_2 son las tres entradas.
- S_0 a S_7 son las ocho salidas. Una de ellas se activará en función del número binario representado a la entrada.
- *EN* es la entrada de autorización o *enable*.

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



4.3.4 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.

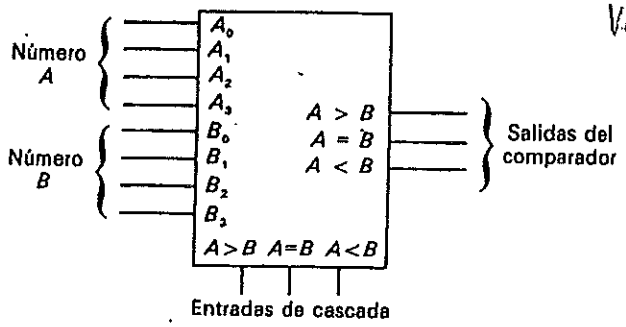
4.4 Comparadores binarios

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:



Ver Jpb85, pg 4.

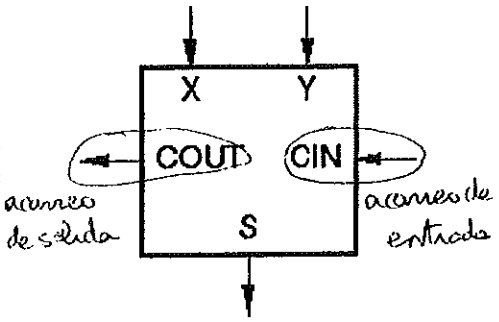
Su tabla de funcionamiento la siguiente.

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

4.5 Sumadores – Full Adder

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

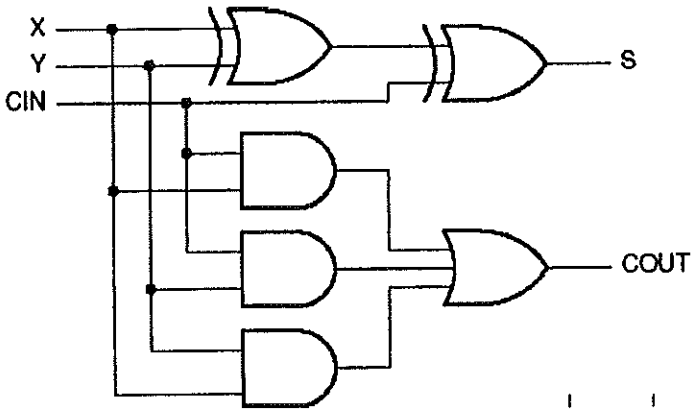


El FA sólo suma dos bits más el acarreo de entrada

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 circuito interno del sumador completo se muestra a continuación:

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



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

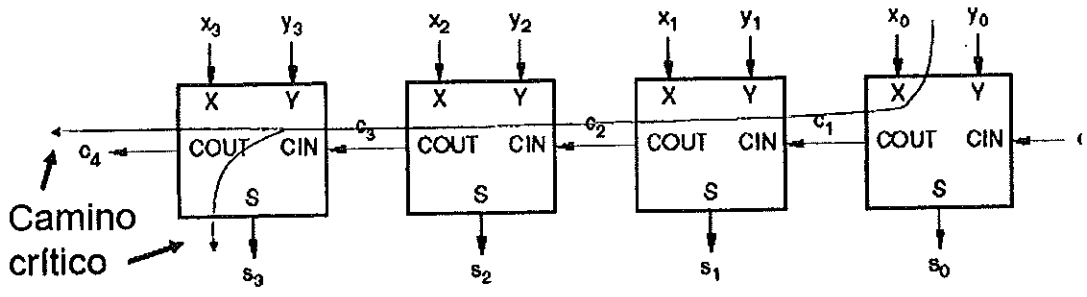
A partir del bloque elemental se pueden construir sumadores de más bits (*n* bits).

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



4.6 Memorias ROM

matriz = memoria

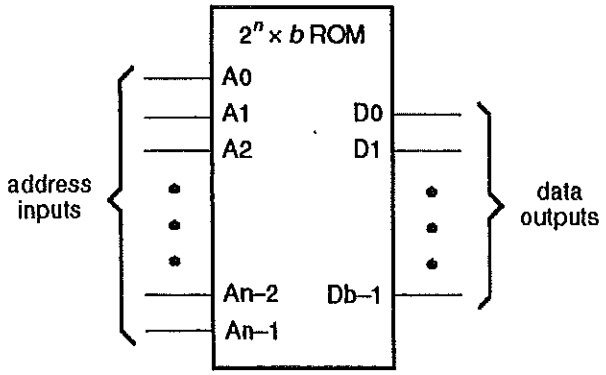
fila = palabra

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

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).

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

4.6.1 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.

4.6.2 Memoria ROM de 8 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.

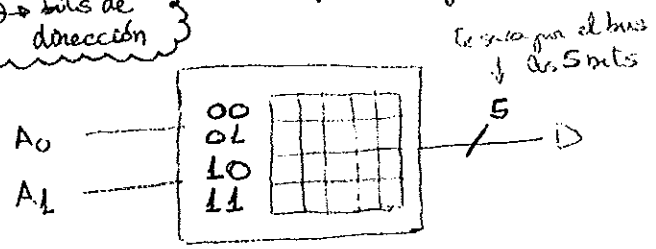
4.6.3 Funcionamiento de las memorias.

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*).

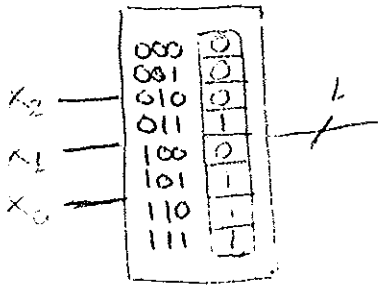
Y así sucesivamente...

- ROM 4×5 → 5 bits por palabra (columnas)
  → 4 palabras (filas de la tabla)

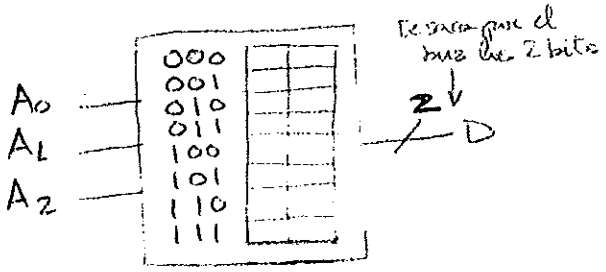
N → bits de dirección



Para implementar una función de (por ejemplo) 3 variables lo único que hay que hacer es usar una ROM 3×1, y poner en cada casilla los valores de la función (la tabla de verdad!



- ROM 8×2
  → 2^3 ⇒ 3 bits de dirección



4.6.4 Uso de las memorias ROM

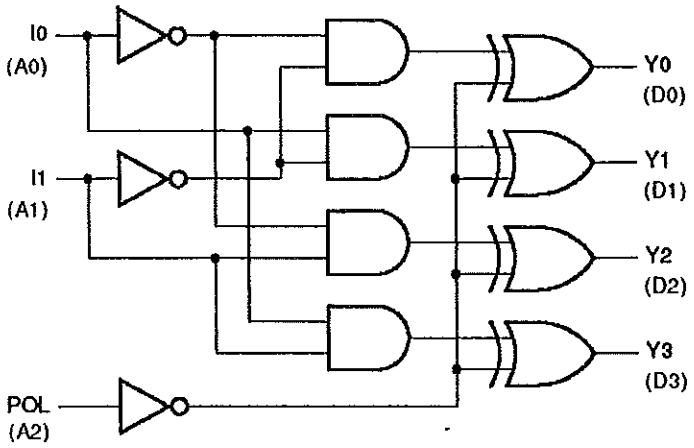
Una Memoria ROM puede implementar cualquier función lógica combinacional.

Para ello basta con hacer los siguiente:

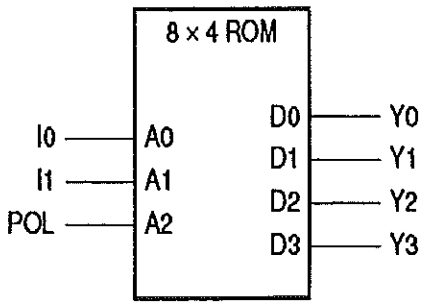
- Las entradas de la función se conectan al bus de direcciones
- Las salidas de la función se conectan al bus de datos

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 5: CIRCUITOS SECUENCIALES

Recuerda lo explicado en el tema 4 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.

5.1 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.

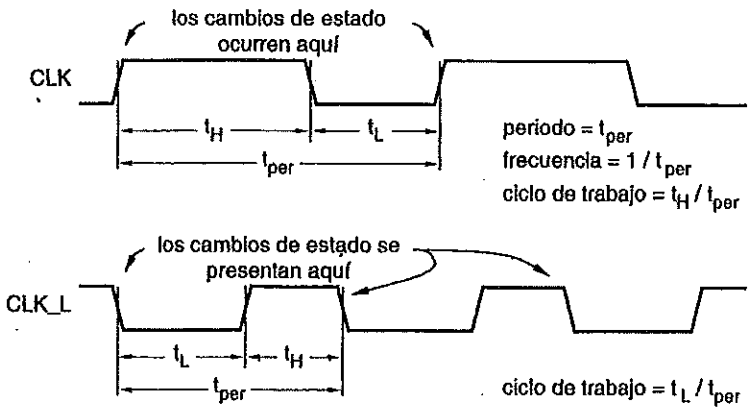
- Las variables de estado se guardan en uno o más bits de información.
- Considerando como entradas las entradas del circuito y las variables de estado, el diseño de un circuito secuencial es igual al de uno combinacional.

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.

5.1.1 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.



$t_H \equiv$ time high
$t_L \equiv$ time low

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.

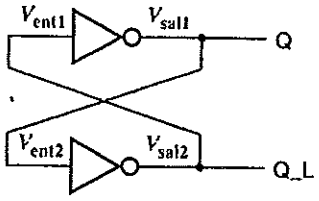
Por convención, una señal de reloj es de estado activo alto si los cambios de estado se presentan en el flanco de subida del reloj o cuando el reloj está en ALTO, y de estado activo bajo en el caso complementario.

- 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.

5.2 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 CEDG.

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.

5.2.1 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 <u>síncronos</u>.

- Todos los diseñadores digiales utilizan el nombre de **<u>flip-flop</u>** 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 <u>asíncronos</u>.

- Por otra parte, la mayoría de los diseñadores digitales utilizan el nombre de **<u>cerrojo</u>** (*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.

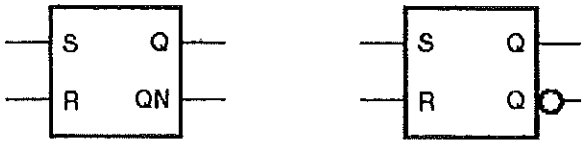
La clasificación de los biestables, desde el punto de vista de su constitución y del número de entradas puede resumirse en: **biestable R-S, biestable J-K, biestable T y biestable D.**

En CEDG a los cerrojos les llamamos básculas.

5.2.2 Báscula R-S (R≡Reset, S≡Set)

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

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 $\bar{Q}$ 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:

variables de estado

R	S	Q_t	$\bar{Q}_t$	Q_{t+1}	
0	0	0	1	0	con R=0 y S=0, lo que esté en Q_t, se conserva para Q_{t+1}
0	0	1	0	1	
0	1	0	1	1	con R=0 y S=1, independientemente de Q_t, ponemos en Q_{t+1}=1
0	1	1	0	1	
1	0	0	1	0	con R=1 y S=0, independ. de Q_t ponemos un Q_{t+1}=0
1	0	1	0	0	
1	1	0	0	-	estado metaestable: las salidas oscilan hasta q aleatoriamente se estabilizan en 0 o 1.

Q_t representa la salida Q en un determinado instante *t*.

$\bar{Q}_t$ representa la salida $\bar{Q}$ 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:

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

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

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, $\bar{Q}$ 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.

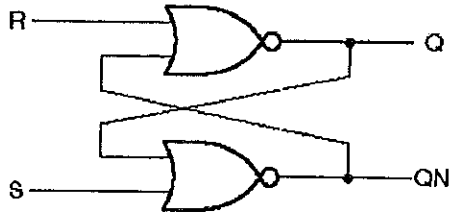
Sin embargo, si negamos ambas entradas de manera simultánea, la báscula se encamina a un estado siguiente impredecible y de hecho puede oscilar o entrar a un estado conocido como metaestable.

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.

5.2.2.1 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

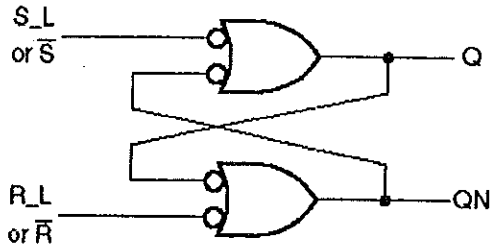


5.2.2.2 Implementación de una báscula R-S con puertas NAND

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

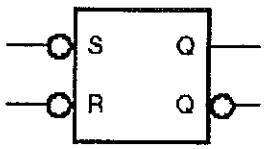
Observa que las puertas OR con las entradas negadas son, en realidad, puertas NAND:

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



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

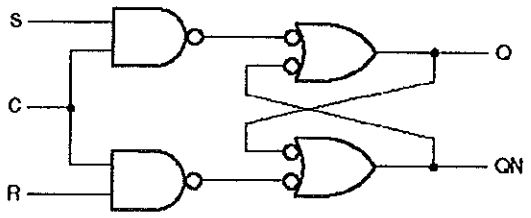
El funcionamiento de esta báscula $\overline{R}-\overline{S}$ es semejante al de la $R-S$, con dos diferencias principales:

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

- $\overline{R}$ 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 $R-S$.

5.2.2.3 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:

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

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.

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

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.

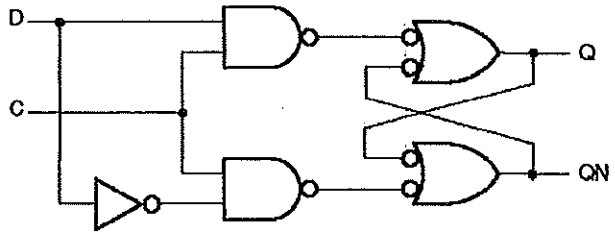
Con frecuencia necesitamos cerrojos simplemente para almacenar bits de información: cada bit es presentado en una línea de señal, y nos gustaría almacenarlo en alguna parte.

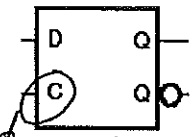
Un cerrojo D puede emplearse en una aplicación de esta naturaleza.

5.2.3 Latch D

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.





entrada de habilitación (enable)
A 0 mantiene lo que haya en Q.
A 1 pone lo que haya en D en Q.

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

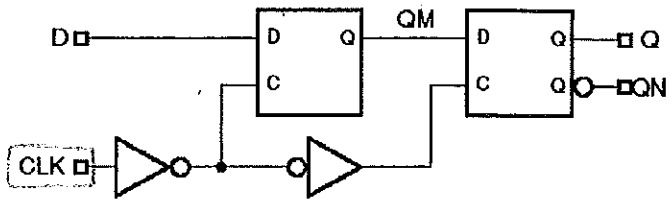
El circuito se denomina con frecuencia un "cerrojo transparente" por esta razón.

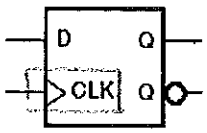
- Cuando C es afirmada, la salida Q sigue la entrada D. En esta situación se dice que el cerrojo está abierto y la trayectoria desde la entrada D a la salida Q es transparente.
- Cuando la entrada C es negada, el cerrojo se cierra y la salida Q retiene su último valor y ya no cambia en respuesta a D, mientras que C permanezca negada.

5.2.4 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:





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.

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**.

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

D	CLK	Q	QN
X	↓	last Q	last QN
X	↓	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

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

La ecuación característica de un biestable D es:

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

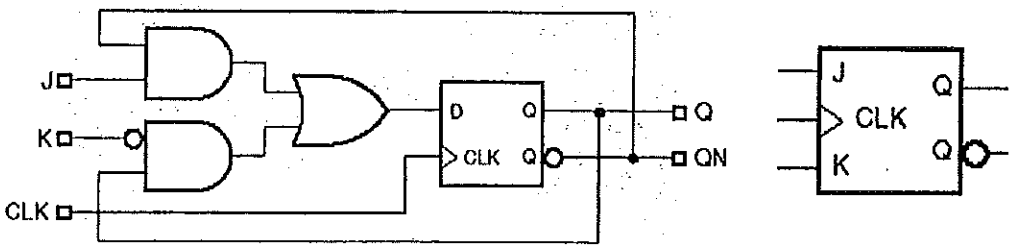
Observaciones:

- 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.

5.2.5 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.

es la misma que la K-S sólo que solucionamos el problema J=K=1

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

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:

En esta tabla se interpreta que:

x = 0 ó 1

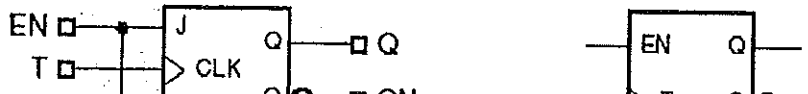
Q_t	Q_{t+1}	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

5.2.6 Flip-flop T

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

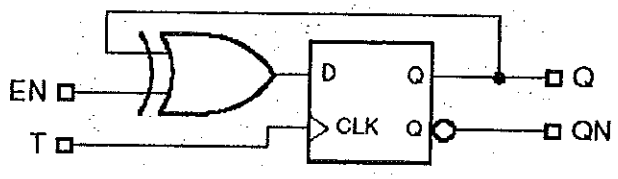
Este circuito es muy importante para realizar contadores.

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



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.



T viene de "*toggle*", conmutación

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:

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

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

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

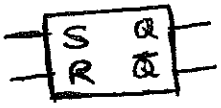
La ecuación característica de un biestable T es:

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

** RESUMEN DE BIESTABLES

- **Cerrojo (latch)**: dispositivo secuencial que no necesita una señal de reloj (CLK)

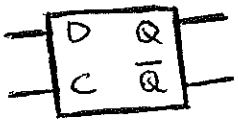
→ **báscula R-S**



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

→ R=S=0 se conserva Q_t
→ R=0; S=1 (activo); ponemos un 1 en Q_{t+1}
→ R=1 (activo); S=0; reseteamos y ponemos 0 en Q_{t+1}
→ estado metaestable: resultado aleatorio

→ **latch (cerrojo) D**

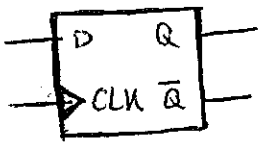


C	D	Q_{t+1}
1	0	0
1	1	1
0	X	Q_t

C ≡ actúa de enable ⇒ C=1 y D=0 ⇒ ponemos un 0. C=1, D=1 ⇒ ponemos un 1.
C=0 (enable desactv.) ⇒ conserva Q_t

- **Biestable (flip flop)**: dispositivo secuencial que necesita una señal de reloj (CLK)

→ **flip-flop D disparado por flanco**



sólo hay cambio con el flanco de subida de CLK ⇒

D	CLK	Q_{t+1}
0	↑	0
1	↑	1
X	0,1,↓	Q_t

↑ en el flanco de subida se pone a la salida lo q haya a la entrada (D)

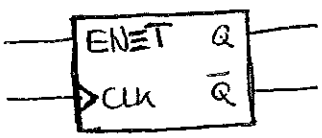
→ **flip-flop J-K**

como el R-S solo que activado por ↑ ⇒ J≡S ; K≡R

J	K	CLK	Q_{t+1}
X	X	0,1,↓	Q_t
0	0	↑	Q_t
0	1	↑	0
1	0	↑	1
1	1	↑	$\overline{Q_t}$

→ CLK=0,1,↓ conserva,
↑: J=K=0 conserva
J=0; K=1 → pone 0
J=1, K=0 → pone 1
J=1, K=1 → niega

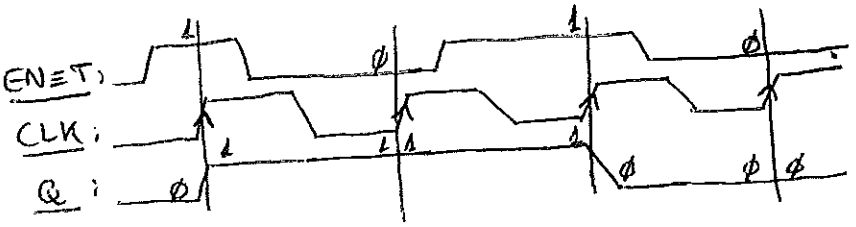
→ **flip-flop T**:



EN≡T	CLK	Q_{t+1}
0	↑	Q_t
1	↑	$\overline{Q_t}$

→ si el enable (T=0 ⇒ conserva (no permite el cambio)
si el EN=T=1 ⇒ cambia $Q_{t+1}=\overline{Q_t}$

q)



5.3 Temporización de circuitos lógicos

En ocasiones puede suceder que aunque un circuito está bien diseñado a nivel lógico después no funciona como se espera en la práctica. Esto normalmente es debido al **retardo** de cada uno de los componentes del circuito .

Se trata de estudiar en este apartado el **período mínimo** o, lo que es lo mismo, la **frecuencia máxima**, que puede tener la señal de reloj para que un circuito funcione correctamente teniendo en cuenta los retardos de los componentes implicados.

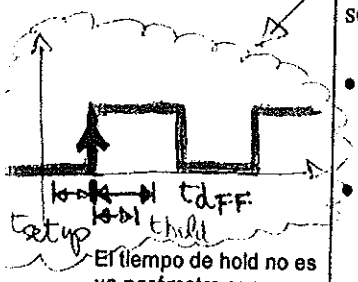
5.3.1 Parámetros característicos de las puertas lógicas

Las puertas son dispositivos asíncronos. Esto significa que la salida cambia en el momento que cambia alguna de sus entradas (si suponemos que su tiempo de propagación es nulo), sin esperar a ninguna señal de reloj.

- Tiempo de propagación (t_d) → Tiempo que tarda una puerta en cambiar su salida desde que cambia alguna de sus entradas. El tiempo de propagación es un parámetro dado por el fabricante.

5.3.2 Parámetros característicos de los biestables

El flanco activo puede ser el de subida o el de bajada



El tiempo de hold no es un parámetro que intervenga en el cálculo de la frecuencia máxima del reloj de un circuito.

Los biestables son dispositivos síncronos. Esto significa que su salida solo cambia cuando se produce un flanco activo de reloj (si suponemos que su tiempo de propagación es nulo).

- Tiempo de propagación (t_{dFF}) → Tiempo que tarda un biestable en cambiar la salida desde que recoge las entradas en el flanco activo del reloj.
- Tiempo de establecimiento (*set up time*, t_{setup}) → Tiempo antes del instante de flanco activo en el cual las entradas del biestable no deben cambiar. = tiempo antes en que las señales han de estar preparadas (antes del flanco activo)
- Tiempo de hold (*hold time*, t_{hold}) → Tiempo después del flanco activo en el cual las entradas del biestable no deben cambiar. = a la entrada del biestable (el biestable tarda un poco en leer las entradas. Desde que empieza a leer la entrada hasta que termina de leer)

Observación:

- El tiempo de propagación, el tiempo de setup y el tiempo de hold son parámetros dados por el fabricante. Es decir, en los ejercicios son dato.

Todos estos parámetros del reloj quedan completamente ilustrados en la figura de la página 5.1

5.3.3 Parámetros característicos del reloj

- Periodo del reloj (T) → Tiempo entre transiciones sucesivas en la misma dirección. Esto es, el tiempo que transcurre entre un flanco de subida y el siguiente flanco de subida, por ejemplo.

Habitualmente se mide en Mhz

- Frecuencia de reloj (f) → Inversa del Período.

En porcentaje, %

- Ciclo de trabajo → es el porcentaje de tiempo que la señal de reloj se encuentra en su nivel ALTO.

  Lo usual es que el ciclo de trabajo sea del 50% lo que significa que el reloj tiene los dos semiciclos alto y bajo de igual duración. En este caso también se dice que el reloj es simétrico.

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.

5.3.4 Cálculos de temporización

5.3.4.1 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.

5.3.4.2 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.

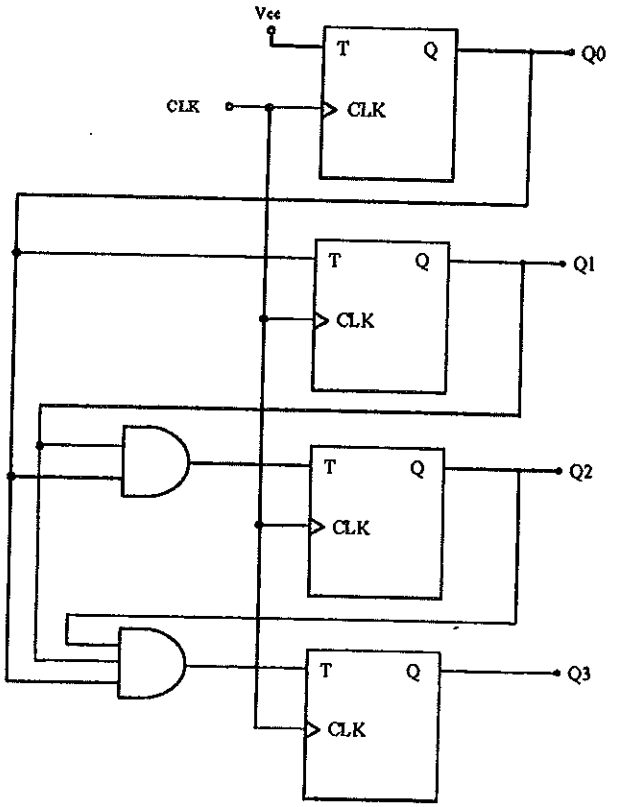
Observaciones para circuitos con más de un biestable:

- 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.

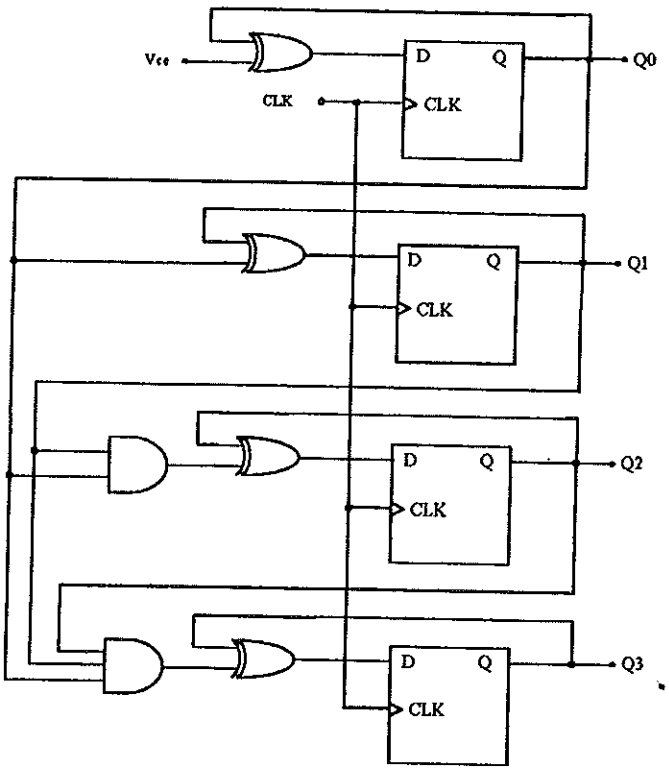
5.4 Contadores

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.

5.4.1 Implementación mediante flip-flops T



5.4.2 Implementación mediante flip-flops D



5.4.3 Contadores comerciales

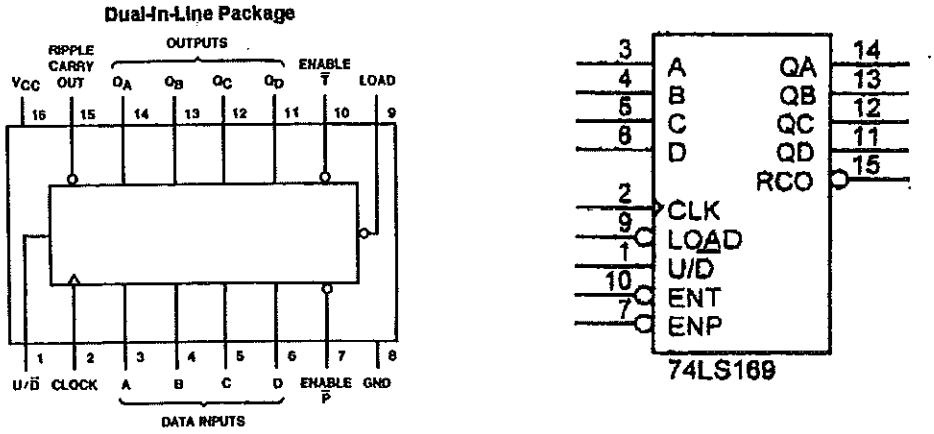
En el caso de los contadores comerciales (como cualquier integrado comercial) el fabricante siempre proporciona una hoja de especificaciones donde aparece un esquema del patillaje y un cronograma (*Timing Diagram*). Este cronograma proporciona normalmente toda la información necesaria para conocer el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

5.4.3.1 Contador 74LS169

El resto de contadores comerciales tienen funcionamientos similares

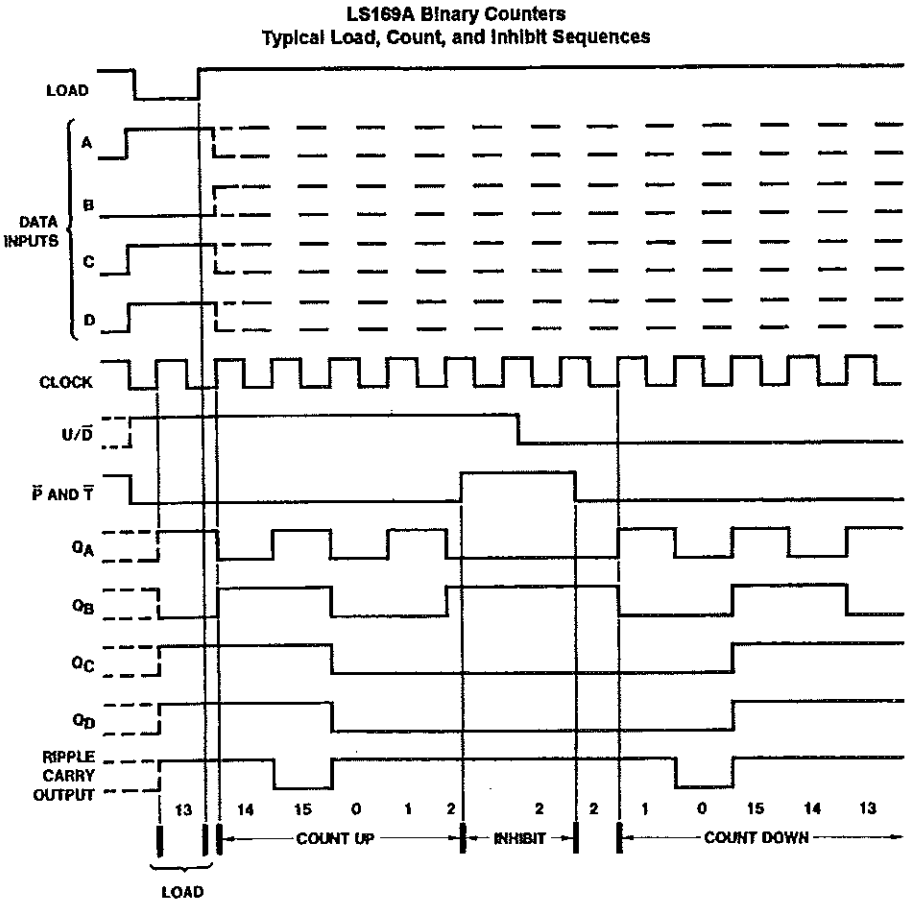
A modo de ejemplo, vamos a estudiar con detalle un modelo determinado, el 74LS169, cuyo esquema y cronograma mostramos a continuación.

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".

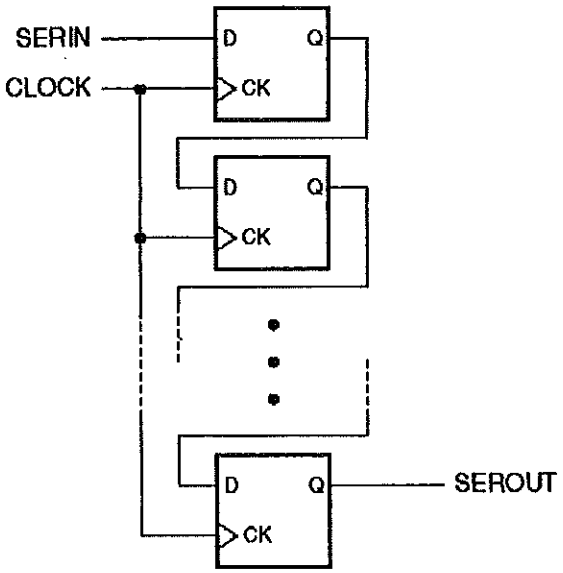
5.5 Registros de desplazamiento

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.

5.5.1 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.

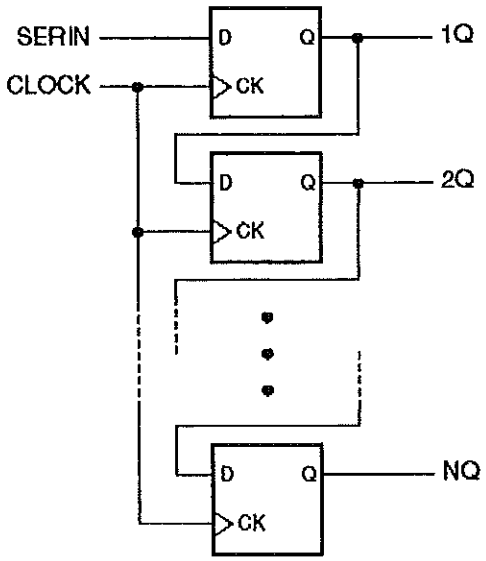


- La entrada serie SERIN especifica un nuevo bit que será desplazado en un extremo para cada tic del reloj.
- Este bit aparece en la salida serie, SEROUT, después de n tics del reloj, y se pierde un tic más tarde.

5.5.2 Registros de desplazamiento de entrada serie y salida paralelo

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

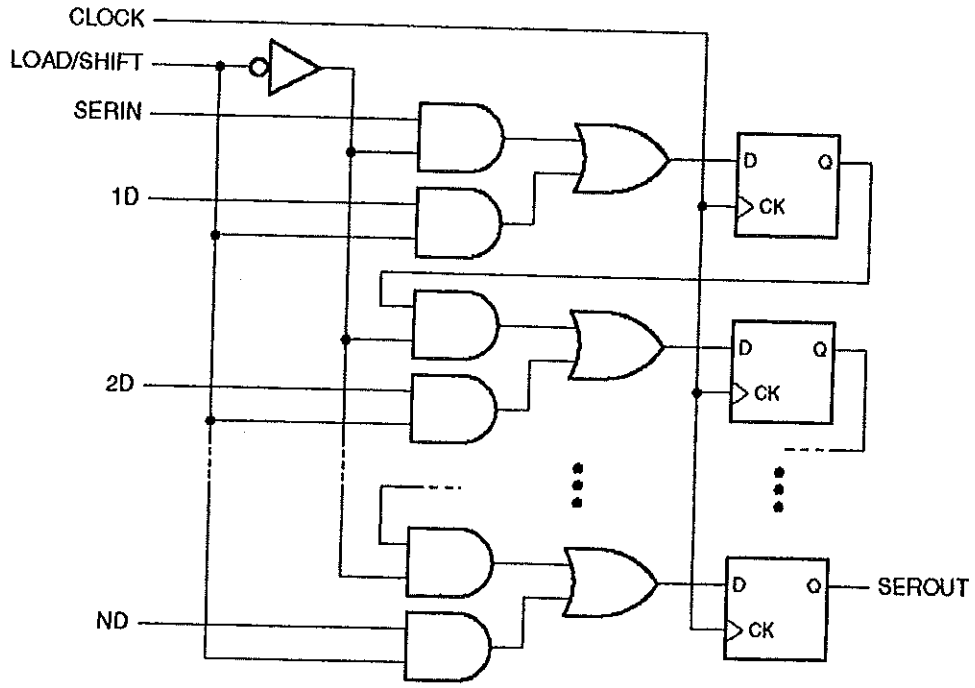
Un registro así puede utilizarse para efectuar la conversión serie a paralelo, muy utilizada en electrónica digital.



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

5.5.3 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:

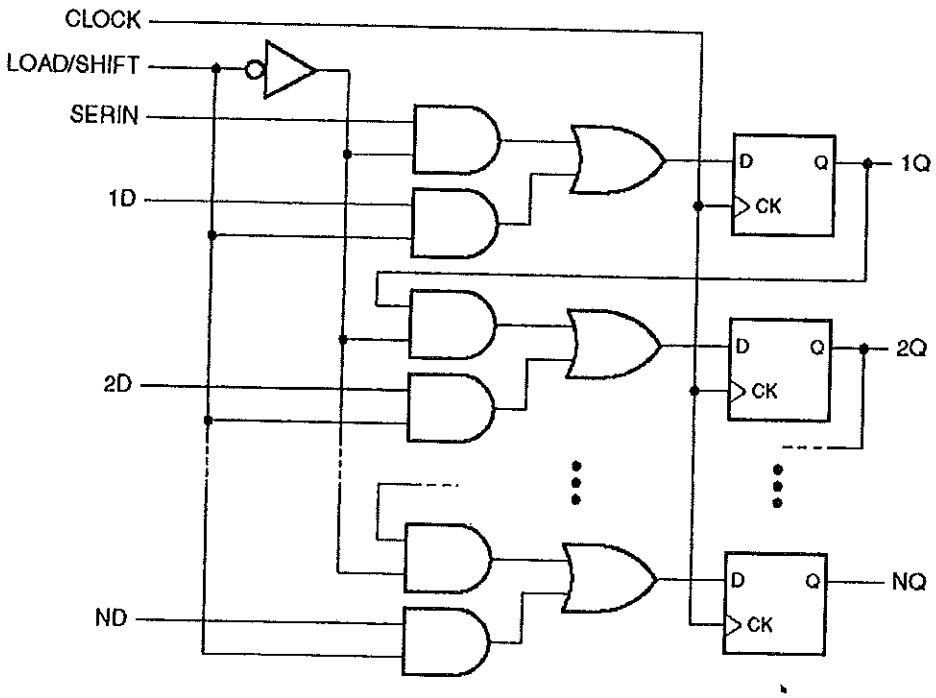


Un registro así puede utilizarse para efectuar la conversión paralelo a serie.

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.

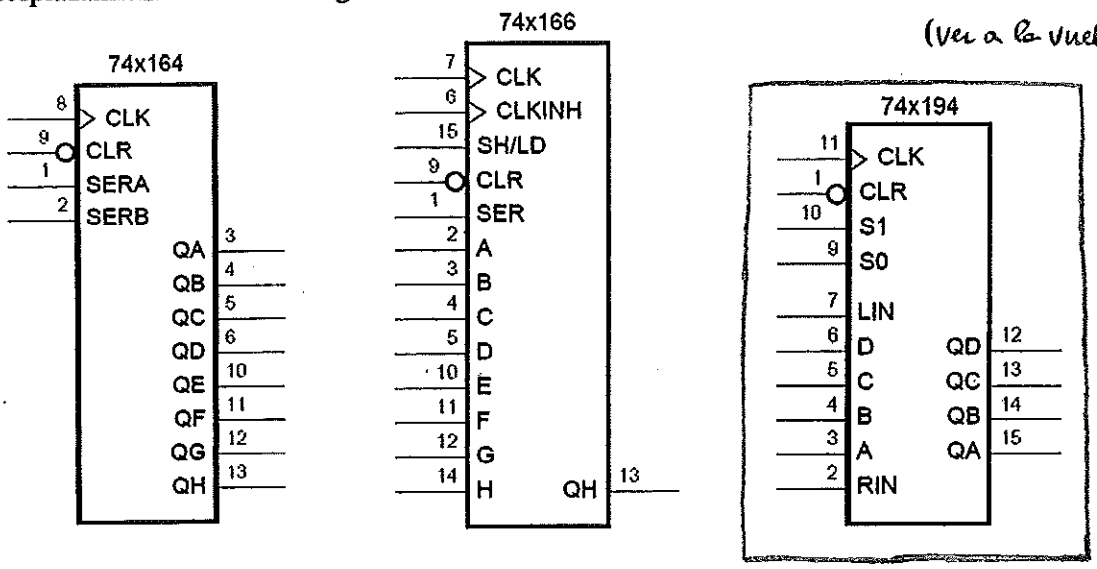
5.5.4 Registros de desplazamiento de entrada paralelo y salida paralelo

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:



5.5.5 Registros comerciales

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



(ver a la vuelta 5)

5.5.5.1 Registro 74x164

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

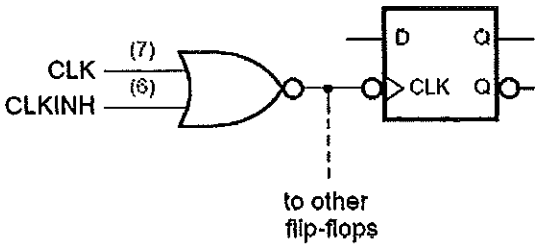
Es un dispositivo de entrada serie y salida paralelo con una entrada de borrado asíncrona (CLR_L). Tiene dos entradas serie que se operan con AND de manera interna, es decir, tanto SERA como SERB deben ser 1 para ser recorridos en el primer bit del registro.

5.5.5.2 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:



5.5.5.3 Registro 74x194

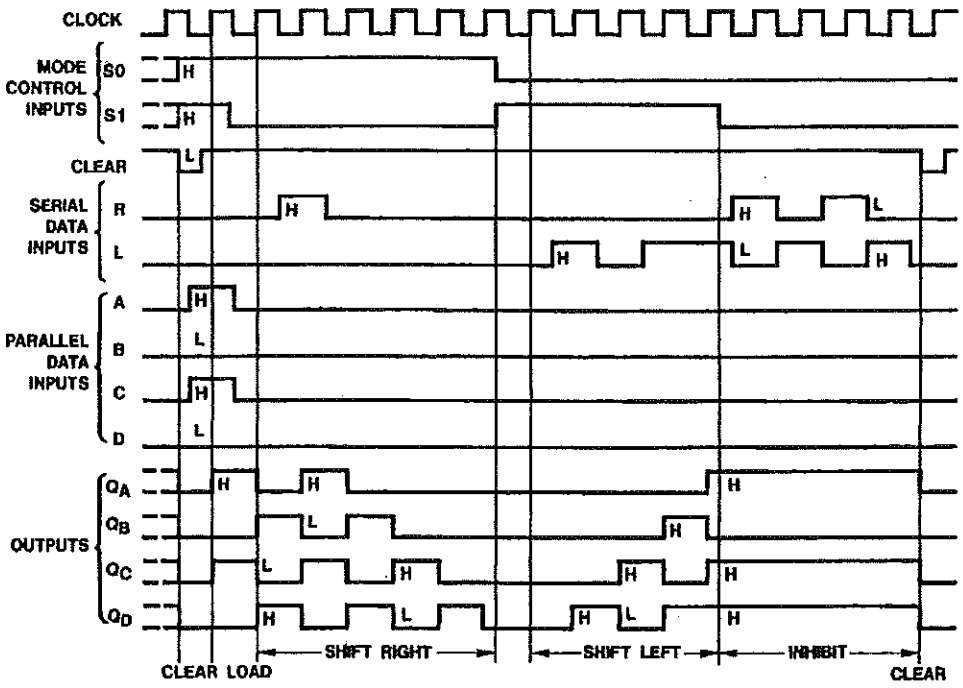
A modo de ejemplo, nosotros vamos a estudiar con más detalle detalle un modelo determinado, el 74LS194. Se trata de un registro de desplazamiento bidireccional de entrada en paralelo y salida en paralelo de 4 bits.

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 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	← LOAD
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.

- 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.

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

- 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.

Es análogo al caso anterior pero en el otro sentido

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 6: TEORÍA DE 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: $f: E \times Q \rightarrow S$

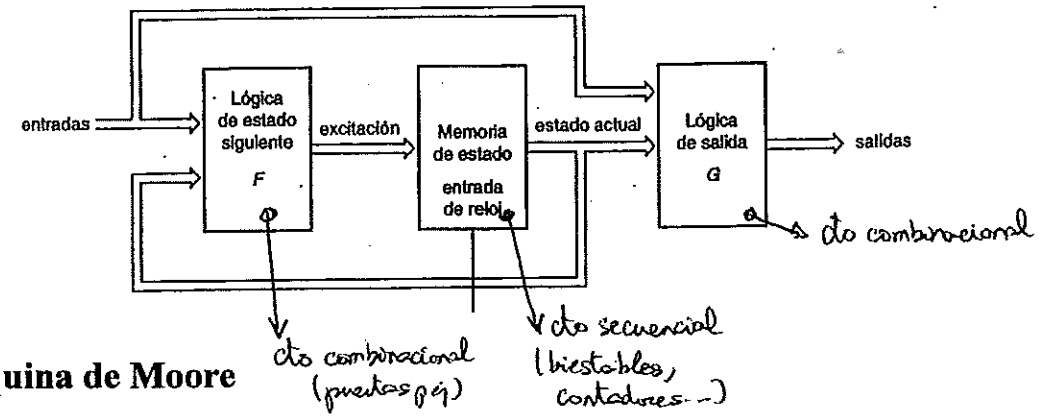
Si el conjunto de estados Q es finito entonces decimos que es un **autómata finito**. Nosotros estudiaremos solamente autómatas finitos.

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.

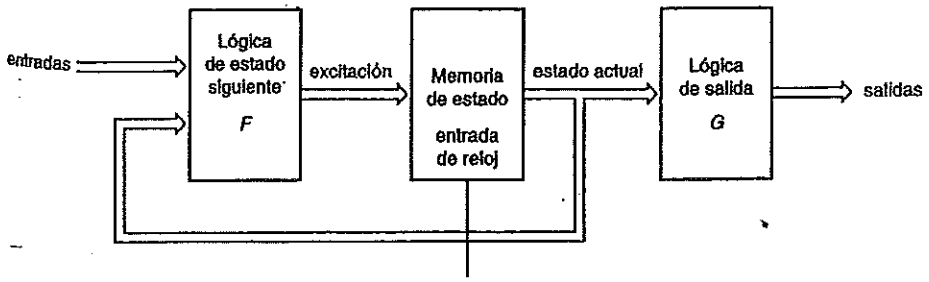
Se puede decir que en una máquina de Mealy las salidas están asociadas a las transiciones, no a los estados.



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.



6.1 Análisis de autómatas

6.1.1 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:

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

- 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.

Máquina de Mealy

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

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.

Máquina de Moore

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

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.

6.1.2 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)	$Q_{t+1} = \overline{Q}_t$

6.1.3 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:

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.

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.
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**.

Este apartado séptimo es opcional.

7. Dibujar un **diagrama de estado** correspondiente a la tabla de estado/salida.

   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.

6.1.4 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.

6.2 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:

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

1. Pasar las especificaciones verbales a un **diagrama de estados.**
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.
6. Simplificar las **ecuaciones de excitación,** a partir de la tabla de excitación.
7. Implementar el circuito.

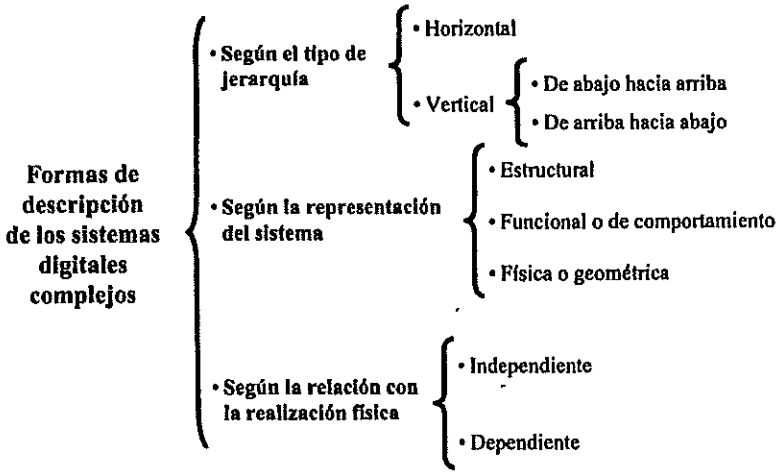
Sólo asimilaremos bien estos pasos cuando los abordemos directamente en ejercicios.

TEMA 7: DESCRIPCIONES FUNCIONALES Y ESTRUCTURALES

7.1 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.

7.1.1 Formas de descripción según la representación del sistema

7.1.1.1 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.

7.1.1.2 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.

7.2 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...).

7.3 Lenguaje VHDL

7.3.1 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.

7.3.2 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

```
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
```

→ definimos la estructura del bicho (las entradas y las salidas)

- Descripción lógica

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

definimos el funcionamiento

asignación de valor

- 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

7.3.3 Bibliotecas y paquetes (esto no entra: es la cabecera y no la dan)

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*...

7.3.4 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

declaramos/ponemos los nombres de las entradas

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

aquí se ponen los tipos de datos

ponemos los nombres de las salidas

Más adelante explicaremos los tipos de datos de VHDL.

Ejemplo:

entradas tipo BIT

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

salidas tipo BIT

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

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

7.3.5 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.

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

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

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
```

aquí declaramos las variables auxiliares que necesitemos

funcionamiento del circuito

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
```

(*) el bit 0 se lo endiño a sum y el bit 1 " " " cout

SIGNAL auxiliar & (1) tipo objeto nombre bit_vector(1 DOWNTO 0) (2) tipo de variable y longitud

aquí está puesto como una asignación de valor (en java a=1)

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)
```

7.3.6 Descripción lógica

7.3.6.1 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:

(2)

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

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

- `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

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

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

7.3.6.2 Tipos de objetos (1)

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

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

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);`

– 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;`

7.3.6.3 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"`

7.3.6.4 Operadores

Operadores Lógicos		Operadores aritméticos	
`and`	función Y lógica	`+`	suma aritmética
`or`	función O lógica	`-`	- resta ritmética (dos operandos) - signo negativo (un operando)
`xor`	función O-exclusiva		
`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

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.

7.3.6.5 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.

→ se pueden ejecutar cuando sea y a la vez: es la gran diferencia con Java y leng. de programación

a) **Process** → siempre que quiera explicar algo con función secuencial = con orden

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
```

→ proceso activado por las 2 señales a y b
a la salida ponemos a AND b

no suele usarse

X 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

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

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)

Ejemplo:

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

c) When ... else

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.

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

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"
```

7.3.6.6 Sentencias secuenciales

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

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).

a) If...then...else

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

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

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.

La sintaxis es la siguiente:

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

Ejemplo:

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
```

flanco activo de reloj

b) Case...when

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;
```

aquí es una implicación

de sintaxis !!

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;
```

en el caso que selección es cuando vale "0000" ó "0001" entonces asigna 0 a y (es decir y = 0)

cuando vale

de asignación !!

de sintaxis (no significa nada)

c) For

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:

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

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;
```

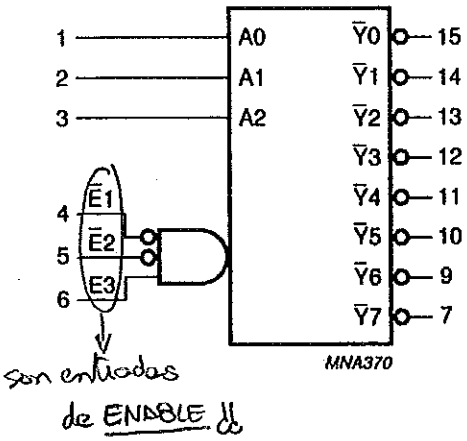
Ejemplo:

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

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

7.3.7 Ejemplos de circuitos combinacionales

7.3.7.1 Decodificador 3 a 8 74xx138



FUNCTION TABLE

INPUTS						OUTPUTS							
E_1	E_2	E_3	A_0	A_1	A_2	Y_0	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_7
H	X	X	X	X	X	H	H	H	H	H	H	H	H
X	H	X	X	X	X	H	H	H	H	H	H	H	H
X	X	L	X	X	X	H	H	H	H	H	H	H	H
L	L	H	L	L	L	L	H	H	H	H	H	H	H
L	L	H	L	L	H	H	L	H	H	H	H	H	H
L	L	H	L	H	L	H	H	L	H	H	H	H	H
L	L	H	L	H	H	H	H	H	L	H	H	H	H
L	L	H	H	L	L	H	H	H	H	L	H	H	H
L	L	H	H	L	H	H	H	H	H	H	L	H	H
L	L	H	H	H	L	H	H	H	H	H	H	L	H
L	L	H	H	H	H	H	H	H	H	H	H	H	L

Cabecera

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

entidad

```
entity V74x138 is
  port (G1, G2A_L, G2B_L: in STD_LOGIC; -- enable inputs
        A: in STD_LOGIC_VECTOR (2 downto 0); -- select inputs
        Y_L: out STD_LOGIC_VECTOR (0 to 7) ); -- decoded outputs
end V74x138;
```

arquitectura

```
architecture V74x138_a of V74x138 is
 signal Y_L_i: STD_LOGIC_VECTOR (0 to 7);
begin
 with A select Y_L_i <=          -- según lo que valga A hacemos una asignación u otra
  "01111111" when "000",
  "10111111" when "001",
  "11011111" when "010",        -- por ejemplo, cuando A valga 010 asigna LLOLLLLL a Y_L_i
  "11101111" when "011",
  "11110111" when "100",
  "11111011" when "101",
  "11111101" when "110",        -- necesario porque A es STD_LOGIC y puede valer más cosas
  "11111110" when "111",
  "11111111" when others;
 Y_L <= Y_L_i when (G1 and not G2A_L and not G2B_L)='1' else   -- solo asignamos el valor Y_L_i a la salida cuando se cumpla que:
  "11111111";                   -- E1 · E2A_L · E2B_L = 1
end V74x138_a;
```

7.3.7.2 Multiplexor de 4 entradas de 8 bits

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

entity mux4in8b is
  port (
    S: in STD_LOGIC_VECTOR (1 downto 0); -- Select inputs, 0-3 ==> A-D
    A, B, C, D: in STD_LOGIC_VECTOR (1 to 8); -- Data bus input
    Y: out STD_LOGIC_VECTOR (1 to 8) -- Data bus output
  );
end mux4in8b;

architecture mux4in8b of mux4in8b is
begin
  with S select Y <=
    A when "00",
    B when "01",
    C when "10",
    D when "11",
    (others => 'U') when others; -- this creates an 8-bit vector of 'U'
end mux4in8b;
```

7.3.7.3 Multiplexor especializado 4 entradas de 3 bits

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

entity mux4in3b is
  port (
    S: in STD_LOGIC_VECTOR (2 downto 0); -- Select inputs, 0-7 ==> ABACADAB
    A, B, C, D: in STD_LOGIC_VECTOR (1 to 18); -- Data bus inputs
    Y: out STD_LOGIC_VECTOR (1 to 18) -- Data bus output
  );
end mux4in3b;

architecture mux4in3p of mux4in3b is
begin
process(S, A, B, C, D)
variable i: INTEGER;
  begin
    case S 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 => 'U'); -- 18-bit vector of 'U'
    end case;
  end process;
end mux4in3p;
```

7.3.7.4 Comparador de 8 bits

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

entity vcompare is
  port (
    A, B: in STD_LOGIC_VECTOR (7 downto 0);
    EQ, NE, GT, GE, LT, LE: out STD_LOGIC
  );
end vcompare;

architecture vcompare_arch of vcompare is
begin
process (A, B)
  begin
    EQ <= '0'; NE <= '0'; GT <= '0'; GE <= '0'; LT <= '0'; LE <= '0';
    if A = B then EQ <= '1'; end if;
    if A /= B then NE <= '1'; end if;
    if A > B then GT <= '1'; end if;
    if A >= B then GE <= '1'; end if;
    if A < B then LT <= '1'; end if;
    if A <= B then LE <= '1'; end if;
  end process;
end vcompare_arch
```

7.3.7.5 Comparador de 8 bits (alternativo)

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

entity comp8 is
  port ( A, B: in STD_LOGIC_VECTOR (7 downto 0);
         EQ, GT: out STD_LOGIC );
end comp8;

architecture comp8_arch of comp8 is
begin
  EQ <= '1' when A = B else '0';
  GT <= '1' when A > B else '0';
end comp8_arch;
```

7.3.7.6 Barrel shifter de 16 bits con desplazamiento circular a la izquierda

```
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;
```

7.3.8 Ejemplos de circuitos secuenciales

7.3.8.1 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;
```

7.3.8.2 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;
```

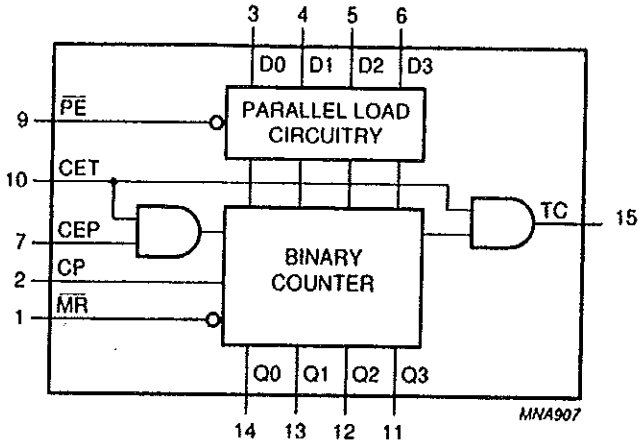
7.3.8.3 Biestable D con Preset y Clear 74xx74

```
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;
```

7.3.8.4 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;
```

7.3.8.5 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;
```

7.3.8.6 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;
```

APÉNDICE 1: PUERTAS LÓGICAS

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

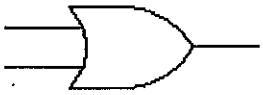
Puerta AND

74HC08

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

Puerta OR

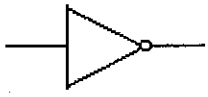
74HC32



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

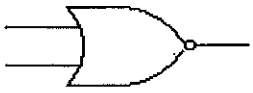
Puerta NAND

74HC00

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

Puerta NOR

74HC02

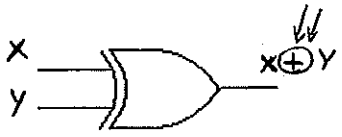


X	Y	X NOR Y
0	0	1
0	1	0
1	0	0
1	1	0

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

(*) Puerta XOR

74HC86



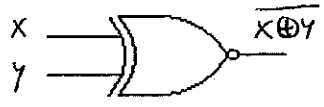
X	Y	X XOR Y
0	0	0
0	1	1
1	0	1
1	1	0

1ª forma canónica

$\bar{X}Y + X\bar{Y} = F$

(*) Puerta XNOR

74HC86



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

1ª forma canónica

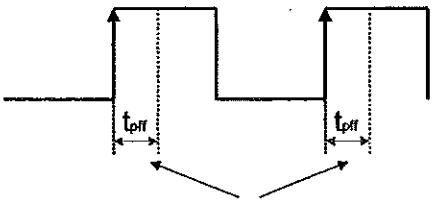
$\bar{X}\bar{Y} + XY = F$

(*) hay que saber identificar en una función cuando aparezcan estos resultados para implementarla con estas puertas (∃ veces, que ∄ mejor opción) !!

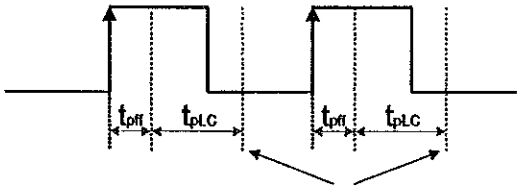
APÉNDICE 2: METAESTABILIDAD, NORMAS DE DISEÑO SÍNCRONO

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.

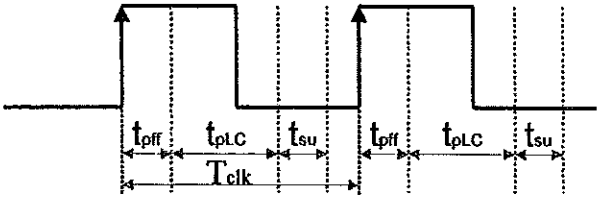


Cambio de estado



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".

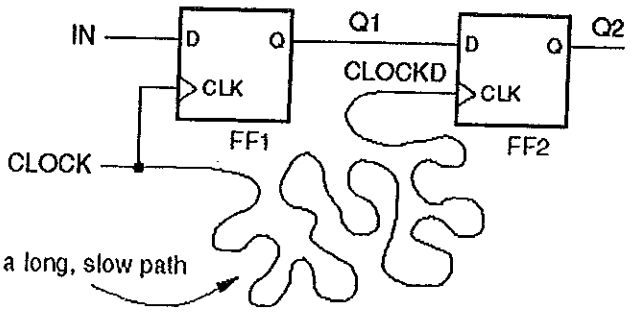
- Se emplean **flip-flops activos en el mismo tipo de flanco** como elementos de memoria del sistema.

- 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.

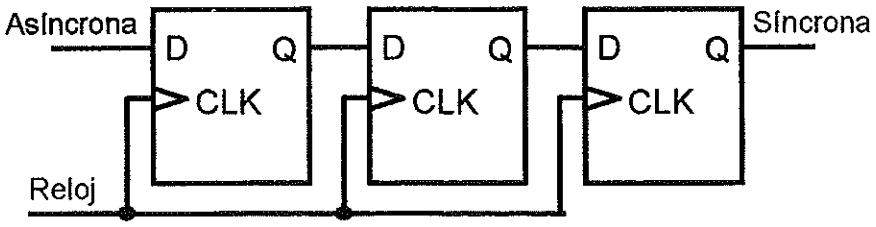
Sincronización de entradas asíncronas

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.

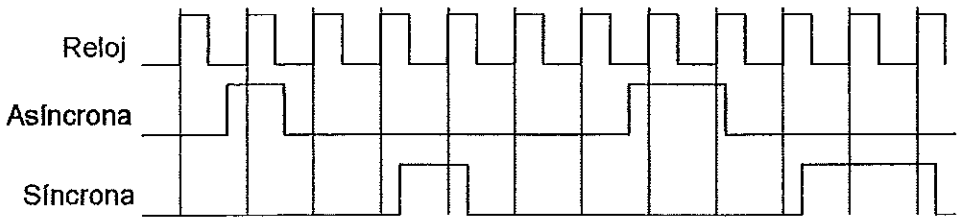
La sincronización consiste en registrar la entrada en un flip-flop conectado al reloj del circuito. Durante esta operación puede ocurrir que se violen los tiempos de set-up o de hold del flip-flop. Como consecuencia, el flip-flop puede registrar o no el evento de entrada o, lo que es peor, entrar en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



CEDG
Ejercicios de clase

TEMA 3: ÁLGEBRA DE CONMUTACIÓN

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)

	A B C	F
(1)	0 0 0	1
(2)	0 0 1	1
(3)	0 1 0	0
(4)	0 1 1	0
(5)	1 0 0	1
(6)	1 0 1	1
(7)	1 1 0	1
(8)	1 1 1	1

$F = \begin{cases} 0 & \text{si } A=0 \wedge B=1 \\ 1 & \text{resto} \end{cases}$

b) b1: primera forma canónica: suma de productos (suma de minterms)

Nos fijamos en las filas de la tabla en las que el valor de la función es 1.

Por cada fila tendremos un término. Variables = 0 niego / = 1 las dejo tal cual

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

(1) $000 \Rightarrow \bar{A}\cdot\bar{B}\cdot\bar{C}$

b2: segunda forma canónica: producto de sumas (suma de maxterms)

Nos fijamos en las filas de la tabla en donde la función vale cero. Para cada fila tendremos un término. las variables = 1 niego / = 0 las dejo tal cual

$F = (A+\bar{B}+C)\cdot(A+\bar{B}+\bar{C})$

(3) $010 \Rightarrow A+\bar{B}+C$

b3: tercera forma canónica: producto de productos

Cogemos la 1ª forma canónica y la negamos 2 veces

$F = \bar{\bar{F}} = \overline{\bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}C + \bar{A}B\bar{C} + \bar{A}BC + AB\bar{C} + ABC} \overset{\text{De Morgan}}{=} (\bar{A}\bar{B}\bar{C})\cdot(\bar{A}\bar{B}C)\cdot(\bar{A}B\bar{C})\cdot(\bar{A}BC)\cdot(AB\bar{C})$

Esta 3ª forma se implementa con puertas NAND!

b4: cuarta forma canónica: suma de sumas

Cogemos la 2ª forma canónica y la negamos dos veces

$F = \bar{\bar{F}} = \overline{(A+\bar{B}+C)(A+\bar{B}+\bar{C})} = \overline{(A+\bar{B}+C)} + \overline{(A+\bar{B}+\bar{C})}$

Esto es lo que usamos cuando implementemos con NOR!

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.

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 0 1 1	1
1 1 0 0	1
1 1 0 1	1
1 1 1 0	1
1 1 1 1	1

← Traducimos el ejercicio a la tabla de verdad

Con la tabla, ahora realizamos las formas canónicas:

1ª forma: $S = \bar{A}BCD + A\bar{B}CD + AB\bar{C}D + ABC\bar{D} + ABCD$

$+ AB\bar{C}\bar{D} + A\bar{B}C\bar{D} + A\bar{B}\bar{C}D + \bar{A}BC\bar{D}$

La expresión es muy extensa como para implementarla con puertas → tenemos que simplificar.

- mediante álgebra Booleana (fatal)
- con el método de Karnaugh

- mapa de Karnaugh

CD \ AB	00	01	11	10
00	0	0	(1)	0
01	0	0	1	(2)
11	0	(3)	1	1 (3)
10	0	0	1	1 (4)

- vamos a simplificar por unos la tabla
- hacemos los sacos (grupos) según las reglas
- por cada saco que hemos obtenido, tendremos un término. Cada término estará formado por el producto de las variables que se mantienen constantes en el recorrido por el saco. Si es 1 la dejamos, si es cero la negamos

saco (1):

A	B	C	D
1	1	0	0
1	1	0	1
1	1	1	1
1	1	1	0

saco (2):

A	B	C	D
1	1	0	1
1	1	1	1
1	0	0	1
1	0	1	1

saco (3):

A	B	C	D
1	1	1	1
1	1	1	0
1	0	1	1
1	0	1	0

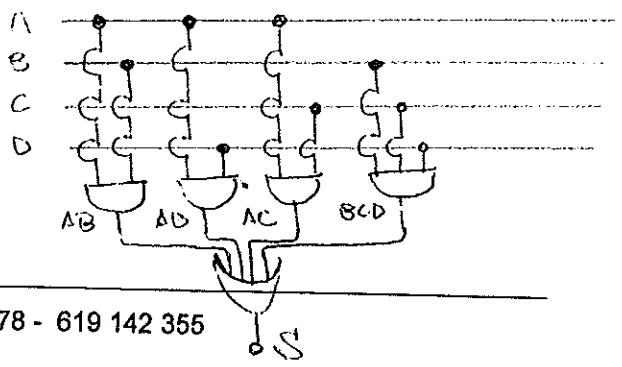
saco (4):

A	B	C	D
0	1	1	1
1	1	1	1

$\Rightarrow$ **$S = \underbrace{AB}_{\text{saco 1}} + \underbrace{AD}_{\text{saco 2}} + \underbrace{AC}_{\text{saco 3}} + \underbrace{BCD}_{\text{saco 4}}$**

Esta expresión es la simplificada del chorizo salido con la 1ª forma canónica

Finalmente, implementamos esta S simplificada con ptas lógicas:



Ejercicio 3

Sea la función booleana definida por:

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

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.

Si nos fijamos en F, tenemos una expresión desarrollada en suma de productos (mintermos !!). Sabemos que en esta expresión cada término corresponde a una fila de la tabla en la que la función vale "1"

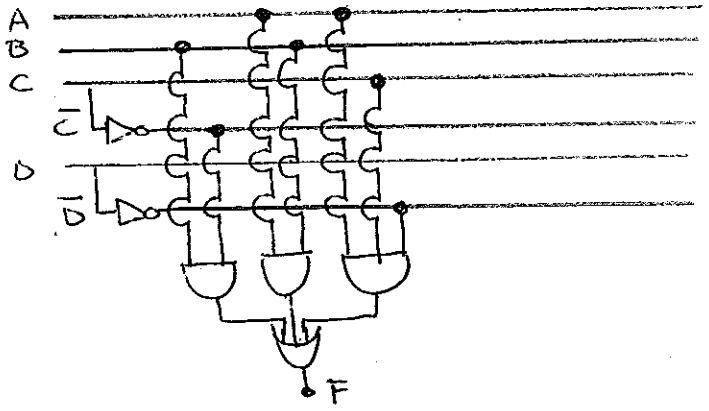
	A	B	C	D	F
	0	0	0	0	0
	0	0	0	1	0
	0	0	1	0	0
	0	0	1	1	0
$\bar{A}B\bar{C}\bar{D}$	0	1	0	0	1
$\bar{A}B\bar{C}D$	0	1	0	1	1
	0	1	1	0	0
	0	1	1	1	0
	1	0	0	0	0
	1	0	0	1	0
$A\bar{B}C\bar{D}$	1	0	1	0	1
	1	0	1	1	0
$AB\bar{C}\bar{D}$	1	1	0	0	1
$AB\bar{C}D$	1	1	0	1	1
$ABC\bar{D}$	1	1	1	0	1
$ABCD$	1	1	1	1	1

a) simplificando por unos

AB \ CD	00	01	11	10
00	0	1 ①	1 ②	0
01	0	1 ①	1 ②	0
11	0	0	1 ② ③	0
10	0	0	1 ③	1

①: B=1, C=0
②: A=1, B=1
③: A=1, C=1, D=0

$$F=B\bar{C}+AB+AC\bar{D}$$

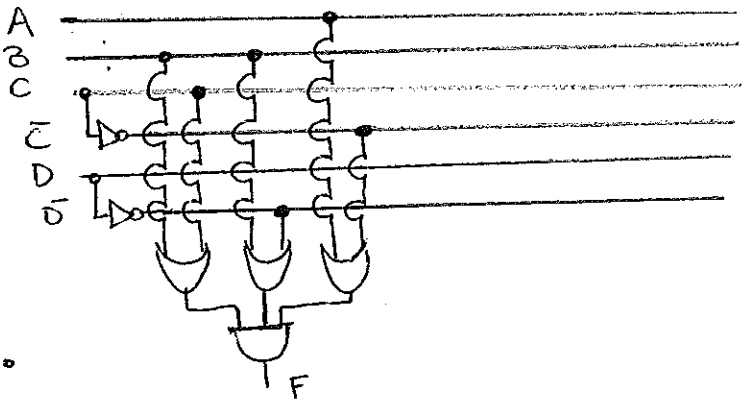


b)

AB \ CD	00	01	11	10
00	0	1	1	0 ②
01	0	1	1	0
11	0	0	1	0
10	0 ③	0 ③	1	1

①: B=0, C=0
②: B=0, D=1
③: A=0, C=1

$$F=(B+C)\cdot(B+\bar{D})\cdot(A+\bar{C})$$

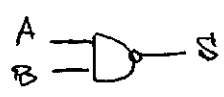


c) $\begin{cases} F = B\bar{C} + AB + A\bar{C}\bar{D} \rightarrow \text{por unos} \\ F = (B+C)(B+\bar{D})(A+\bar{C}) \rightarrow \text{por ceros} \end{cases}$, simplificación y forma canónica !!

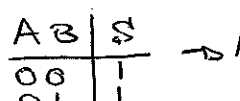
⇒ obtenemos la expresión obtenida en la simplificación por unos y negamos 2 veces (lo mismo con ceros para d))

$$F = \overline{\overline{B\bar{C} + AB + A\bar{C}\bar{D}}} \overset{\text{De Morgan}}{=} \overline{(\overline{B\bar{C}}) \cdot (\overline{AB}) \cdot (\overline{AC\bar{D}})}$$

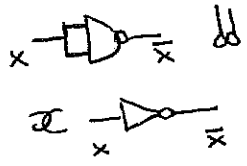
- ¿Cómo hacemos un negador (NOT) a partir de una NAND?



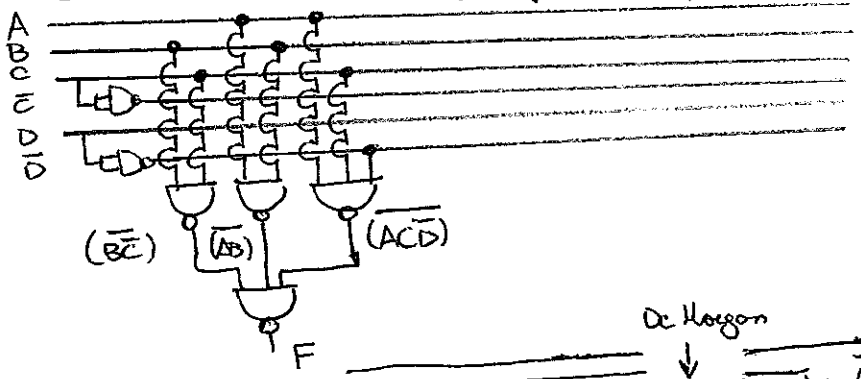
A	B	S
0	0	1
0	1	1
1	0	1
1	1	0



A	A	S
0	0	1
1	1	0



Electrónicamente consume y chupa menos que un NOT

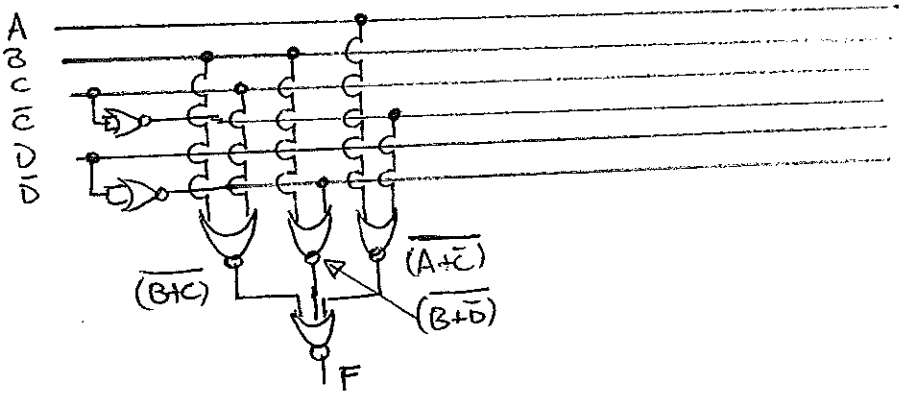


d) Sólo con NOR ⇒ $F = \overline{(B+C)(B+\bar{D})(A+\bar{C})} \overset{\text{De Morgan}}{=} \overline{(\overline{B+C}) + (\overline{B+\bar{D}}) + (\overline{A+\bar{C}})}$

¿Cómo hacemos un negador (NOT) a partir de NOR?

A	B	S
0	0	1
0	1	0
1	0	0
1	1	0

A	A	S
0	0	1
1	1	0

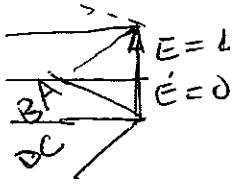


Ejercicio 4

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

Esta tabla, nos hacemos 2 mapas de Karnaugh con E (imagino 3D)



E=0

DC \ BA	00	01	11	10
00	1	0	0	1
01	1	1	0	0
11	0	1	0	1
10	0	0	0	0

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

sacas 3D

resto 2D

B=0, C=1

D=0, C=0, A=0

E=0, D=0, C=1, A=0

E=1, D=0, C=0, B=1

E=1, D=1, C=1, A=1

$$F = B\bar{C} + \bar{A}\bar{C}\bar{D} + \bar{A}CD\bar{E} + B\bar{C}\bar{D}\bar{E} + ACDE$$

Nota, al agrupar los 1s tenemos que cumplir las mismas normas vistas para mapas de Karnaugh de 4 y 3 variables

TEMA 4: 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) $F(a,b,u,w,x,y) = abu + a\bar{b}x + \bar{a}by + \bar{a}\bar{b}w$

a) $F = \bar{a}b + a\bar{b}$

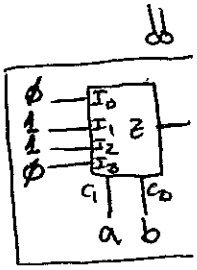
- forma estándar salida MUX 4x2 $\Rightarrow z = \bar{C_1}\bar{C_0}I_0 + \bar{C_1}C_0I_1 + C_1\bar{C_0}I_2 + C_1C_0I_3$

$\Rightarrow$ Como coinciden el nº de variables con el nº de entradas de control (2) asignamos a cada entrada de control una variable e identificamos

- identificando con nuestras variables

$$\begin{cases} F = \bar{a}\bar{b}(\emptyset) + \bar{a}b(1) + a\bar{b}(1) + ab(\emptyset) \\ z = \bar{C_1}\bar{C_0}I_0 + \bar{C_1}C_0I_1 + C_1\bar{C_0}I_2 + C_1C_0I_3 \end{cases}$$

$\Rightarrow$



Nota: La función que nos piden implementar es la operación OR EXCLUSIVA (XOR) $\Rightarrow F = \bar{a}b + a\bar{b} = a \oplus b$

b) $F = ab + ac + \bar{b}c$

$\Rightarrow$ Tenemos una variable más, que entradas de control $\Rightarrow$ Vamos a tener que conectar una de las variables a las entradas de datos

- A bolea, elegimos a y c como entradas de control $\Rightarrow$ Tenemos que poner en cada término que no aparezcan las variables de control, éstas

$1 = \bar{x} + x$

$F = ab + ac + \bar{b}c = ab(c+\bar{c}) + ac + \bar{b}c(a+\bar{a}) =$

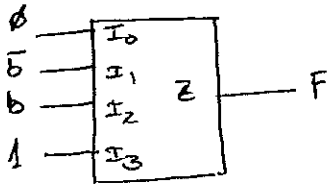
(falta c, falta a)

$= abc + ab\bar{c} + ac + a\bar{b}c + \bar{a}\bar{b}c =$

$= ac(b + 1 + \bar{b}) + a\bar{c}b + \bar{a}c\bar{b} = \boxed{ac + a\bar{c}b + \bar{a}c\bar{b} = F}$

- identificamos

$$\begin{cases} z_1 = \bar{C_1}\bar{C_0}I_0 + \bar{C_1}C_0I_1 + C_1\bar{C_0}I_2 + C_1C_0I_3 \\ F = \bar{a}\bar{c}\,\emptyset + \bar{a}c\,\bar{b} + a\bar{c}\,b + ac\,1 \end{cases} \leftarrow F = ac + a\bar{c}b + \bar{a}c\bar{b}$$



Nota si en el enunciado dijeran que NO podemos usar ninguna otra puerta, esta configuración NO valdría. Tendríamos que probar otra asignación de variables

$F = ab + ac + \bar{b}c$

⇒ <u>vamos a poner a y b como entradas de control</u>

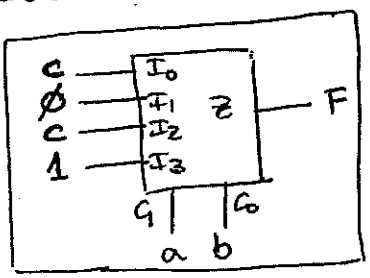
$x + x = x$ ‼

$$F = ab + ac(b+\bar{b}) + \bar{b}c(a+\bar{a}) = ab + abc + a\bar{b}c + a\bar{b}c + \bar{a}\bar{b}c =$$

$$= ab(1+c) + a\bar{b}c + \bar{a}\bar{b}c = ab + a\bar{b}c + \bar{a}\bar{b}c$$

<u>identificamos</u>

$$\begin{cases} z = \bar{G}_1\bar{G}_0 I_0 + \bar{G}_1 G_0 I_1 + G_1\bar{G}_0 I_2 + G_1 G_0 I_3 \\ F = \bar{a}\bar{b}\underset{\smile}{c} + \bar{a}b\underset{\smile}{\emptyset} + a\bar{b}\underset{\smile}{c} + ab\underset{\smile}{1} \end{cases}$$

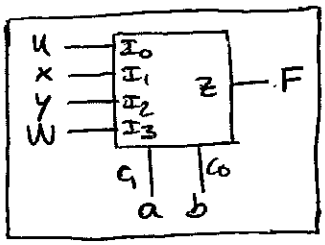


‼

Nota: como mucho en un MUX podremos programarlo siempre y haya solo una variable más a implementar como en el caso anterior; a no ser que esté preparado como en el siguiente ejercicio

c) $F = ab\underset{\smile}{u} + a\bar{b}\underset{\smile}{x} + \bar{a}b\underset{\smile}{y} + ab\underset{\smile}{w}$

$z = \bar{G}_1\bar{G}_0\underset{\smile}{I_0} + \bar{G}_1 G_0\underset{\smile}{I_1} + G_1\bar{G}_0\underset{\smile}{I_2} + G_1 G_0\underset{\smile}{I_3}$ } identificamos ⇒



‼

TEMA 5: CIRCUITOS SECUENCIALES

Ejercicio 1 | **Diseñar un biestable J-K utilizando un biestable T con entrada de reloj.**

Tabla del J-K

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

variables de estado → nos fijamos en lo que pasa de Q_t a Q_{t+1}

En aquellos filas en las que cambia la salida de Q_t a Q_{t+1}, en la función T escribimos un 1.

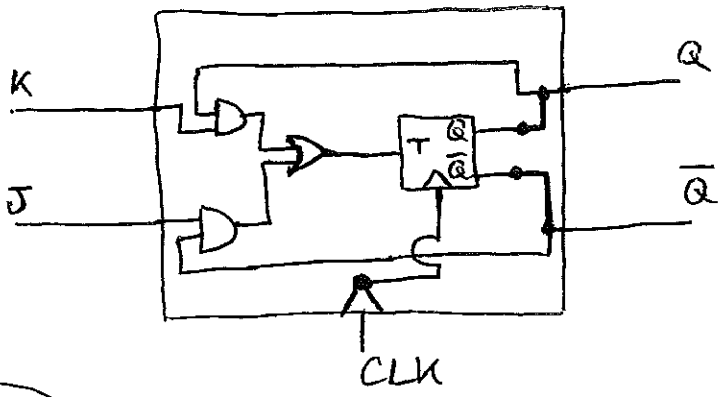
T	Q_{t+1}
0	Q_t
1	$\overline{Q_t}$

= tabla de verdad de un biestable T

Ahora implementamos la función lógica T (simplificando por Karnaugh)

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

$$T = \overline{Q_t} \cdot J + Q_t K$$



How to queremos J-K implementado con T ⇒

⇒ ponemos la tabla de funcionamiento de un JK y a continuación ponemos la columna T. Ahora se trata de ver qué pasa de Q_t a Q_{t+1} para poderlo relacionar con el funcionamiento de un T.

q' | Q_t | Q_{t+1} |
|---|---|
| 0 | 0 | → se conserva ⇒ T = 0 |
| 1 | 0 | → cambia ⇒ T = 1 |

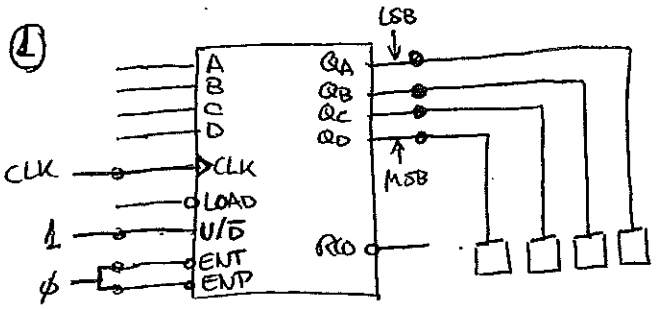
Ahora implementamos la función T en función de nuestras variables de estado, para este caso, J, K y Q_t

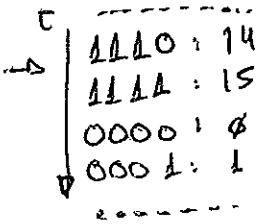
Ejercicio 3

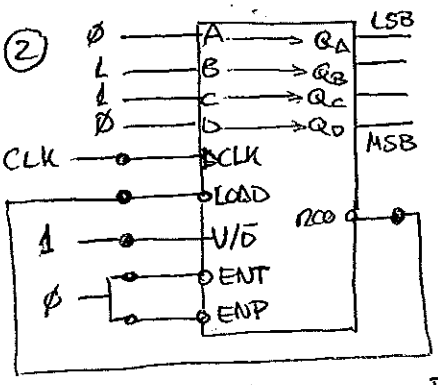
ver anexo

Utilizando un contador 74LS169 y la lógica adicional que precise realice circuitos que cumplan las siguientes especificaciones. No tenga en cuenta la inicialización de los mismos.

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. → igual q 1) pero cargas φ φ φ
4) Contador que cuente de 6 a 11 cíclicamente.
5) Contador que cuente de 15 a 0 cíclicamente. → igual a 1) pero con $\bar{L} = U/\bar{D}$
6) Contador que cuente de 15 a 6 cíclicamente. → mismo q 4) " " " "
7) Contador que cuente de 11 a 0 cíclicamente. → igual q 3º pero $U-\bar{D} = \phi$ y cargas 1 φ 1
8) Contador que cuente de 11 a 6 cíclicamente. → 2 detectores para 6 y 11 + búsqueda R-S → a similar a 9)
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.



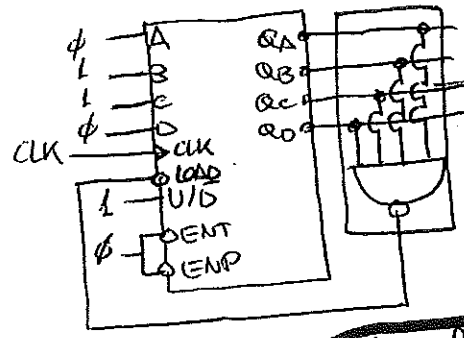




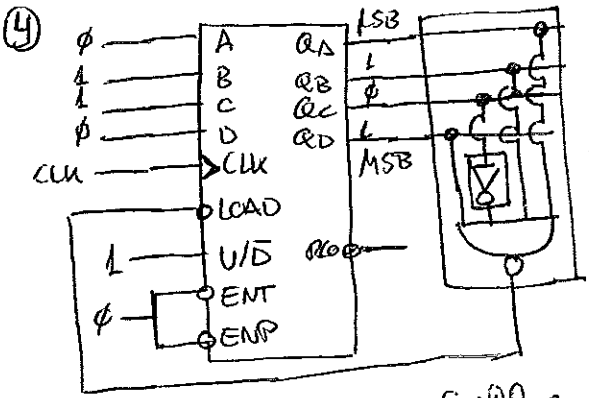
EXTRA: sin usar RCO

cuando $Q_D\ Q_C\ Q_B\ Q_A$ = 1 1 1 1 ⇒ RCO = φ ⇒ LOAD = φ

siguiente φ 1 1 φ ⇒ RCO = 1 ⇒ LOAD = 1 sigue contando



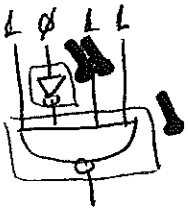
cto detector de tope de cuenta (1111)



la regla general para detectar unos es enchufar una NAND (ver 4 salvo que un φ). Donde sea un φ, le metemos un negador

ej/ queremos detectar un D'11 → B'1011

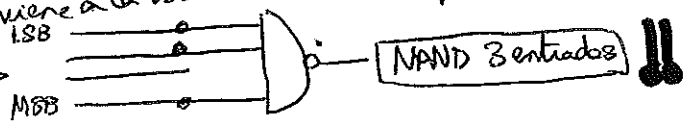
⇒



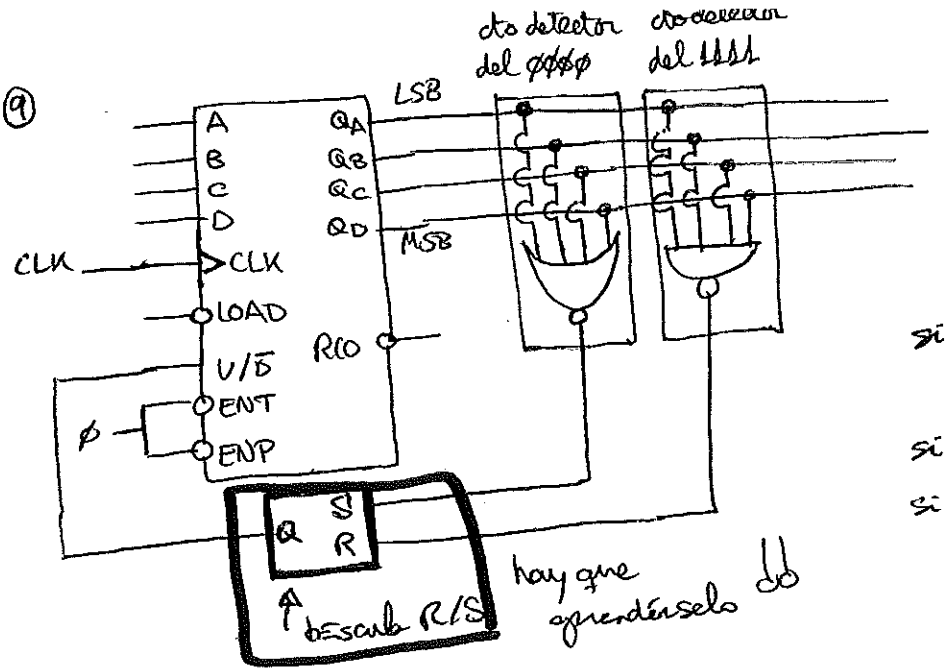
// Como vamos del 6 al 11:

0	1	1	0 : 6
0	1	1	1 : 7
1	0	0	0 : 8
1	0	0	1 : 9
1	0	1	0 : 10
1	0	1	1 : 11

basta con comprobar que por las 3 líneas nos viene a la vez un 1



⑨



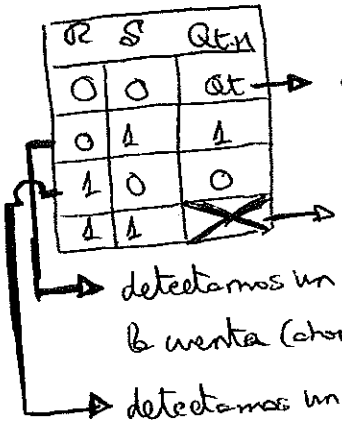
si $R\,S = 0 \Rightarrow$ conserva lo q haya en Q

si $S = 1 \Rightarrow$ pone en Q un 1

si $R = 0 \Rightarrow$ pone en Q un 0

Báscula R-S

Usamos una R-S y no un biestable J-K pq no queremos que la señal dependa del reloj: queremos que la señal U/D esté preparada ANTES de que venga el ciclo (en cuanto activemos R-S)



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

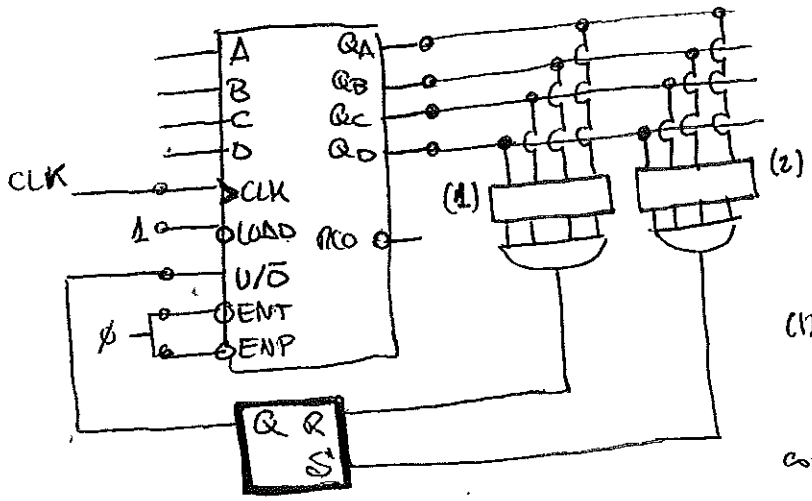
→ estamos en un nº que no es ni 0 ni 15 (seguimos la cuenta en el mismo sentido)

→ esta situación NO se puede dar NUNCA (no se van a detectar a la vez 0 y 15 por el dto)

→ detectamos un 0 (bajando) y cambiamos la cuenta (ahora hacia arriba)

→ detectamos un 15 (subiendo) y cambiamos la cuenta, ahora irá hacia abajo

(10,11,12)

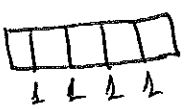


(10) 13, 14, 15, 14, 13 ... 7, 6, 7, 8, 9

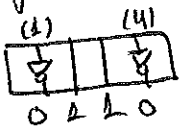
tope "pa arriba" — tope "pa abajo"

Vamos a implementar con el caso general, aunque ∃ mejores optimizaciones

(1)



(2)



↓ mejor

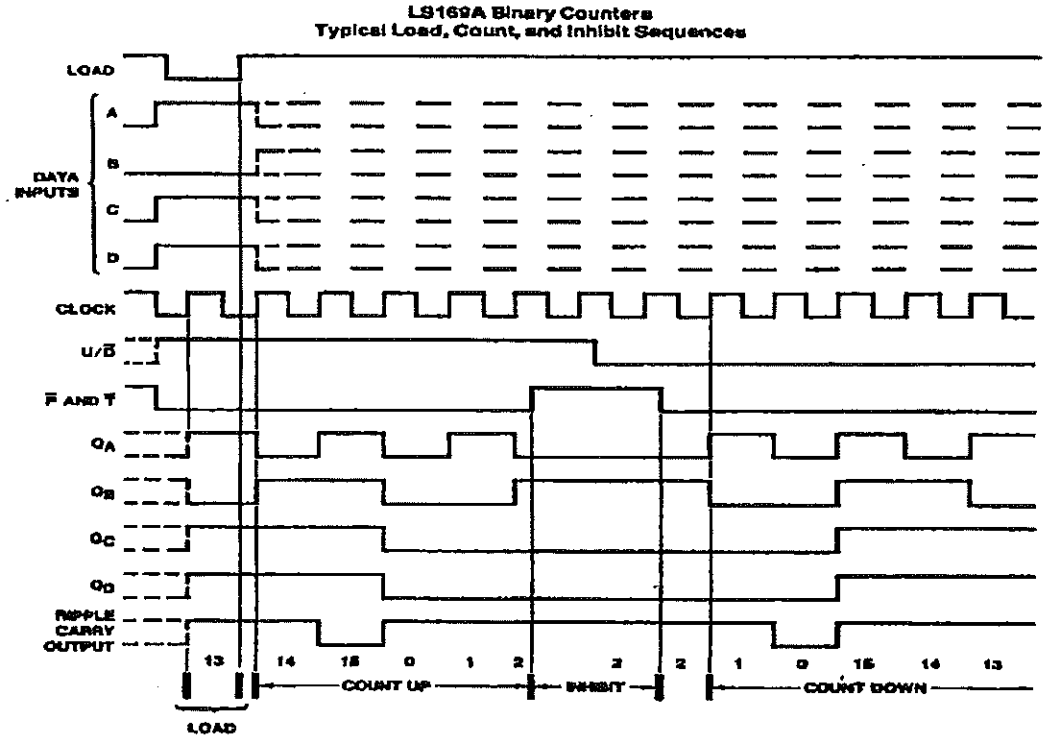
como 6: 0110
7: 0111
8: 1000
9: 1001
10: 1010
11: 1011
12: 1100
13: 1101
14: 1110
15: 1111

→ solo aparecen en secuencia estos 2 ceros de

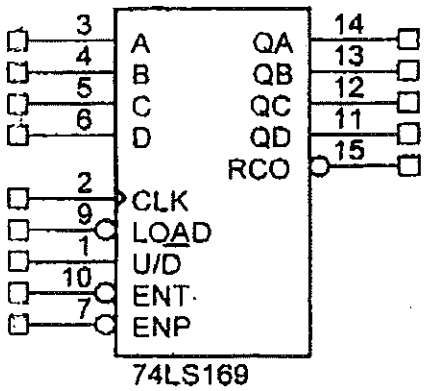
DATOS PARA LOS EJERCICIOS 3 y 4

CRONOGRAMA DEL CONTADOR 74LS169

Timing Diagram



DESCRIPCIÓN DEL CONTADOR 74LS169



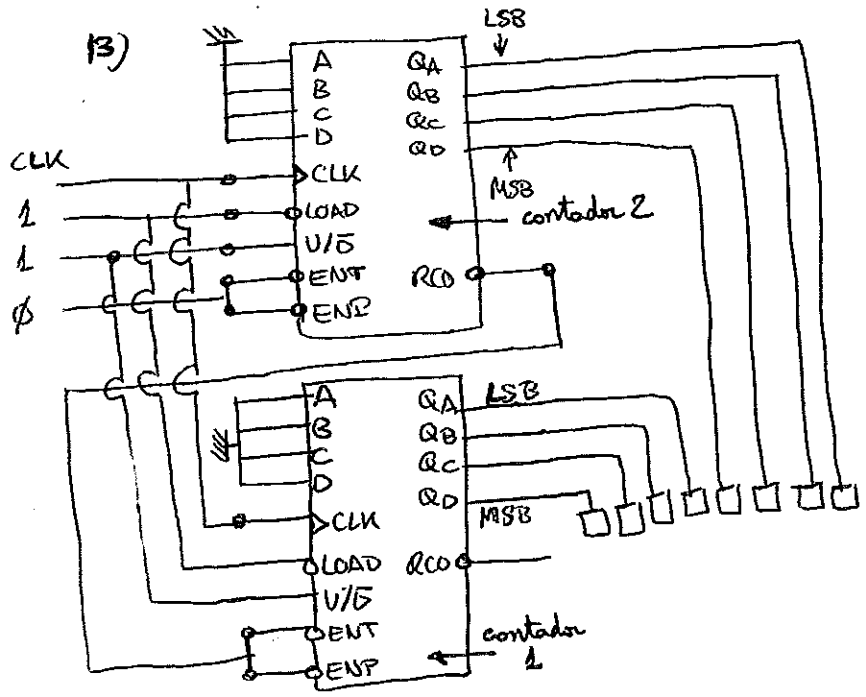
ver anexo

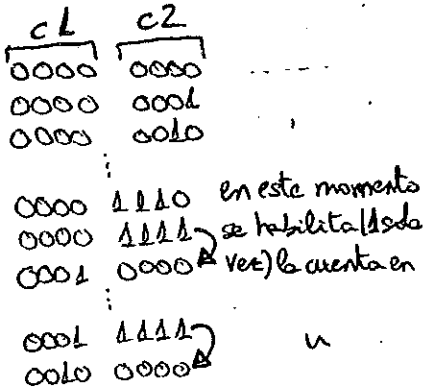
Ejercicio 4

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.

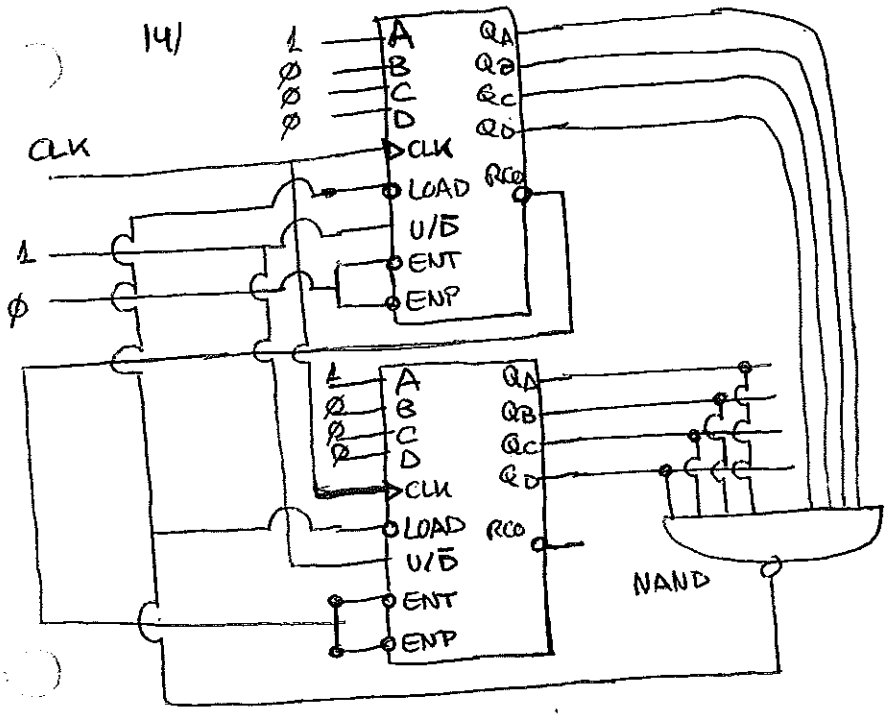
13)



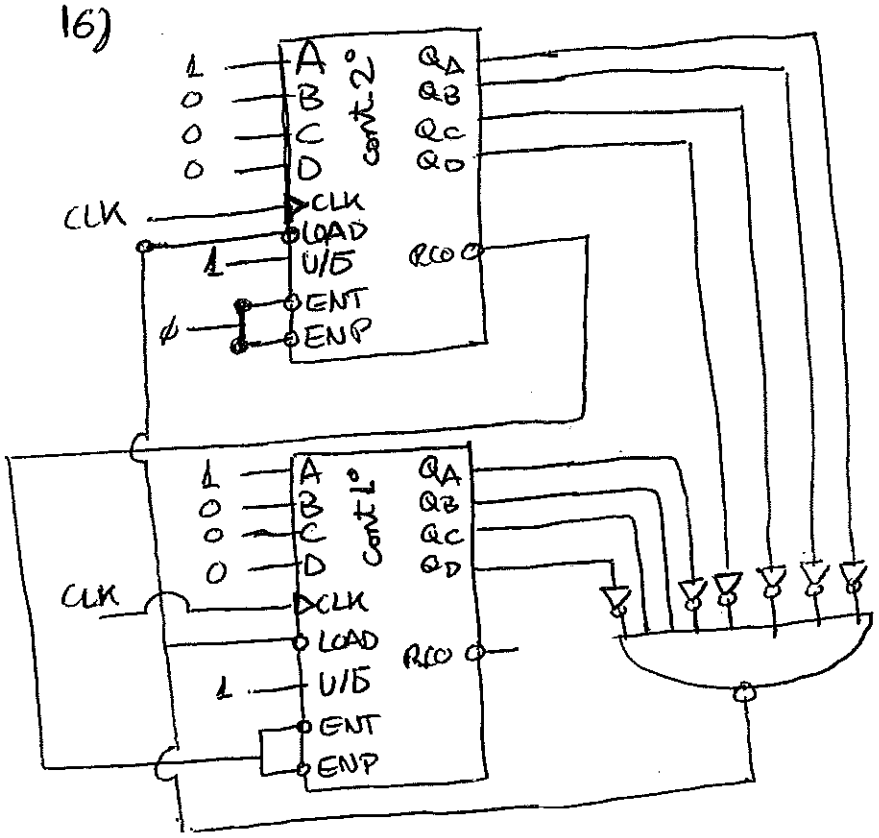


Las entradas que no usemos ⇒
⇒ las ponemos a tierra (activas)

14)



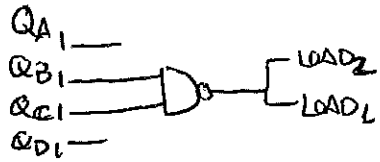
16)



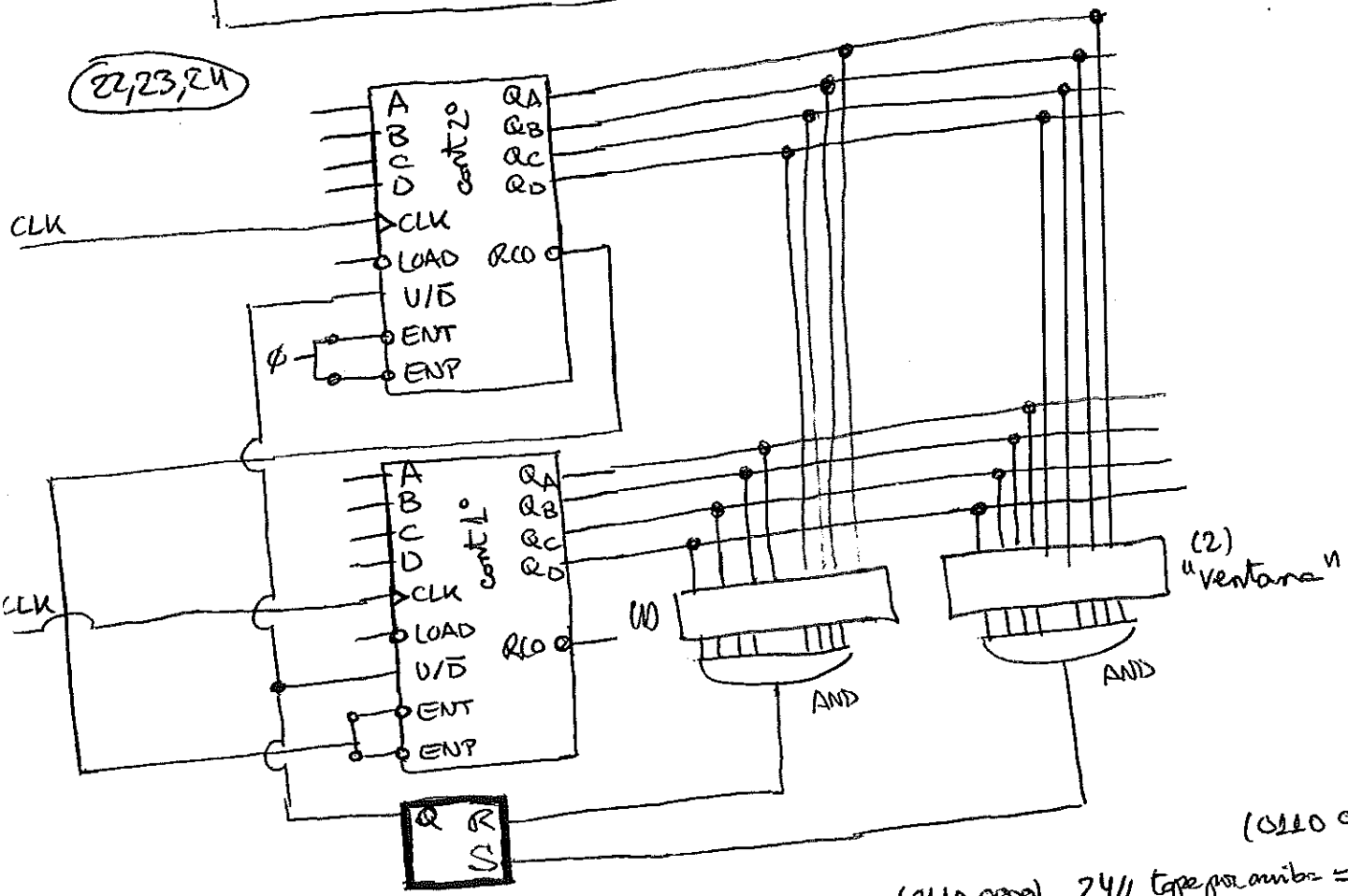
detectamos: 0110 0000

cargamos: 0001 0001

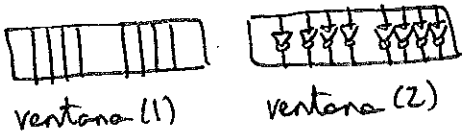
realmente solo nos hace falta detectar estas 2 unos



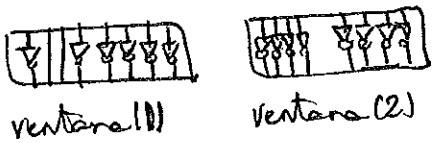
22,23,24



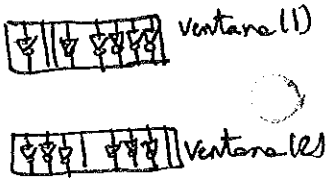
22// tope por arriba = 255 (1111 1111)
a ↘ abajo = 17 (0001 0001)



23// tope por arriba = 96 (0110 0000)
a ↘ abajo = 0 (0000 0000)



24// tope por arriba = 96 (0110 0000)
tope por abajo = 17 (0001 0001)



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 actual $Q_3\ Q_2\ Q_1\ Q_0$	Q_{t+1} estado next $Q_3 Q_2\ Q_1 Q_0$	Biest 3 $J_3\ K_3$	Biest 2 $J_2\ K_2$	Biest 1 $J_1\ K_1$	Biest ϕ $J_\phi\ K_\phi$
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	1 1 1 1	X 0	X 0	X 0	1 X
1 1 1 1	0 0 0 0	X 1	X 1	X 1	X 1

tabla normal J-K ¿que pongo en Q_{t+1}?

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

tabla de transiciones

tabla p. 57

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

tabla de excitaciones

¿q pongo en J-K?

⇒ las variables son las del estado actual

⇒ Nuestro trabajo es implementar cada una de las funciones desde J_3, K_3 hasta J_ϕ, K_ϕ de la manera + simplificada

** $J_0 = 1$, $K_0 = 1$

** nos fijamos en Q_0 y en las columnas J_1 y K_1. Como en las X podemos poner lo que queramos ⇒ $Q_\phi = J_1 = K_1$

** J_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

elegimos los valores de X para que las agrupaciones de unos sean las + grandes posibles

Q_3Q_2 \ Q_1Q_0	00	01	11	10
00	X	X	X	X
01	0	0	1	0
11	0	0	1	0
10	X	X	X	X

K_2 ⇒ $J_2 = Q_1 \cdot Q_0$

$K_2 = Q_1 \cdot Q_0$

** J_3:

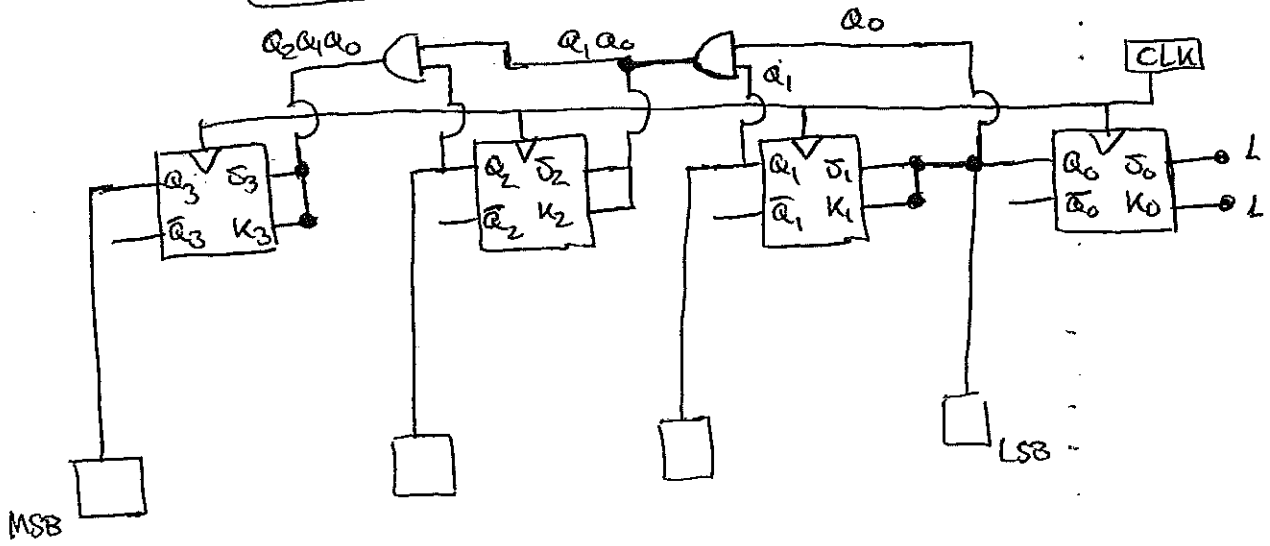
Q_3Q_2 \ Q_1Q_0	00	01	11	10
00	0	0	0	0
01	0	0	1	0
11	X	X	X	X
10	X	X	X	X

K_3:

Q_3Q_2 \ Q_1Q_0	00	01	11	10
00	X	X	X	X
01	X	X	X	X
11	0	0	1	0
10	0	0	0	0

$J_3 = Q_2Q_1Q_0$

$K_3 = Q_2Q_1Q_0$



TEMA 6: TEORÍA DE AUTÓMATAS

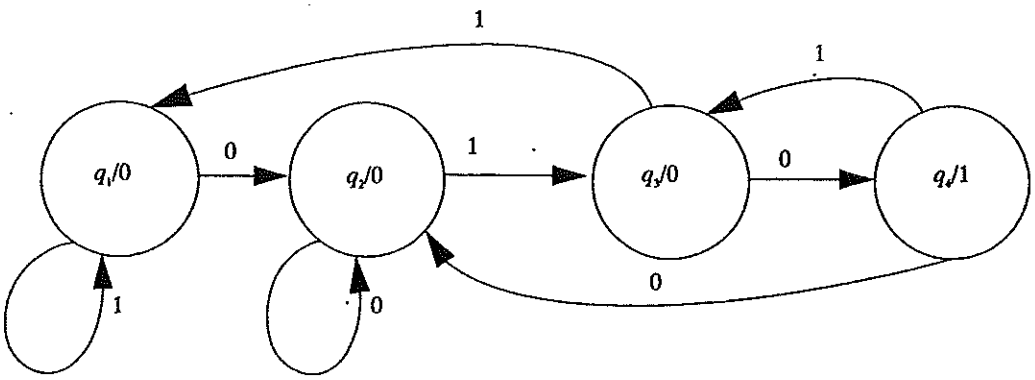
Ejercicio 1

Diseñar un circuito secuencial con una entrada y una salida que reconozca la cadena "010". utilizando biestables J-K

Definimos los siguientes estados:

q_1 = "El último bit ha sido un 1"
q_2 = "El último bit ha sido un 0"
q_3 = "Los dos últimos bits han sido 01 en ese orden"
q_4 = "Los tres últimos bits han sido 010 en ese orden"

El diagrama de Moore del Autómata finito reconocedor de la cadena "010" es el siguiente.



Como hay cuatro estados necesitaremos dos biestables. Hagamos la siguiente asignación de estados $q_1 = 00$; $q_2 = 01$; $q_3 = 10$; $q_4 = 11$. Con ello podemos escribir la tabla de transiciones en binario y, al mismo tiempo, para cada transición, las entradas de los biestables. Implementaremos el circuito con biestables JK.

En un autómata de Moore cada estado tiene una única salida asociada

e	$q(t)$		s	$q(t+1)$		J_1	K_1	J_2	K_2
0	0	0	0	0	1	0	x	1	x
0	0	1	0	0	1	0	x	x	0
0	1	0	0	1	1	x	0	1	x
0	1	1	1	0	1	x	1	x	0
1	0	0	0	0	0	0	x	0	x
1	0	1	0	1	0	1	x	x	1
1	1	0	0	0	0	x	1	0	x
1	1	1	1	1	0	x	0	x	1

Minimizamos por Karnaugh las cuatro ultimas columnas que son las correspondientes a los biestables J-K. Llamamos Q_1 y Q_2 a las (salidas de los biestables) al primero y segundo dígito de q.

Para J_1 :

	00	01	11	10
0	0	0	x	x
1	0	1	x	x

$$J_1 = e \cdot Q_2$$

Para K_1 :

	00	01	11	10
0	x	x	1	0
1	x	x	0	1

$$K_1 = \bar{e} \cdot Q_2 + e \cdot \overline{Q_2} = e \oplus Q_2$$

Para J_2 :

e \ Q_1Q_2	00	01	11	10
0	1	x	x	1
1	0	x	x	0

$$J_2 = \overline{e}$$

Se puede ver a ojo, sin utilizar Karnaugh, en la tabla de transiciones.

Para K_2 :

e \ Q_1Q_2	00	01	11	10
0	x	0	0	x
1	x	1	1	x

$$K_2 = e$$

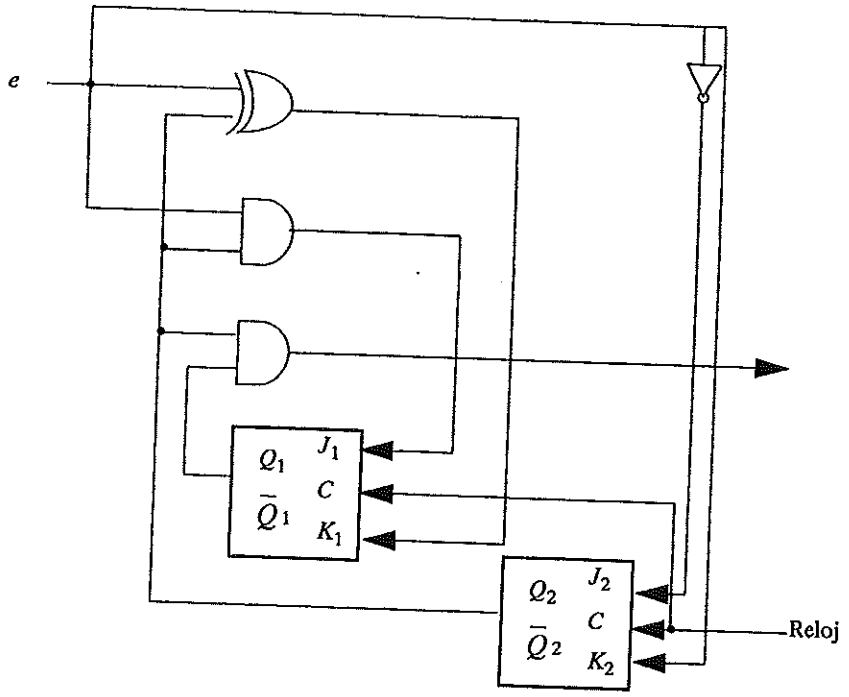
Se puede ver a ojo, sin utilizar Karnaugh, en la tabla de transiciones.

Para S :

e \ Q_1Q_2	00	01	11	10
0	0	0	1	0
1	0	0	1	0

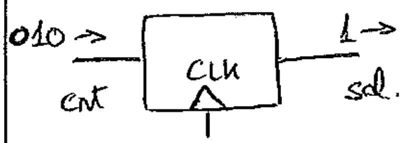
$$S = Q_1 \cdot Q_2$$

El circuito implementado es el siguiente:



Ejercicio 2

Diseñar un circuito secuencial con una entrada y una salida que reconozca la cadena "010".



Es lo que nos piden que diseñemos

1) Definimos los estados:

$q_1 \equiv$ "el último bit recibido ha sido un 1"

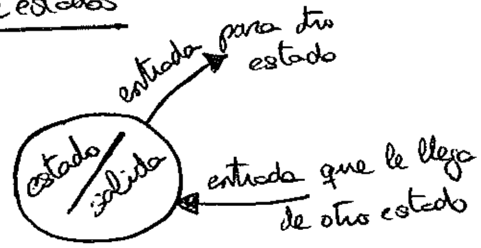
$q_2 \equiv$ " " " " " " " " ∅"

$q_3 \equiv$ " los dos últimos bits recibidos han sido ∅ y 1"

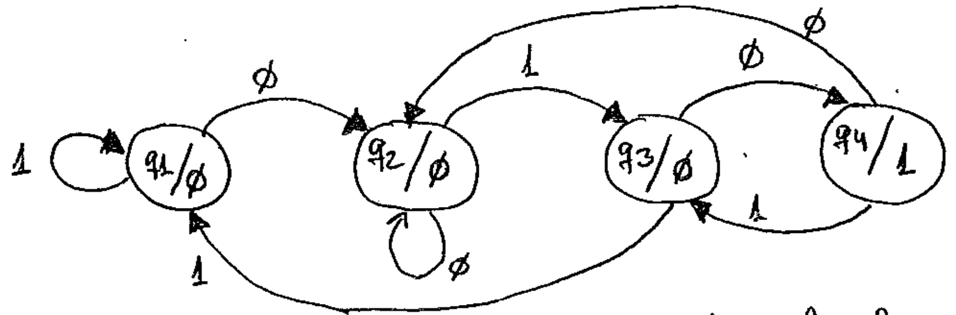
$q_4 \equiv$ " los 3 " " " " " ∅, 1, ∅"

2) Diagrama de estados

Leyenda:



⇒



∅, 1, 1
∅, 1, ∅, ∅
∅, 1, ∅, 1 dd

3) Asignamos nombres binarios a los estados (da igual cual pongas a que estado)

$q_1 \equiv 00$ $q_2 \equiv 01$ $q_3 \equiv 10$ $q_4 \equiv 11$

dd Como numeramos los estados usando 2 dígitos, necesitaremos dos biestables dd para generar las transiciones de cada uno de los dígitos

e	q(t)		S	q(t+1)		D_1	D_2
0	0	0	0	0	1	0	1
0	0	1	0	0	1	0	1
0	1	0	0	1	1	1	1
0	1	1	1	0	1	0	1
1	0	0	0	0	0	0	0
1	0	1	0	1	0	1	0
1	1	0	0	0	0	0	0
1	1	1	1	1	0	1	0

(*) (marca junto a la primera fila)

e ≡ entrada, s ≡ salida

variables de estado; salida; estado siguiente; tabla de excitaciones

↳ tabla de transiciones ↲

- $D_1 \equiv$ este biestable se encarga del MSB del estado
- $D_2 \equiv$ del LSB

(*) q pasa si en el estado $q(t) = 00 \equiv q_1$, p… nos llega una entrada = ∅, p ej.

Nota: tabla transiciones biest. D

D	Q(t)	Q(t+1)
0	0	0
0	1	0
1	0	1
1	1	1

$Q_{t+1} = D$

Por ser una **máquina de Moore** la salida no depende de la entrada (no directamente), sólo depende del estado ACTUAL, así que su tabla de verdad se puede simplificar:

$q(t)$	S
0 0	0
0 1	0
1 0	0
1 1	1

Por tanto la salida se pondrá cuando CLK ↓

Nota: usamos máquina de Moore porque no obligan a usar ninguna máquina en especial (Moore o Mealy)

4) Simplificamos por Karnaugh las funciones D_1 y D_2 (entradas de biestables) y S (salida del circuito)

Notación: $q(t) = Q_1 Q_2$ (MSB LSB)

D_1:

Q_1Q_2 / e	00	01	11	10
0	0	0	0	1
1	0	1	1	0

$D_1 = \bar{e}\, Q_1 \bar{Q}_2 + e\, Q_2$

D_2:

Q_1Q_2 / e	00	01	11	10
0	1	1	1	1
1	0	0	0	0

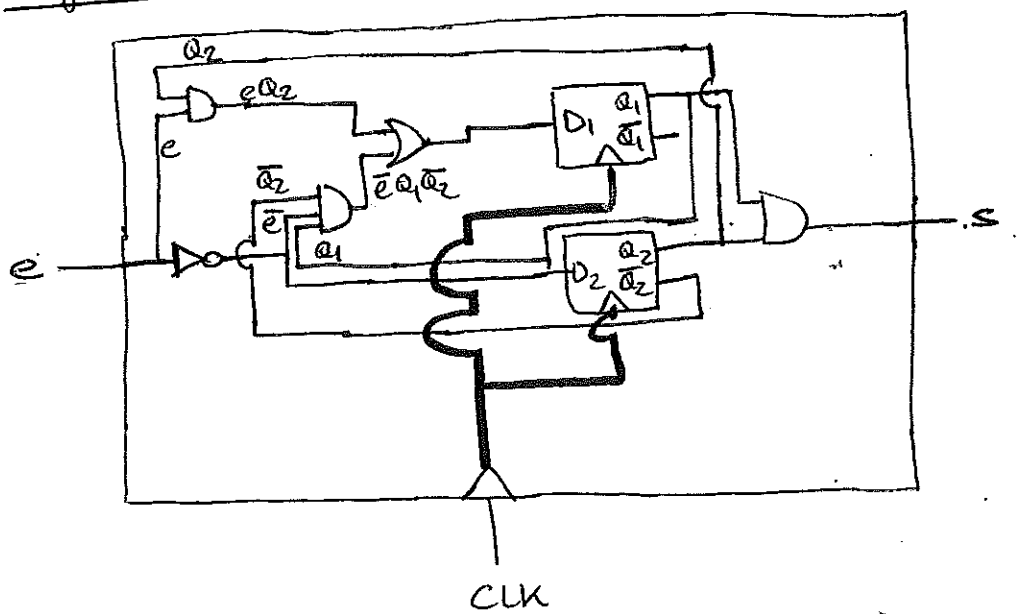
$D_2 = \bar{e}$

S: → no hace realmente falta Karnaugh: $S = Q_1 \cdot Q_2$

Q_1 / Q_2	0	1
0	0	0
1	0	1

+D

5) Implementación del circuito

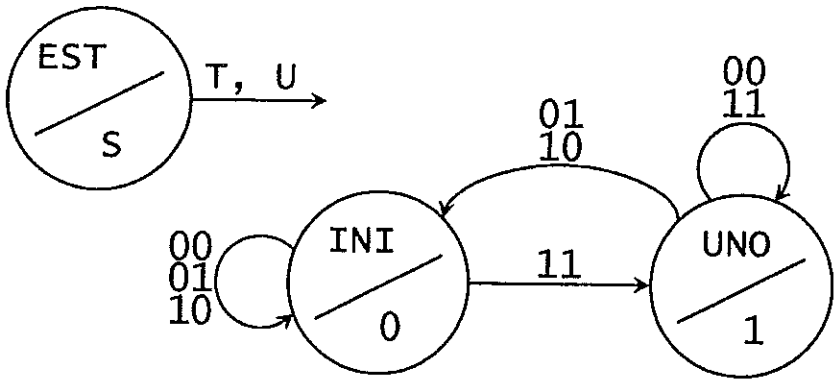


Ejercicio 3

Resuelto

(Chuso)

Dibuje el diagrama de estados de un autómata de Moore que tiene dos entradas de 1 bit cada una (**T** y **U**) y una única salida de un solo bit (**S**). Este autómata deberá poner su salida a 1 cada vez que se activen simultáneamente las señales **T** y **U**. Las salida sólo volverá a cero cuando **T** y **U** sean diferentes entre si.



Ejercicio 2

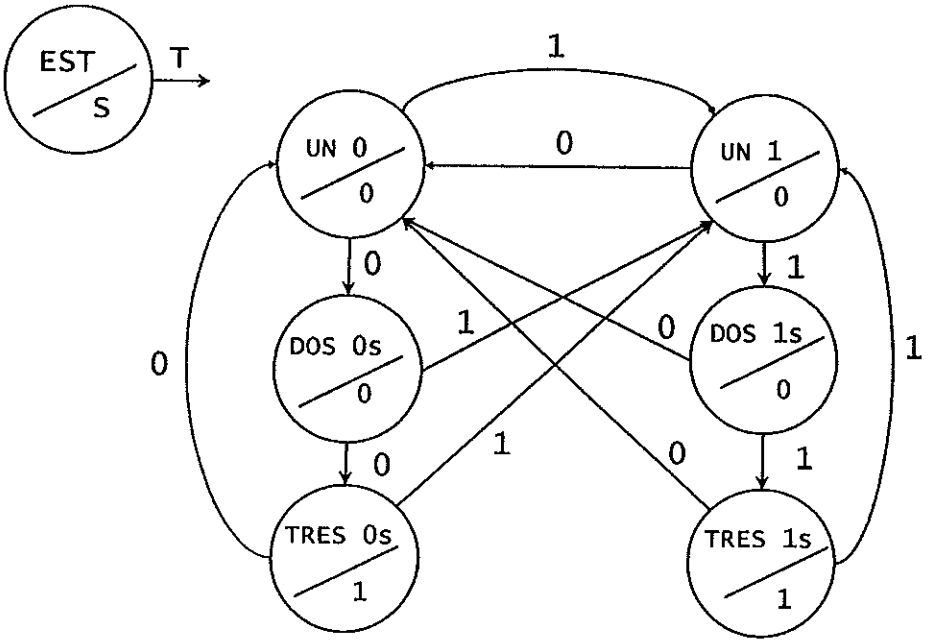
1. Realizar un biestable J-K síncrono a partir de un biestable D síncrono.
2. Realizar un biestable D síncrono a partir de un J-K síncrono.

Explique los diseños utilizando tablas de verdad y de excitaciones.

Ejercicio 4
Resuelto

Dibuje el diagrama de estados de un autómata de Moore que posee una entrada (T) de un solo bit por la que se recibe en sincronía con el reloj una serie de datos. El autómata debe activar la salida (S) cuando se detecte una secuencia de 3 bits consecutivos iguales (o lo que es lo mismo la secuencia 000 o la secuencia 111).

Cuando se detecte una de estas dos secuencias debe activarse la salida durante un pulso de reloj comenzando a partir de ese instante a detectar una nueva secuencia.



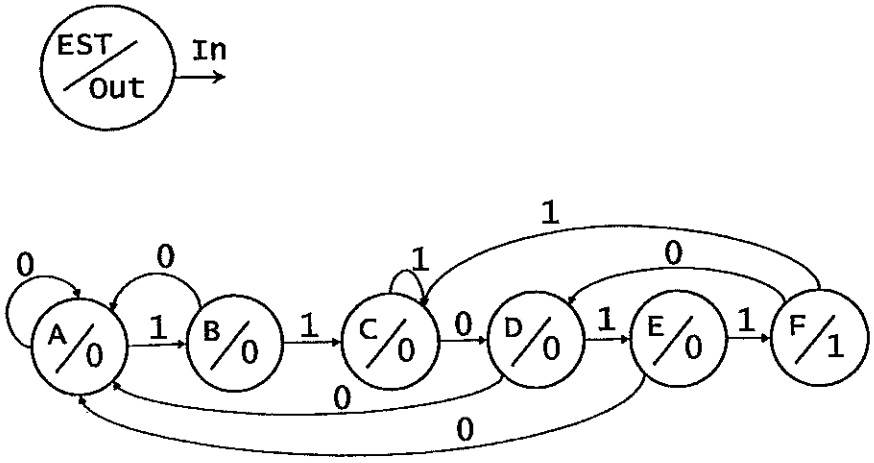
Ejercicio 5

Resuelto

(chungo)

Dibuje el diagrama de estados de un autómata de Moore que modele un detector de la secuencia 11011. Dicha secuencia se recibe a través de una entrada (In) por la que llegan los bits en serie, en sincronismo con el reloj. El detector debe poner su salida (Out) a nivel alto durante un pulso de reloj, cada vez que se detecte la secuencia.

Los bits que forman parte de una secuencia, pueden formar parte también de la siguiente, es decir, si se recibe la serie bits ...11011011..., la salida Out debería activarse dos veces, después del quinto y después del octavo bit.

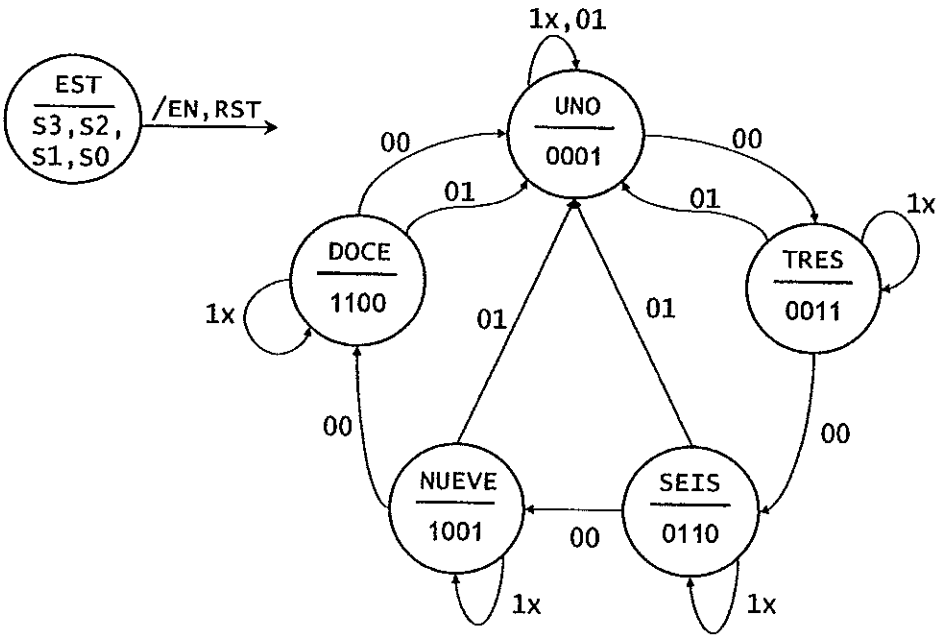


Ejercicio 6

Resuelto

Realice el diagrama de estados de un contador que funcione de la siguiente manera:

- El contador realiza de forma cíclica una cuenta ascendente generando la siguiente secuencia de números: 1, 3, 6, 9 y 12.
- Dispone de una entrada de control que es una habilitación de la cuenta y debe ser activa a nivel bajo (/EN).
- Dispone de una segunda señal de control que resetea la cuenta llevando al contador al primer estado de cuenta. Esta señal de reset debe ser activa a nivel alto (RST). En caso de que el contador esté deshabilitado y se active la señal de reset el contador debe permanecer en el valor de cuenta en el que se encuentre.

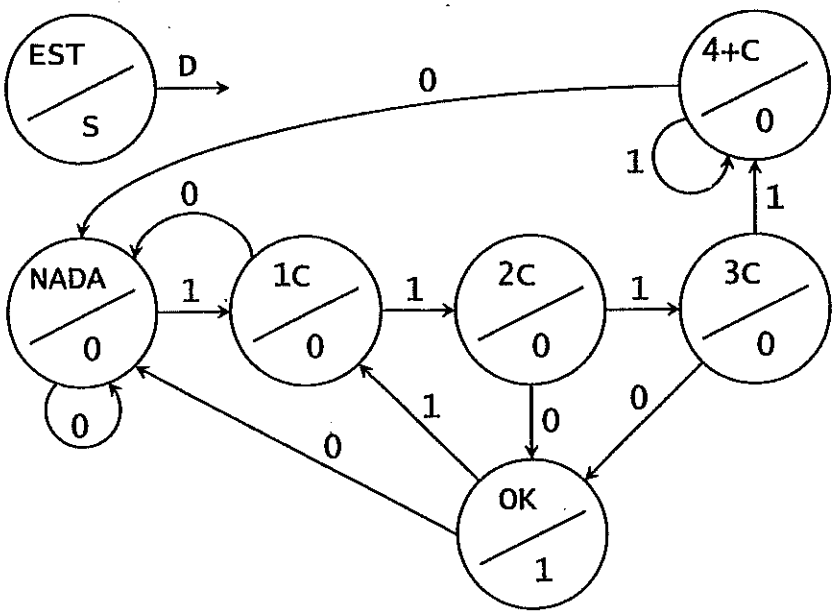


Ejercicio 7

Resuelto

(chungo)

Dibuje el diagrama de estados de un autómata de Moore que tiene una única entrada de 1 bit (**D**) y una única salida también de un solo bit (**S**). Este autómata deberá activar su salida durante 1 solo ciclo de reloj al terminar un pulso a nivel alto de la entrada, siempre que ese pulso haya durado exactamente 2 ó 3 ciclos de reloj. Suponga que la señal de entrada solo puede variar en sincronía con el reloj del sistema.

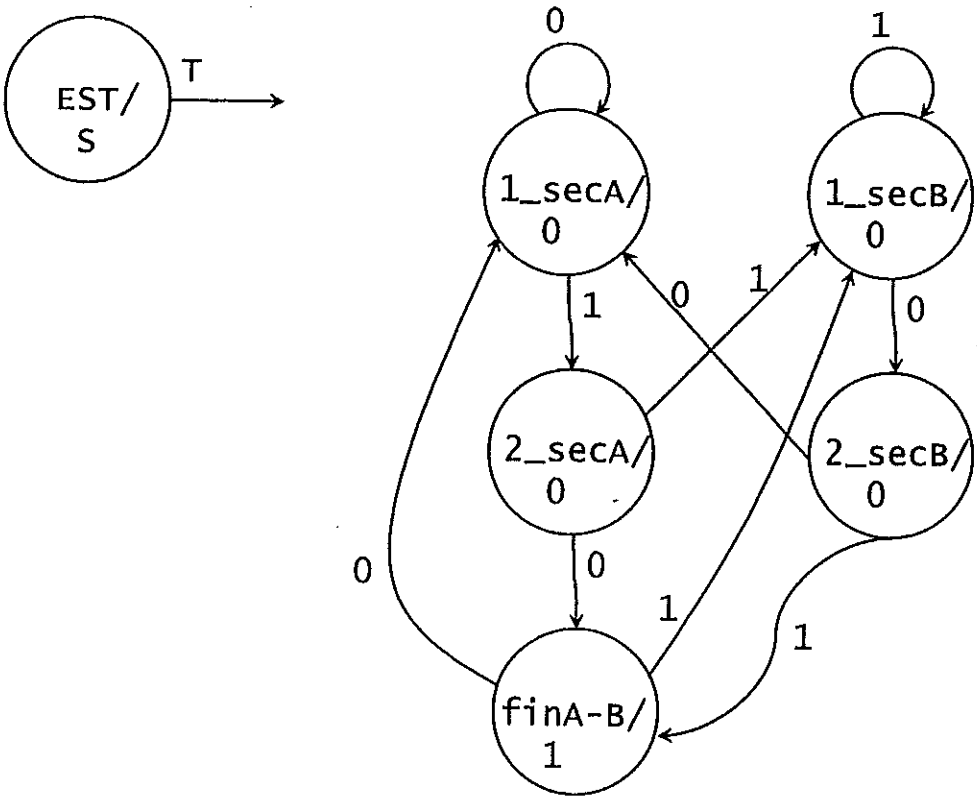


Ejercicio 8
Resuelto

Dibuje el diagrama de estados de un autómata de Moore que posee una entrada (T) de un bit por la que se recibe en sincronía con el reloj una serie de datos. El autómata debe activar la salida (S) cuando se detecten las secuencias 010 ó 101.

Cuando se detecte una de estas dos secuencias debe activarse la salida durante un pulso de reloj comenzando a partir de ese instante a detectar una nueva secuencia.

(Moore)



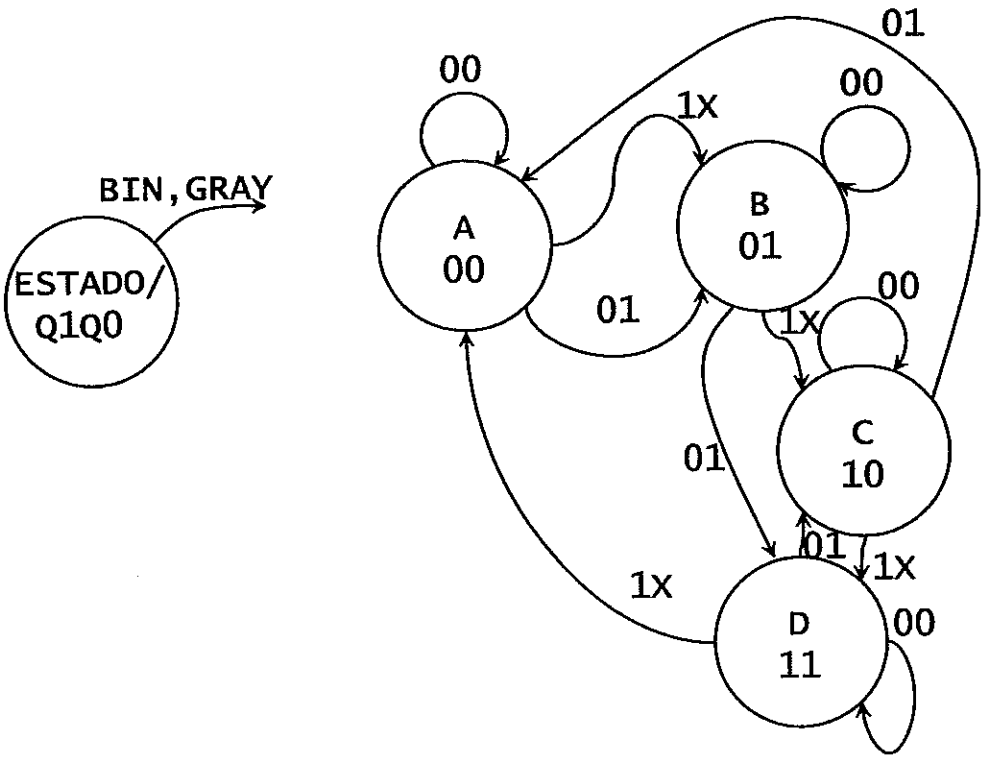
Ejercicio 9

Resuelto

(el largo)

Realice el diagrama de estados de un autómata de Moore que funcione como un contador que siga la secuencia: 00, 01, 10, 11, 00... cuando la entrada BIN esté activa (a nivel alto) y siga la secuencia 00, 01, 11, 10, 00... cuando la entrada GRAY esté activa (a nivel alto). Si ninguna de estas dos entradas está activa, el contador permanecerá en el estado en que se encuentre.

Nota: La entrada BIN tiene preferencia sobre GRAY.



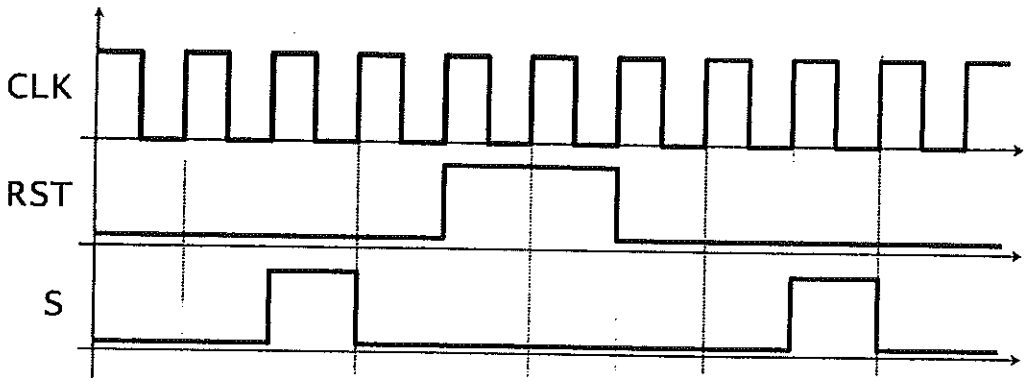
Ejercicio 10

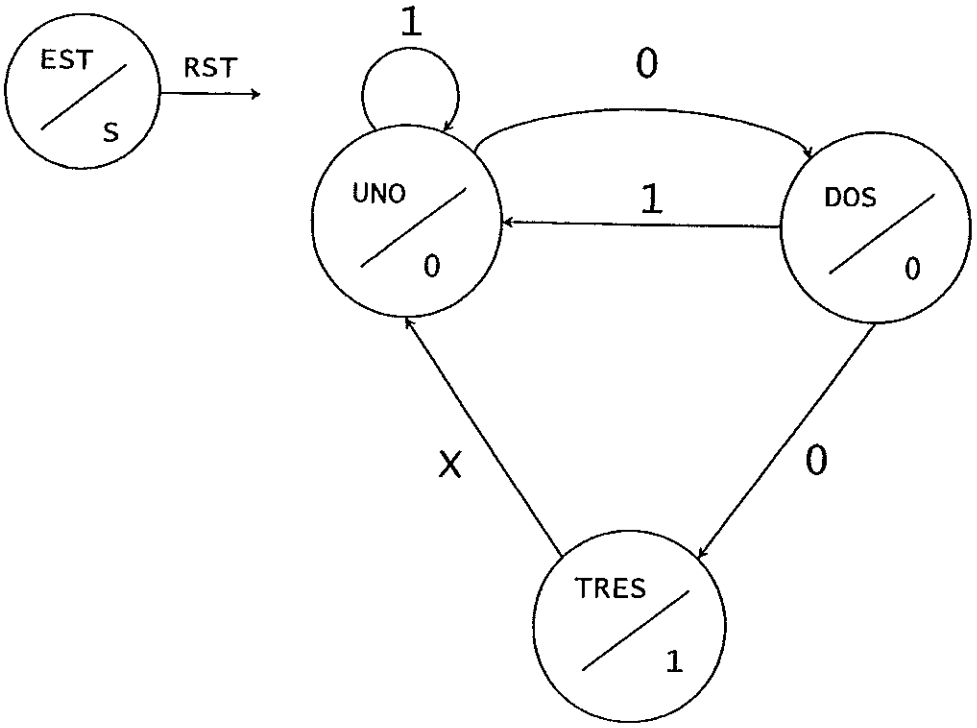
Resuelto

(chuvoso)

Dibuje el diagrama de estados de un autómata de Moore que posee una entrada (RST) de un solo bit que se activa y desactiva siempre en sincronía con la señal de reloj. El autómata debe generar una señal periódica de periodo 3 ciclos de reloj a través de su salida (S) en la que los 2 primeros ciclos de reloj debe estar a nivel bajo y el otro a nivel alto.

Si durante el funcionamiento se activa la señal RST (activa a nivel alto) el sistema debe comenzar de nuevo permaneciendo en el estado inicial hasta que se desactive la señal de RST. El comportamiento del sistema se puede apreciar en la siguiente figura.



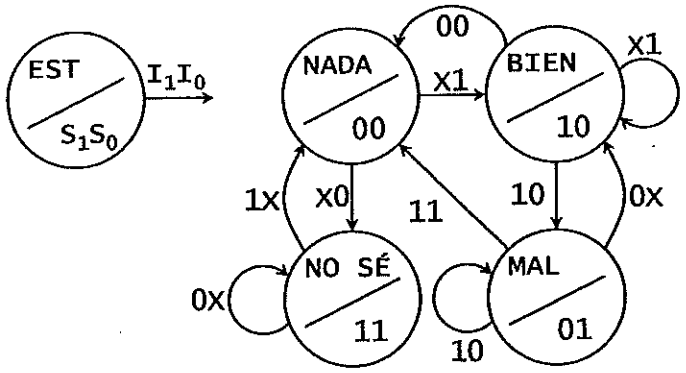


Ejercicio 11

Resuelto

(fácil)

Obtenga la tabla de transiciones para el autómata descrito por el diagrama de estados de la figura siguiente.



Indique claramente la codificación de estados que emplee.

Codificación de estados:

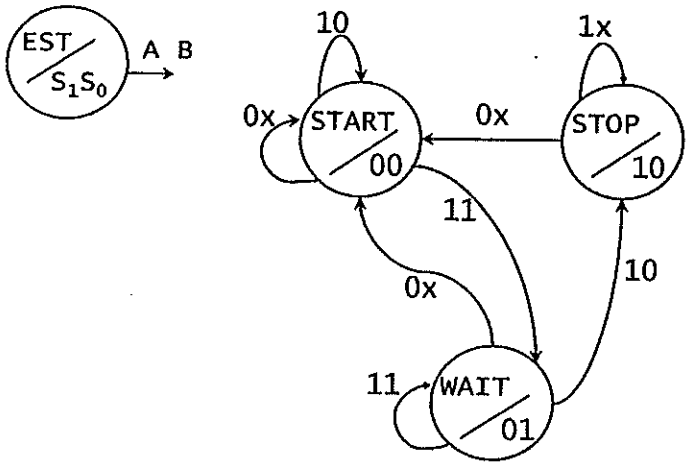
NADA: 00
MAL: 01
BIEN: 10
NO SÉ: 11

Q_1	Q_0	I_1	I_0	D_1	D_0	S_1	S_0
0	0	0	0	1	1	0	0
0	0	0	1	1	0		
0	0	1	0	1	1		
0	0	1	1	1	0		
0	1	0	0	1	0	0	1
0	1	0	1	1	0		
0	1	1	0	0	1		
0	1	1	1	0	0		
1	0	0	0	0	0	1	0
1	0	0	1	1	0		
1	0	1	0	0	1		
1	0	1	1	1	0		
1	1	0	0	1	1	1	1
1	1	0	1	1	1		
1	1	1	0	0	0		
1	1	1	1	0	0		

Ejercicio 12

Resuelto

Obtenga la tabla de transiciones, incluyendo las salidas, para el autómata descrito por el diagrama de estados de la figura siguiente. Indique claramente la codificación de estados que emplee.



START	0 0
WAIT	0 1
STOP	1 0

Estado Actual		Entradas		Estado Futuro		Salidas	
$Q_1(t)$	$Q_0(t)$	A	B	$Q_1(t+1)$	$Q_0(t+1)$	S_1	S_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	1	0	0	0	0	0
0	0	1	1	0	1	0	0
0	1	0	0	0	0	0	1
0	1	0	1	0	0	0	1
0	1	1	0	1	0	0	1
0	1	1	1	0	1	0	1
1	0	0	0	0	0	1	0
1	0	0	1	0	0	1	0
1	0	1	0	1	0	1	0
1	0	1	1	1	0	1	0

Ejercicio 13

Resuelto

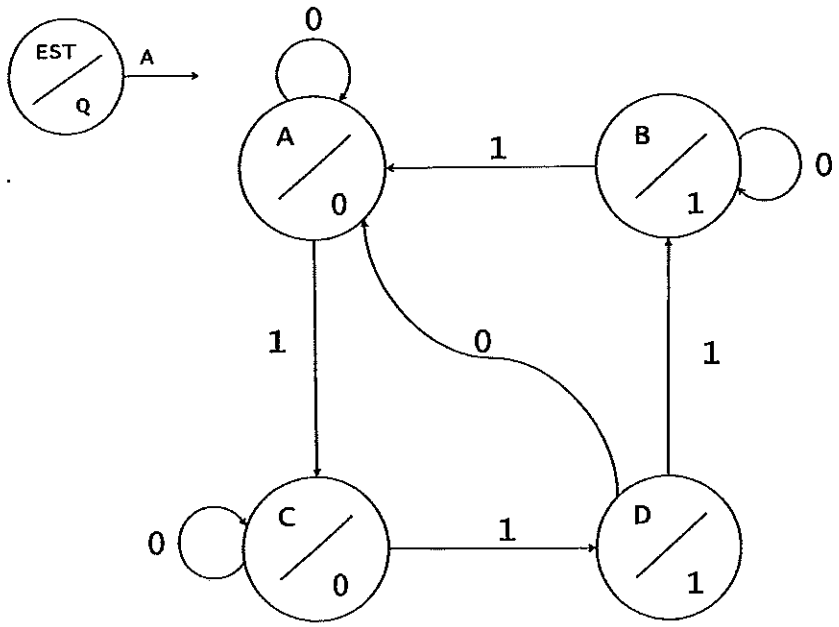
(fácil)

La tabla de transiciones de la figura siguiente se corresponde con la de un autómata de Moore. Represente el diagrama de estados que da lugar a esta tabla.

Estado Presente		Entrada	Estado Futuro		Salida
$Q_1(t)$	$Q_0(t)$		$Q_1(t+1)$	$Q_0(t+1)$	
0	0	0	0	0	0
0	0	1	1	0	0
0	1	0	0	1	1
0	1	1	0	0	1
1	0	0	1	0	0
1	0	1	1	1	0
1	1	0	0	0	1
1	1	1	0	1	1

Codificación de estados:

Estado A: $Q_1=0$ $Q_0=0$
Estado B: $Q_1=0$ $Q_0=1$
Estado C: $Q_1=1$ $Q_0=0$
Estado D: $Q_1=1$ $Q_0=1$



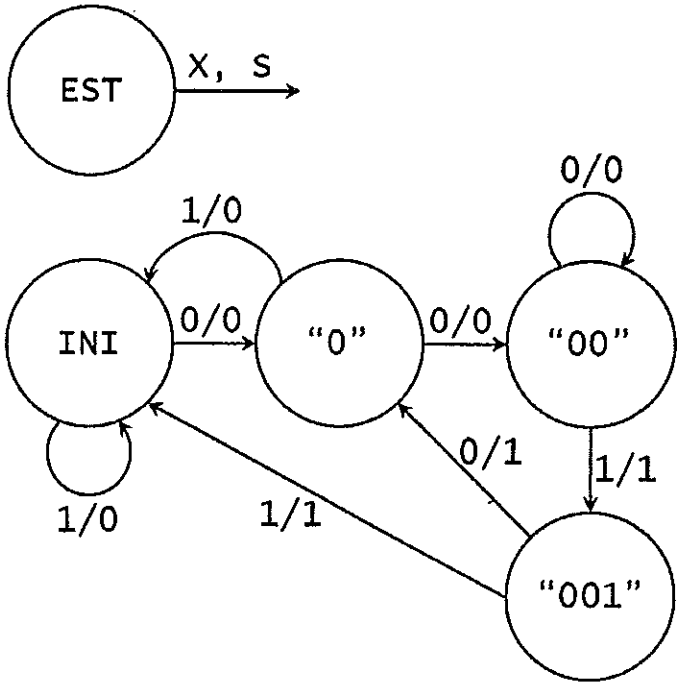
Ejercicio 14

Resuelto

Dibuje el diagrama de estados de un autómata de Mealy que tiene 1 entrada (**X**) de un solo bit por la que se reciben bits en serie y una salida (**S**) también de un solo bit.

Este autómata generará en su salida la secuencia "11" cada vez que se detecte en la entrada la secuencia "001", permaneciendo **S** a cero en caso contrario.

Mealy
(s cambia)



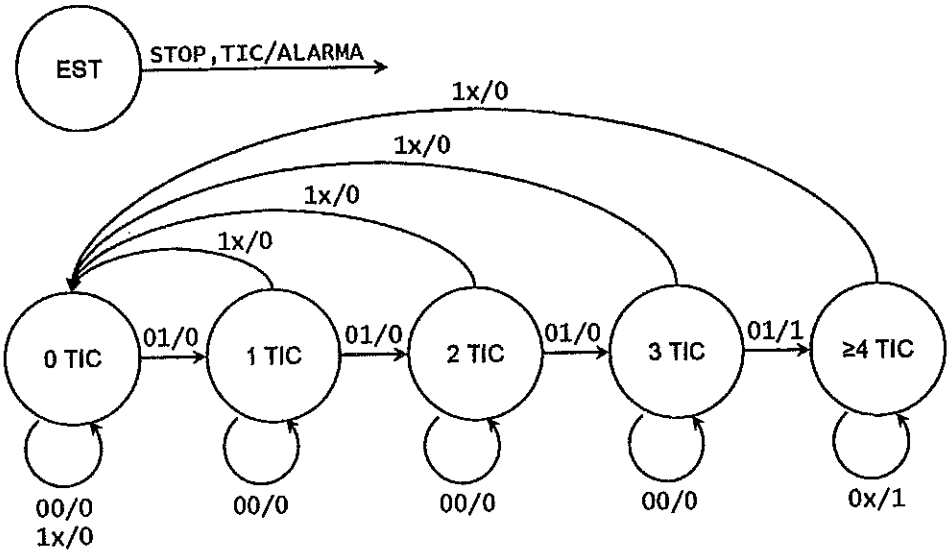
Ejercicio 15

Resuelto

Mealy (+chungo)

Dibuje el diagrama de estados de un autómata de Mealy que tiene dos entradas (TIC y STOP) y una salida (ALARMA) todas ellas de un bit y activas a nivel alto. El comportamiento de cada una de estas señales es el siguiente:

- La señal TIC se activa en sincronía con la señal de reloj y durante un ciclo de éste una vez cada segundo.
- La señal STOP puede activarse en cualquier momento en sincronía con la señal de reloj y durante un ciclo de este. Esta señal es más prioritaria que la señal TIC.
- La señal de ALARMA debe activarse cuando se hayan recibido 4 pulsos de TIC desde la última activación de STOP y desactivarse cuando se reciba un pulso de STOP. Si se activa la señal de STOP antes de que se haya finalizado la cuenta de los 4 pulsos debe comenzarse de nuevo.



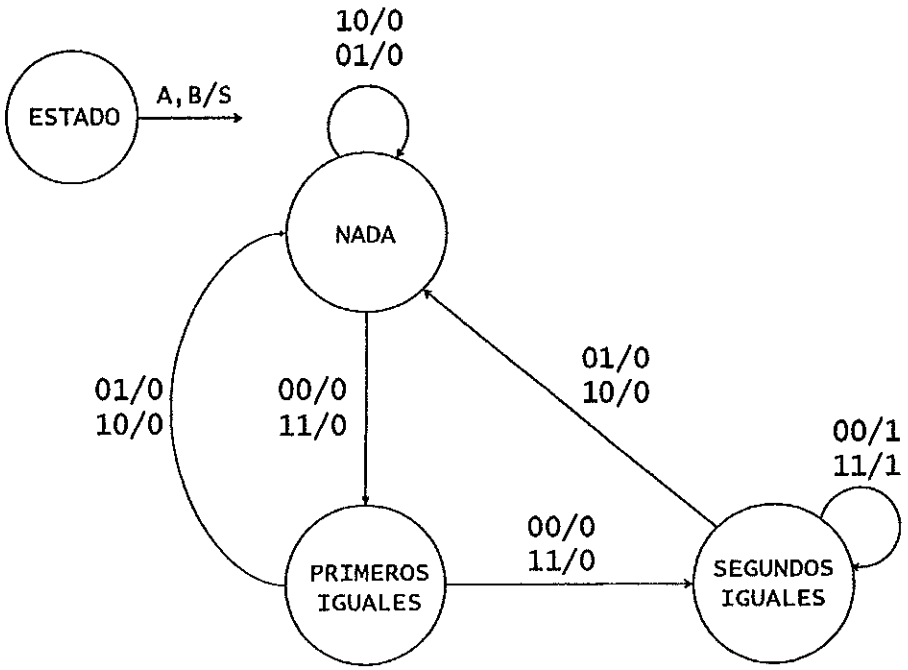
Ejercicio 16

Resuelto

Dibuje el diagrama de estados de un autómata de Mealy para realizar un detector de secuencias que recibe a través de dos entradas A y B, y siempre en sincronía con el reloj, los datos de la secuencia que se desea detectar.

El sistema posee una única salida (S) activa a nivel alto que debe activarse cuando los bits recibidos por A y B sean iguales (los dos cero o los dos uno) durante al menos tres ciclos de reloj. Una vez detectada la secuencia, la salida debe permanecer activa hasta que se detecten por primera vez dos datos diferentes (un cero y un uno o viceversa).

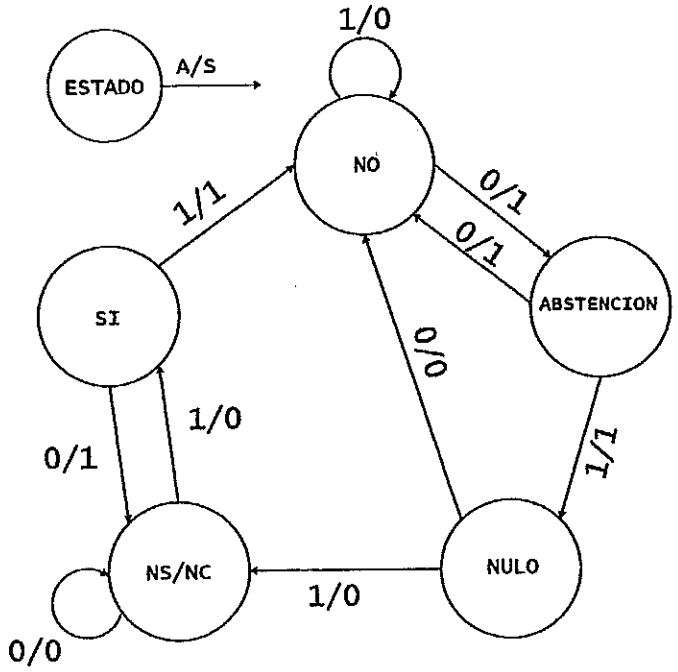
Mealy
(+ chungo)



Ejercicio 17

Resuelto

Obtenga la tabla de transiciones incluyendo las salidas para el autómata descrito por el diagrama de estados de la figura siguiente. Emplee la tabla que se indica para codificar los estados.



Codificación Estados	Q2	Q1	Q0
NO	0	0	0
SI	0	0	1
NS/NC	0	1	0
NULO	0	1	1
ABSTENCION	1	0	0

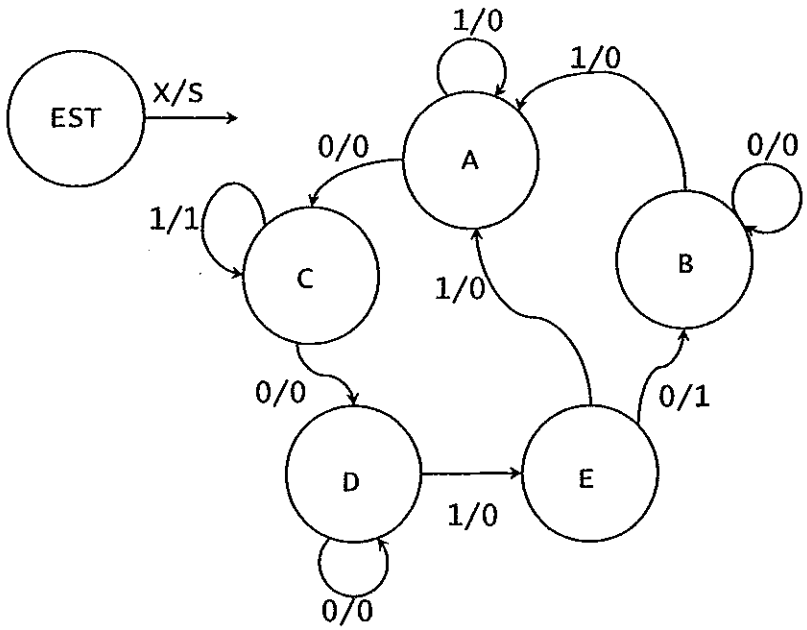
Estado Actual			Entrada	Estado Futuro			Salida
Q2	Q1	Q0	A	Q2	Q1	Q0	S
0	0	0	0	1	0	0	1
0	0	0	1	0	0	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	0	0	1
0	1	0	0	0	1	0	0
0	1	0	1	0	0	1	0
0	1	1	0	0	0	0	0
0	1	1	1	0	1	0	0
1	0	0	0	0	0	0	1
1	0	0	1	0	1	1	1

Ejercicio 18

Resuelto

Obtenga la tabla de transiciones del circuito representado por el diagrama de estados de la figura siguiente.

Nota: Codifique los estados en binario, por orden alfabético.



Estado, Q2 Q1 Q0

A 0 0 0
B 0 0 1
C 0 1 0
D 0 1 1
E 1 0 0

Q2	Q1	Q0	X	D2	D1	D0	S
0	0	0	0	0	1	0	0
0	0	0	1	0	0	0	0
0	0	1	0	0	0	1	0
0	0	1	1	0	0	0	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	0	1
0	1	1	0	0	1	1	0
0	1	1	1	1	0	0	0
1	0	0	0	0	0	1	1
1	0	0	1	0	0	0	0
1	0	1	0	X	X	X	X
1	0	1	1	X	X	X	X
1	1	0	0	X	X	X	X
1	1	0	1	X	X	X	X
1	1	1	0	X	X	X	X
1	1	1	1	X	X	X	X

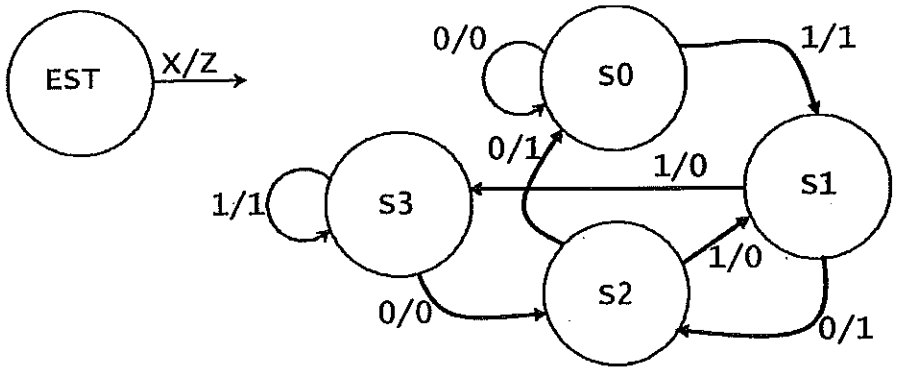
Ejercicio 19

Resuelto

(fácil)

Obtenga la tabla de transiciones del circuito representado por el diagrama de estados de la figura siguiente.

Nota: Realice la siguiente codificación de estados: S0=00, S1=01, S2=10, S3=11.



Q1	Q0	X	D1	D0	Z
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	1	0	1
0	1	1	1	1	0
1	0	0	0	0	1
1	0	1	0	1	0
1	1	0	1	0	0
1	1	1	1	1	1

Ejercicio 20

Resuelto

(Fácil)

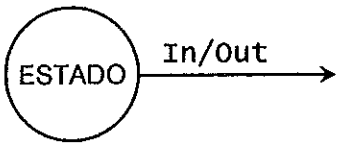
Analice la tabla de transiciones siguiente:

Estado Actual		Entrada	Estado Futuro		Salida
$Q_1(t)$	$Q_0(t)$	In	$Q_1(t+1)$	$Q_0(t+1)$	Out
0	0	0	0	0	1
0	0	1	1	0	0
0	1	0	0	1	1
0	1	1	1	0	1
1	0	0	0	1	1
1	0	1	1	0	0
1	1	0	1	1	0
1	1	1	1	0	0

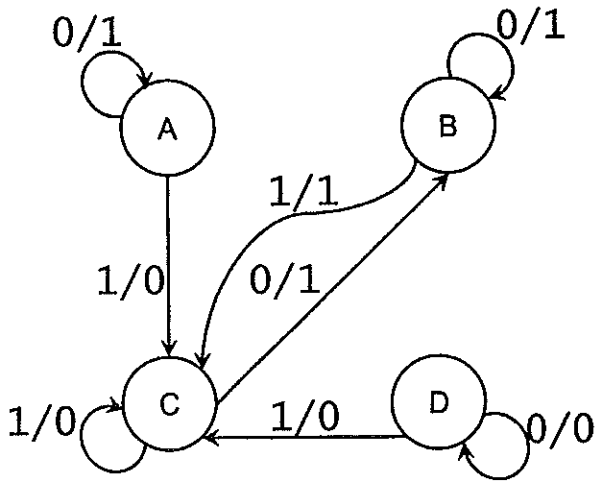
Indique, justificando su respuesta, si corresponde a un autómata de Mealy o de Moore.

Represente el diagrama de estados correspondiente a la tabla. Indique claramente la codificación de estados que emplee.

Es un autómata de Mealy porque la salida depende del estado y de la entrada. Por ejemplo, estando en el estado "00" la salida puede valer 1 ó 0, dependiendo del valor de la entrada. Lo mismo ocurre en el estado "10".



Q_1	Q_0	Estado
0	0	A
0	1	B
1	0	C
1	1	D



Ejercicio 21

Resuelto

(fácil)

La tabla de transiciones de la figura siguiente se corresponde con la de un autómata de Mealy. Represente el diagrama de estados que da lugar a esta tabla.

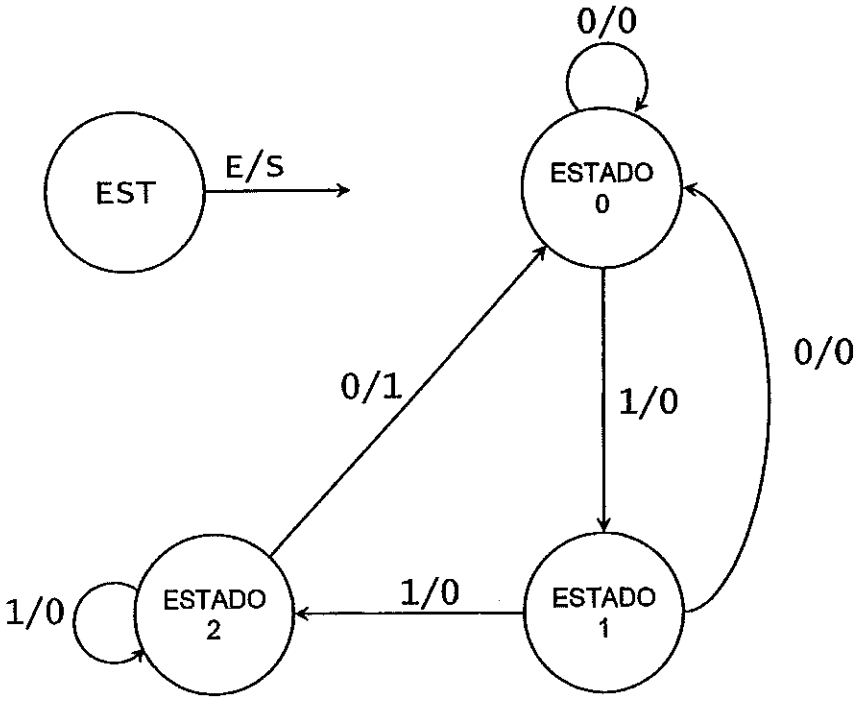
Estado Presente		Entrada	Estado Futuro		Salida
$Q_1(t)$	$Q_0(t)$		$Q_1(t+1)$	$Q_0(t+1)$	
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	0
0	1	1	1	0	0
1	0	0	0	0	1
1	0	1	1	0	0

Codificación de estados:

Estado 0: Q0=Q1=0

Estado 1: Q1=0 Q0=1

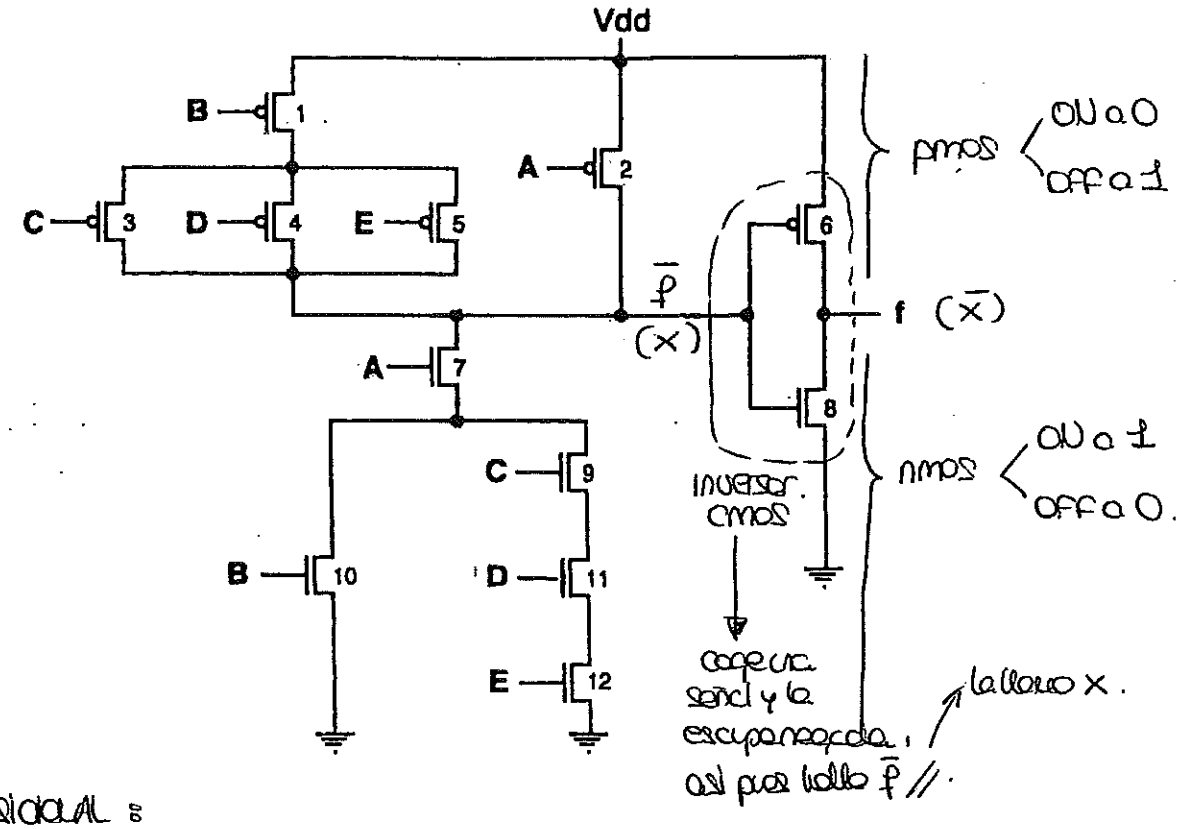
Estado 2: Q1=1 Q0=0



CEDG
Problemas de examen

FEBRERO 1996 CMOS.

5. Determinar la función realizada por el circuito CMOS de la figura razonando la respuesta.



* Ejercicio Raro: hay q inversar !!

Uso LÓGICA PROPOSICIONAL:

$\boxed{X=1} \leftrightarrow (T_2=ON) \vee (T_1=ON \wedge (T_3=ON \vee T_4=ON \vee T_5=ON)) \leftrightarrow$

$\leftrightarrow (A=0) \vee (B=0 \wedge (C=0 \vee D=0 \vee E=0)) \leftrightarrow$

$\leftrightarrow \bar{A} + \bar{B} \cdot (\bar{C}+\bar{D}+\bar{E}) = X$ puedo simplificar ...

$X = \bar{A} + \bar{B}(\bar{C}+\bar{D}+\bar{E}) = \bar{A} + \bar{B}(\overline{CDE}) = \bar{A} + \overline{B+CDE} = \overline{A \cdot (B+CDE)}$

pero estoy buscando $f = \bar{X} = A \cdot (B+CDE)$.

Y si estaba buscando la $\bar{X}$ ... podría haber hecho el circuito para $X=0$:

$\boxed{X=0} \leftrightarrow (T_7 ON) \wedge (T_{10} ON \vee (T_9 ON \wedge T_{11} ON \wedge T_{12} ON)) \leftrightarrow$

$\leftrightarrow (A=1) \wedge (B=1 \vee (C=1 \wedge D=1 \wedge E=1)) \leftrightarrow$ $\boxed{\wedge \equiv Y = \cdot \quad \vee \equiv O = +}$

$\leftrightarrow A \cdot (B + (C \cdot D \cdot E)) = \bar{X}$ ¡y lo tenía directamente!

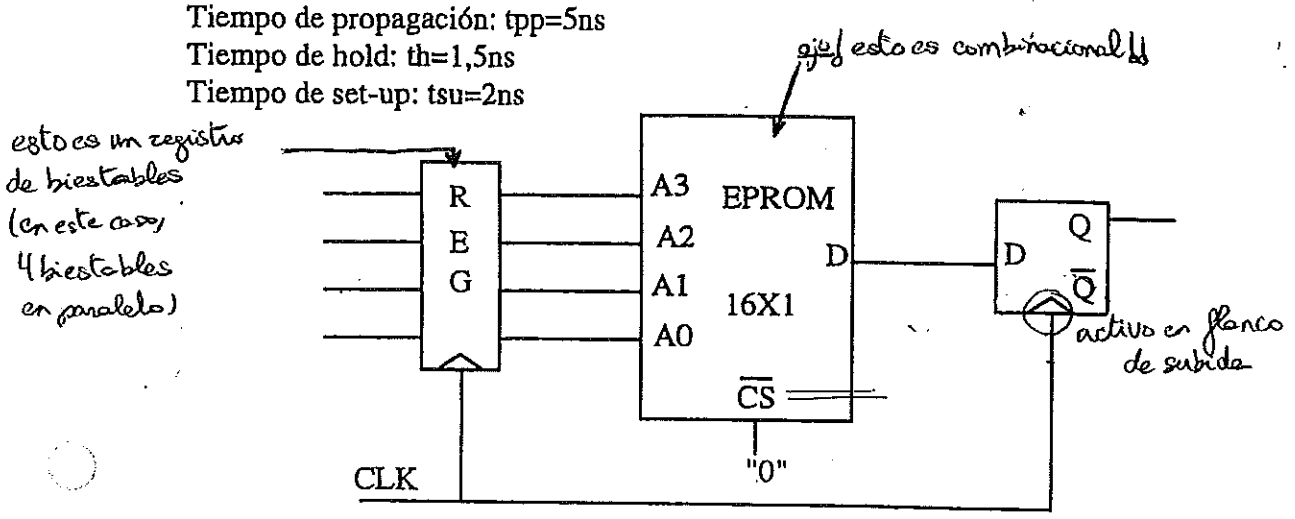
$\boxed{\bar{X} = A(B+CDE)} = f$ //

(pero efectivamente, por cubrir circuitos solo).

PROBLEMA 3: (15 puntos)

El circuito de la figura representa un sistema simplificado de lectura en una memoria EPROM de 16X1 bits. Los Flip-flops del registro de entrada y el de salida son idénticos y sus características son:

Tiempo de propagación: tpp=5ns
Tiempo de hold: th=1,5ns
Tiempo de set-up: tsu=2ns

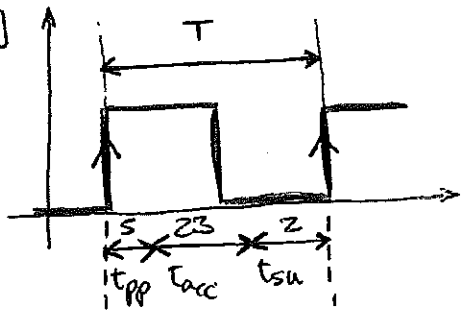


En cuanto a la memoria, su tiempo de acceso es Tacc=23ns (Tiempo que transcurre desde que ponemos una dirección en su bus de direcciones hasta que aparece el dato correcto en su bus de datos) y su salida es *tri-state* cuando el *chip select* esté desactivado. Con estos datos se pide:

3.1) Calcular la Frecuencia máxima de la señal de reloj que garantice un correcto funcionamiento del circuito.

3.2) Suponiendo que se quiera aumentar la frecuencia de la señal de reloj, y que únicamente se disponga de memorias EPROM de 16X1 descritas anteriormente, de EPROM de 8X1 con un tiempo de acceso de 13ns y cualquier tipo de puerta lógica con un tiempo de retardo de 3 nseg., rediseñe el circuito y calcule la máxima frecuencia que podrá alcanzar ahora la señal de reloj. **(SE VALORARÁ LA SENCILLEZ DEL CIRCUITO)**

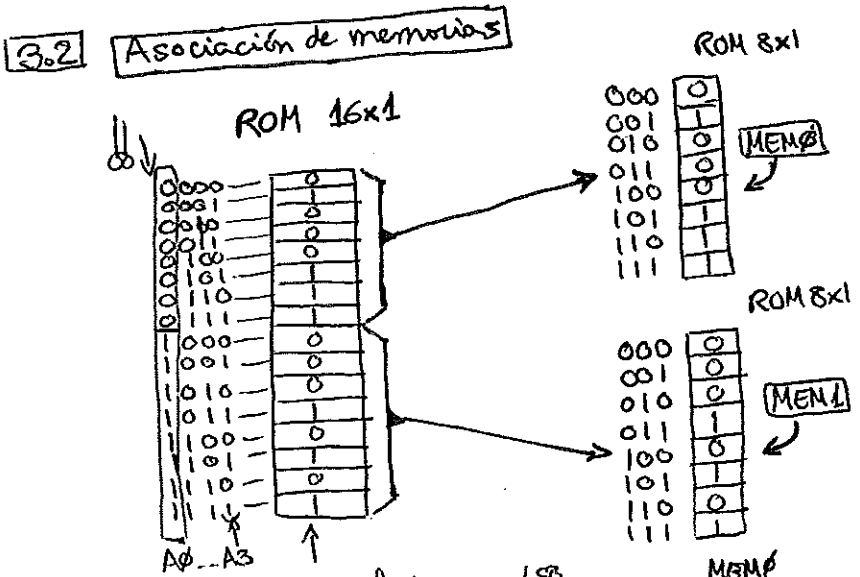
3.1



$T_{mín} = 5+23+2 = 30$ nseg.

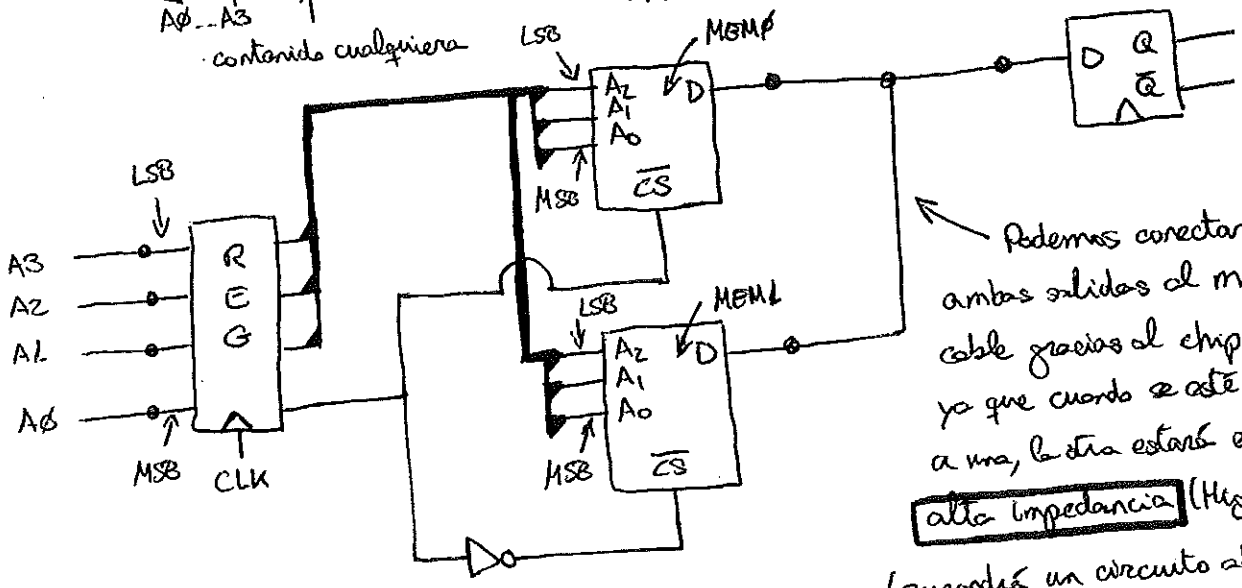
$f_{máx} = \frac{1}{30}$ nseg $= 33{,}3$ MHz

3.2 Asociación de memorias



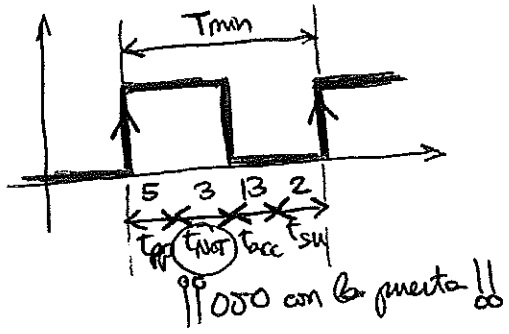
El bit AØ (el MSB) de direcciones va a hacer de chip-select, habilitando en cada caso la memoria a la que tiene que irse a buscar el dato

AØ = 0 → MEMØ
AL = L → MEM1



Podemos conectar ambas salidas al mismo cable gracias al chip-select ($\overline{CS}$) ya que cuando se esté accediendo a una, la otra estará en **alta impedancia** (High Z)
(supondrá un circuito abierto)

Calculamos ahora la máxima frecuencia:



Todo con la puerta !!

$T_{min} = 5+3+13+2 = 23\ ns$

$f_{max} = \frac{1}{T_{min}} = 43{,}48\ MHz$

FEBRERO 1998

3. Se desea diseñar un sistema para el control de un puesto de peaje automático en una autopista (Ver figura 1). Inicialmente, a la entrada del puesto hay un semáforo en verde indicativo de que el vehículo puede pasar (IN = 1).

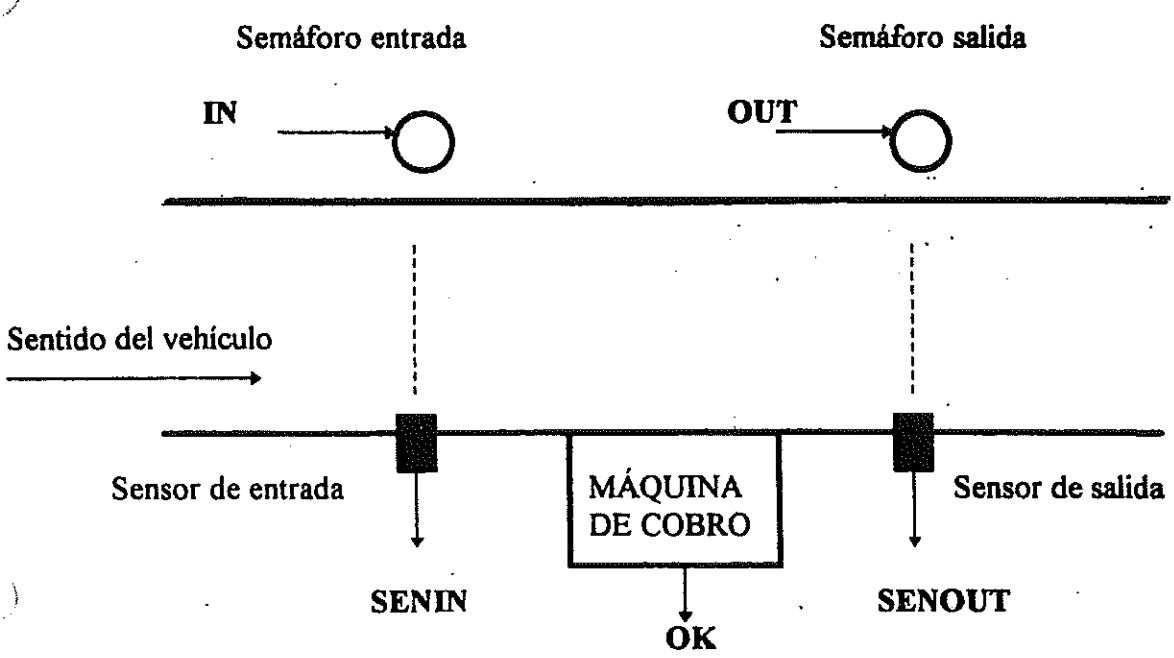
Cuando el vehículo entra a la sección del peaje, un sensor detecta el paso y genera una señal (SENIN = 1), de duración un ciclo de reloj; momento en el que se debe cambiar a rojo el semáforo de entrada (IN = 0).

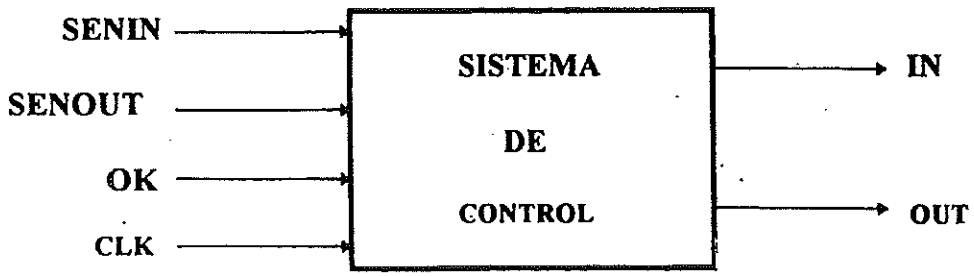
La máquina de cobro, espera recibir la cantidad correcta de dinero, momento en el que activa la señal (OK = 1) durante un ciclo de reloj. Mientras que no se deposite el dinero correcto, el semáforo de salida permanecerá en rojo (OUT = 0), y cuando se haya depositado la cantidad correcta, éste pasará a verde (OUT = 1).

Un sensor de salida detecta el paso del vehículo, y activa la señal SENOUT = 1 durante un ciclo de reloj, momento en el ~~que se tiene~~ que se tiene que poner en verde el semáforo de entrada (IN = 1) y en rojo el de salida (OUT = 0), para permitir la entrada de un nuevo vehículo.

Todas las señales son síncronas con la señal de reloj CLK.

Nota: Ningún vehículo se saltará nunca un semáforo en rojo.





Se pide:

3.1. Diseñe el diagrama de estados del sistema de control. Indique claramente qué representa cada uno de los estados que pinte. Indique también el tipo de autómata que ha obtenido (Moore o Mealy).
Tenga en cuanta que en cada estado hay entradas que son posible, y otras que no lo son.

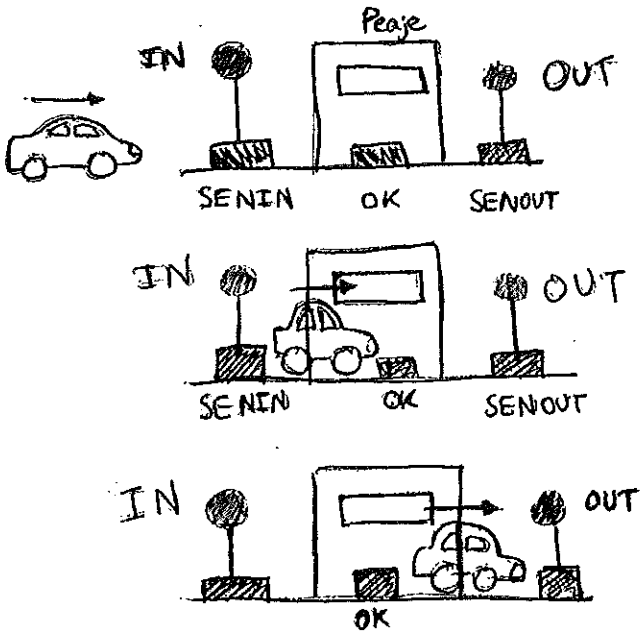
3.2. Complete la tabla de transiciones, empleando biestables tipo J-K.

ESTADO (T)		ENTRADA			ESTADO (T+1)		SALIDA		EXCITACIÓN BIESTABLES			
Q1	Q2	SENIN	SENOUT	OK	Q1	Q2	IN	OUT	J1	K1	J2	K2
0	0	0	0	0	0	0	1	0	0	X	0	X
0	0	0	0	1	—	—	—	—	—	—	—	—
0	0	0	1	0	—	—	—	—	—	—	—	—
0	0	0	1	1	—	—	—	—	—	—	—	—
0	0	1	0	0	0	1	1	0	0	X	1	X
0	0	1	0	1	—	—	—	—	—	—	—	—
0	0	1	1	0	—	—	—	—	—	—	—	—
0	0	1	1	1	—	—	—	—	—	—	—	—
0	1	0	0	0	0	1	0	0	0	X	X	0
0	1	0	0	1	1	0	0	0	1	X	X	1
0	1	0	1	0	—	—	—	—	—	—	—	—
0	1	0	1	1	—	—	—	—	—	—	—	—
0	1	1	0	0	—	—	—	—	—	—	—	—
0	1	1	0	1	—	—	—	—	—	—	—	—
0	1	1	1	0	—	—	—	—	—	—	—	—
0	1	1	1	1	—	—	—	—	—	—	—	—
1	0	0	0	0	1	0	0	1	X	0	0	X
1	0	0	0	1	—	—	—	—	—	—	—	—
1	0	0	1	0	0	0	0	1	X	1	0	X
1	0	0	1	1	—	—	—	—	—	—	—	—
1	0	1	0	0	—	—	—	—	—	—	—	—
1	0	1	0	1	—	—	—	—	—	—	—	—
1	0	1	1	0	—	—	—	—	—	—	—	—
1	0	1	1	1	—	—	—	—	—	—	—	—
1	1	0	0	0	—	—	—	—	—	—	—	—
1	1	0	0	1	—	—	—	—	—	—	—	—
1	1	0	1	0	—	—	—	—	—	—	—	—
1	1	0	1	1	—	—	—	—	—	—	—	—
1	1	1	0	0	—	—	—	—	—	—	—	—
1	1	1	0	1	—	—	—	—	—	—	—	—
1	1	1	1	0	—	—	—	—	—	—	—	—
1	1	1	1	1	—	—	—	—	—	—	—	—

Nota: En caso de que obtenga un autómata de Moore, no rellene las columnas correspondientes a las salidas. Haga una pequeña tabal que suministre las salidas (IN y OUT), tan solo en función del estado.

3.3. Obtener las expresiones simplificadas de las salidas **IN** y **OUT**, así como de las entradas al biestable 2: **J2** y **K2**.

3.1 Tenemos 3 estados:



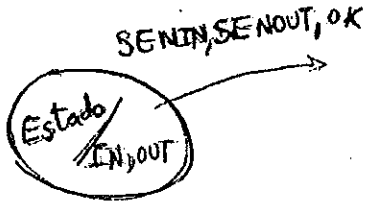
Autómata con 3 entradas (SENIN, SENOUT, OK)
2 salidas (IN, OUT)

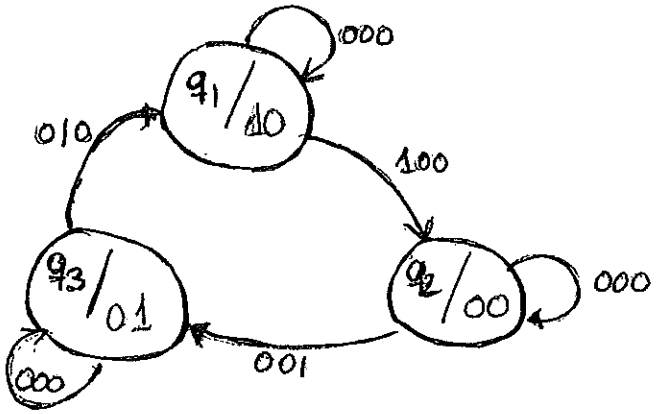
$q_1 \equiv$ peaje vacío

$q_2 \equiv$ vehículo en peaje (aún no ha pagado)

$q_3 \equiv$ vehículo en peaje (ya ha pagado)

Leyenda:





Hemos planteado una máquina de Moore, en la que la salida está asociada sólo al estado (no a la entrada, directamente)

3.2 Asignamos números binarios a los estados

$q_1 \equiv 00$
$q_2 \equiv 01$
$q_3 \equiv 10$

(ojo) $\neq$ estado 11

Nota: todas las demás filas no tienen sentido porque:

> el estado 11 NO existe

> no existen entradas con más de un sensor activado

$\Rightarrow$ | - x X | no importa lo q salga

Para la tabla de excitaciones recordemos la tabla de transiciones de un 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

(sigue en la otra página)

(sigue en la otra página)

Por última, la tabla de salida se puede "sacar" porque no depende de todas las variables:

Q_1	Q_2	IN	OUT
0	0	1	0
0	1	0	0
1	0	0	1
1	1	X	X

3.3 **IN** = {poniendo X=∅} = $\overline{Q_1} \cdot \overline{Q_2} = \overline{Q_1 + Q_2}$

{poniendo X=1} = $\overline{Q_1 \oplus Q_2} = \overline{Q_1}\,\overline{Q_2} + Q_1 Q_2$

OUT = {poniendo X=1} = Q_1 (la + simple ↑)

{ " X=∅} = $Q_1 \cdot \overline{Q_2}$ (NOT y AND).

para J2:

SI ≡ SENIN
SO ≡ SENOUT

$Q_1 = \emptyset$

Q_2 SI / SO OK	00	01	11	10
00	0	1	X	X
01	X	X	X	X
11	X	X	X	X
10	X	X	X	X

$J_2 = SENIN$

$Q_1 = 1$

Q_2 SI / SO OK	00	01	11	10
00	0	X	X	X
01	X	X	X	X
11	X	X	X	X
10	0	X	X	X

para K2:

$Q_1 = \emptyset$

Q_2 SI / SO OK	00	01	11	10
00	X	X	X	0
01	X	X	X	1
11	X	X	X	X
10	X	X	X	X

$K_2 = OK$

$Q_1 = 1$

Q_2 SI / SO OK	00	01	11	10
00	X	X	X	X
01	X	X	X	X
11	X	X	X	X
10	X	X	X	X

SUMADORES
MULTIPLICADOR

2 Diseño de un multiplicador rápido

Queremos diseñar un multiplicador de dos palabras de cuatro bits A3..A0 y B3..B0 que nos permita obtener el resultado P7..P0 de una forma rápida.

2.1 Definición de la operación de multiplicación (*5 puntos*)

Rellene la siguiente tabla con los productos parciales que sean necesarios para obtener el resultado P (Producto).

los ϕ_i y las cajas de colores, sup para el ejercicio 2.2

				A3	A2	A1	A0
				* B3	B2	B1	B0
T0		acarreo	ϕ_1	A3·Bϕ	A2·Bϕ	A1·Bϕ	Aϕ·Bϕ
T1	ϕ_3		A3·B1	A2·B1	A1·B1	Aϕ·B1	
T2	ϕ_2	A3·B2	A2·B2	A1·B2	Aϕ·B2		
T3	A3·B3	A2·B3	A1·B3	Aϕ·B3			
Producto	P_6	P_5	P_4	P_3	P_2	P_1	P_ϕ

multiplicación de toda la vida, como nos enseñaron en el cole...

P_7 para posible acarreo

2.2 Realización del multiplicador (*15 puntos*)

este lo hago aparte

Para realizar la multiplicación se van a realizar las siguientes operaciones.

- Suma en paralelo, en dos sumadores independientes, para obtener los términos U0=T0+T1 (Sumador_1) y U1=T2+T3 (Sumador_2).
- Una suma a continuación para obtener el resultado P=U1+U2 en otro sumador (Sumador_3).

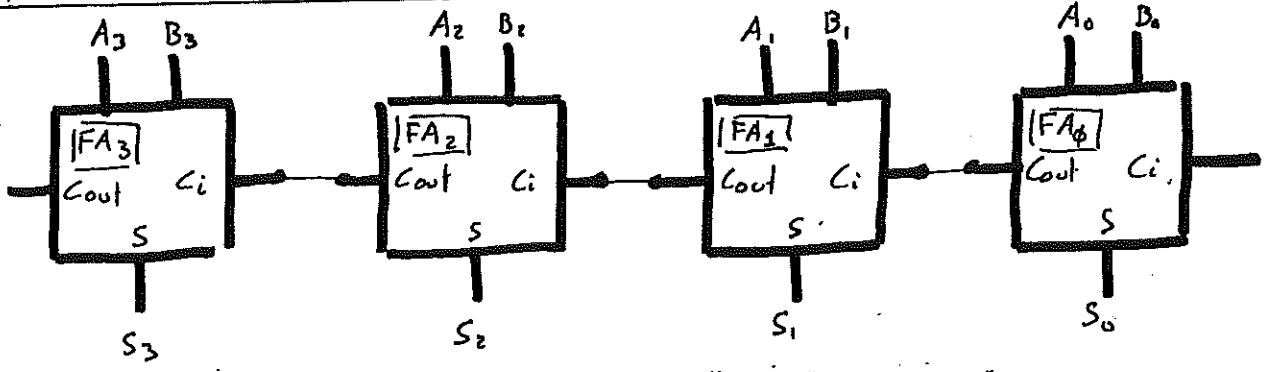
Suponiendo que dispone de los productos parciales **Ai*Bj**, dibuje el esquema del multiplicador con todas sus señales y dimensione la longitud de palabra (número de bits) de cada uno de los sumadores para que ésta sea mínima. NOTA: Razone la longitud de palabra necesaria en cada uno de los sumadores suponiendo que éstos no tienen entrada de acarreo.

2.3 Diseño de un sumador (*5 puntos*)

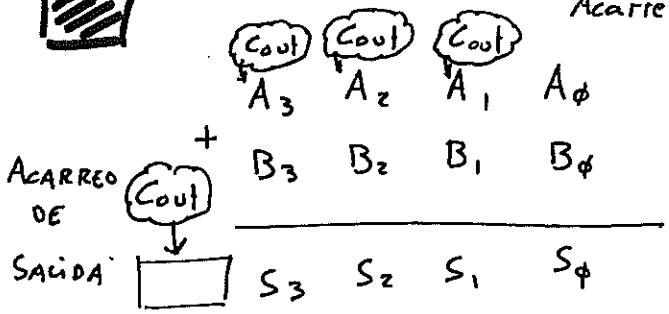
VER PRIMERO EL EJERCICIO 2 DEL TEMA 4 (recordatorio)

Diseñe un sumador paralelo con acarreo serie (Ripple Carry) de dos palabras de cuatro bits utilizando el mínimo número de sumadores completos (Full Adder), obtenga el retardo del mismo suponiendo que $t_{sum} = 1.5\ t_{carry}$ y marque el camino crítico. Razone las decisiones que tome.

Lo que nos piden, justo, lo tenemos en la teoría (T-4.8)



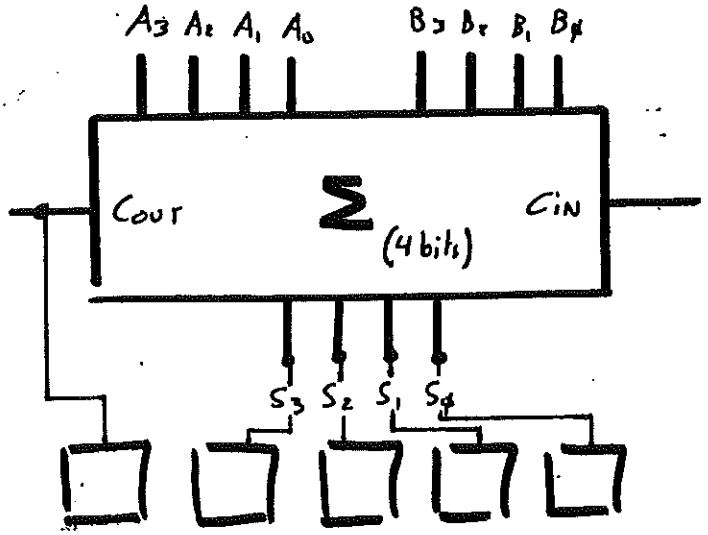
Acarreos del 1º, 2º y 3er sumador, respectivamente



NOTA:

Recuerda SIEMPRE que un sumador es un circuito combinacional, NO secuencial

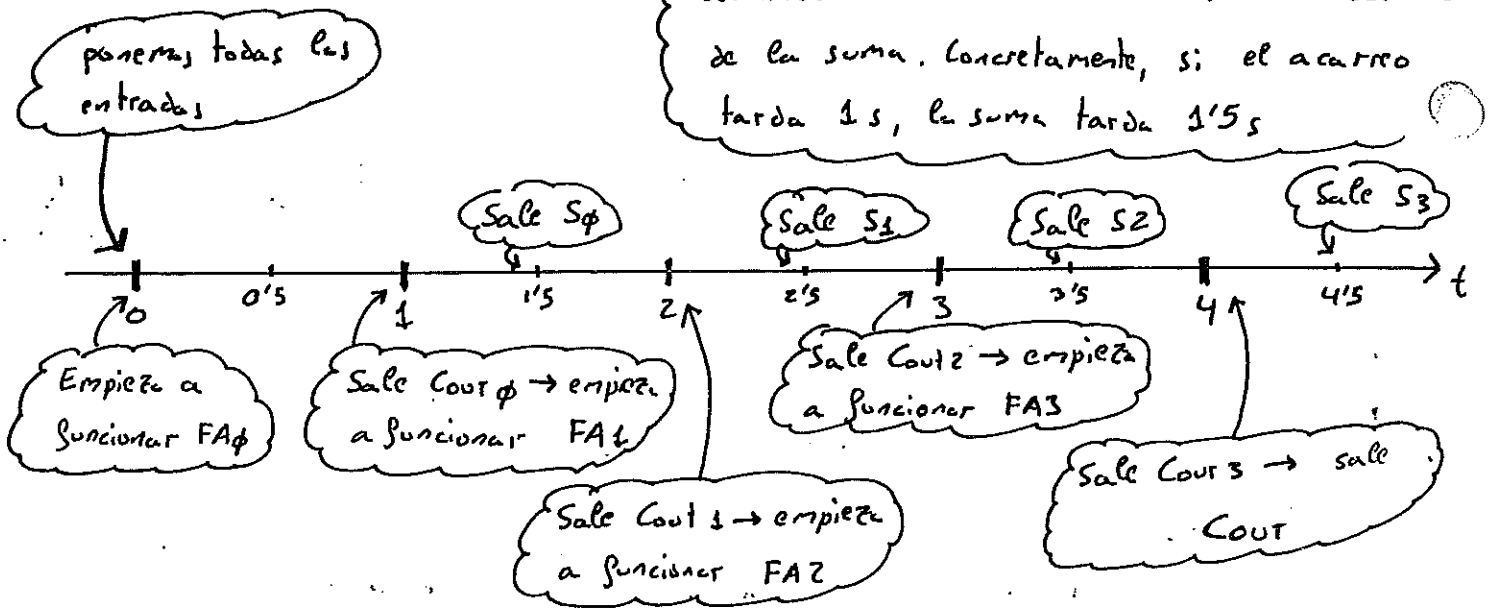
Lo que acabamos de diseñar es:



Camino crítico: es aquel que determina el retardo máximo

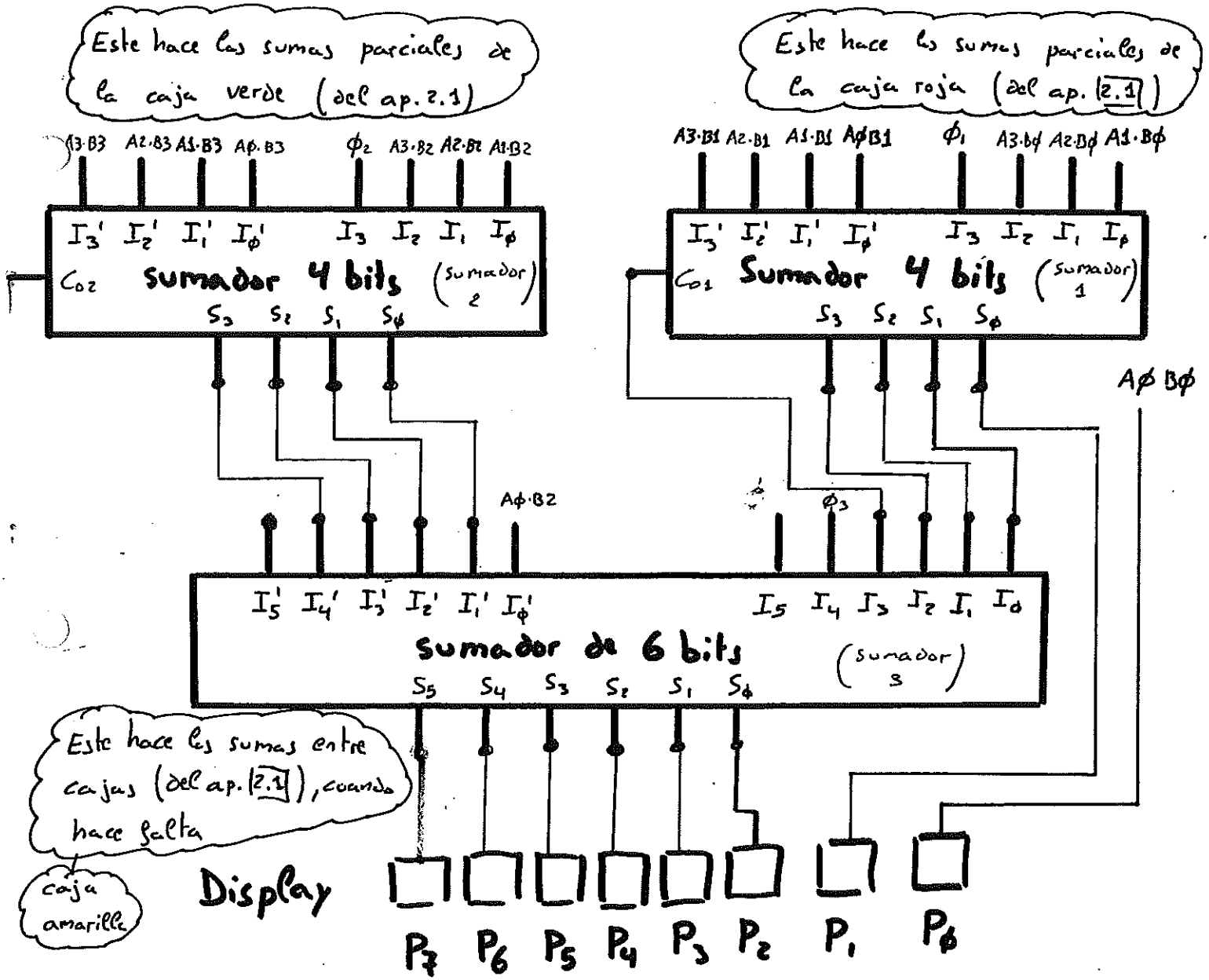
DATO: $t_{sum} = 1'5\ t_{carry}$

Esto quiere decir que, en cada full-adder, el acarreo de salida sale antes que el resultado de la suma. Concretamente, si el acarreo tarda 1 s, la suma tarda 1'5 s



$$t_{retardo} = t_{carry\ FA0} + t_{carry\ FA1} + t_{carry\ FA2} + t_{sum\ FA3} =$$

$$= 3 \cdot t_{carry} + t_{sum} = 3 \cdot t_{carry} + 1'5 \cdot t_{carry} = 4'5\ t_{carry}$$



Basta con mirarlo ordenadamente para entenderlo:

- Sumador 1: hace las sumas de la "caja roja"
  - → $A\phi \cdot B\phi$ no se suma con nadie, directamente a $P\phi$
  - → $A1 \cdot B\phi + A\phi \cdot B1$ va directamente a $P1$ (sin pasar por el sumador 3)
- Sumador 2: hace las sumas de la "caja verde"
  - → $A\phi \cdot B2$ no lo suma con nadie (irá directo al sumador final, el 3) — no entra en la caja verde, va directo al sumador
  - → $A3 \cdot B3$ podría ir directo a $P6$, en vez de eso le sumamos "ϕ".
  - → $A3 \cdot B2 + A2 \cdot B3$ no puede ir directo a $P5$, le falta el acarreo del sumador 1.
- Sumador 3: hace las sumas de la "caja amarilla"
  - → $(A3 \cdot B2 + A2 \cdot B3)$ lo suma con el acarreo del sumador 1
  - → El acarreo del sumador anterior puede ir directo a D final. Lo mismo con A

ponemos "ϕ" para completar las cajas

3 Temporización (*15 puntos*)

En el circuito de la figura 6 se conocen los siguientes parámetros temporales:

- Tiempo de propagación de los flip-flops (t_{ff}): 3ns
- Tiempo de set-up de los flip-flops (t_{su}): 1ns
- Tiempo de retardo de cada puerta (t_{puerta}): 1ns

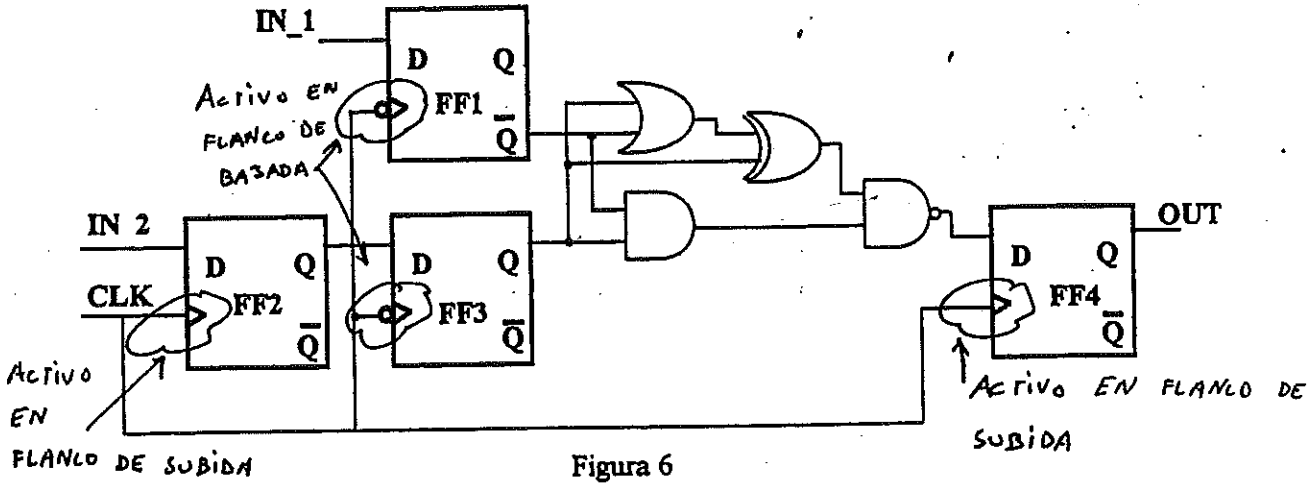
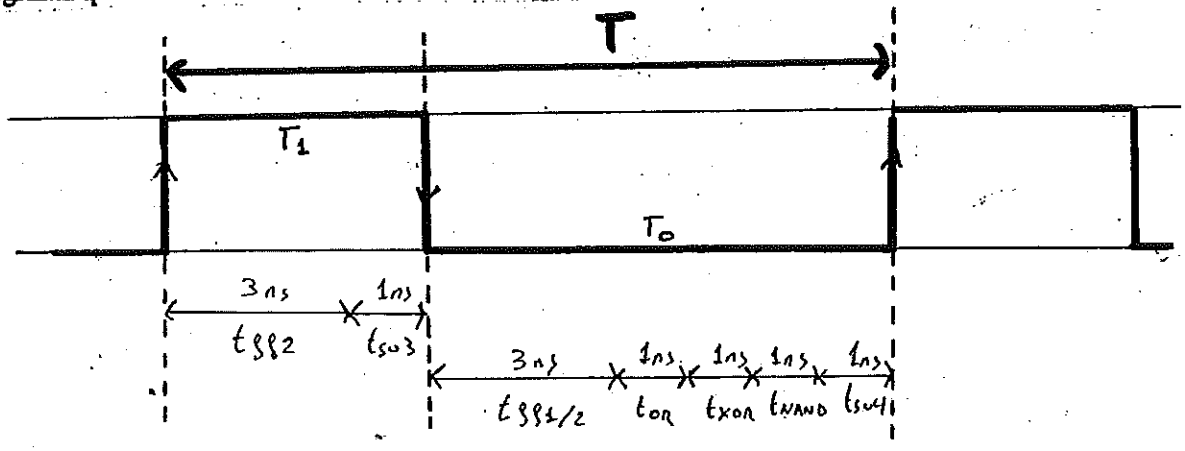


Figura 6

Determinar la máxima frecuencia de reloj que se puede aplicar al sistema, así como el tiempo máximo de hold que podría existir en dichas condiciones. Justifique la respuesta sobre un cronograma que muestre el funcionamiento del circuito.

[2a]



$T_{1min} = 3 + 1 = 4\ ns$

$T_{0min} = 3 + 1 + 1 + 1 + 1 = 7\ ns$

$T_{min} = T_{1min} + T_{0min} = 11\ ns \Rightarrow \boxed{f_{max} = \frac{1}{T_{min}} = 90'9\ MHz}$

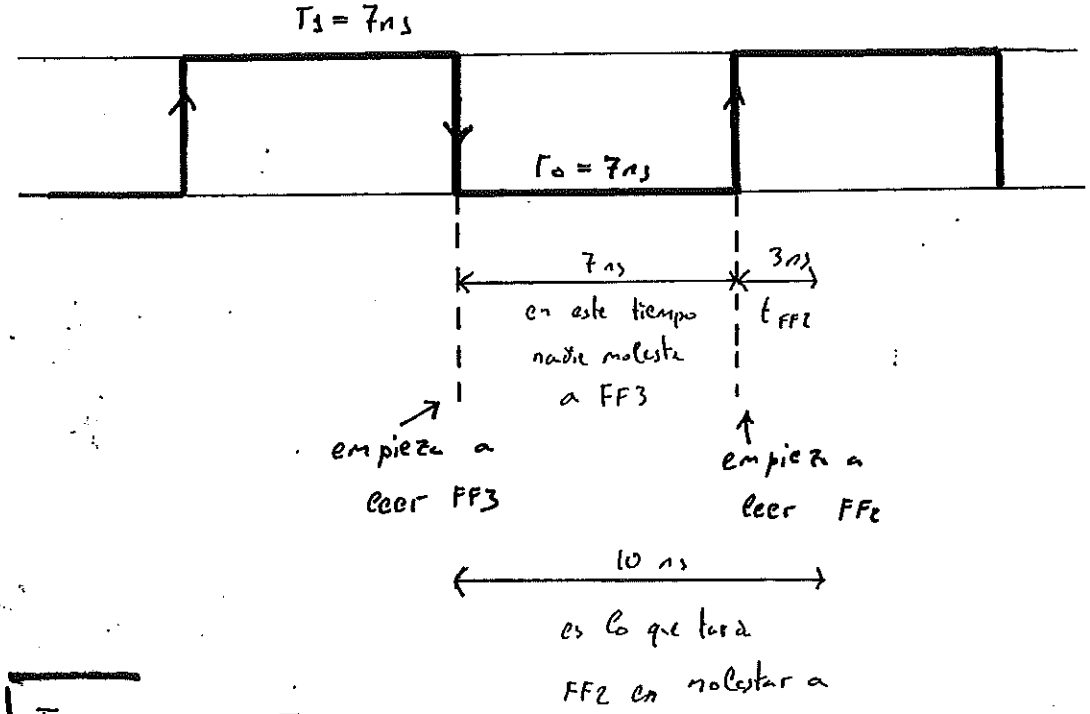
Nota: si quisiéramos un reloj con 50% de DC:

$\frac{T_{min}}{2} = \max\{T_{1min}, T_{0min}\} = 7\ ns$

$T_{min} = 14\ ns \Rightarrow f_{max} = 71'4\ MHz$

esta sue la respuesta oficial.

1b Vamos a suponer los datos obtenidos en el apartado anterior, suponiendo también 50% de ciclo de trabajo (DC)



$T_{HOLD-MAX} = 7 + 3 = 10\ ns$

como mucho, en las características de los biestables, este tiene que ser el tiempo de hold que necesiten, x q como necesiten más no funcionará.

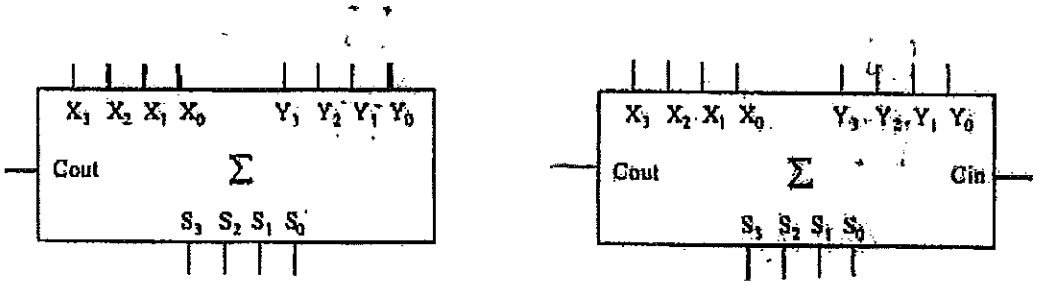
FEBRERO 2001

2. Se desea realizar un sumador/restador de dos números A y B de 5 bits cada uno, representados en complemento a 2. De esta forma, cuando la señal de control M sea igual a 0 deberá realizarse la suma A + B, mientras que si M es igual a 1 se efectuará la resta A – B. Por otra parte, se desea evitar problemas de desbordamiento, para lo cual el resultado de salida deberá tener el número de bits suficiente para poder representar todos los posibles resultados de las operaciones.

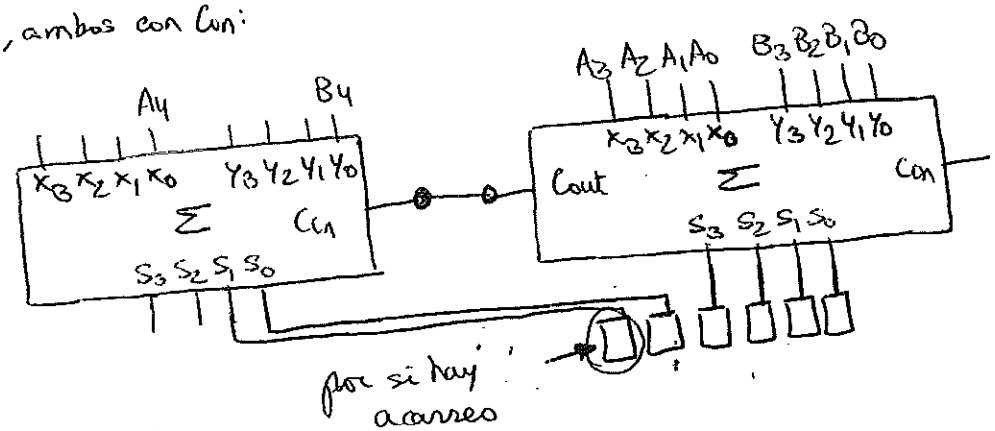
Para realizar el diseño se dispone de puertas XOR de 2 entradas y de dos sumadores *ripple carry* de 4 bits como los que se muestran en la figura 4(advierta que uno de los sumadores carece de acarreo de entrada).

Complete la figura 4 para realizar el sumador/restador utilizando el menor número de puertas XOR adicionales. Indique cuál es el *mínimo número* de bits de salida que garantiza la ausencia de desbordamiento en las operaciones y señale claramente qué terminales escogería como resultado de salida.

M



De momento vamos a implementar sólo un sumador de números positivos con sumadores, ambos con Cin:

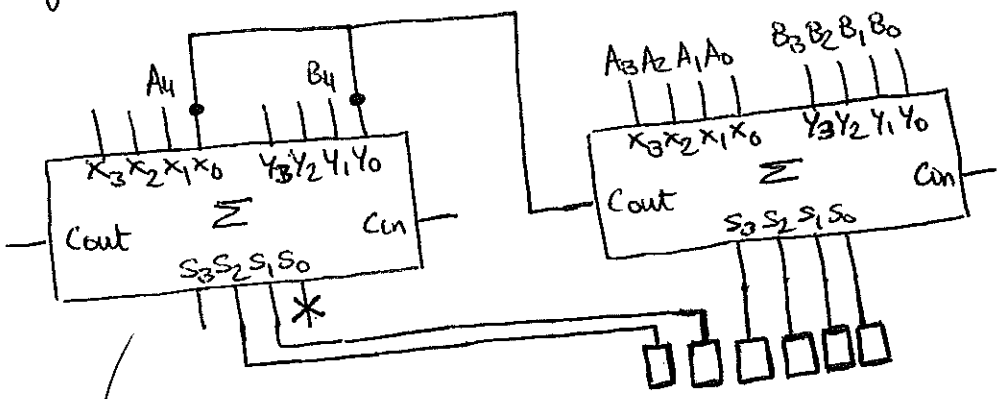


→ nº de bits a la salida

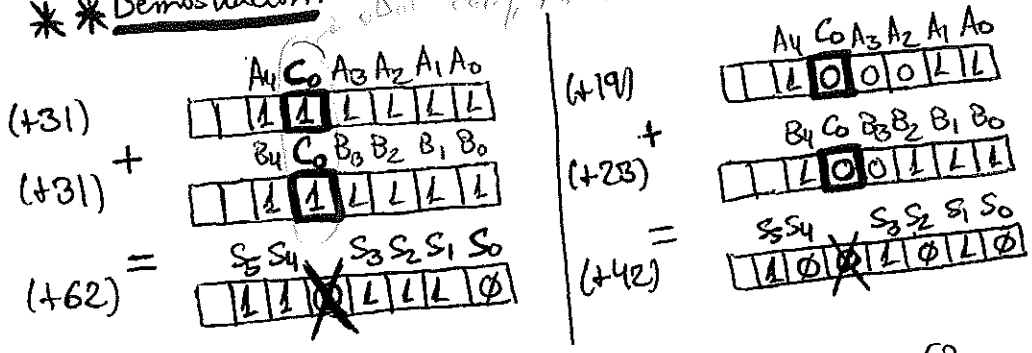
```
  11111  (31)
+ 11111  (31)
 111110  (62)
```

↑ (bit MSS)

Ahora un sumador de enteros positivos pero sin acarreo de entrada en el de cifras significativas



**Demostración:



Finalmente: sumador/restador de números en C2

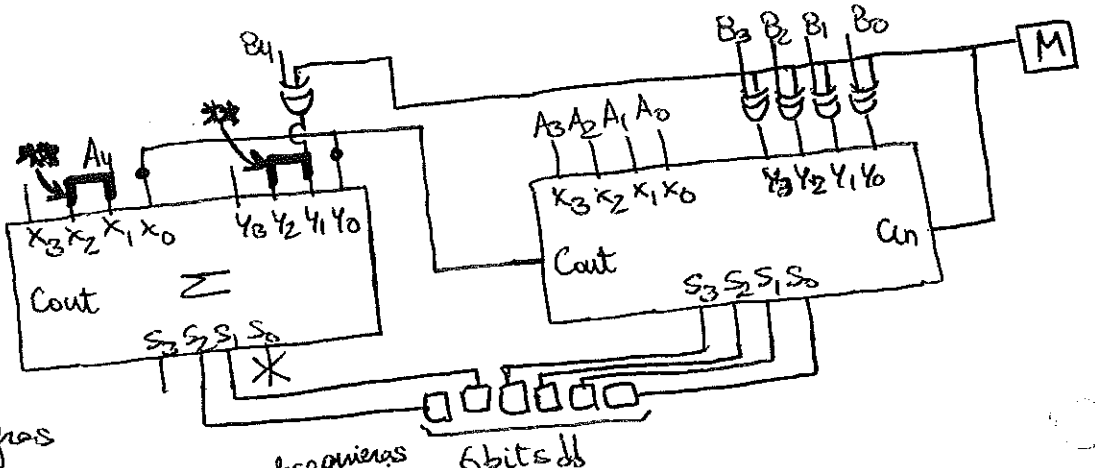
$M=0 \quad S = A+B$

$M=1 \quad S = A-B = A+(-B) = A+\overline{B}+1$

Así, cuando $M=1$ tenemos que:

- sumar 1 ⇒ conectamos M al acarreo de entrada
- negar la B ⇒ probamos poniendo la tabla de verdad de $M \oplus B$ ⇒

M	B	M⊕B
0	0	0 (deja pasar la B tal cual)
0	1	1
1	0	1 (negamos la B tal cual)
1	1	0



Tenemos a la entrada 5 cifras y 6 a la salida ⇒ necesitamos una **extensión de signo**: con positivos añades ceros: 00010 (los queremos) con negativos añades unos: 11010 (ver siguiente hoja ↓)

Nota: al operar números en CA2, si llegamos a la conclusión de que necesitamos 6 bits a la salida, tenemos que operar con números de 6 bits a la entrada. Como eran números de 5 bits hacemos **extensión de signo**. Ejemplo: $\emptyset\emptyset\ \boxed{\emptyset}\ 0\,1\,0\,1\,0$ → positivo; $1\,1\ \boxed{1}\ 1\,1\,0\,1\,1$ → negativo

si hicieran falta más meteríamos + ceros o unos (positivo/negativo)

sin extensión

$$\begin{array}{r} \boxed{1}\,0\,0\,0\,0 \\ +\ \boxed{0}\,0\,0\,1\,1 \\ \hline \boxed{0}\,1\,0\,0\,1\,1 \end{array} \quad \begin{array}{l} -16 \\ +\ 3 \\ \hline +19 \end{array}$$

con extensión

$$\begin{array}{r} 1\,1\,0\,0\,0\,0 \\ +\ 0\,0\,0\,0\,1\,1 \\ \hline 1\,1\,0\,0\,1\,1 \end{array} \quad \begin{array}{l} -16 \\ +\ 3 \\ \hline -13\ \checkmark \end{array}$$

Por cierto, que el ejemplo anterior sigue funcionando con extensión:

$$\begin{array}{r} 1\,1\,0\,0\,0\,0 \\ +\ 1\,1\,0\,0\,0\,0 \\ \hline \cancel{1}\,1\,0\,0\,0\,0\,0 \end{array} \quad \begin{array}{l} -16 \\ +\ -16 \\ \hline -32\ \checkmark \end{array}$$

FEBRERO 2001

Problema 4

Realizar el diseño de la figura, para lo cual se han empleado componentes con los siguientes parámetros temporales:

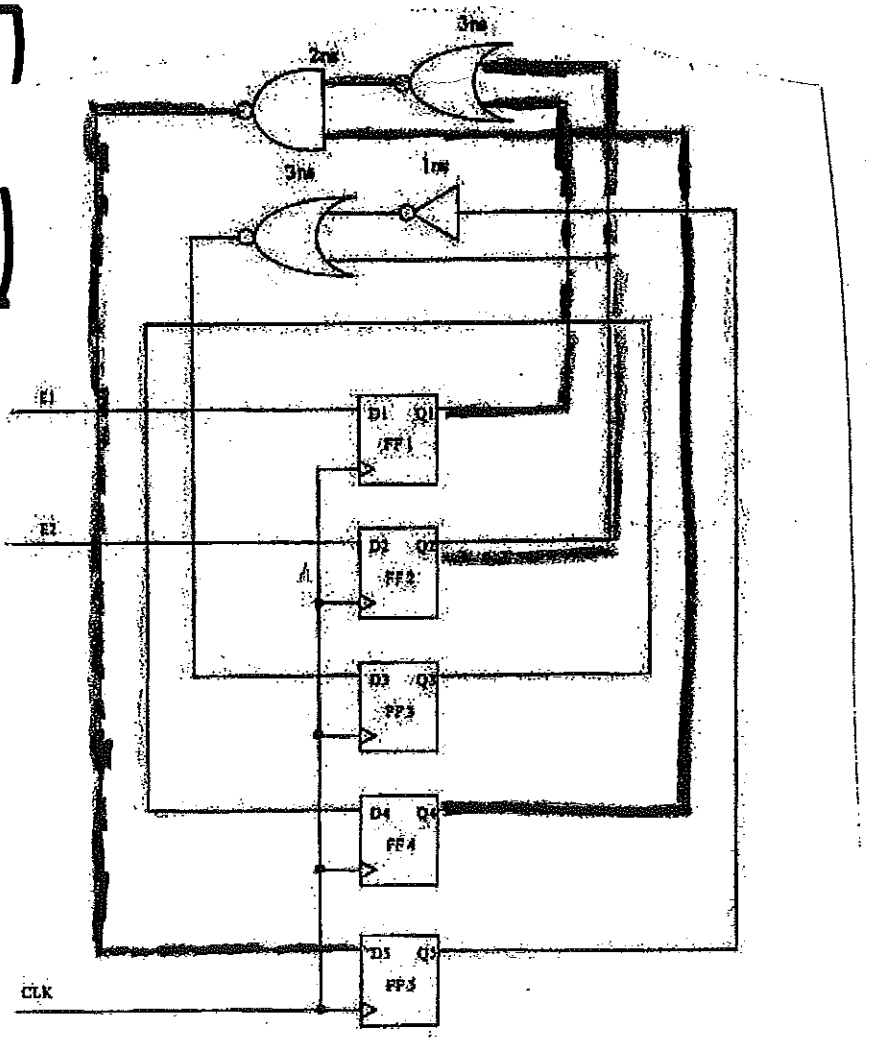
Flip-Flops	Puertas
$t_{FF1} = t_{FF2} = 5$ ns	$t_{nand} = 2$ ns
$t_{FF3} = t_{FF4} = 10$ ns	$t_{nor} = 3$ ns
$t_{FF5} = 3$ ns	$t_{inv} = 1$ ns
$t_{setup} = 2$ ns	

4.1 Caminos críticos
Determine cuáles son los tiempos máximos y mínimos que tardarán en estabilizarse las entradas de datos de los biestables D_3, D_4 y D_5 desde que se produce un flanco de subida de reloj.

4.2 Frecuencia máxima
Calcule la máxima frecuencia de reloj a la que puede funcionar el circuito. Indique claramente cuál es el camino crítico que la determina.

4.3 Tiempo de hold
Calcule el máximo tiempo de hold que pueden tener los biestables para garantizar el correcto funcionamiento del circuito, e indique el camino que lo origina.

TODOS LOS RELOJES SON DE FLANCO DE SUBIDA



4.1 Caminos críticos: tiempos de estabilización de las entradas:

D_3 → $D_{3max} = max\{ \underbrace{t_{FF5} + t_{inv} + t_{nor}}_{\text{hay dos caminos que llegan a } D_3}, \underbrace{t_{FF2} + t_{nor}}_{} \} =$

$= max\{3 + 1 + 3,\ 5 + 3\} = max\{7, 8\} = 8\ ns$

→ $D_{3min} = min\{7, 8\} = 7\ ns$

D_4 → $D_{4max} = t_{FF3} = 10\ ns$

→ $D_{4min} = t_{FF3} = 10\ ns$

} sólo hay un camino que llegue a D_4

D_5 → $D_{5min} = min\{ \underbrace{t_{FF4} + t_{NAND}}_{\text{hay dos caminos posibles}},\ \underbrace{min\{t_{FF1}, t_{FF2}\} + t_{NOR} + t_{NAND}}_{\text{las entradas de la NOR vienen de FF distintos}} \} =$

(elegimos la que cambia más rápido)

$= min\{10 + 2,\ min\{5, 5\} + 3 + 2\} =$

$= min\{12, 10\} = 10\ ns$

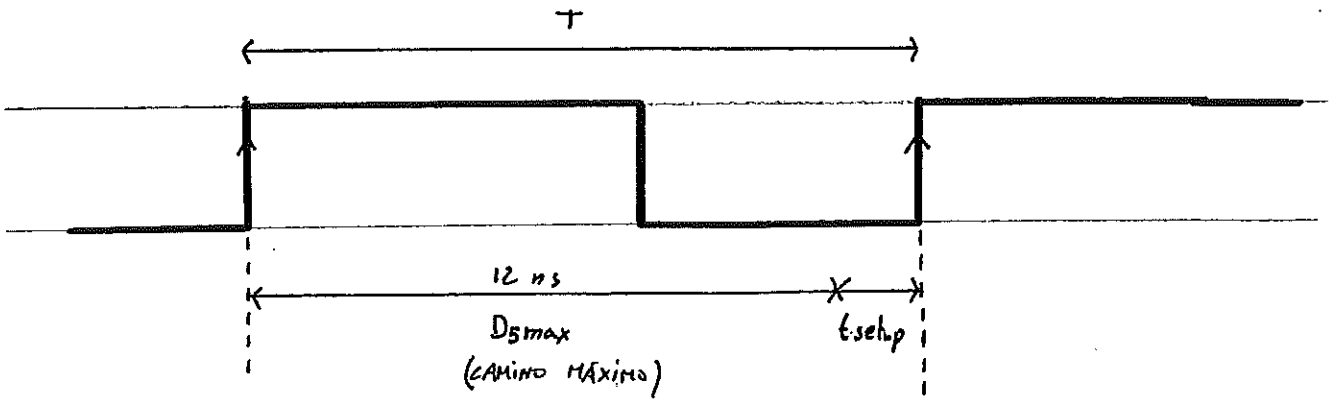
→ $D_{5max} = max\{12, 10\} = 12\ ns$

4.2 Frecuencia máxima (buscamos el camino máximo, y le sumamos tsetup.)

$T_{min} = max\{D_{3max}, D_{4max}, D_{5max}\} + t_{setup} = max\{8, 10, 12\} +$

$+ t_{setup} = 12 + 2 = 14\ ns$

$$f_{max} = \frac{1}{T_{min}} = \frac{1}{14\ ns} = 71'4\ MHz$$



ls de los tres biestables ya están calculados en (caminos mínimos) tenemos que fijar un $t_{HOLDMAX}$ para TODOS los biestables

$s_{min} = 7\,ns$

$t_{min} = 10\,ns$

$_{min} = 10\,ns$

El tiempo de HOLD máximo de TODOS los biestables del circuito, será el tiempo más restrictivo de todos estos:

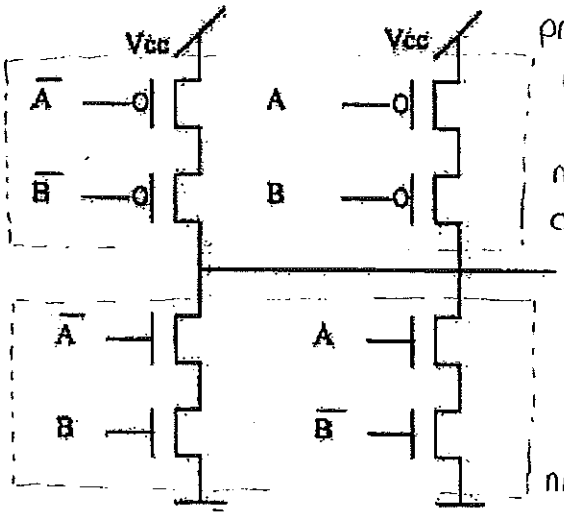
$$min\{t_{HOLDMAX3}, t_{HOLDMAX4}, t_{HOLDMAX5}\} = 7\,ns$$

SEPTIEMBRE 2001 CMOS .

1.1. Obtenga razonadamente la función lógica que realiza el circuito de la figura 1 y rellene la tabla adjunta con el resultado de la misma. Diga de qué función lógica se trata.

Probamos Vopción :

A	B	S
0	0	
0	1	?
1	0	
1	1	

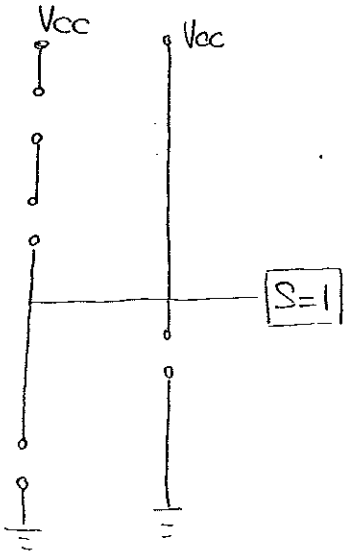


pmos.
Estos bichos se activan con un 0 : cuando les meto un 0 son CORTOS, cuando les meto 1 son ABIERTOS

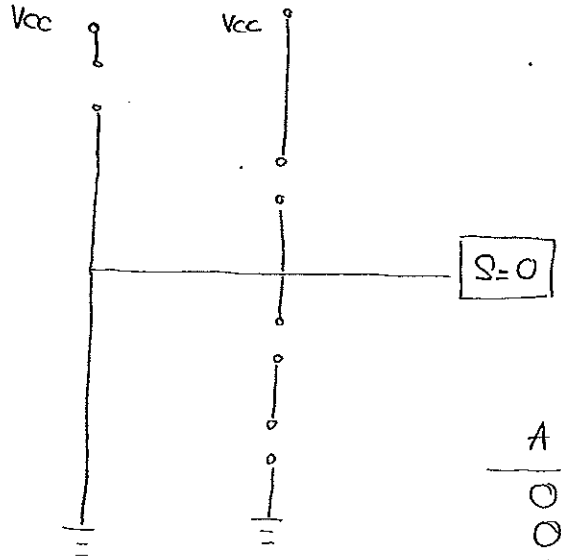
Estos bichos se activan con un 1 : cuando les meto un 1 son CORTOS, si les meto 0 son ABIERTOS

nmos.

A=0 B=0



A=0 B=1

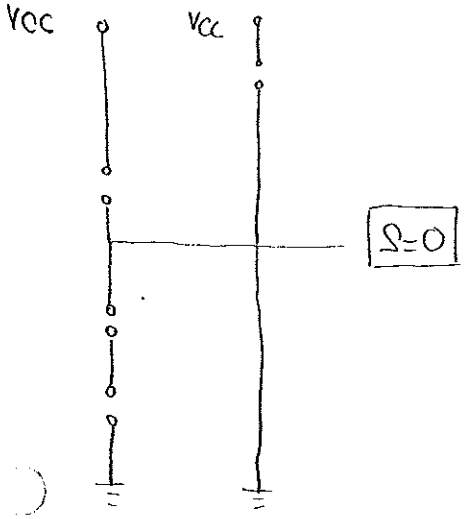


$S=\overline{A \oplus B}$

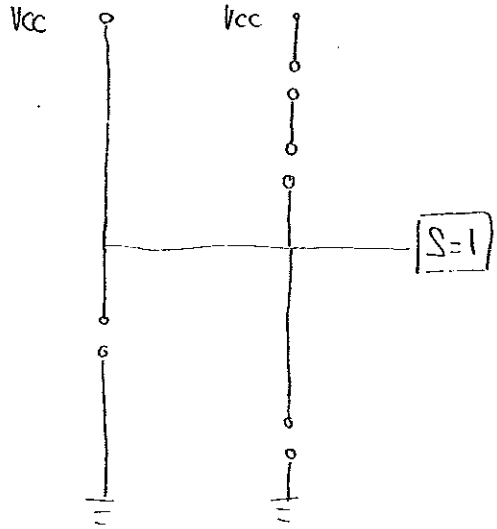
III

A	B	S
0	0	1
0	1	0
1	0	0
1	1	1

A=1 B=0



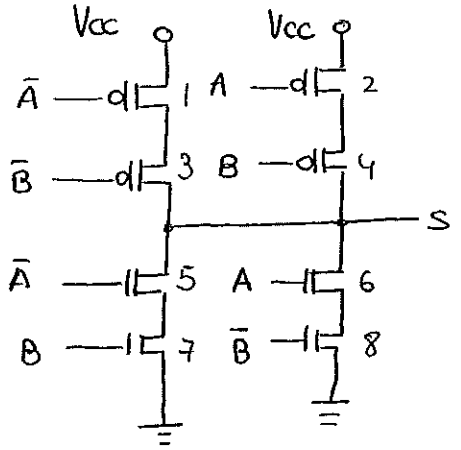
A=1 B=1



YA PUEDO "RELLENAR" LA TABLA DE VERDAD

PERO ...

2ª FORMA de hacer este problema : LÓGICA PROPOSICIONAL.



para que por S salga Vcc ≡ $\boxed{S=1}$ de los cables!

para que por S salga TIERRA ≡ $\boxed{S=0}$ tb de los cables!

$\boxed{S=1} \leftrightarrow (T_1=ON \wedge T_3=ON) \vee (T_2=ON \wedge T_4=ON) \leftrightarrow$

$\leftrightarrow (\bar{A}=0 \wedge \bar{B}=0) \vee (A=0 \wedge B=0) \leftrightarrow$

$\leftrightarrow (A=1 \wedge B=1) \vee (A=0 \wedge B=0) \leftrightarrow$

$\leftrightarrow (A \cdot B) + (\bar{A} \cdot \bar{B}) = S = \overline{A \oplus B}$

SÓLO HACE FALTA SACAR UNO!!

$\boxed{S=0} \leftrightarrow (T_5=ON \wedge T_7=ON) \vee (T_6=ON \wedge T_8=ON) \leftrightarrow$

$\leftrightarrow (\bar{A}=1 \wedge B=1) \vee (A=1 \wedge \bar{B}=1) \leftrightarrow$

$\leftrightarrow (A=0 \wedge B=1) \vee (A=1 \wedge B=0) \leftrightarrow$

$\leftrightarrow (\bar{A} \cdot B) + (A \cdot \bar{B}) = \bar{S} = A \oplus B \leftrightarrow S = \overline{A \oplus B}$

* ¿Ves cómo se hace?

" ∨ " = OR = +

" ∧ " = AND = •

Unidad a 1 : sin negar

Unidad a 0 : negada

SEPTIEMBRE 2001

TIEMPO

1.2. Una forma de obtener pulsos con una duración determinada para poder utilizarlas como ventanas de tiempo es la que se presenta en la figura 2. En ella se realiza una función OR exclusive de una señal con ella misma retardada un tiempo determinado. Suponiendo que la señal de entrada varía entre 0V y 5V, que la tensión de conmutación de las entradas es de 2,5 V y R = 10K, obtenga razonadamente el valor de la capacidad C para obtener un pulso a la salida con una duración de 10 ms y dibuje un cronograma con las formas de onda de las señales de salida (3) y las dos entradas (1) y (2).

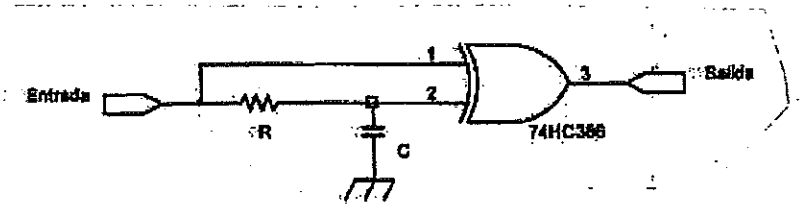
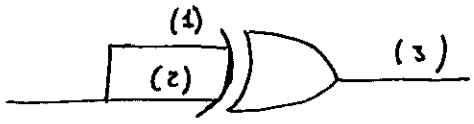
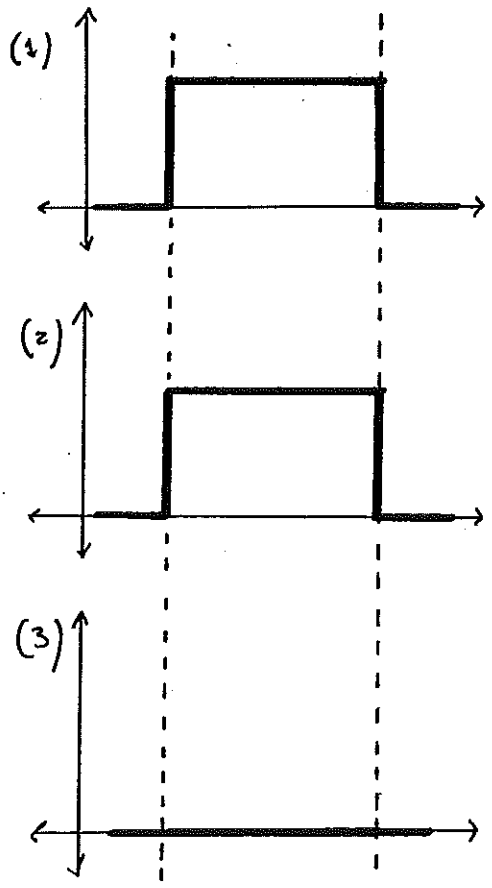


Figura 2

Previo :

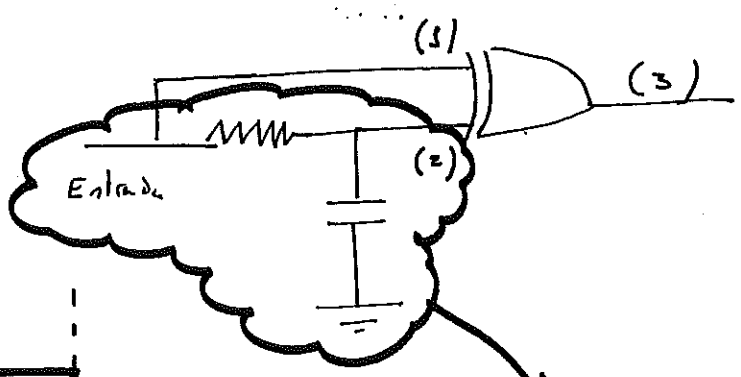
ejercicio ideal

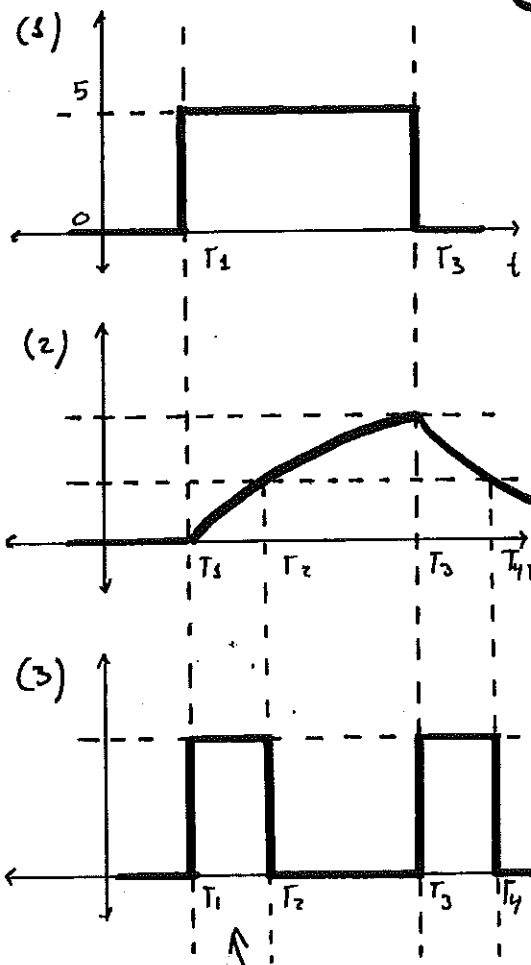




A	B	Xor
0	0	0
0	1	1
1	0	1
1	1	0

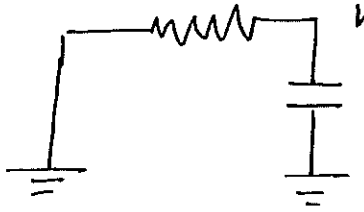
Ejercicio real





buscamos que este pulso dure 10ms

Circuito antes de conmutar de 0 a 5V



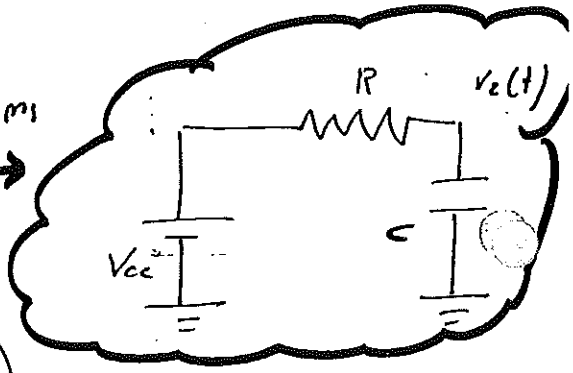
Recordatorio:

PVI: $\begin{cases} \tau \frac{dy}{dt} + y = c \\ y(0) = CI \end{cases}$

Solución:

$y(t) = (ci - cf)\, e^{-\frac{t}{\tau}} + ci$

Circuito justo después de conmutar de 0 a 5 V



$$V_2(t) = (CI - CF)\cdot e^{-\frac{t}{\tau}} + CF$$

$$V_2(t) = 5\left(1 - e^{-\frac{t}{RC}}\right)$$

$CI = 0$

$CF = 5$

$\tau = R \cdot C$

Además nos dan un dato: $R = 10\,K$

Nos exigen que cuando $t = 10ms$, $V_2(t) = 2'5\,V \Rightarrow V_2(10ms) = 2'5$

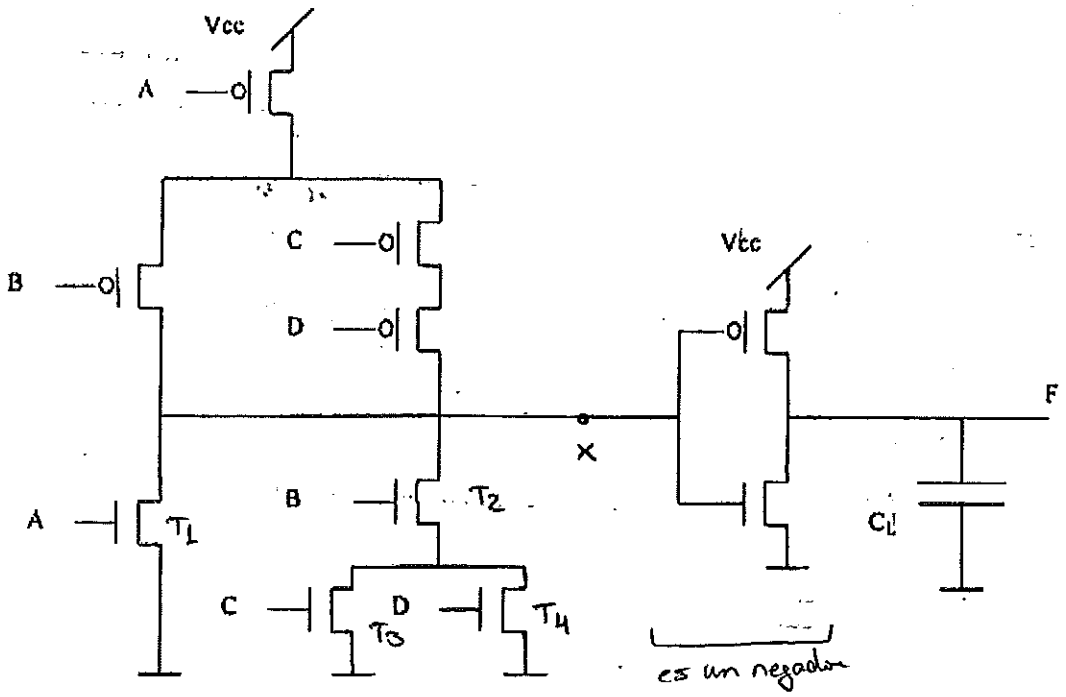
Sustituyendo: $2'5 = 5\left(1 - e^{-\frac{0'01}{10^4 C}}\right) \Rightarrow 0'5 = 1 - e^{-\frac{0'01}{10^4 C}} \Rightarrow$

$\Rightarrow e^{-\frac{0'01}{10^4 \cdot C}} = 0'5 \Rightarrow \ln(0'5) = -\frac{1}{10^6 C} \Rightarrow$

$\Rightarrow \boxed{C = 1'443 \cdot 10^{-6}\ F}$

FEBRERO 2002

1.1. Obtenga razonadamente la expresión de la función lógica **F** que realiza el circuito de la figura 1.



1.2. Obtenga justificadamente los tiempo de propagación de la función lógica F en los siguientes casos: t_{pLHmax} y t_{pHLmin}, marcando en la figura 1 los caminos de la señal que corresponden a cada caso. La resistencia de ON de los transistores pMOS y nMOS es $R_p = 200\Omega$ y $R_n = 100\Omega$ respectivamente, la de OFF despreciable en ambos casos (> 10MΩ), la capacidad de puerta de los transistores es $C_{GD} = 1nF$, la capacidad $C_L = 100nF$ y la tensión de conmutación del inversor es de 0.5 V_{CC}.

1.1 $F = \overline{X}$ para $X=0 \Leftrightarrow T_1 = ON \vee (T_2 = ON \wedge (T_3 = ON \vee T_4 = ON)) \Leftrightarrow$

$\Leftrightarrow A = 1 \vee [B = 1 \wedge (C = 1 \vee D = 1)] \Leftrightarrow$

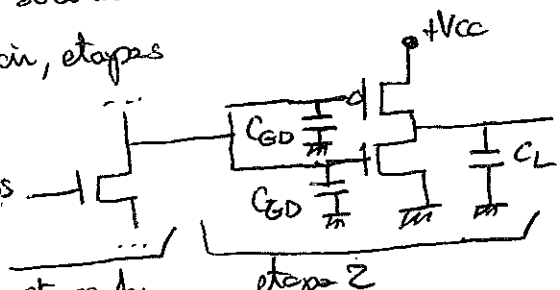
$\Leftrightarrow \boxed{A + B(C+D) = \overline{X} = F}$ ← para $x=1$ + largo y difícil

1.2 $R_{p-ON} = 200\Omega$ $C_{GD} = 1nF$ Tensión de conmutación = $V_{CC}/2$ (para los tiempos de propagación)

$R_{n-ON} = 100\Omega$ $C_L = 100nF$

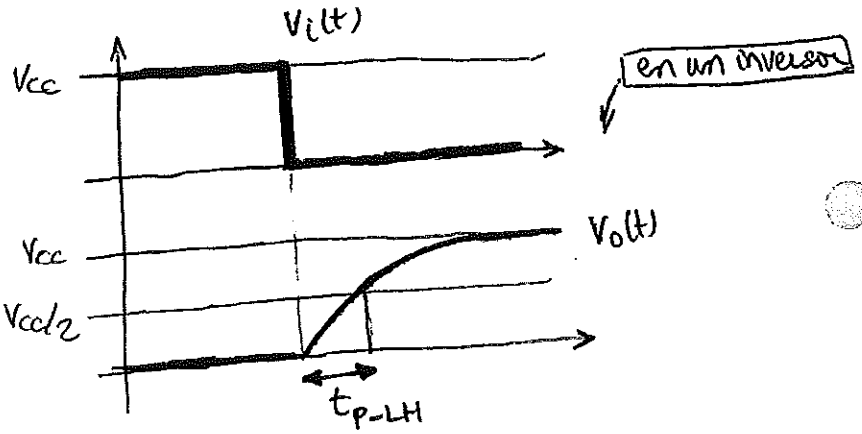
Nota: C_{GD} = capacidad a la entrada del TRT. Sólo interesa cuando enganchamos cosas a la izqda y derechas, es decir, etapas

⇒ la dificultad de este problema es la 3ª de capacidades en los TRT. Aun así, cuando estudiamos el comportamiento dinámico de un dto, sólo nos interesan las capacidades que quedan a la salida del mismo. Para este ejercicio vamos a estudiar por separado el t_p de las 2 etapas:



- 2ª etapa: estudiamos los FETs del inversor con la capacidad C_L
- 1ª etapa: " " " del circuito de la izqda actuando sobre las capacidades del inversor

[a] ¿$t_{p-LH-máx}$?



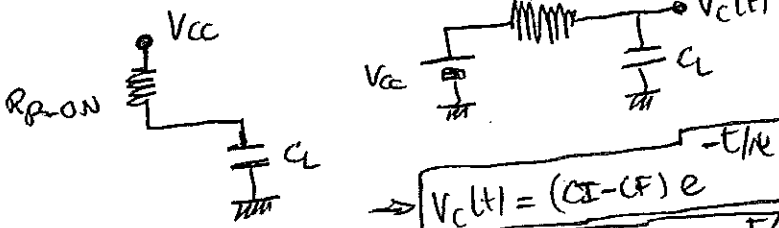
Recordatorio: t_{p-LH}

Como estudiamos las 2 etapas por separado:

$$t_{p-LH-máx} = t_1 + t_2$$

→ **Cálculo de t_1:** es el t_{p-LH} del inversor (2ª etapa):

Directamente estudiamos el transitorio (HIGH):



$C_I = 0$

$C_F = V_{cc}$

$\tau = R_{PON} \cdot C_L$

$$V_c(t) = (C_I - C_F)\, e^{-t/\tau} + C_F = -V_{cc}\, e^{-t/R_{PON}C_L} + V_{cc} =$$

$$\Rightarrow \boxed{V_c(t) = V_{cc}\left(1 - e^{-t/R_{PON}\cdot C_L}\right)}$$

Buscamos ahora el tiempo tal que $V_c(t) = \frac{V_{cc}}{2}$

$$V_c(t_1) = \frac{V_{cc}}{2} = V_{cc}\left(1 - e^{-t_1/R_{PON}\cdot C_L}\right) = 0{,}5 \Rightarrow e^{\frac{-t_1}{R_{PON}C_L}} = 0{,}5$$

$$\boxed{t_1 = \ln(0{,}5)\, R_{PON}\cdot C_L\, (-1) = 13{,}86\ \mu s}$$

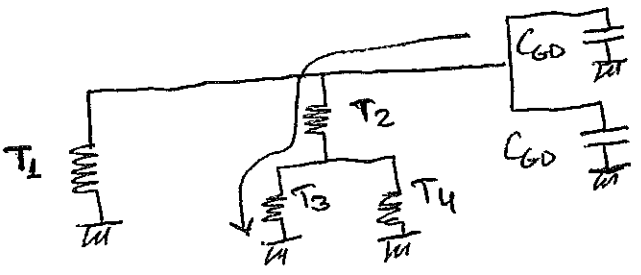
→ **Cálculo de t_2:** es el t_{p-HL} del circuito CMOS de la izquierda (1ª etapa) porque esta señal será HIGH → LOW porque esta señal será la entrada del inversor en el que acabamos de provocar un cambio LOW → HIGH

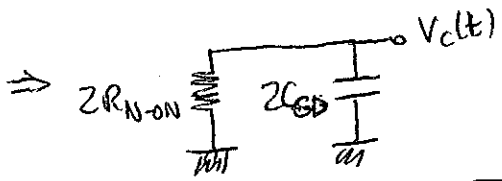
⇒ Directamente estudiamos el transitorio:

¿camino resistivo más grande?

el que pasa por T_2 y T_3 o T_2 y T_4.

ya que si $T_3 = T_4 = ON$ ⇒ R/2 !! porque están en paralelo !!





$C_I = V_{cc}$

$C_F = 0$

$\tau = 4R_{N-ON}\cdot C_{GD}$

$$V_c(t_2) = \frac{V_{cc}}{2} = V_{cc}\, e^{-t_2/4R_{N-ON}C_{GD}}$$

$$\Rightarrow t_2 = \ln(0{,}5)\cdot 4R_{N-ON}C_{GD}(-1)$$

$$\boxed{t_2 = 0{,}277\ \mu s}$$

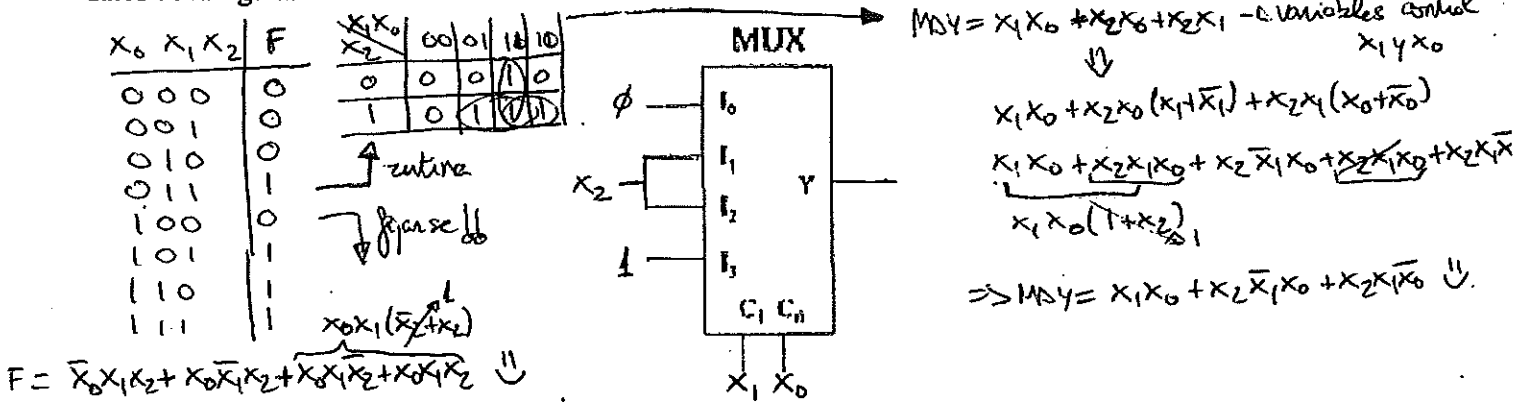
→ Finalmente: $t_{p-LH-máx} = t_1 + t_2 = 14{,}14\ \mu s$

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:

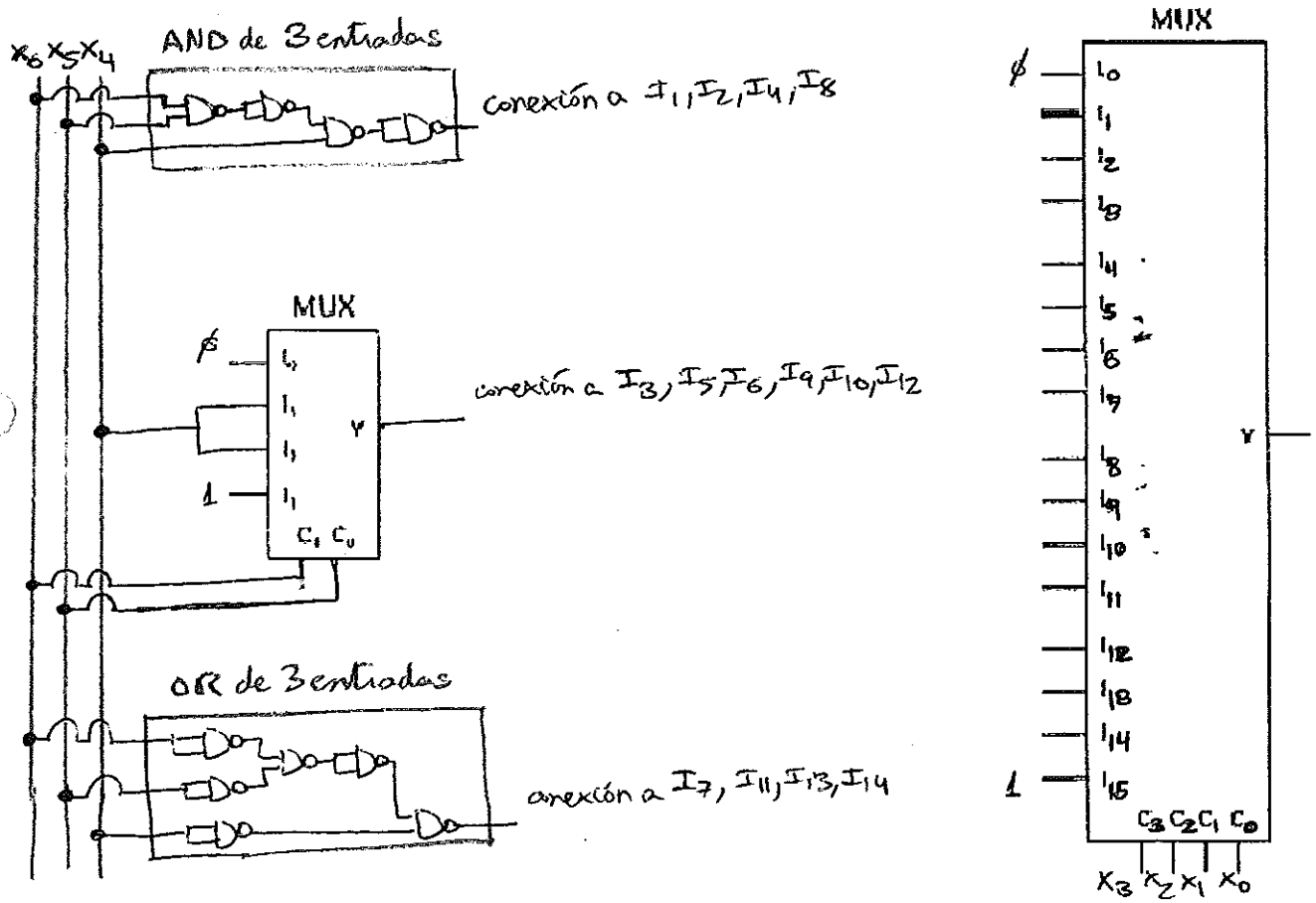
- MAY(1,1,0) = 1 (pues hay más unos que ceros)
- MAY(1,0,0) = 0 (pues no hay más unos que ceros)

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.

2.3 queremos implementar

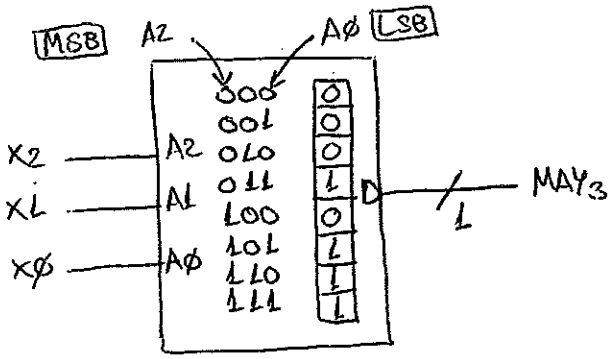
Tenemos una función:

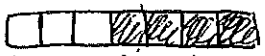
→ de tres variables ⇒ $n=3$

→ de un solo bit ⇒ $b=1$

→ de 8 filas (2^3) ⇒ 8 palabras

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



2.2 función MAY_7 ⇒ 7 cifras 

casos posibles, fijándonos en $x_3 \dots x_0$:

caso [1] · tener 4 ceros: directamente $MAY_7 = 0$

caso [2] · tener 4 unos: " $MAY_7 = 1$

caso [3] · tener 3 ceros y 1 uno: bastará un cero más para que $MAY_7 = 0$

caso [4] · tener 3 unos y 1 cero: " - uno " " " $MAY_7 = 1$

[resto] · tener 2 unos y 2 ceros: entonces la decisión será de MAY_3 (fijándonos en x_6, x_5, x_4).

MUX 16x4:1 $C_3\ C_2\ C_1\ C_0$

I_0 0 0 0 0 → caso [1] ⇒ $I_0 = \phi$

I_1 0 0 0 1 → caso [3]

I_2 0 0 1 0 → caso [3]

[resto] I_3 0 0 1 1

I_4 0 1 0 0 → caso [3]

I_5 0 1 0 1

I_6 0 1 1 0

I_7 0 1 1 1 → caso [4]

I_8 1 0 0 0 → caso [3]

I_9 1 0 0 1

I_{10} 1 0 1 0

I_{11} 1 0 1 1 → caso [4]

I_{12} 1 1 0 0

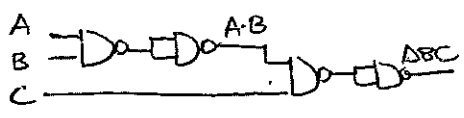
I_{13} 1 1 0 1 → caso [4]

I_{14} 1 1 1 0 → caso [4]

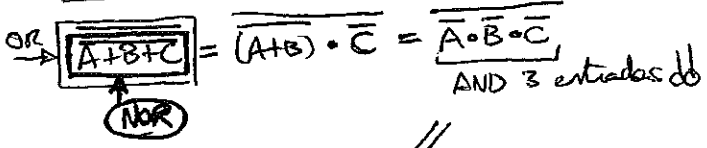
I_{15} 1 1 1 1 → caso [2] ⇒ $I_{15} = 1$

- para el caso [3] basta con tener un cero más ⇒ de lo que te venga con haber un cero más ⇒ salida = ϕ ⇒ esto es una AND !!
- para el caso [4]: basta con tener un uno más ⇒ de lo que te venga con haber un uno más ⇒ salida = 1 ⇒ esto es una OR !!

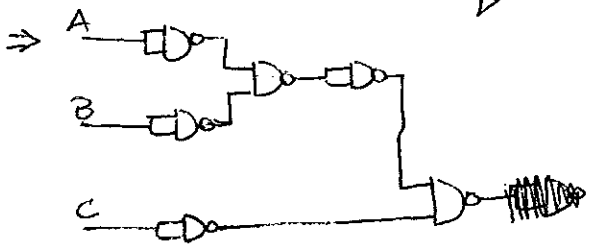
AND 3 entradas con NAND



OR 3 entradas con NAND



⇒



¡ MUY IMP !

Ver Nota 3

3 Problema 3

El objetivo de este problema es diseñar un circuito que funcione como los dígitos de las unidades y decenas de segundos de un cronómetro digital, es decir que cuente desde 0 hasta 59 y vuelta a empezar desde 0. El valor de la cuenta debe entregarse con dos dígitos en formato BCD.

3.1 El circuito de cuenta de segundos lo vamos a realizar con dos contadores hexadecimales de 4 bits como los de la Figura 3.1. Dichos contadores poseen una entrada síncrona de ENABLE activa a nivel bajo, una entrada síncrona LOAD, de carga, activa a nivel bajo y una entrada asíncrona CLEAR activa a nivel bajo. Se dispone de una señal de reloj CLK1 de frecuencia 1 Hz que se utilizará como referencia de tiempo.

El contador U0 mostrará el dígito D0 que indica las unidades de segundos (valores entre 0 y 9) mientras que U1 mostrará el dígito D1 que indica las decenas de segundos (valores entre 0 y 5), ambos en formato BCD.

Realice todas las conexiones necesarias en el esquema de la Figura 4 y, en caso de necesitar puertas lógicas, utilice el mínimo número de puertas NAND de dos entradas. Indique claramente que señales corresponden a D0 y D1. **(5 puntos)**.

NOTA: Las entradas síncronas se activan/ejecutan con los flancos de reloj mientras que las asíncronas se ejecutan instantáneamente, independientemente de la señal del reloj.

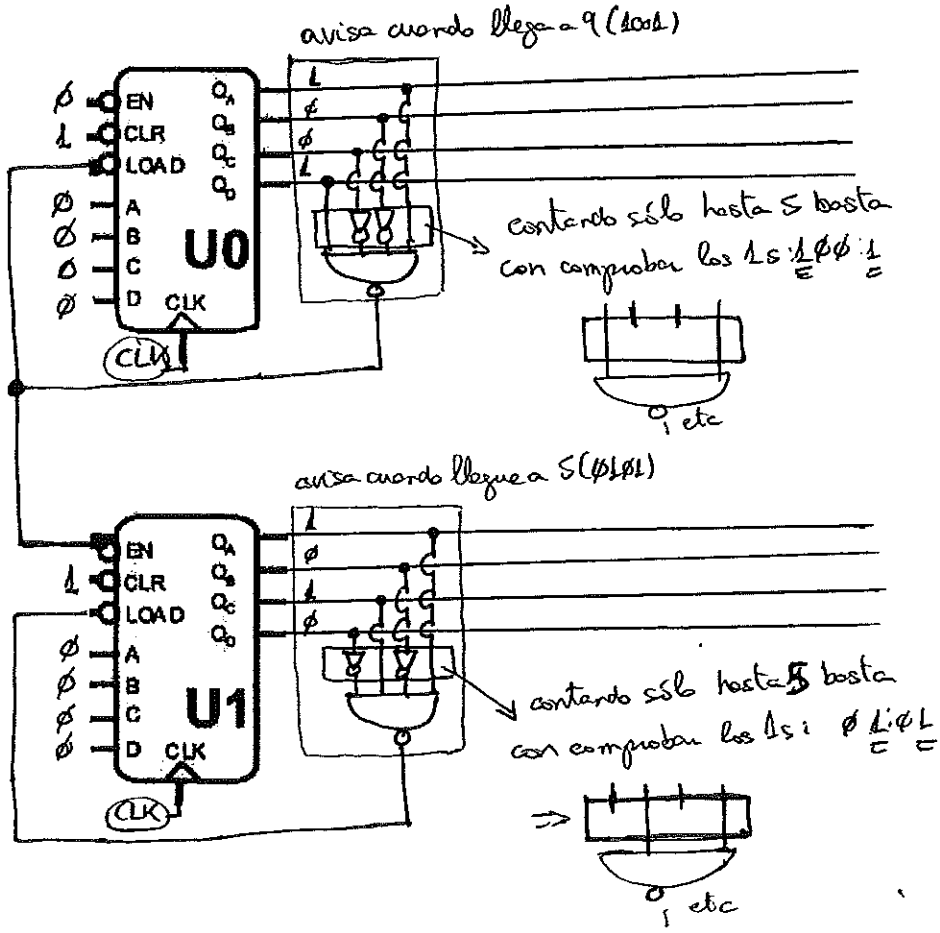
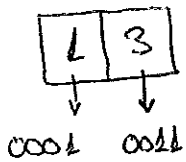


Figura 4

Nota: **SISTEMA DE NUMERACIÓN BCD**

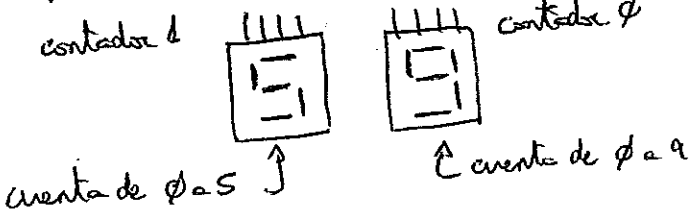


$0'13 = BCD\ 0001\ 0011$

$0'59 = BCD\ 0101\ 1001$

0:	0000
1:	0001
2:	0010
3:	0011
4:	0100
5:	0101
6:	0110
7:	0111
8:	1000
9:	1001

Nos piden hacer un contador que cuente "zero"



3.2 En este apartado se trata de implementar las funciones de START, STOP y RESET. Dichas señales provienen de cuatro pulsadores como el de la Figura 5 De esta forma las señales están a nivel bajo (0 V) en condición de reposo, pasando a nivel alto (5 V) mientras se acciona el correspondiente pulsador. Al dejar de pulsar, la señal vuelve al estado de reposo (0 V). Considere que la duración de una pulsación dura varios ciclos de reloj. Las funciones de cada una de las entradas son las siguientes:

START: Inicio de la marcha del cronómetro.

STOP: Parada de la marcha del cronómetro.

RESET: Debe inicializar el cronómetro a cero segundos. En caso de que estuviera en marcha deberá pararlo y ponerlo a cero, a la espera de que se active la señal **START**.

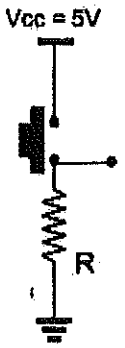


Figura 5

Conecte las diferentes señales y componentes de la manera adecuada en el esquema de la figura 6 para implementar las tres funciones. Aparte de los mismos contadores U0 y U1 del apartado anterior, se dispone adicionalmente de un biestable J-K disparado por flanco de reloj, con entrada de CLEAR activa a nivel alto. En caso de necesitar puertas lógicas, utilice el mínimo número de puertas NAND y OR de dos entradas. (**10 puntos**).

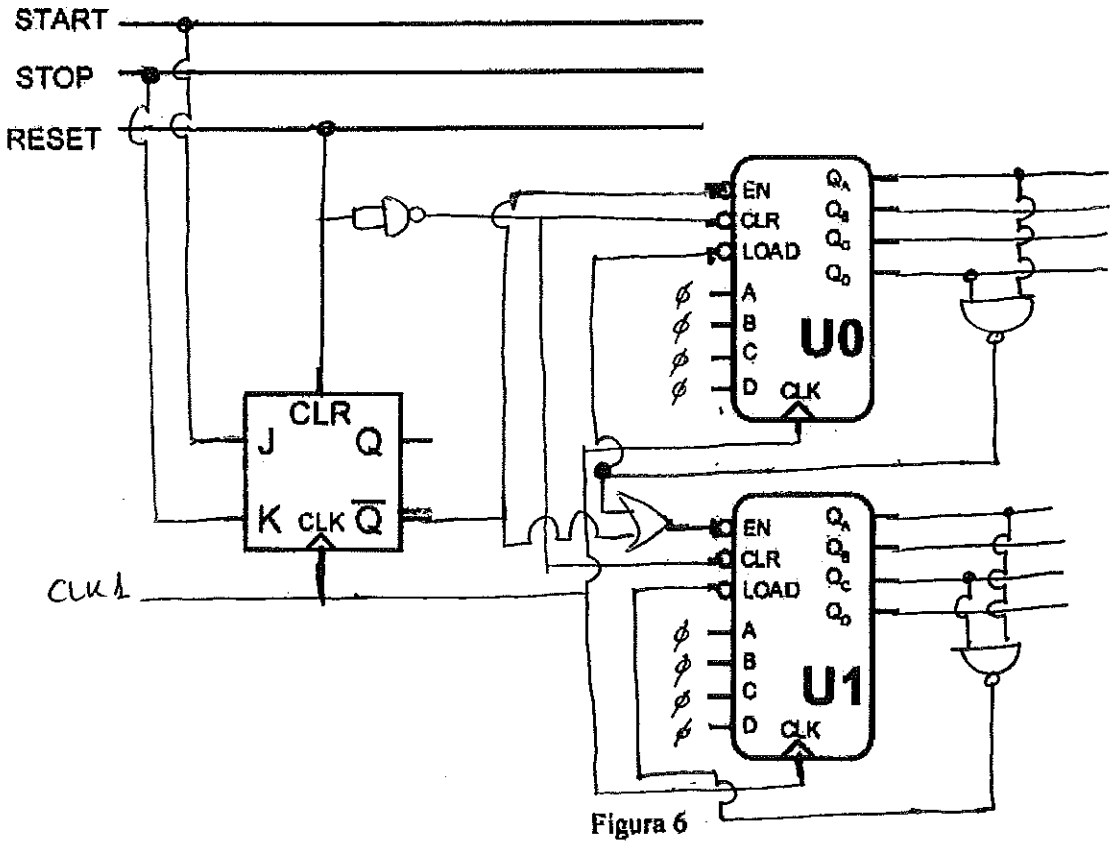


Figura 6

- START: activo a nivel alto, habilita el ENABLE de los contadores (activo a lvl bajo)
- STOP; activo a nivel alto, deshabilita ENABLE de los contadores
- RESET; " " " " " " " y activa CLEAR de los contadores (activo lvl bajo)

Tabla del biestable JK con clear dado

CLR	J	K	Q_{t+1}	$\overline{Q_{t+1}}$	
0	0	0	Q_t	$\overline{Q_t}$	→ mantendrá la opción anterior cuando no usemos ninguna
0	0	1	0	1	→ habilitará la cuenta cuando K=1 (pulsador)
0	1	0	1	0	→ habilitará la cuenta cuando J=1
0	1	1	$\overline{Q_t}$	Q_t	→ vamos a suponer q no se activarán J y K a la vez
1	X	X	0	1	→ desactivará la cuenta cuando CLEAR=1

⇒ Vamos a usar la salida $\overline{Q}$ del biestable como señal de habilitación,

conectando: START → J RESET → CLR ojo! cdo RESET tiene que parar los contadores (contador activo a nivel alto)
STOP → K

Nota la señal ENABLE del contador U1 tiene que coexistir con la habilitación que habíamos montado en el apartado anterior (procedente de U8)
Usamos una OR, que SOLO dará 0 (ENABLE) cuando le lleguen los 2 a la vez 0

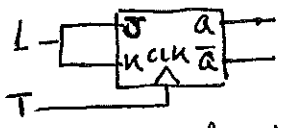
3.3 Finalmente, se trata de añadir la función de LAP a dicho cronómetro. Esta señal proviene de otro pulsador similar a los descritos en el apartado anterior y funciona de la siguiente manera:

- Una primera presión del pulsador debe mantener ("congelar") la cuenta que haya en ese momento en D0 y D1, pero los contadores NO paran de contar.
- Una segunda acción en este pulsador actualiza los valores de D0 y D1 y siguen contando.

Para implementar esta función se dispone de un segundo biestable tipo J-K, disparado por flanco de reloj y con CLEAR activo a nivel alto y de un registro de 8 bits, biestables tipo D, con ENABLE de reloj activo a nivel alto. Realice las conexiones necesarias sobre la figura 7. **(5 puntos)**.

NOTA: Considere que antes de empezar a utilizar el cronómetro se activa siempre primero la señal de RESET.

CONSTRUCCIÓN DE UN CONMUTADOR MEDIANTE J-K: p S-7



A medida que vaya pulsando T, irán conmutando las salidas

Figura 7

Vamos a hacer pasar la señal de los contadores a través de 8 biestables tipo D que congelarán la salida cuando los deshabilite y volverán a dejarla pasar cuando los habilite
Lo único que necesitamos es que el botón LAP vaya conmutando esta habilitación (en la entrada EN del registro de biestables)

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

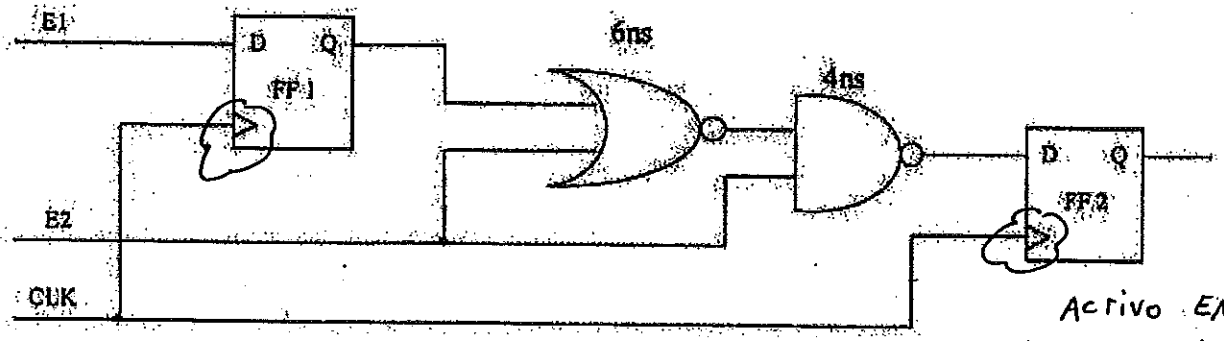
t_{propFF}=10ns

$t_{propNOR}$=6ns

$t_{propNAND}$=4ns

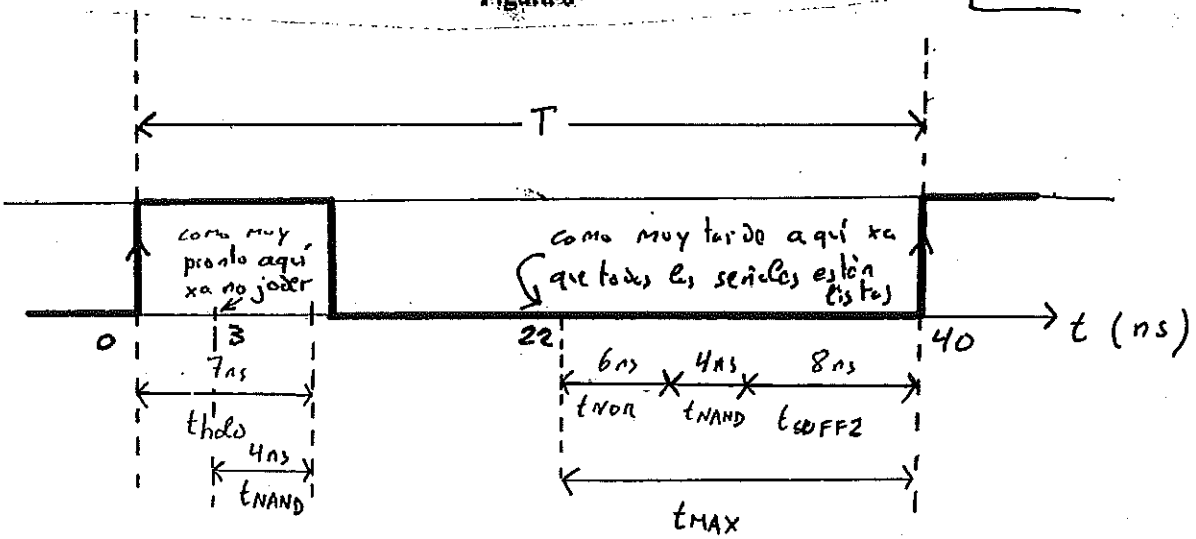
determine los instantes de tiempo máximo (t_{max}) y mínimo (t_{min}) relativos al flanco de subida del reloj entre los cuales la señal E_2 podría variar, permitiendo un correcto funcionamiento del circuito. Justifique su respuesta mediante un cronograma. (10 puntos).

$T = 40\,ns$ (no tiene xq ser el 50%)



ACTIVO EN FLANCO DE SUBIDA

Figura 8



TIEMPO MÁXIMO Mi señal, tiene que estar en FF2 antes del siguiente flanco, respetando además el t_{setup} del biestable. Además, tarda en llegar a éste (porque atraviesa dos puertas) ↓ ojo, por el camino más pesado siempre

$t_{MAX} = 40 - (8+4+6) =$ " tengo que dar tiempo a llegar ← al más lento

$= 22\,ns$

TIEMPO MÍNIMO

Mi señal no puede perturbar la entrada del biestable antes de que se cumpla su tiempo de hold. Aun así, tarda un poco en llegar a "molestar" (porque tiene que atravesar una puerta → por el camino más rápido, por donde antes llegará la señal a "molestar")

$$T_{min} = 7 - 4 = \frac{3\,ns}{2}$$

$$3\,ns < t < 22\,ns$$

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'

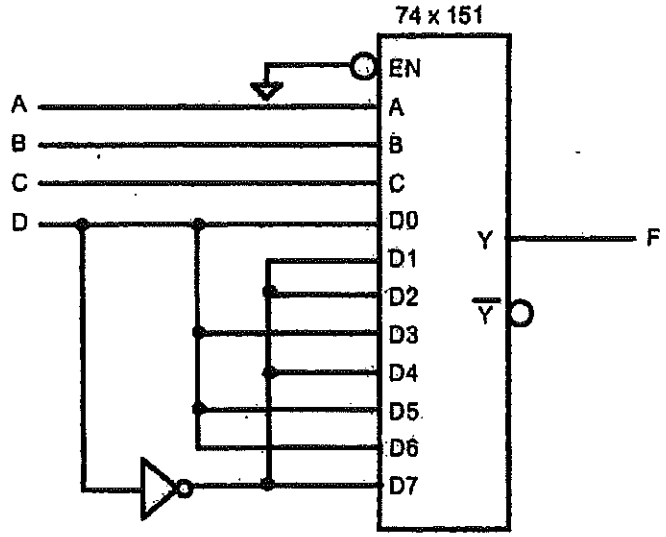


Figura 1

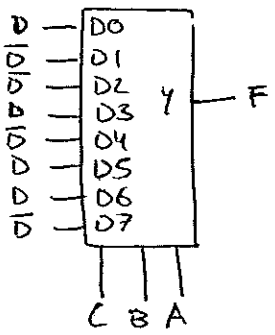
1.1. Función lógica

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)



D	C	B	A	F
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
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	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

$$F = \bar{C}\bar{B}\bar{A}\not{D}_0^{D} + \bar{C}\bar{B}A\not{D}_1^{\bar{D}} + \bar{C}B\bar{A}\not{D}_2^{\bar{D}} + \bar{C}BA\not{D}_3^{D}$$
$$+ C\bar{B}\bar{A}\not{D}_4^{\bar{D}} + C\bar{B}A\not{D}_5^{D} + CB\bar{A}\not{D}_6^{D} + CBA\not{D}_7^{\bar{D}}$$

podemos saltarnos la tabla de verdad rellenando directamente Karnaugh:

DC \ BA	00	01	11	10
00	0	1	0	1
01	1	0	1	0
11	0	1	0	1
10	1	0	1	0

2) Si intentamos simplificar por Karnaugh volvemos a la expresión de F obtenida anteriormente (1ª forma canónica) que no está simplificada

Siempre que quede este "tablero de ajedrez", simplificaremos algebraicamente. En este caso implementaremos la función con puertas XOR

$$F = \bar{C}(\bar{B}\bar{A}D + \bar{B}A\bar{D} + B\bar{A}\bar{D} + BAD) + C(\bar{B}\bar{A}\bar{D} + \bar{B}AD + B\bar{A}D + BA\bar{D}) =$$
$$= \bar{C}[\bar{B}\underbrace{(\bar{A}D + A\bar{D})}_{A\oplus D} + B\underbrace{(\bar{A}\bar{D} + AD)}_{\overline{A\oplus D}}] + C[\bar{B}\underbrace{(\bar{A}\bar{D} + AD)}_{A\oplus D} + B\underbrace{(\bar{A}D + A\bar{D})}_{\overline{A\oplus D}}]$$

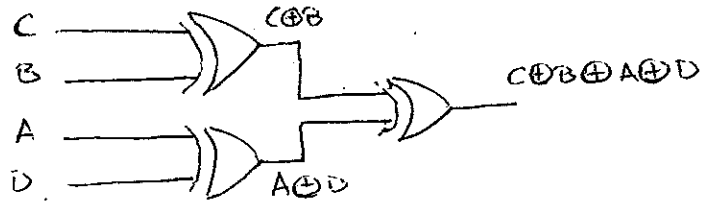
$$F = \overline{C}\left[\overline{B}(A \oplus D) + B\overline{(A \oplus D)}\right] + C\left[\overline{B(A \oplus D)} + B(A \oplus D)\right]$$

$\overline{XY}$ $\quad$ $X\overline{Y}$ $\quad$ $\overline{XY}$ $\quad$ XY

$B \oplus (A \oplus D)$ $\qquad$ $\overline{B \oplus (A \oplus D)}$

$\overline{XY}$ $\qquad$ dd $\qquad$ $X\overline{Y}$

$$\boxed{F = C \oplus [B \oplus (A \oplus D)]} = (C \oplus B) \oplus (A \oplus D)$$ ← distributiva



SEPTIEMBRE 2002 CMOS

4.1. Implementación

Diseñe el circuito que implemente en tecnología CMOS la función lógica $F = \overline{((A+B)\cdot C)}$ utilizando el mínimo número posible de transistores MOS (tanto nMOS como pMOS).

4.2. Temporización

El circuito del apartado anterior ataca un inversor con una capacidad de entrada C_{in} de 10pF. Obtenga razonadamente el tiempo máximo de subida t_r de dicho circuito.

Suponga: R_{on} transistores nMOS = 100
R_{on} transistores pMOS = 200

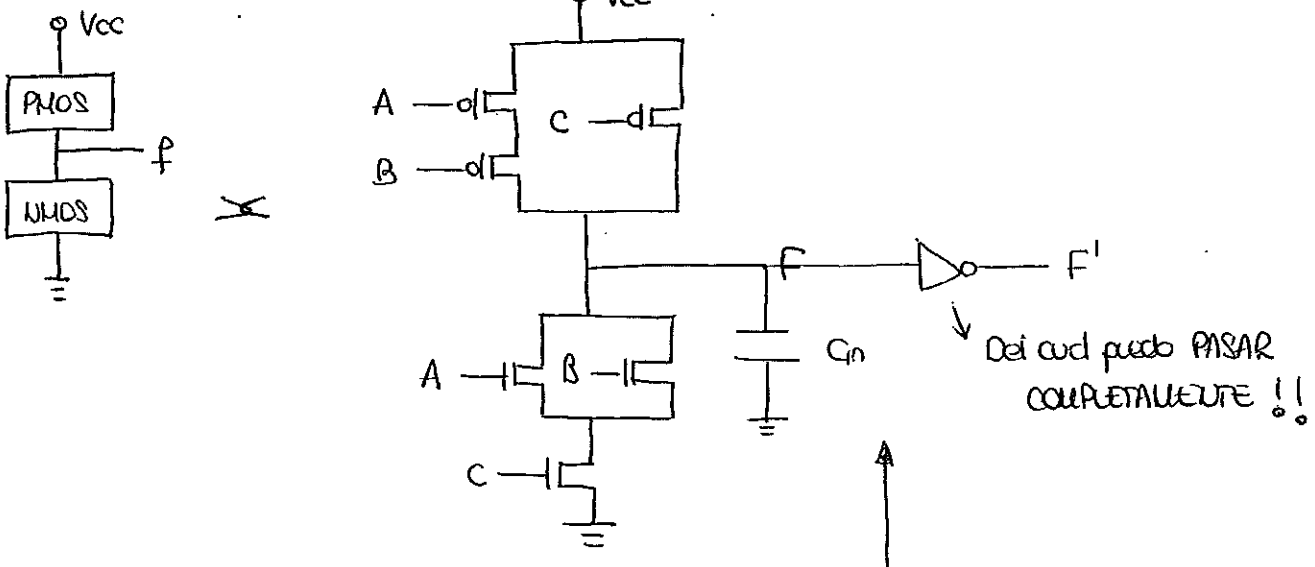
Nota: Se considera el t_r de una señal como el tiempo que tarda en pasar del 10% al 90% de su valor máximo.

4.1. Implementación CMOS de $F = \overline{(A+B)\cdot C} = \overline{(A+B)} + \overline{C} = \overline{A}\cdot\overline{B} + \overline{C}$.

$F = \overline{A}\cdot\overline{B} + \overline{C}$ — $\overline{A}\cdot\overline{B}$: SERIE de 2; $\overline{C}$: 1; PARALELO → BLOQUE PMOS

$F = \overline{A}\cdot\overline{B} + \overline{C}$ — $\overline{A}\cdot\overline{B}$: PARAL 2; $\overline{C}$: 1; SERIE → Bloque NMOS

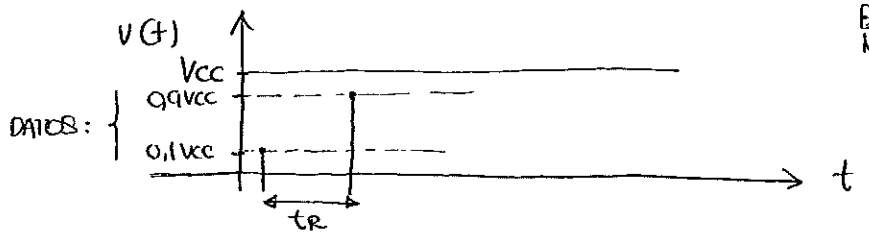
Recordar! NEGAR Variable



4.2. Con el ckt implementado de 4.1. acabamos en INVERSOR con $C_{in} = 10pF$

(pues siempre que me pregunten por TIEMPO, necesito CARGA).

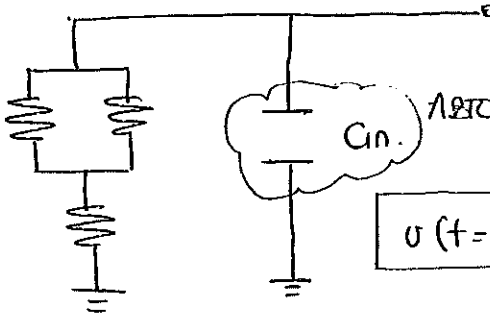
- Nos piden TIEMPO SUBIDA de $v(t)$, t_R: cómo voy de BAJO a ALTO, cuando 1º el ckt



BLOQUE NMOS: [puerta $v(t) = F = 0$] y después sale el [de $F = 1$, $v(t) = 5V$.] BLOQUE PMOS.

☑ El circuito, para $t<0$, está en RÉGIMEN ESTACIONARIO

(PASO DEL PULSO)



(Voy de LOW a HIGH, así que primero 0V).

$$v(t=0^-) = v(t=0^+) = 0V$$

(PRINC. CONTINUIDAD v_c).

DC: el CONDENSADOR...

$i_c = C\frac{dv_c}{dt}$ ($dv_c' = 0$)

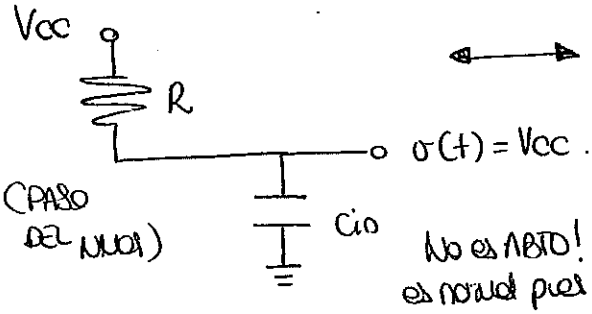
EBAS

$Z_c = \frac{1}{j\omega C}$ ($\omega = 0$)

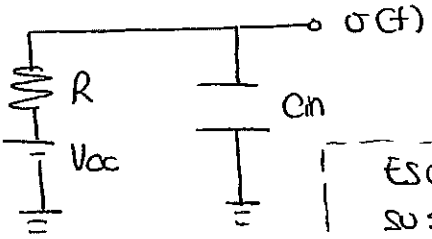
IACR

... no porque como no pone, es CTO ABIERTO

☑ Ahora planteamos la salida de 5V (hemos SUBIDO a HIGH).



No es ABIERTO! es normal pues ACABO DE CONECTAR!!



Esta RED RC: su solución es una EDO...

$$\tau \cdot \frac{d}{dt} v(t) + v(t) = CF$$

$$V(0) = CI. \quad (PVI)$$

Como siempre me tocará una red RC, la ecuación es siempre la misma: $v(t) = (Ci - CF)e^{-t/\tau} + CF$ (Solución de un PVI de red RC).

$\tau = R \cdot Cin$

$Ci = 0$ $\longrightarrow$ $v(t) = (0 - Vcc)e^{-t/RCin} + Vcc = Vcc\,(1 - e^{-t/RCin})$

$CF = Vcc$

* Simplemente, quiero hallar R... VER NOTA, ABAJO (*)

Necesitamos una R_{MAX} por cualquier una $t_{R\,MAX}$, estamos eligiendo el camino a $v(t)$ de mayor resistencia: $R = R_{ON} + R_{ON} = 2R_{ON}$. ($R_{ON}$ pues pues estoy con solido); y calculamos $v(t) = 0{,}1\,Vcc$ y $v(t) = 0{,}9\,Vcc$: ... pienso esto y pto.

☑ $0{,}1\,Vcc = Vcc\,(1 - e^{-t/2R_{ON}\cdot Cin}) \longrightarrow t = -2R_{ON}\,Cin\,(\ln 0{,}9) = 0{,}4\,ns = t_1$

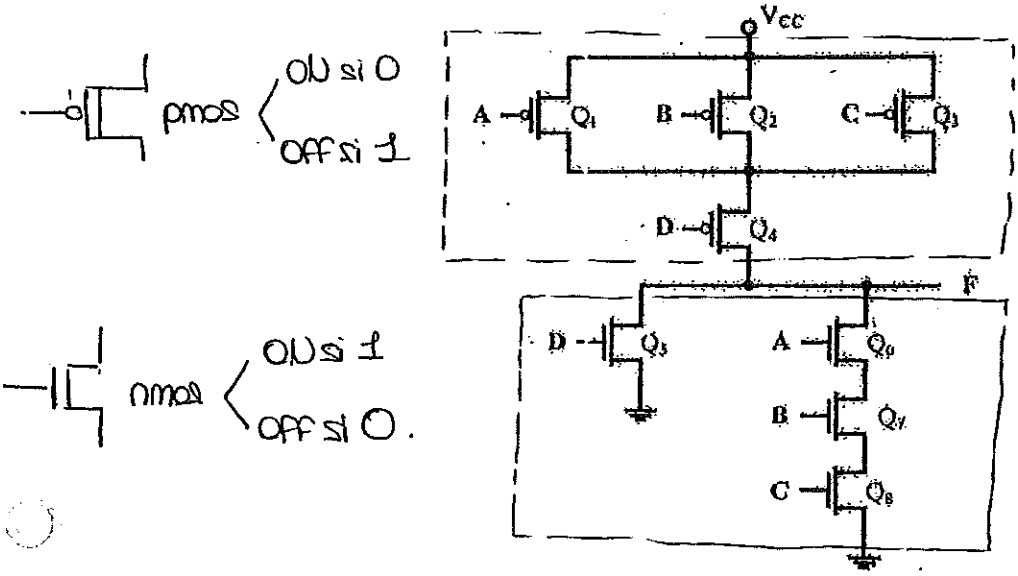
☑ $0{,}9\,Vcc = Vcc\,(1 - e^{-t/2R_{ON}\cdot Cin}) \longrightarrow t = -2R_{ON}\,Cin\,(\ln 0{,}1) = 9{,}2\,ns = t_2$.

Finalmente, $\boxed{t_R = t_2 - t_1 = 8{,}8\,ns}$.

(*) Nota en la nota matemática: $+t_R$ a $+\tau$, a $-$ coef. de la t en D.A.

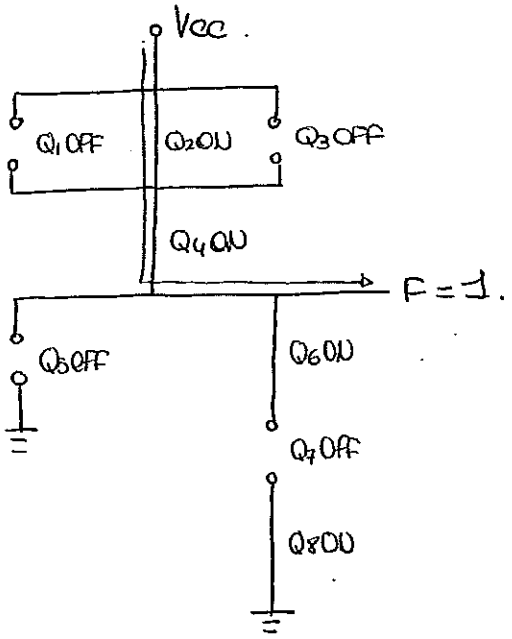
FEBRERO 2003 CMOS.

1.1. Indique el valor lógico de la señal F así como el estado de conducción (ON) o no conducción (OFF) de los transistores del circuito de la figura para la siguiente combinación de valores lógicos: A = 1; B = 0; C = 1; D = 0. Suponga que un valor lógico '0' equivale a una tensión de 0 V, un valor lógico '1' equivale a una tensión de 5 V y que V_{CC} = 5V.



1.2. Determine razonadamente la expresión de la función lógica F que realiza el circuito de la figura.

1.1. A=1, B=0, C=1, D=0.



1.2. Me piden V caso: uso LÓGICA PROPOSICIONAL pensada para F=1, p.ej.

$F=1 \leftrightarrow (Q_4=ON) \wedge (Q_1=ON \vee Q_2=ON \vee Q_3=ON) \leftrightarrow \ldots$

... $\leftrightarrow (D=0) \wedge (A=0 \vee B=0 \vee C=0) \leftrightarrow \overline{D} \cdot (\overline{A}+\overline{B}+\overline{C}) = F$.

Podemos simplificar: $F = \overline{D}(\overline{A}+\overline{B}+\overline{C}) = \overline{D}\,\overline{(A \cdot B \cdot C)} = \overline{D + ABC}$ //

FEBRERO 2003

(*) DATO MUY IMPORTANTE ! :

t_{pHL} : tiempo que tarda en cambio de L a H en entrada, en transmitirse a la salida, que pasa de H a L.

PROBLEMA 2 (10 PUNTOS)

Obtenga justificadamente el valor de la resistencia de conducción de los transistores NMOS, $R_{conNMOS}$, de la puerta NAND de dos entradas de la figura 2, suponiendo que cargada con una capacidad C_L = 50pF presenta un tiempo de propagación t_{pHL} de 8ns al aplicarle la tensión V_I mostrada en la figura 3. Dibuje sobre la figura 3 la forma aproximada de la tensión de salida V_O.

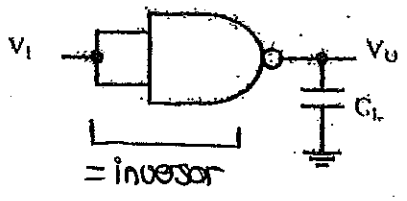


Figura 2

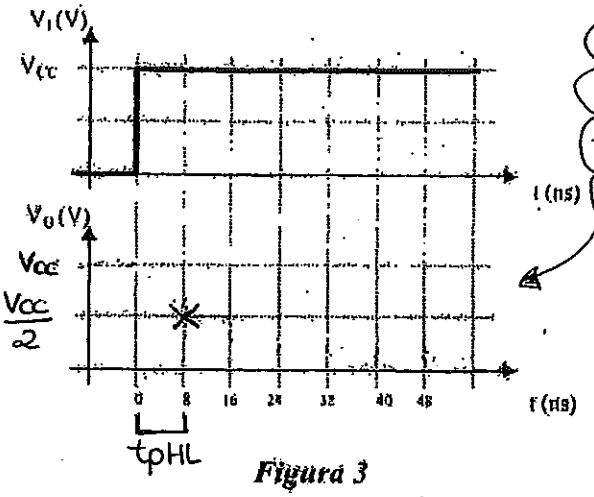


Figura 3

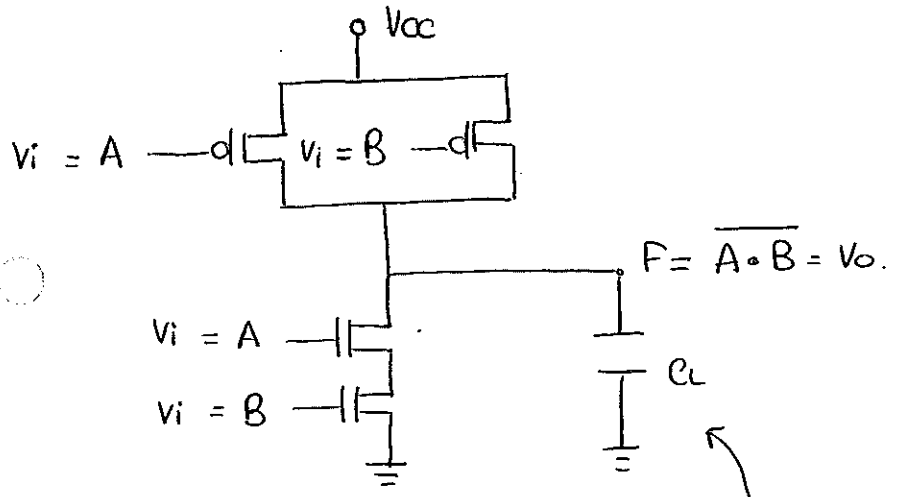
op ! de modo práctico, en 8ns estaré la salida a $\frac{V_{CC}}{2}$

Primero, representamos una NAND con tecnología CMOS : IMPLEMENTACIÓN !! (no mono).

A (NAND) B = $\overline{A \cdot B} = \bar{A} + \bar{B}$

Bloque PMOS : PARALELO DE c. negados ($\bar{\bar{A}}=A$, $\bar{\bar{B}}=B$)

Bloque NMOS : SERIE DE c. negados ($\bar{\bar{A}}=A$, $\bar{\bar{B}}=B$)

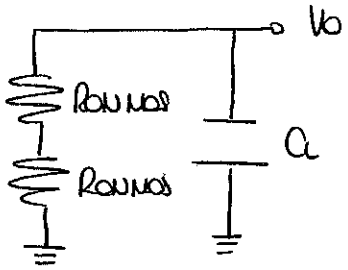


Nuestro circuito es, el de arriba, con un condensador ! y en cambio nuevos $\begin{cases} A=B=V_I \\ F=V_O \end{cases}$

Vamos a estudiar el PASO DE ALTO A BAJO :

- Con V_O = ALTO tecnicual, hasta t=0 → si t<0, $\boxed{V_O = V_{CC}} = C_i$
- Ante el cto pasa BAJO ! será red RC : si t>0, $V_O(t) = (C_i - C_F)e^{-t/\tau} + C_F$.

CTO BAJO :



donde ...

$$\begin{cases} C_i = V_{CC} \\ C_F = 0 \\ \tau = R \cdot C = 2R_{ON/NOR} \cdot C \end{cases}$$

Así, $v_o(t) = V_{CC} \cdot e^{-t/2R_{ON}C}$

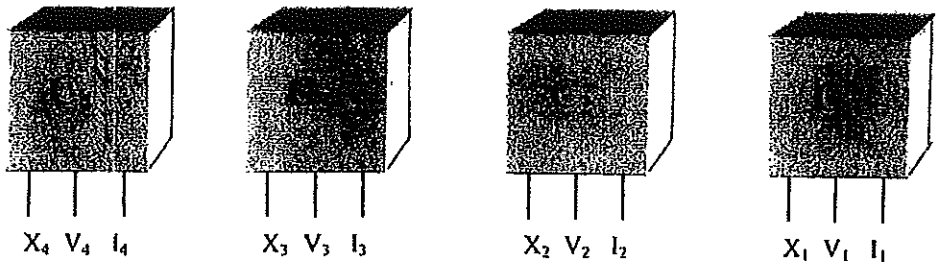
Usamos el dato: $V_o(t = 8ns) = \frac{V_{CC}}{2}$ sustituido en la EDO, despejo R_{ON}:

$$V_{CC} \cdot e^{-8ns/2R_{ON} \cdot C} = \frac{V_{CC}}{2} \; ; \; \frac{-8ns}{2R_{ON} \cdot C} = \ln(0'5) \; ; \; \underline{\text{dato}}: C = 50pF \ldots$$

$$\boxed{R_{ON} = 115'4\,\Omega}$$

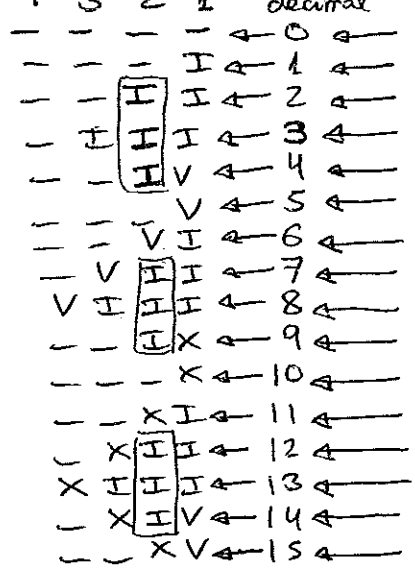
FEBRERO 2003

3. 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 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 ="I", se encenderá el neón correspondiente a la cifra romana "I" del elemento C_2, y así sucesivamente. La cifra romana completa se alienará a la derecha, dejando en blanco los elementos no usados en la parte izquierda del visualizador.



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).

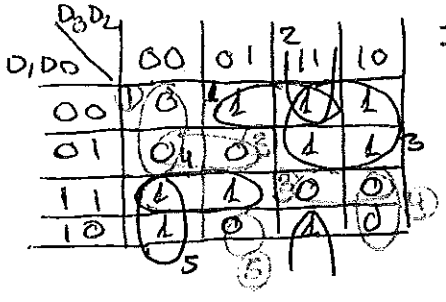
1. Obtenga la tabla de verdad correspondiente a I_2, rellenando la tabla (añadido por parte del profe)



→ pantalla vacía

D_3	D_2	D_1	D_0	I_2	F	G
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	1	0	1	0	0
0	0	1	1	1	1	0
0	1	0	0	1	0	1
0	1	0	1	0	0	1
0	1	1	0	0	0	0
0	1	1	1	1	0	0
1	0	0	0	1	1	0
1	0	0	1	1	0	0
1	0	1	0	0	0	0
1	0	1	1	0	0	0
1	1	0	0	1	0	0
1	1	0	1	1	1	0
1	1	1	0	1	0	1
1	1	1	1	0	0	1

2. Obtenga la expresión simplificada de I_2 usando el método de *Karnaugh*. Realice la simplificación por "unos" gráficamente y por "ceros". Escriba a continuación las expresiones correspondientes.



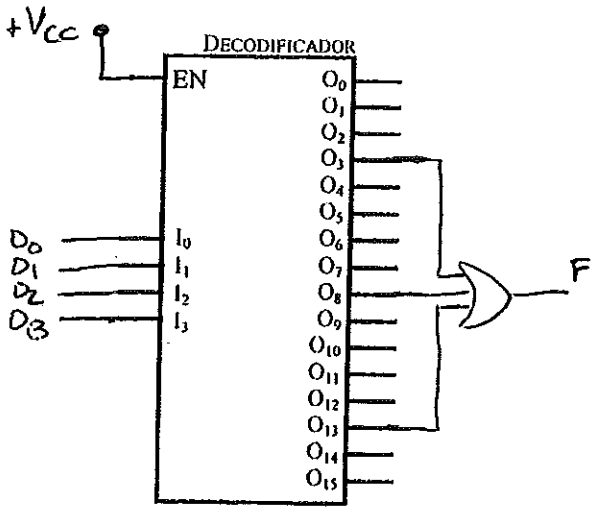
$I_2 = D_2\bar{D}_1\bar{D}_0 + D_3D_2\bar{D}_0 + D_3\bar{D}_1 + \bar{D}_3D_1D_0 + \bar{D}_3\bar{D}_2D_1$ (1,2,3,4,5)

por unos ↑

$I_2 = (D_3+D_2+D_1)(D_3+D_1+\bar{D}_0)(\bar{D}_3+\bar{D}_1+\bar{D}_0)(\bar{D}_3+D_2+\bar{D}_1)$
$(D_3+\bar{D}_2+\bar{D}_1+D_0)$ (①,②,③,④,⑤)

por ceros ↑

3. Utilizando el decodificador de cuatro entradas de la figura (cuya tabla de verdad se encuentra junto a la misma), 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'**.



I_3 bit mas significativo

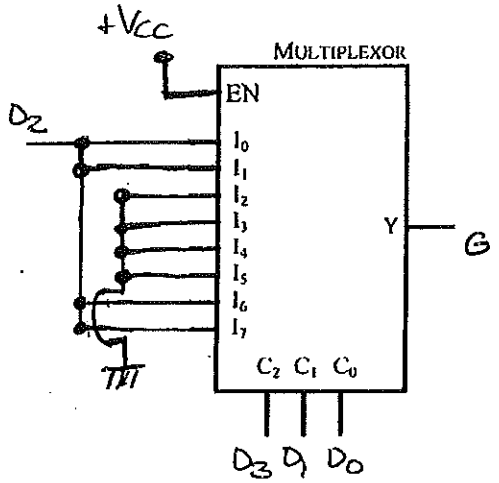
EN	I_3	I_2	I_1	I_0	O_0	O_1	O_2	O_3	O_4	O_5	O_6	O_7	O_8	O_9	O_{10}	O_{11}	O_{12}	O_{13}	O_{14}	O_{15}
0	X	X	X	X	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
1	0	1	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
1	0	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
1	0	1	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

$F = (\overline{D_3}\,\overline{D_2}\,D_1 D_0) + (D_3\overline{D_2}\,\overline{D_1}\,\overline{D_0}) + (D_3 D_2 \overline{D_1} D_0)$

0011 1000 1101

Para implementar una función lógica con un decodificador, solo tenemos que conectar mediante una puerta OR todas las salidas que se corresponden con filas de la tabla en las que F vale 1

4. Utilizando el multiplexor de 8 a 1 en la figura (cuya tabla de verdad se encuentra junto a la misma) 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 la señal **S** negada como **S'**.



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

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.

4) $G = \bar{D_3} D_2 \bar{D_1} \bar{D_0} + \bar{D_3} D_2 \bar{D_1} D_0 + D_3 D_2 D_1 \bar{D_0} + D_3 D_2 D_1 D_0$

$Y = \bar{C_2}\bar{C_1}\bar{C_0} I_0 + \bar{C_2}\bar{C_1} C_0 I_1 + \bar{C_2} C_1 \bar{C_0} I_2 + \bar{C_2} C_1 C_0 I_3$

$+ C_2 \bar{C_1} \bar{C_0} I_4 + C_2 \bar{C_1} C_0 I_5 + C_2 C_1 \bar{C_0} I_6 + C_2 C_1 C_0 I_7$

¿cuál de las variables es la que está no negada? → D_2

$\Rightarrow I_0 = I_1 = I_6 = I_7 = D_2$; resto $= \phi$

$C_2 = D_3$; $C_1 = D_1$; $C_0 = D_0$

plantamos 1s y ϕs en G de la tabla → 0100; 0101; 1110; 1111

5) para F: en la fila 3: ∪ I I I
" " " 8: V I I I
" " " 13: X I I I

A la vista de la Tabla de Verdad, tenemos que buscar el elemento en exclusividad iluminado, en las filas de la tabla en las que la función vale 1

para G: en la fila 4: ∪ ∪ I V
" " " 5: ∪ ∪ ∪ V
" " " 14: ∪ X I V
" " " 15: ∪ ∪ X V

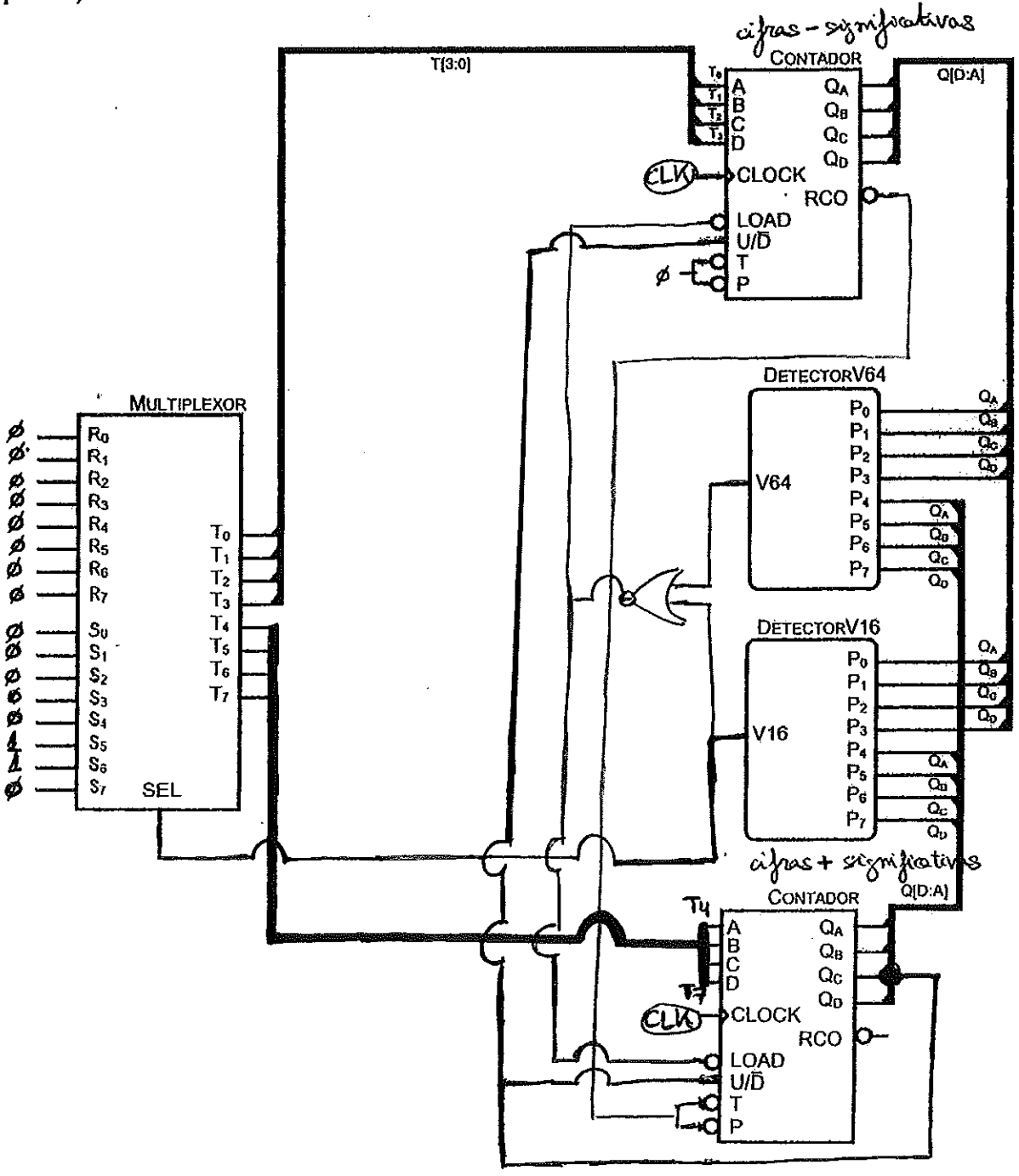
- ☒ F es la función que gobierna I_3 de
- ☒ G " " " " " V_1 de

PROB 4 FEB 2003 (faltan apartados 1 y 2)

3. Ahora se desea diseñar un contador que, a partir de una señal de reloj CLK, permita obtener la secuencia de cuenta:

   0, 1, 2, 3, ..., 14, 15, 16, 96, 95, ..., 65, 64, 0, 1, 2, 3, ..., 14, 15, 16, 96, 95, ..., 65, 64, 0, ...

   Disponiendo de los detectores V16 y V64 del apartado anterior, de un multiplexor de dos entradas de 8 bits (cuya tabla de verdad se encuentra al final de este examen) y de las puertas lógicas que considere necesarias, realice **todas** las conexiones pertinentes sobre la **figura 9** para que el circuito funcione como se desea. **Por favor, sea limpio en el esquema.** (15 puntos)



Nos piden un contador de 8 bits "raro":

- cuando llega (subiendo) al 16 debe: → cargar 96
  → ponerse a bajar
- cuando llega (bajando) al 64 debe: → cargar ∅
  → ponerse a subir

Vamos a abordar por separado los diferentes problemas:

1 Tenemos diferentes cantidades a cargar!

El MUX se encargará de elegir que número (∅ ó 96) carga a las entradas de datos de los contadores

Qué entrada seleccionar es el trabajo de la señal SEL, que depende de los detectores V64 y V16:

V16	V64	SEL	
0	0	X	→ no se realiza carga
0	1	∅	} según hemos puesto los nos en el MUX
1	0	1	
1	1	X	→ no posible

DETALLE || En esta tabla, para implementar la función SEL, las casillas que tienen X pueden valer lo que sea, pueden valer ∅ ó 1 como queramos. Así podemos decir que SEL = V16

2 ¿Cuando cargamos?

La señal LOAD tmb depende de los detectores V16 y V64:

V16	V64	LOAD	
0	0	1	
0	1	0	
1	0	0	
1	1	X	← caso no posible

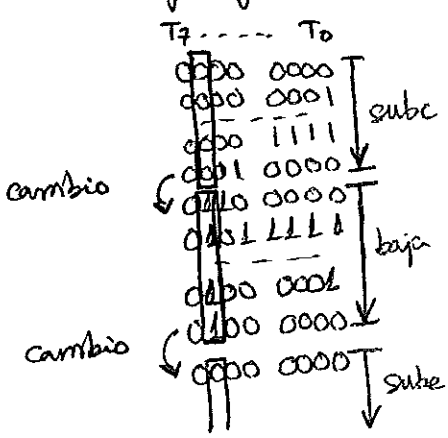
Ojo LOAD activo a nivel bajo ⚠

Como la X puede ser ∅ ó 1 ⇒

⇒ (V16) NOR (V64) = LOAD

3 La subida/bajada (U/D̄)

Para implementar U/D̄ no podemos implementarlo por biestables si no por puertas lógicas.

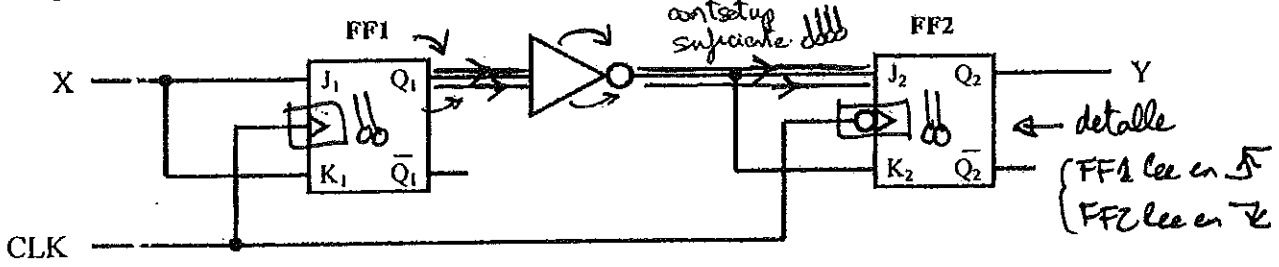


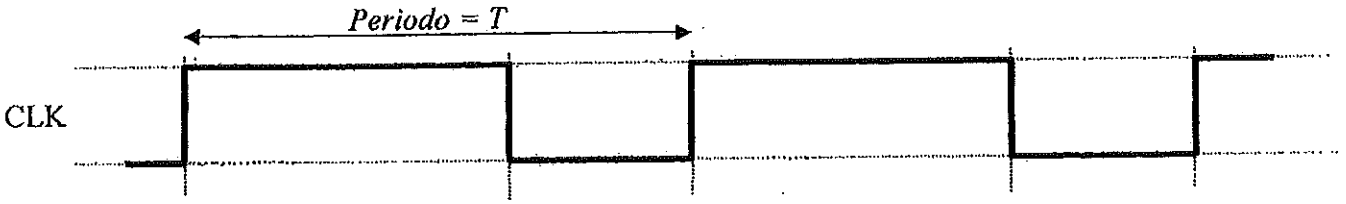
nos fijamos en la columna T6 (correspondiente al 2c del contador de las cifras + significativas). Vemos que cuando sube es un ∅ y cuando baja es un 1

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%.

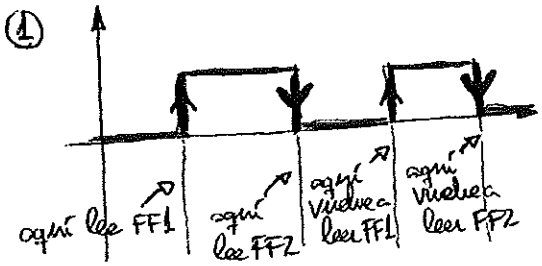




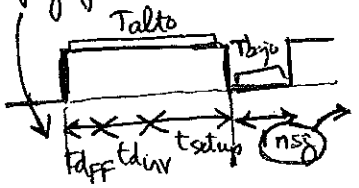
2. Calcule razonadamente el tiempo de hold máximo tholdmax que podrá tener el biestable FF2, para que el circuito funcione correctamente. Asuma las mismas restricciones que las descritas en la nota del apartado anterior y justifique su respuesta a partir del cronograma que ha realizado en el apartado anterior.

FIJARSE { Flip flop 1: activo flanco subida; Flip flop 2: " " bajada } { t_{dinv} = 1ns; t_{dFF} = 2ns; t_{setup} = 3ns }

①



desde que lee el flip flop 1, hasta que lo haga el flip flop 2, tenemos que estar con el tsetup



La restricción no es del dato (pq FF2 no sale ninguna señal que alimente a otro FF), pero la impone el enunciado

⇒ $T_{mun} = t_{dFF} + t_{dinv} + t_{setup} + 1nseg = 7nseg$

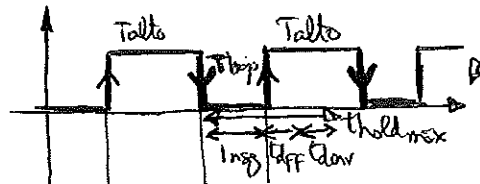
$f_{max} = 1/T_{mun} = 143MHz$

** Ejercicio extra: ¿Y si además deseasen tener un duty cicle = 50%?

Tenemos 2 opciones: reducir Talto hasta 1nseg = Tbajo pero IMPOSIBLE

aumentar Tbajo = Talto ✓ ⇒ $T_{mun} = 12nseg$ ⇒ $f_{max} = 83,3MHz$

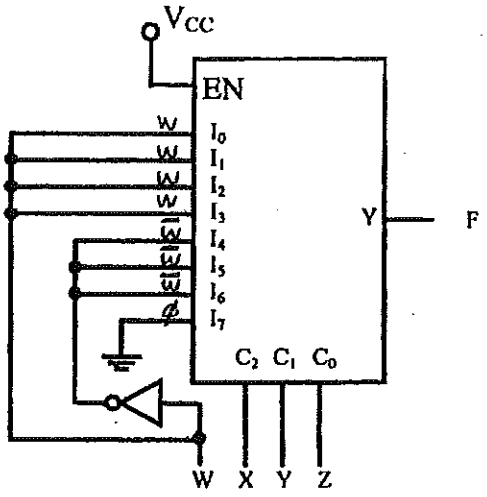
②



Desde que FF2 empieza a leer, transcurre 1nseg "tranquilo" en el que FF1 aún no ha empezado a leer !!

JUNIO 2003

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.

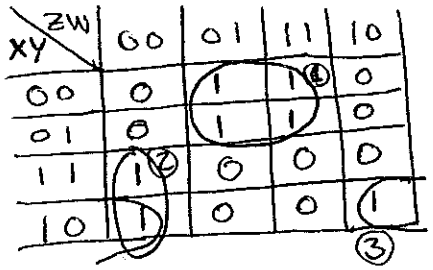
3.2. Obtenga la expresión simplificada de F usando el método de Karnaugh. Realice la simplificación por "unos" gráficamente sobre la tabla 2 y por "ceros" sobre la tabla 3. Escriba a continuación las expresiones correspondientes.

1) $Y = \bar{C}_2\bar{C}_1\bar{C}_0 I_0 + \bar{C}_2\bar{C}_1 C_0 I_1 + \bar{C}_2 C_1 \bar{C}_0 I_2 + \bar{C}_2 C_1 C_0 I_3 + C_2\bar{C}_1\bar{C}_0 I_4 + C_2\bar{C}_1 C_0 I_5 + C_2 C_1 \bar{C}_0 I_6 + C_2 C_1 C_0 I_7$

$F = \bar{X}\bar{Y}\bar{Z}W + \bar{X}\bar{Y}ZW + \bar{X}Y\bar{Z}W + \bar{X}YZW + X\bar{Y}\bar{Z}\bar{W} + X\bar{Y}Z\bar{W} + XY\bar{Z}\bar{W} + XYZ\phi$

2)

X	Y	Z	W	F
0	0	0	0	ϕ
0	0	0	1	1
0	0	1	0	ϕ
0	0	1	1	1
0	1	0	0	ϕ
0	1	0	1	1
0	1	1	0	ϕ
0	1	1	1	1
1	0	0	0	1
1	0	0	1	ϕ
1	0	1	0	1
1	0	1	1	ϕ
1	1	0	0	1
1	1	0	1	ϕ
1	1	1	0	ϕ
1	1	1	1	ϕ

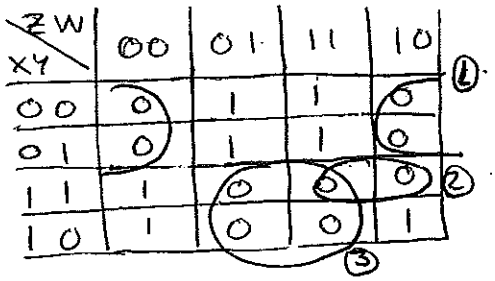


① XYZW: 0001, 0011, 0101, 0111

② XYZW: 1100, 1000

③ XYZW: 1000, 1010

$\Rightarrow F = \bar{X}W + X\bar{Z}\bar{W} + X\bar{Y}\bar{W}$



① XYZW: 0000, 0010, 0100, 0110

② XYZW: 1111, 1110

③ XYZW: 1101, 1111, 1001, 1011

$\Rightarrow F = (X+W)(\bar{X}+\bar{Y}+\bar{Z})(\bar{X}+\bar{W})$

ojo puede haber otras soluciones válidas para el ϕ: $F = (X+W)(\bar{X}+\bar{W})(\bar{X}+\bar{Z}+W)$

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:

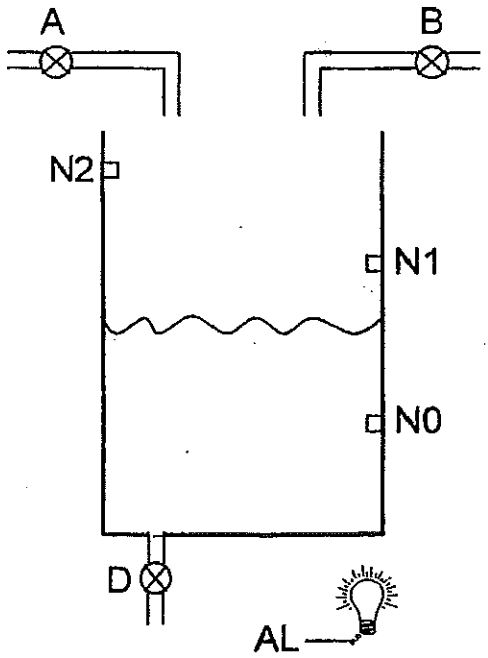


Figura 3

- **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:

(c) • Si el agua desciende por debajo de **N0**, se deben ~~de~~ activar las dos electroválvulas simultáneamente **A** y **B**, permitiendo el llenado del tanque a través de las dos entradas.
(b) • Si el agua alcanza **N0**, solo se activará **A**, desactivándose **B**.
(c) • Si el agua alcanza **N0** y **N1**, se desactivarán las dos señales **A** y **B**.
(d) • Si el agua alcanza **N0, N1** y **N2**, se desactivarán **A** y **B** y se activará **D**.
(e) • 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

	N2	N1	N0	D	AL
(c) →	0	0	0	0	0
(b) →	0	0	1	0	0
(c) →	0	1	0	0	1
(c) →	0	1	1	0	0
(e) →	1	0	0	0	1
(e) →	1	0	1	0	1
(e) →	1	1	0	0	1
(d) →	1	1	1	1	0

Implementar las funciones D, AL, que son booleanas,
D(N2, N1, N0) ; AL(N2, N1, N0)

D

N2N1 / N0	00	01	11	10
0	0	0	0	0
1	0	0	1	0

$D = N2 \cdot N1 \cdot \overline{N0}$

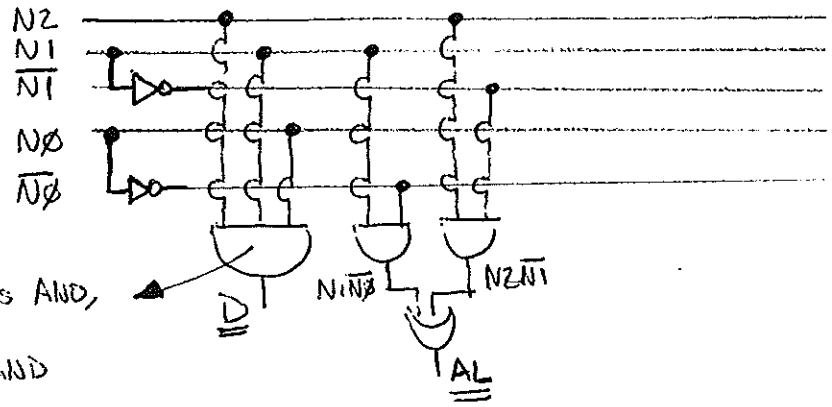
AL

N2N1 / N0	00	01	11	10
0	0	1 ①	1 ①	1 ②
1	0	0	0	1

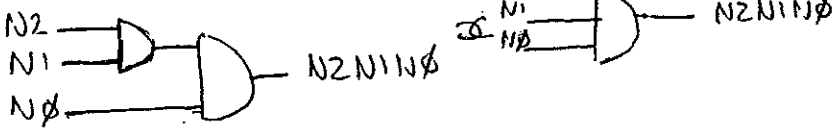
① : N2 N1 N0 → 0 1 0 ; 1 1 0
② : N2 N1 N0 → 1 0 0 ; 1 0 1

$\Rightarrow AL = N1\,\overline{N0} + N2\,\overline{N1}$

$$\Rightarrow \begin{cases} D = N2 \cdot N1 \cdot N\emptyset \\ AL = N1\overline{N\emptyset} + N2\overline{N1} \end{cases}$$



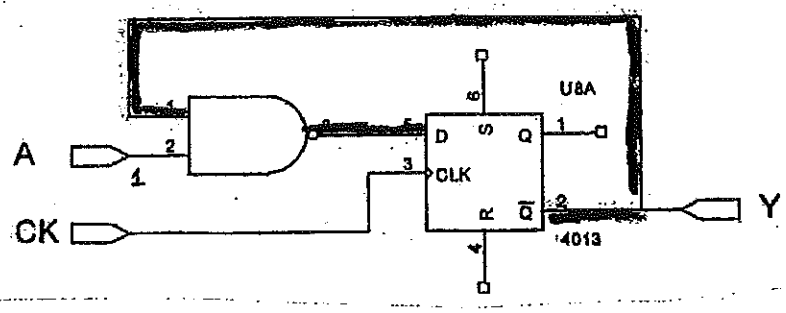
Nota: la puerta de 3 entradas AND, la podemos implementar con AND de dos entradas



SEPTIEMBRE 2003

PROBLEMA 4

Sobre el circuito de la figura y suponiendo inactivas las señales **R** y **S**:

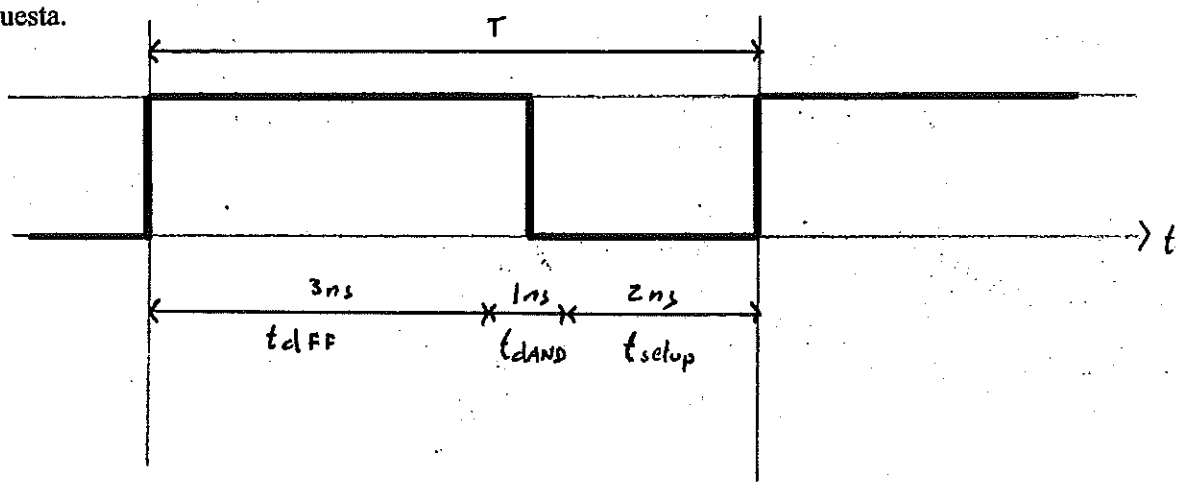


PARÁMETROS

$t_{dAND} = 1$ ns
$t_{dFF} = 3$ ns
$t_{setup} = 2$ ns

4.1 Obtenga justificadamente la máxima frecuencia de funcionamiento del mismo.
4.2 Obtenga justificadamente cuál será el t_{hold} máximo que podrá tener el biestable para que el circuito funcione correctamente.
4.3 Justifique razonadamente cuál es la misión de la entrada A.
4.4 Describa cualitativamente el funcionamiento del circuito y dibuje un cronograma del mismo para justificar su respuesta.

4.1

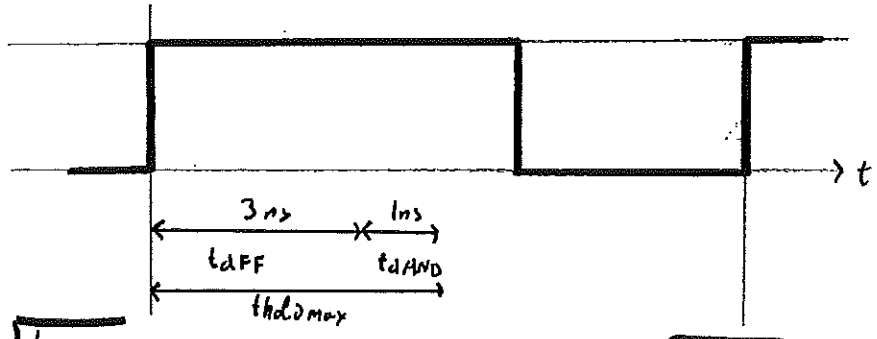


$T_{min} = t_{dFF} + t_{dAND} + t_{setup} = 3 + 1 + 2 = 6$ ns

$f_{max} = \frac{1}{T_{min}} = \frac{1}{6\,ns} = 166'67$ MHz

Sólo hay un camino posible!!

4.2



$t_{hold\,max} = t_{dFF} + t_{dAND} = 3 + 1 = 4$ ns

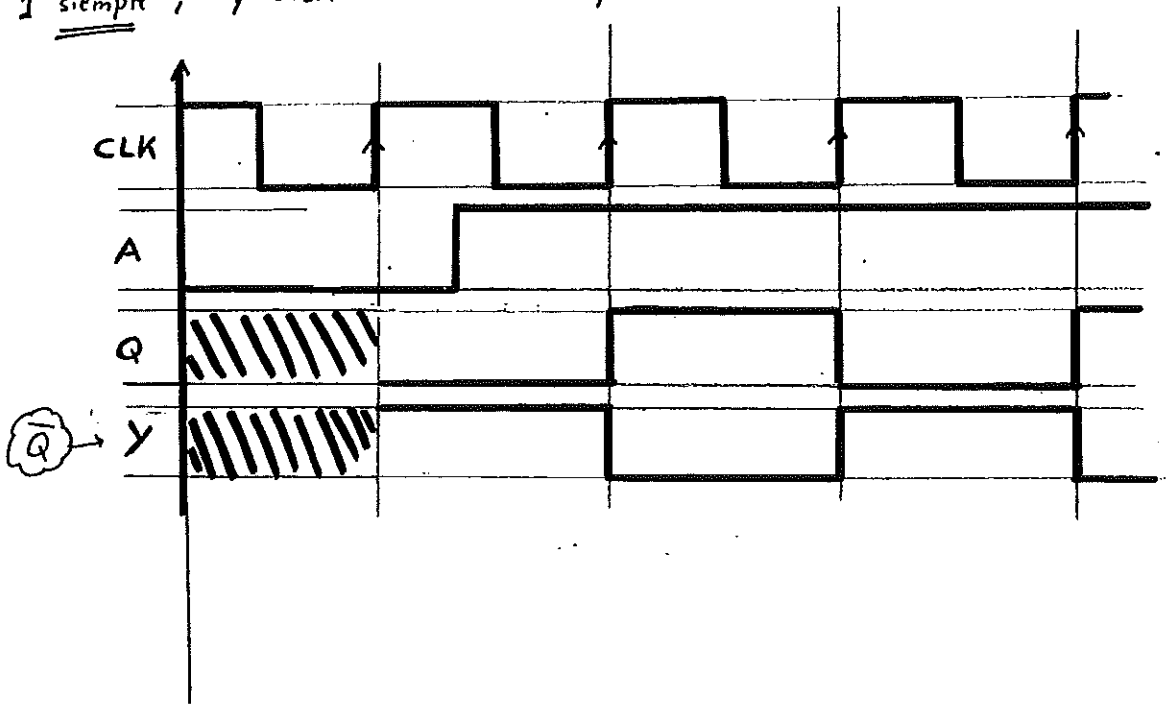
4.3 OJO!! La puerta está mal en el enunciado, en realidad es una AND

$D = \bar{Q} \cdot A$ ⟶ $\boxed{D = Q_{t+1}}$ Ecuación característica de un flip-flop D

$\boxed{Q_{t+1} = \bar{Q}_t \cdot A}$ $\boxed{Y_{t+1} = \bar{Q}_{t+1}}$ OJO, QUE LA SALIDA QUE USAMOS ES LA NEGADA.

- $A = 0 \longrightarrow Q_{t+1} = \bar{Q}_t \cdot 0 \Rightarrow Q_{t+1} = 0 \Rightarrow Y_{t+1} = 1$
- $A = 1 \longrightarrow Q_{t+1} = \bar{Q}_t \cdot 1 \Rightarrow Q_{t+1} = \bar{Q}_t \Rightarrow Y_{t+1} = Q_t = \bar{Y}_t$

La señal A actúa como ENABLE, ya que cuando vale '0' la salida Y vale 1 siempre, y cuando está a '1', la señal Y cambia de '1' a '0' y viceversa.



Con esto también queda contestado el apartado **4.4**

FEBRERO 2004

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*)

$R_1 = \bar{B}_1\bar{B}_0A_1 + B_1B_0A_1\bar{A}_0$

A1	A0	B1	B0	C1	C0	R1	R0	E	AUX
0	0	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0
0	0	1	1	0	0	0	0	0	0
0	1	0	0	0	1	0	1	1	1
0	1	0	1	0	1	0	0	0	0
0	1	1	0	0	0	0	1	0	1
0	1	1	1	0	0	0	1	0	1
1	0	0	0	1	0	1	0	1	0
1	0	0	1	1	0	0	0	0	0
1	0	1	0	0	1	0	0	0	0
1	0	1	1	0	0	1	0	0	0
1	1	0	0	1	1	1	1	1	1
1	1	0	1	1	1	0	0	0	0
1	1	1	0	0	1	0	1	0	1
1	1	1	1	0	1	0	0	0	0

$R_0 = \bar{B}_1\bar{B}_0A_0 + B_1\bar{B}_0A_0 + \bar{A}_1A_0B_1 = (\bar{B}_1 + B_1)\bar{B}_0A_0 + \bar{A}_1A_0B_1$

$E = \bar{B}_1\bar{B}_0$

$R_1 = A_1\bar{B}_1\bar{B}_0 + A_1\bar{A}_0B_1B_0$

$R_0 = \bar{A}_1A_0\bar{B}_1 + A_0\bar{B}_0$

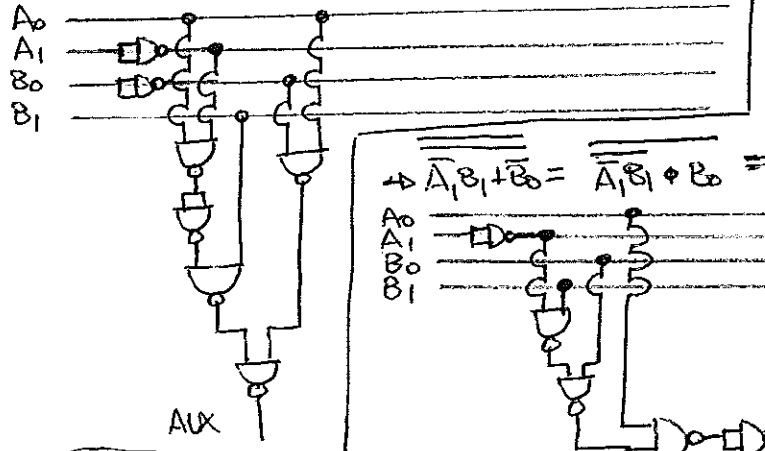
1.2 Implemente la función **AUX** de la *Tabla 1* con el mínimo número de puertas **NAND** de dos entradas. Nota: Sólo se dispone de las entradas A1, A0, B1 y B0 y no de sus complementarios. (*5 puntos*)

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*)

1.2: AUX es la misma función que R_0. Para implementar esta función con puertas NAND$_2$, partimos de la simplificación por UNOS de R_0 y negamos dos veces

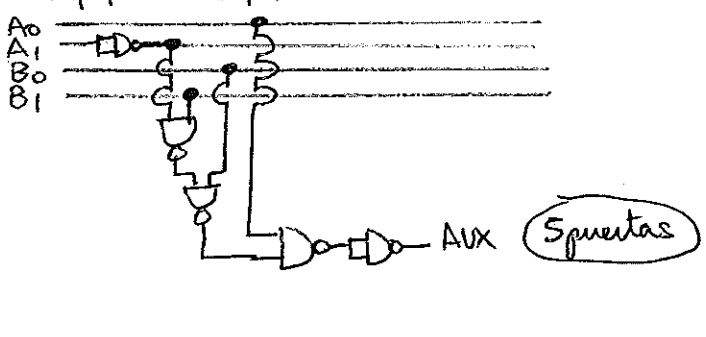
$R_0 = \overline{\bar{A}_1A_0B_1 + A_0\bar{B}_0} = \overline{\bar{A}_1A_0B_1} \cdot \overline{A_0\bar{B}_0}$



Hay una optimización mejor aunque con lo anterior nos darían 7-8/10 ptos.

$AUX = \bar{A}_1A_0B_1 + A_0\bar{B}_0 = A_0(\bar{A}_1B_1 + \bar{B}_0) \rightarrow$

$\rightarrow \bar{A}_1B_1 + \bar{B}_0 = \overline{\bar{A}_1B_1} \cdot B_0 \Rightarrow AUX = A_0\left(\overline{\bar{A}_1B_1 \cdot B_0}\right)$

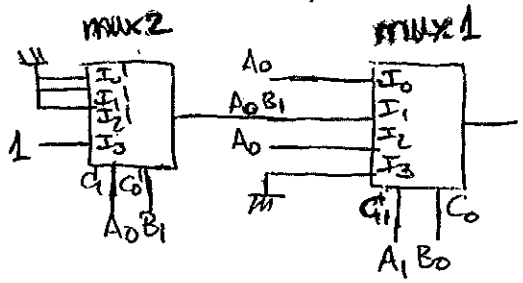


1.3: $AUX = \bar{A}_1 A_0 B_1 + A_0 \bar{B}_0$ con 2 MUX de 4 entradas (4x2)

- **mi método**: A_1 y B_0 variables de control

$$\Rightarrow AUX = \bar{A}_1 A_0 B_1 B_0 + \bar{A}_1 A_0 B_1 \bar{B}_0 + A_0 \bar{B}_0 A_1 + A_0 \bar{B}_0 \bar{A}_1$$

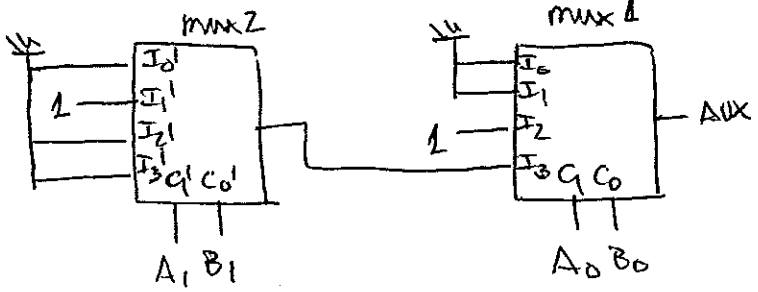
$$= \bar{A}_1 \bar{B}_0 A_0 (B_1 + 1) + \bar{A}_1 B_0 A_0 B_1 + A_1 \bar{B}_0 A_0$$



- **primer método (no mecánico)**

nos fijamos en la tabla, A_ϕ y B_ϕ

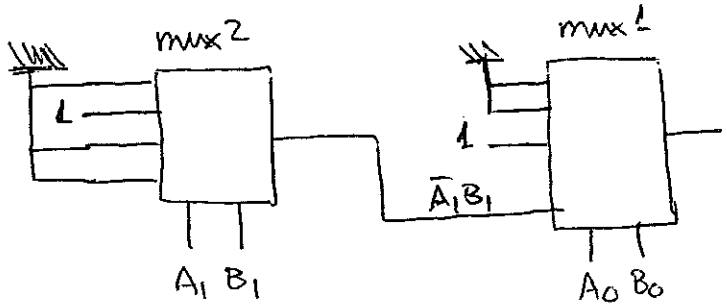
- cuando $A_0 = \phi$ y $B_0 = \phi \Rightarrow AUX = \phi$ (valgan lo q valgan A_1 y B_1)
- cuando $A_0 = \phi$ y $B_0 = 1 \Rightarrow AUX = \phi$ (" " " " " ")
- cuando $A_0 = 1$ y $B_0 = \phi \Rightarrow AUX = 1$ (" " " " " ")
- cuando $A_0 = 1$ y $B_0 = 1 \Rightarrow$ el resultado depende de otras variables



- **segundo método ("mecánico")**

elegimos A_0 y B_0 como variables de control

$$AUX = \bar{A}_1 A_0 B_1 (B_0 + \bar{B}_0) + A_0 \bar{B}_0 = A_0 B_0 \bar{A}_1 B_1 + A_0 \bar{B}_0 \bar{A}_1 B_1 + A_0 \bar{B}_0 = A_0 B_0 \bar{A}_1 B_1 + A_0 \bar{B}_0$$



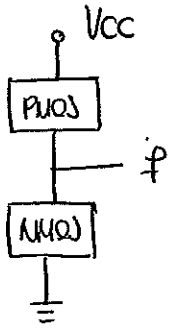
FEBRERO 2004 P1.

IMPLEMENTACIÓN DE FUNCIÓN LÓGICA POR MEDIO DE CMOS.

(1.4.) $AUX = A\emptyset\,\overline{B\emptyset} + \overline{A1}\,B1\,A\emptyset$

- $A\emptyset\,\overline{B\emptyset}$: SERIE de 2 mos
- $\overline{A1}\,B1\,A\emptyset$: SERIE de 3 mos.
- EN PARALELO!

} perc el BLOQUE de PMOS:
- • – SERIE
- + – PARALELO
- variables NEG.



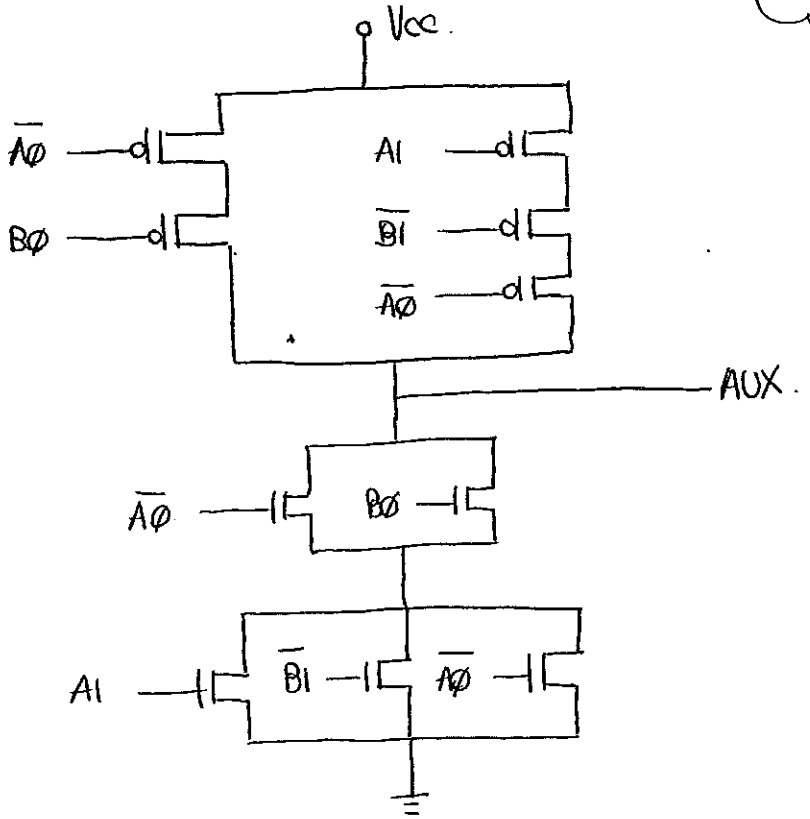
$AUX = A\emptyset\,\overline{B\emptyset} + \overline{A1}\,B1\,A\emptyset$

- $A\emptyset\,\overline{B\emptyset}$: PARAL de 2mos
- $\overline{A1}\,B1\,A\emptyset$: PARAL de 3mos
- EN SERIE!

} perc. el bloque de NMOS:
- • – PARALELO
- + – SERIE
- variables NEG!

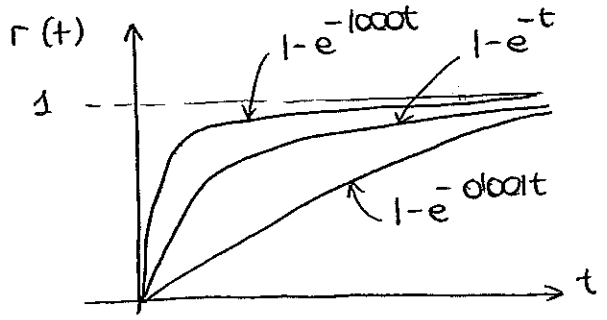
Así, el circuito CMOS nos queda ...

Recordar! NEGAR LAS VARIABLES



* NOTA: puede estar simplificado así $AUX = A\emptyset\,(\overline{B\emptyset} + \overline{A1}\,B1)$

✱ NOTA MATEMÀTICA al problema SEPT 2002 (P4).



Vemos caso a que hace menor o.A. del coef. de la t (= mayor t) es la más plana. → será la de MÁS tr.

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)*

Representamos la tabla de verdad de un biestable D recordar ej clase tema 5, ej. 1

E	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

modo retención

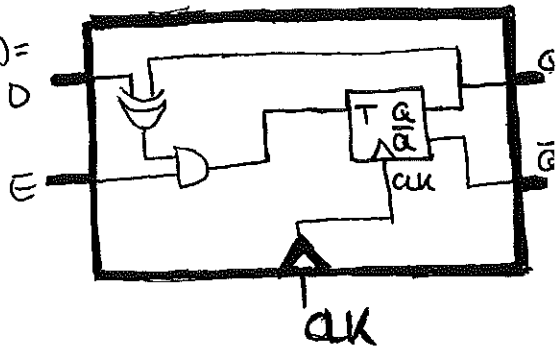
modo transparente

⇒ Implementamos la función simplificando por Karnaugh

Q \ ED	00	01	11	10
0	0	0	1	0
1	0	0	0	1

$T = (E\bar{D}\bar{Q}) + (ED\bar{Q}) =$

$= E \cdot (D \oplus Q)$



Nos tenemos que preguntar si tenemos que provocar una transición en la que cambia la salida o no: { cambia la salida → T=1 ; no " " → T=0

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)*

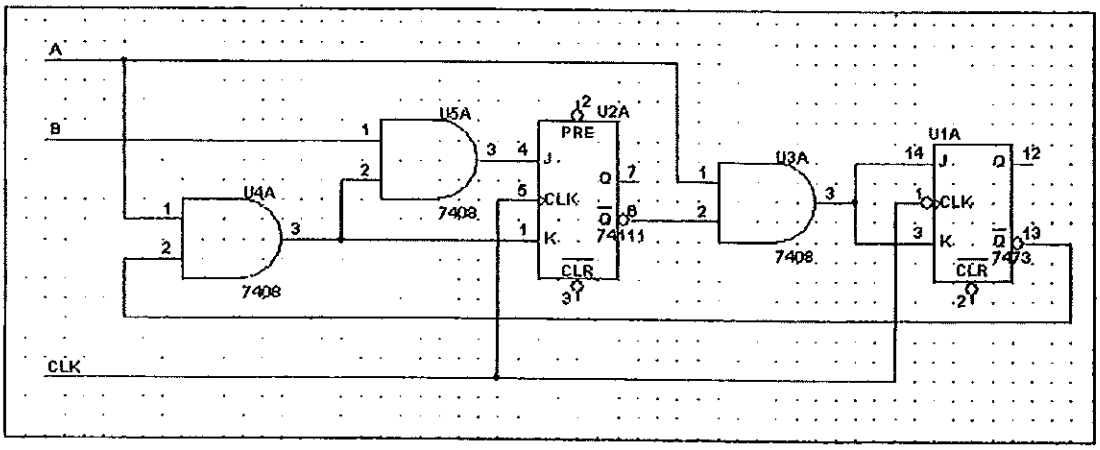


Figura 8

Nota: Considere que las señales **CLR*** y **PRE*** están desactivadas.

Parámetros: $t_{dpuertas} = 1$ ns

$t_{dFF} = 3$ ns

$t_{setup} = 2$ ns

FALTA 3.3

BIESTABLES
TEMPORIZACIÓN

devolver a clase 15

PROBLEMA 3

LO HICIMOS EN LA CLASE 5

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)*

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	1	0
1	1	0	1	1
1	1	1	1	0

Retención

Transparente

↳ HA HABIDO CAMBIO?

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

Simplificación por Karnaugh (de la función T)

EN, D / Q_t	00	01	11	10
0	0	0	(1)	0
1	0	0	0	(1)

Voy a llamar E a ENABLE ya que no se confunda

$T = E D \bar{Q}_t + E \bar{D} Q_t =$

$= E (D \bar{Q}_t + \bar{D} Q_t) = E (D \oplus Q_t)$

recordamos la tabla de verdad de un biestable T

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)*

CLASE 15

Activo por flanco de subida

Activo por flanco de bajada.

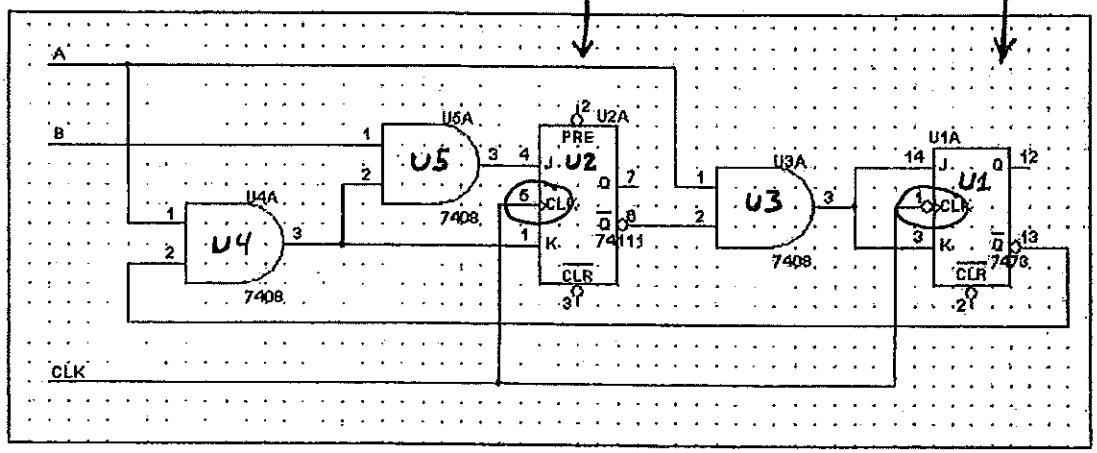
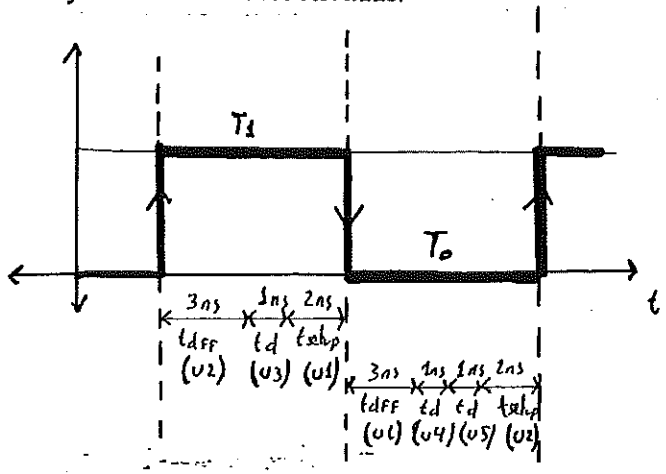


Figura 8

Nota: Considere que las señales **CLR*** y **PRE*** están desactivadas.

Parámetros: $t_{dpuertas} = 1$ ns
$t_{dFF} = 3$ ns
$t_{setup} = 2$ ns



$T_{1min} = 3+1+2 = 6$ ns

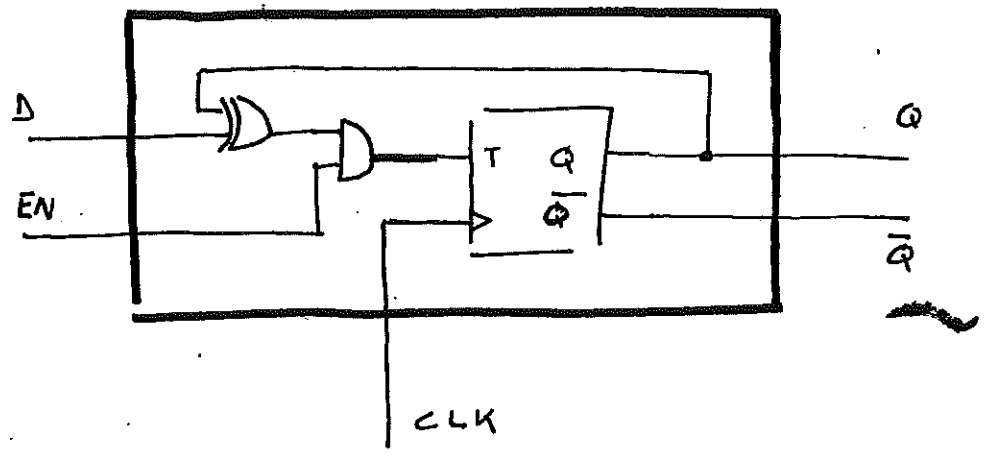
$T_{0min} = 3+1+1+2 = 7$ ns

$\frac{T_{min}}{2} = \max\{6, 7\} = 7$ ns

$T_{min} = 14$ ns

$f_{max} = \frac{1}{14ns} = 71'4$ MHz

Continuación 3.1



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*)

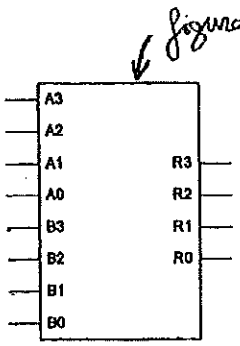
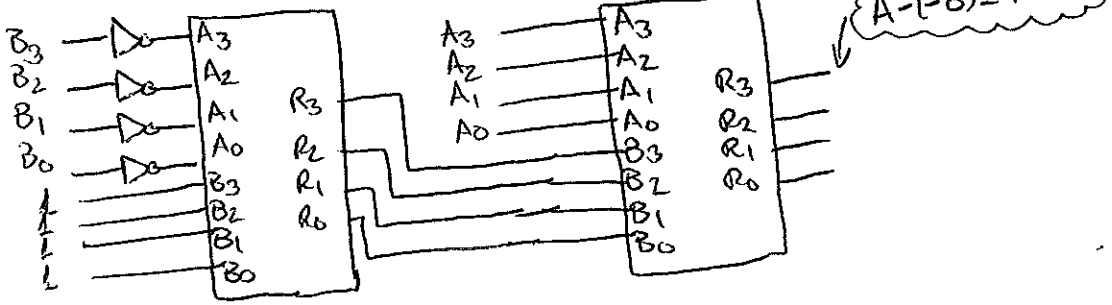


figura 2 X)

$S = A+B = A-(-B) = A-(\bar{B}+1) = A-(\bar{B}-(-1))$

+óptimo pero no oficial: $A-(-B) = A-(0-B)$ y ¿qué es este 0?

Representamos un (-1) en compl a 2: 0001 → 1110 → 1111 → (-1)



PROBLEMA 3

Se desea implementar un generador de números seudo-aleatorios utilizando un registro de desplazamiento de 3 bits basado en biestables de tipo D, según la *Figura 3*.

La secuencia a generar será la siguiente, codificada en binario como "**ABC**" donde **C** es el bit menos significativo:

0, 4, 6, 7, 3, 1, 0, 4, 6, 7, 3, 1, ...

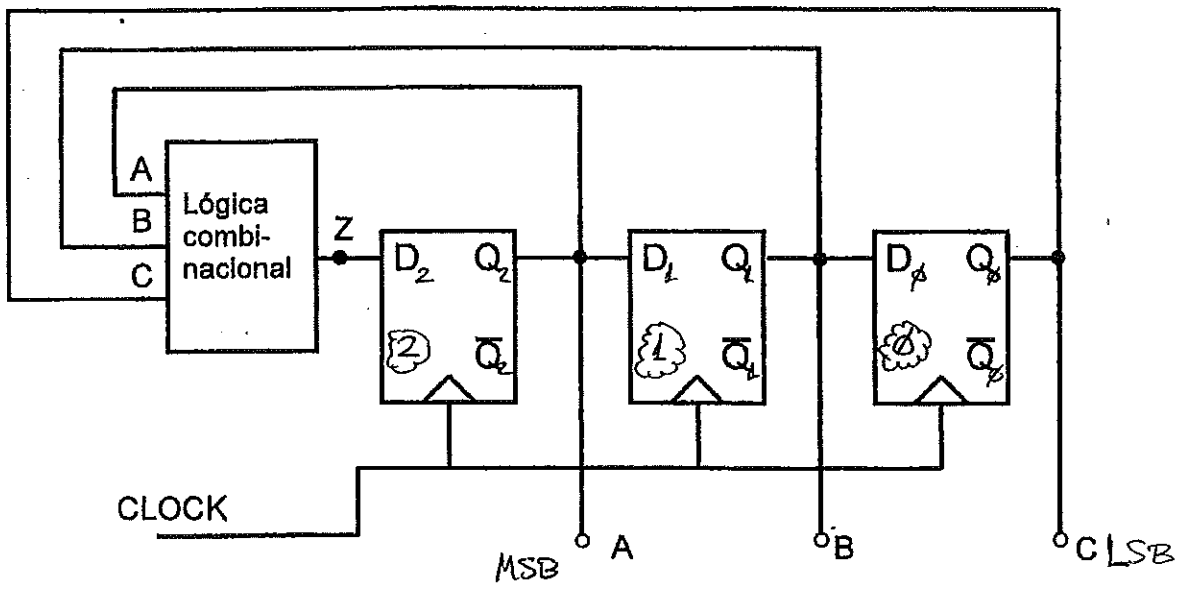


Figura 3

3.1 Rellene la tabla de verdad de la señal **Z** en función de **A**, **B** y **C**. (*10 puntos*)

A	B	C	Z
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

W
1
1
0
0
1
0
0
X

3.2 Dibuje el mapa de Karnaugh y utilice dicha simplificación para determinar la expresión lógica de **Z**. **Nota:** En caso de no haber respondido el primer apartado utilice la tabla de verdad de la señal **W**. (*5 puntos*)

3.3 Implemente la lógica combinacional para generar **Z** utilizando únicamente multiplexores de 4 entradas (*Figura 4*). **Nota:** En caso de no haber respondido el primer apartado utilice la tabla de verdad de la señal **W**. (*5 puntos*)

3.1 Tabla de excitaciones y transiciones de dicho contador: 0,4,6,7,3,1,0,4,6,7...

Q_t est. actual Q_2 Q_1 Q_0	Q_{t+1} est sig. Q_2 Q_1 Q_0	biest 2 D_2	biest 1 D_1	biest 0 D_0
0 0 0	1 0 0	1	0	0
1 0 0	1 1 0	1	1	0
1 1 0	1 1 1	1	1	1
1 1 1	0 1 1	0	1	1
0 1 1	0 0 1	0	0	1
0 0 1	0 0 0	0	0	0

$Q_{t+1}=D$ ↗

tabla de transiciones — tabla de excitaciones

Notación: $Z \equiv D_2$, $A \equiv Q_2$, $B \equiv Q_1$, $C \equiv Q_0$

3.2

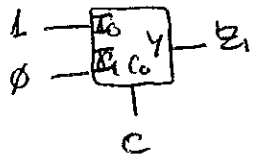
A	B	C	Z
0	0	0	1
0	0	1	0
0	1	0	X
0	1	1	0
1	0	0	1
1	0	1	X
1	1	0	1
1	1	1	0

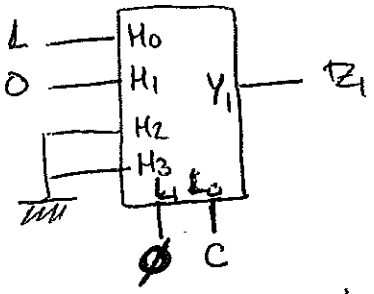
C \ AB	00	01	11	10
0	1	X	1	1
1	0	0	0	X

$Z = \overline{C}$ X

3.3 Implementamos Z_1 mediante un MUX 4x2

aunque tendremos 1 con MUX 2x1





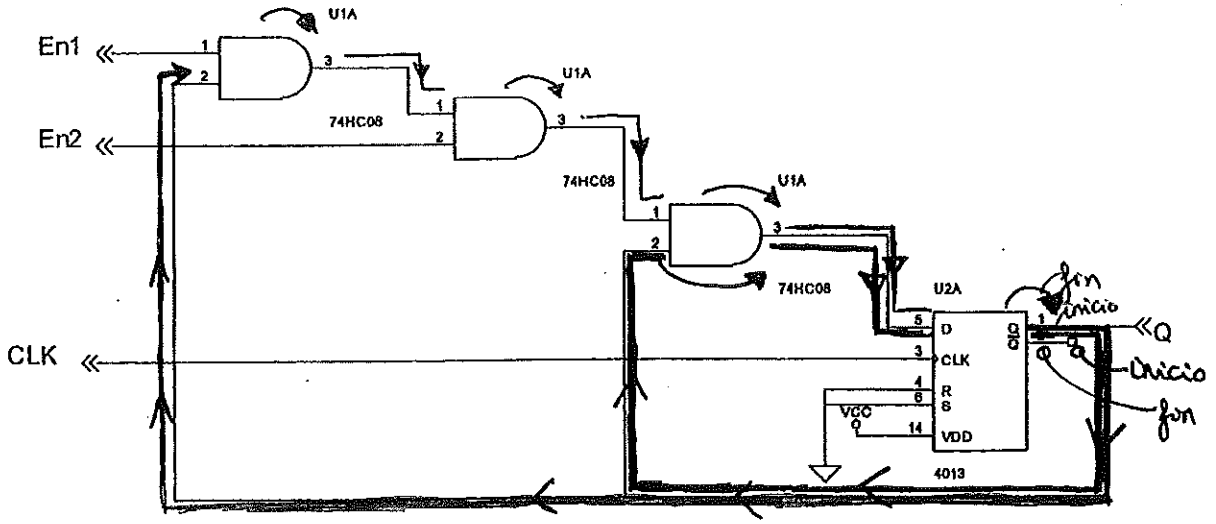
$Y = \overline{C_0} I_0 + C_0 I_1$

$Y_1 = \overline{L_1}\overline{L_0} H_0 + \overline{L_1} L_0 H_1 + L_1 \overline{L_0} H_2 + L_1 L_0 H_3$ → destabilitamos las entradas que no usemos

poniendo ese ø garantizamos que no se accede a H_2 y H_3 ⇒ da = que poner en H_2 y H_3, pues no las usamos

SEPTIEMBRE 2004

PROBLEMA 5



$t_{dAND} = 1$ ns
$t_{dff} = 2$ns
$t_{setup} = 2$ ns

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*)

Elegimos el camino más largo incluyendo t_{setup}

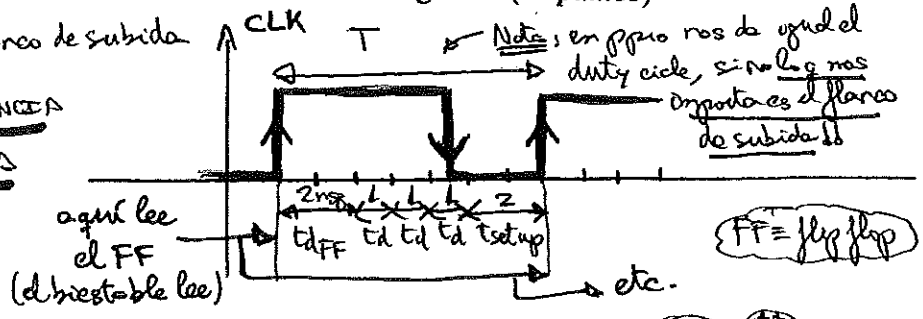
(1) El biestable es activo en flanco de subida

$t_{dAND} = 1$ nsg

$t_{dFF} = 2$ nsg

$t_{setup} = 2$ nsg

● FRECUENCIA MÁXIMA

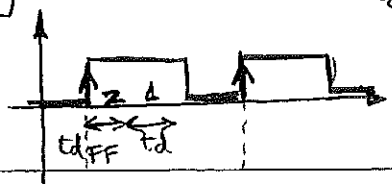


(**) Los 7nsg obtenidos, es el tiempo q le tenemos q dar a la señal que recorre el camino más largo para que llegue, a tiempo, a la entrada del biestable y estar así preparada para la llegada del siguiente flanco activo)

$T_{min} = t_{dFF} + 3t_{dAND} + t_{setup} = 2 + 3 \cdot 1 + 2 = 7$nsg (**)

$f_{max} = \frac{1}{T_{min}} = \frac{1}{7nsg} = 143$ MHz

(2) MÁXIMO TIEMPO DE HOLD



$t_{holdmax} = t_{dFF} + t_{dAND} = 2 + 1 = 3$nsg

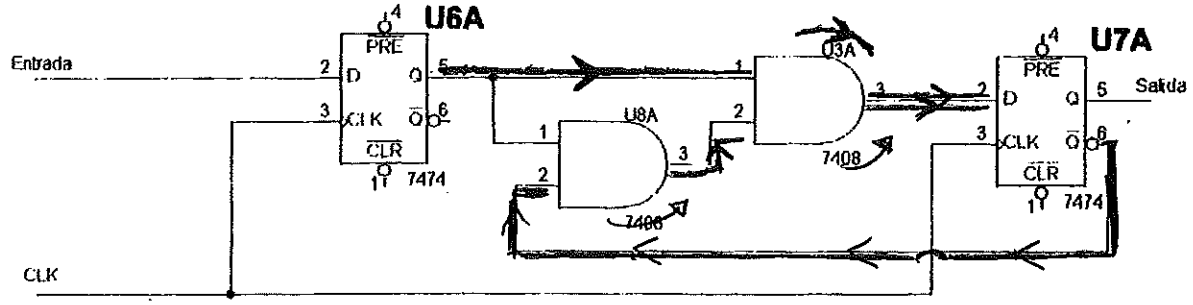
Este es el tiempo que va a tardar la señal más rápida en hacer cambiar la entrada del flip flop(FF). Por lo tanto, como mucho, el FF debe tener un thold de 3nsg, porque sabemos que en ese tiempo SEGURO que varía su entrada

Elegimos el camino más corto y NO incluimos el tiempo de SET UP

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 El reloj es el mismo para los dos (activo a nivel alto)

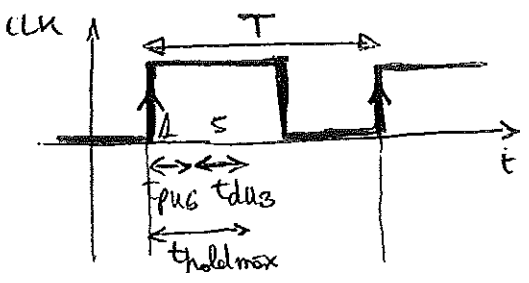
posibles caminos:

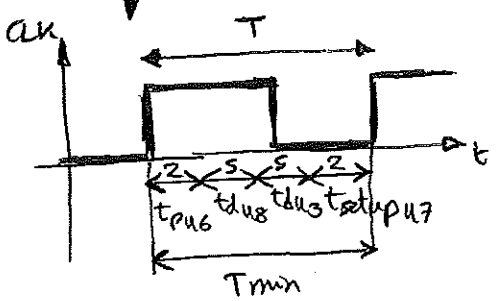
T_{min} ○ U7A $\xrightarrow{\overline{Q}}$ U8A → U3A → U7A: $T_{min1} = t_{pu7} + t_{du8} + t_{du3} + t_{setup\,u7} = 2+5+5+2 = 14\,ns$

- Entrada → U6A → U8A → U3A → U7A: $T_{min2} = t_{pu6} + t_{du8} + t_{du3} + t_{setup\,u7} = 1+5+5+2 = 13\,ns$
- Entrada → U6A → U3A → U7A: $T_{min3} = t_{pu6} + t_{du3} + t_{setup\,u7} = 1+5+2 = 8\,ns$

$T_{min} = \max\{T_{min1}, T_{min2}, T_{min3}\} = 14\,ns \Rightarrow f_{MÁX} = 71{,}4\,MHz$ → tiempo del camino más largo desde un biestable hasta otro (puede ser el mismo) ($t+t_{setup}$)

② $t_{holdmax} = T_{min3} - t_{setup} = 6\,ns$





FEBRERO 2005

OSO] No especifican cual es la salida ⇒ identificamos con el giro del motor { E=0 parado / E=1 girando

Problema 3

Un motor que puede estar parado o en marcha dispone para su control de dos pulsadores **A** y **P**, de arranque y parada respectivamente. El funcionamiento es el siguiente:

-Si el motor estuviera parado y se pulsara **A**, el motor no comenzaría a girar hasta que se soltara **A**.
-Si el motor estuviera girando y se pulsara **P**, el motor no pararía de girar hasta que se soltara **P**.
-Si el motor estuviera girando y se pulsara **A**, el motor seguiría girando.
-Si el motor estuviera parado y se pulsara **P**, el motor seguiría parado.

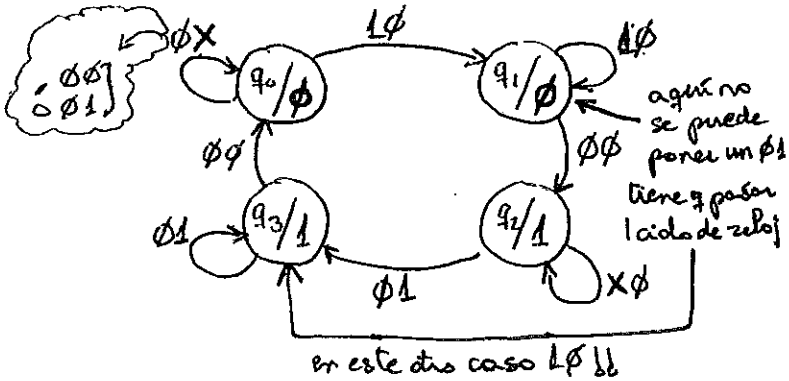
Asumimos que:
- El pulsar un botón pone la señal correspondiente (**A** ó **P**) a "1", y el soltarlo la pone a "0".
- No se pueden pulsar simultáneamente los dos pulsadores.
- Entre pulsaciones consecutivas pasará como mínimo un ciclo de reloj.

3.1 Obtenga **justificadamente** el diagrama de estados en caso de utilizar una máquina de Moore. Se valorará el grado de simplificación.

3.2 Obtenga **justificadamente** el diagrama de estados en caso de utilizar una máquina de Mealy. Se valorará el grado de simplificación.

3.1 Moore

$q_0 \equiv$ motor parado sin A pulsado
$q_1 \equiv$ " " con A "
$q_2 \equiv$ " girando sin P "
$q_3 \equiv$ " " con P "



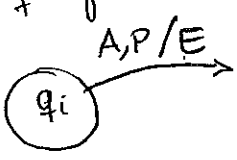
Leyenda:

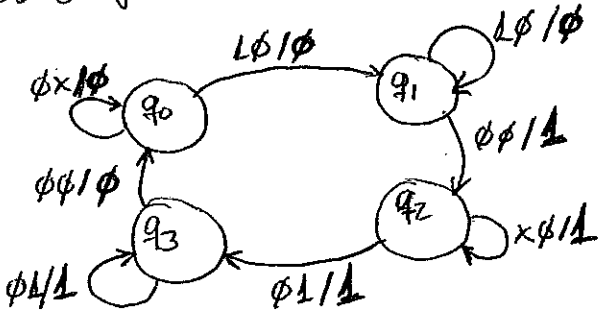
Nota: si no hiciera falta soltar el botón para activar la marcha/la parada sólo harían falta 2 estados
SIEMPRE que haga falta soltar un botón para que haga efecto necesitamos un estado intermedio que represente a la espera al botón pulsado

3.2 Mealy

Mismos estados que para Moore

Leyenda:





Extra: todas las flechas que le llegan al estado (todas las transiciones que llegan al estado) tienen la misma salida ⇒ podemos pasarlo a Moore

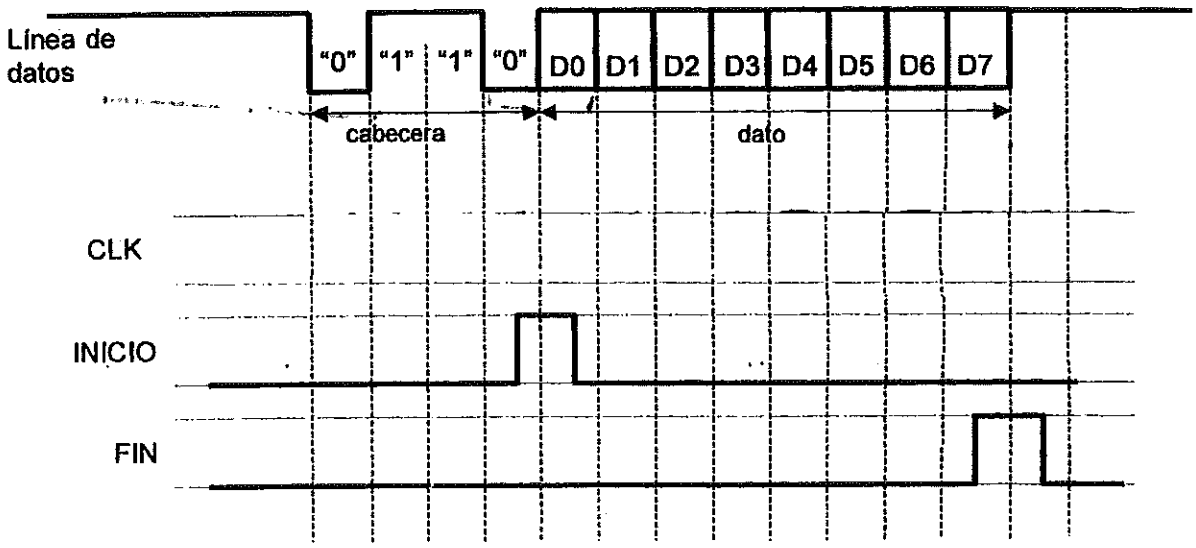
FEBRERO 2005

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:

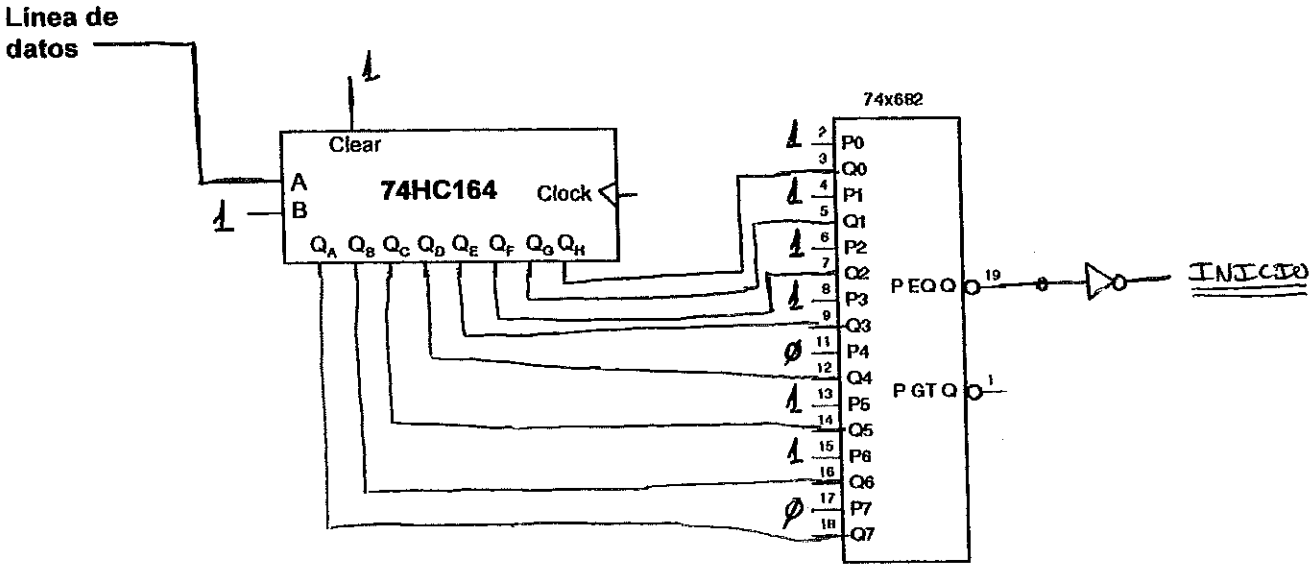
- La línea de datos en reposo se encuentra en estado alto ("1").
- Cada vez que se quiere transmitir una nueva trama, primero se transmite una cabecera de 4 bits predeterminados, seguidos de los 8 bits de datos.
- Los 4 bits de cabecera tienemn código **0110**.
- Al detectarse la cabecera se generará 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.



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 en la Figura 2.

Nota. El registro de desplazamiento 74HC164 está formado por biestables tipo D con tiempos de setup no despreciables. En el anexo del examen se encuentran las características del 74HC164 y del 74HC682. Utilice la información de dichas hojas.



PEQQ = P equals to Q
PGTQ = P greater than Q

Nota: tabla de verdad del registro

Inputs			Outputs
Clear	CLK	AB	$Q_A Q_B \cdots Q_H$
1	↑	11	1 $Q_{AN} \cdots Q_{GN}$
1	↑	ø x	ø $Q_{AN} \cdots Q_{GN}$
1	↑	x ø	ø $Q_{AN} \cdots Q_{GN}$
1			

↑ flanco activo del reloj

el registro siempre desplaza a derechos

Respecto a las entradas serie A y B:
se trata de dos entradas conectadas internamente mediante una AND
(se puede ver en la tabla que: $1 \cdot 1 = 1$, $\emptyset \cdot X = X \cdot \emptyset = \emptyset$)
Si queremos usar una de ellas como entrada serie de datos basta con que conectemos dicha entrada a A y pongamos **B=1**

El clear es activo a nivel bajo y pone todo a ø

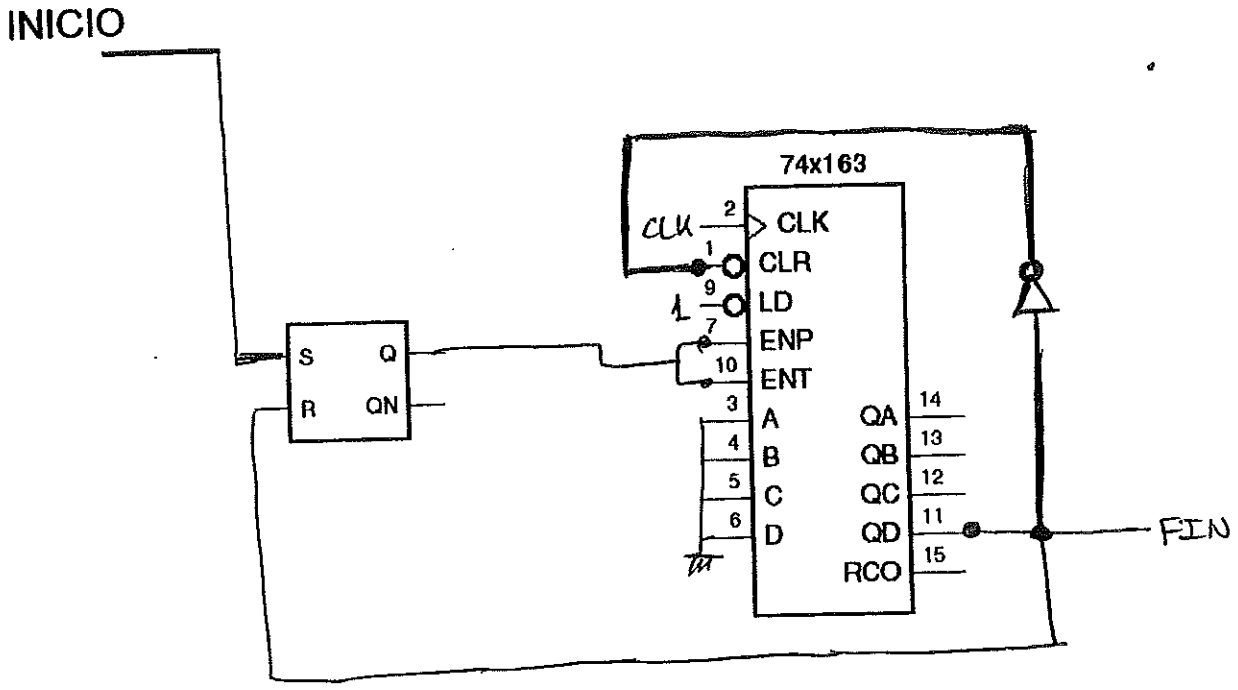
4.1 Tenemos que detectar la secuencia: øø11 → 11111111
cabecera que va a entrar — línea en reposo

Con lo que activaremos INICIO cuando el estado del registro sea: øø11øø11
aquí la cabecera acaba de entrar al registro

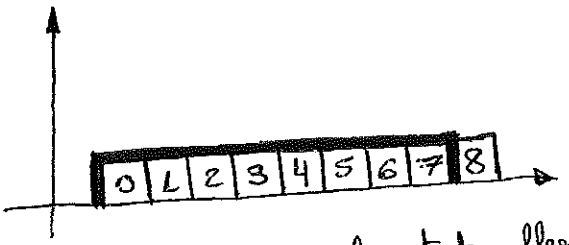
Vamos a utilizar el comparador para avisar justo cuando el estado del registro sea esa cadena. Cuidado con:

- → CLEAR activo a nivel bajo ⇒ CLEAR = 1.
- → Ojo a las entradas A y B: queremos pasar en serie los datos que vienen por la línea: { línea de datos = A ; B = 1 }
- → El comparador saca un cero cuando detecta igualdad. Para generar INICIO negaremos la salida **PEQQ**

4.2 Utilizando un contador binario 74HC163 (información en el anexo), una báscula Set – Reset, las puertas inversoras necesarias, y partiendo de la señal INICIO implemente el circuito que genere la señal FIN sobre el esquema de la Figura 4. Dibuje todas las conexiones necesarias para un funcionamiento correcto. Se valorará el grado de simplificación.



Queremos implementar un circuito que a partir de un "1" en INICIO cuente 8 ciclos de reloj y después genere una señal.



Tenemos que esperar a contar de ϕ a 7 (8 ciclos completos) y justo después generar la señal fin

En resumen, cuando el contador llegue a 8 tenemos que:

→ generar la señal FIN: 1 0 0 0 ← es la primera vez que aparece
directamente la usaremos como señal FIN

→ resetear la cuenta: activando el CLEAR del contador

→ deshabilitar el contador hasta que vuelva a pulsarse la señal INICIO

Habilitación / Inhibición: INICIO → habilita poniendo 1 en ENT y ENP
FIN → deshabilita poniendo ϕ en ENT y ENP

⇒ estas opciones a una báscula R/S para que se mantenga hasta que queramos lo contrario

SN54HC682, SN74HC682
8-BIT MAGNITUDE COMPARATORS

- **Compare Two 8-Bit Words**
- **100-kΩ Pullup Resistors Are on the Q Inputs**
- **Package Options Include Plastic Small-Outline (DW) and Ceramic Flat (W) Packages, Ceramic Chip Carriers (FK), and Standard Plastic (N) and Ceramic (J) 300-mil DIPs**

description

These magnitude comparators perform comparisons of two 8-bit binary or BCD words. The 'HC682 feature 100-kΩ pullup termination resistors on the Q inputs for analog or switch data.

FUNCTION TABLE

DATA INPUTS P, Q	OUTPUTS $\overline{P=Q}$	OUTPUTS $\overline{P>Q}$
P = Q	L	H
P > Q	H	L
P < Q	H	H

The P < Q function can be generated by applying $\overline{P=Q}$ and $\overline{P>Q}$ to a 2-input NAND gate.

MM54HC163/MM74HC163 Synchronous Binary Counter with Synchronous Clear

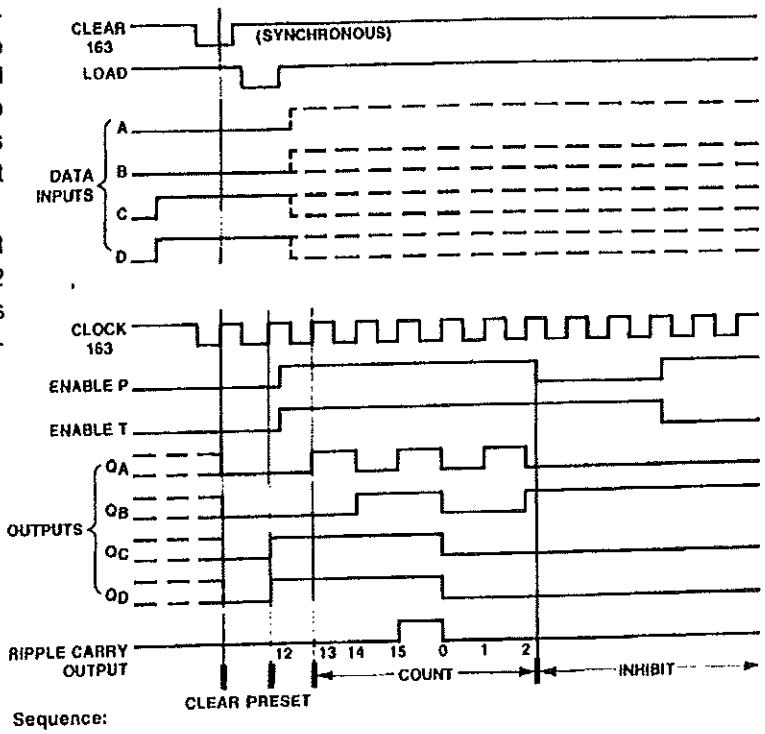
These counters may be preset using the LOAD input. Presetting of all four flip-flops is synchronous to the rising edge of CLOCK. When LOAD is held low counting is disabled and the data on the A, B, C, and D inputs is loaded into the counter on the rising edge of CLOCK. If the load input is taken high before the positive edge of CLOCK the count operation will be unaffected.

All of these counters may be cleared by utilizing the CLEAR input. The clear function on the MM54HC162/MM74HC162 and MM54HC163/MM74HC163 counters are synchronous to the clock. That is, the counters are cleared on the positive edge of CLOCK while the clear input is held low.

H high level, L low level
X don't care, ↑ low to high transition

'HC162/HC163

CLK	CLR	ENP	ENT	Load	Function
↑	L	X	X	X	Clear
X	H	L	L	H	Count & RC disabled
X	H	L	H	H	Count disabled
X	H	L	L	H	Count & RC disabled
↑	H	X	X	L	Load
↑	H	H	H	H	Increment Counter



Sequence:
(1) Clear outputs to zero
(2) Preset to binary twelve
(3) Count to thirteen, fourteen, fifteen, zero, one and two
(4) Inhibit

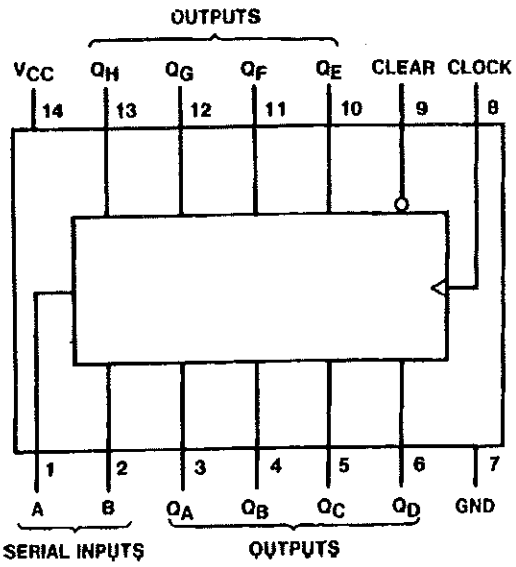
National Semiconductor

MM54HC164/MM74HC164
8-Bit Serial-in/Parallel-out Shift Register

General Description

The MM54HC164/MM74HC164 utilizes advanced silicon-gate CMOS technology. It has the high noise immunity and low consumption of standard CMOS integrated circuits. It also offers speeds comparable to low power Schottky devices.

This 8-Bit shift register has gated serial inputs and CLEAR. Each register bit is a D-type master/slave flip flop. Inputs A & B permit complete control over the incoming data. A low at either or both inputs inhibits entry of new data and resets the first flip flop to the low level at the next clock pulse. A high level on one input enables the other input which will then determine the state of the first flip flop. Data at the serial inputs may be changed while the clock is high or low, but only information meeting the setup and hold time requirements will be entered. Data is serially shifted in and out of the 8-Bit register during the positive going transition of the clock pulse. Clear is independent of the clock and accomplished by a low level at the CLEAR input.



Truth Table

Inputs				Outputs			
Clear	Clock	A	B	Q_A	Q_B	...	Q_H
L	X	X	X	L	L		L
H	L	X	X	Q_{AO}	Q_{BO}		Q_{HO}
H	↑	H	H	H	Q_{An}		Q_{Gn}
H	↑	L	X	L	Q_{An}		Q_{Gn}
H	↑	X	L	L	Q_{An}		Q_{Gn}

H = High Level (steady state), L = Low Level (steady state)

X = Irrelevant (any input, including transitions)

↑ = Transition from low to high level.

Q_{AO}, Q_{BO}, Q_{HO} = the level of Q_A, Q_B, or Q_H, respectively, before the indicated steady state input conditions were established.

Q_{An}, Q_{Gn} = The level of Q_A or Q_G before the most recent ↑ transition of the clock; indicated a one-bit shift.

FEBRERO 2005

Problema 5

A continuación se describe un componente en lenguaje VHDL.

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
entity V74xxxx 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 V74xxxx;
architecture V74xxx_arch of V74xxxx 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 V74xxx_arch;
```

cabecera ←
entidad ←
arquitectura ←
cuando venga un flanco de reloj y sea de subida entonces:
Definimos una señal "temporal" (para hacer cuentas intermedias)
estas son las señales que activarán el proceso
cuando varíe
si CLR_L = Ø ponemos todo a Ø
si LD_L = Ø asignamos los 4 bits de datos a la entrada IQ en la señal IQ
si ENT=ENP=1 ⇒ incrementamos el valor de IQ
si IQ=15 y ENT=1 ⇒ ponemos 1 en la salida RCO
en caso contrario RCO = Ø
al final sacamos la señal IQ por la salida Q

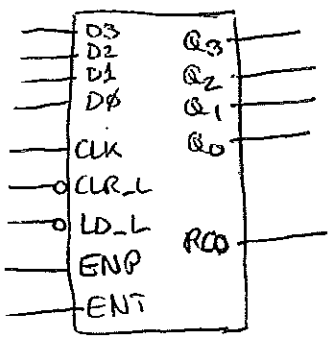
Nota: El tipo UNSIGNED es equivalente al tipo STD_LOGIC_VECTOR que permite la función adicional "+" que es la función "suma de números naturales".

5.1 Sobre el bloque de la Figura 5, dibuje las entradas y salidas del componente y explique detalladamente su funcionamiento.
Nota: Las entradas se dibujan a la izquierda y las salidas a la derecha del bloque.

5.2 Modifique el código VHDL para que:

a) El reloj se active con el flanco de bajada.
b) La señal RCO sea activa a nivel bajo.
c) Funcione en módulo 10. En este caso, la señal RCO debe ser activa cuando el componente esté en su último estado.

5.1



Tal y como hemos comentado en el código, se trata de un contador módulo 16 (que cuenta de Ø a 15) Tiene 4 salidas que contarán de Ø a 15
Peculiaridades:
- contador siempre ascendente
- entrada CLR activa a nivel bajo síncrona
- entrada LOAD síncrona, activa a nivel bajo (que efectúa la carga D a Q)
- entradas de habilitación activas a nivel alto
- reloj activo en flanco de subida
- salida RCO de final de cuenta (poniendo un 1)

5.2 a) if (CLK'event and CLK='0') then
↳ cambio realizado

b) if (IQ=15) and (ENT='1') then RCO <= '0';
else RCO <= '1';
endif;

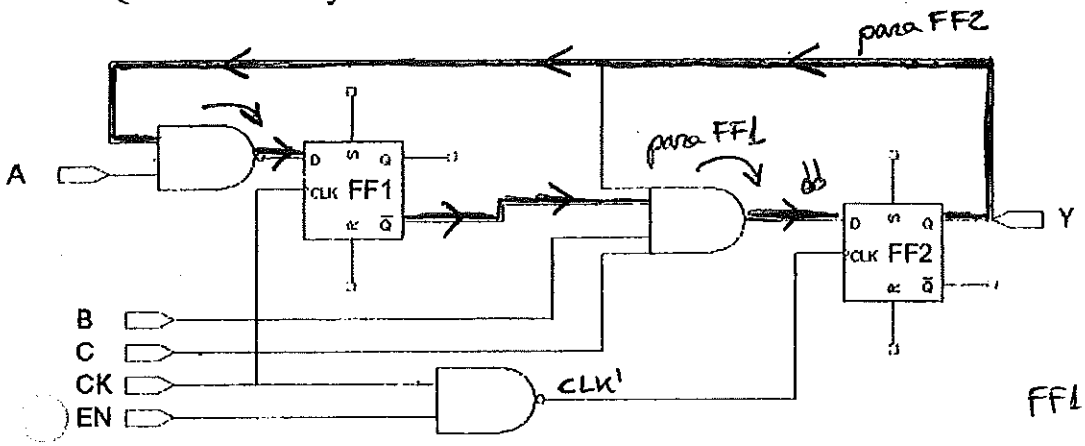
c) elsif (ENT and ENP) = '1' then;
if IQ<= 8 then IQ <= IQ+1;
elsif IQ =9 then IQ <= (others => '0');
endif;
endif;
if (IQ = 9) and

" todos a cero "

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

$$\begin{cases} t_{dNAND2} = 1\ ns \\ t_{dAND4} = 3\ ns \\ t_{dFF1} = t_{dFF2} = 3\ ns \\ t_{setup-FF1} = t_{setup-FF2} = 2\ ns \end{cases}$$

FF1 y FF2 activos en flanco de subida

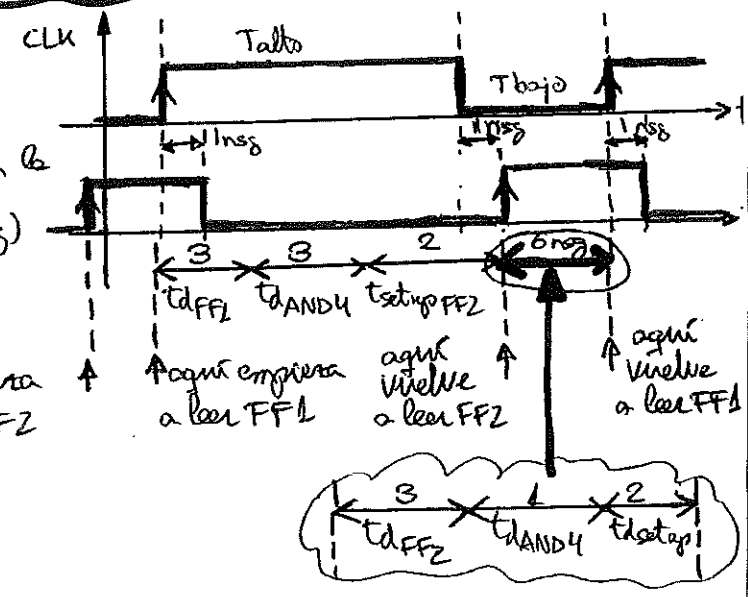
1.1 Obtenga justificadamente, sobre un cronograma, la máxima frecuencia de funcionamiento del mismo.

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.

// Primera norma del diseño síncrono: NUNCA se enchufa puertas lógicas ni leches al reloj, SIEMPRE hay que enchufar el reloj directamente a los componentes



① CLK' = $\overline{CLK \cdot 1}$ = $\overline{CLK}$

Además la puerta introduce un retardo en la señal CLK' con respecto a CLK (de 1ns)

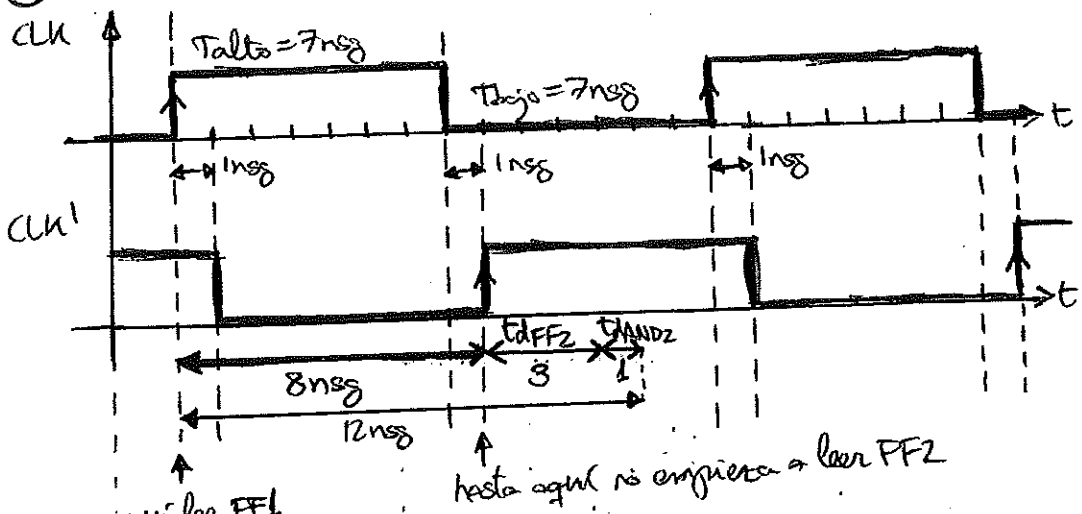
Del cronograma obtenemos:

CLK: Talto = 8ns − 1ns = 7ns
Tbajo = 6ns + 1ns = 7ns

⇒ Tmin = 14ns ⇒ f_{max} = 71,43 MHz

② sabemos que Duty Cicle = 50% (para la escala)

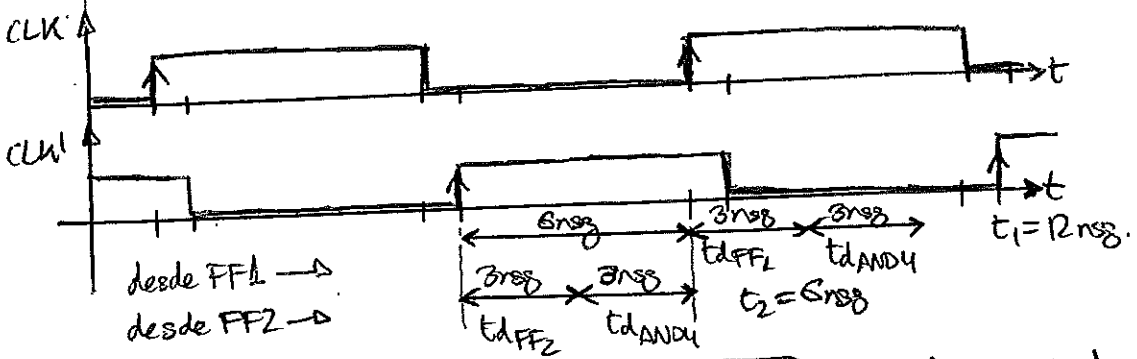
FF1



thold MÁX (FF1) = 8 + 3 + 1 = 12 nsg

↳ tiempo que tarda en empezar a leer FF2 desde que empezó a leer FF1

FF2



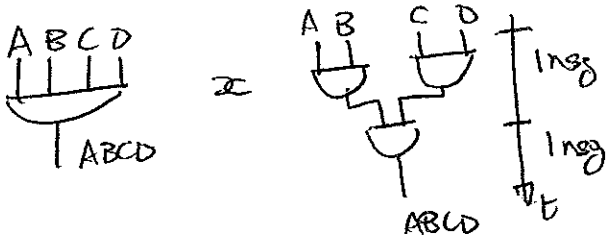
thold máx = min {t1, t2} = min {12, 6} = 6nsg la señal más rápida que "molesta" a FF2 tarda solo 6nsg en llegar. Este es el thold-max

③

la señal tarda en propagarse

$t_d = 3 \cdot 1\,nsg = 3nsg$

④



la señal tarda en propagarse

$t_d = 2 \cdot 1\,nsg = 2nsg$

SEPTIEMBRE 2005

Problema 3

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.
Ejemplo: Número 3 = estado (011) = salida (011).

3.2 Complete la siguiente tabla de transiciones y excitaciones en base al diagrama de estados del contador obtenido en el apartado 3.1 utilizando biestables tipo D.

(Q2 Q1 Q0) = estado actual
(Q2* Q1* Q0*) = nuevo estado
(D2 D1 D0) = entradas de los biestales tipo D
(B2 B1 B0) = Salidas del contador

U/D	Q2	Q1	Q0	Q2*	Q1*	Q0*	D2	D1	D0	B2	B1	B0
0	0	0	0									
0	0	0	1									
0	0	1	0									
0	0	1	1									
0	1	0	0									
0	1	0	1									
0	1	1	0									
0	1	1	1									
1	0	0	0									
1	0	0	1									
1	0	1	0									
1	0	1	1									
1	1	0	0									
1	1	0	1									
1	1	1	0									
1	1	1	1									

3.3 Obtenga las expresiones simplificadas por Karnaugh para las entradas de los tres biestables, **(D2 D1 D0)**, y para las salidas del contador, **(B2 B1 B0)**.

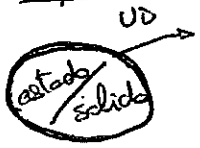
3.4 Dibuje la implementación final del circuito necesario para obtener **D0** y **D1** con el mínimo número de puertas XOR, XNOR, AND y/o OR de dos entradas. Disponemos de la señal U/D y de su complementaria.

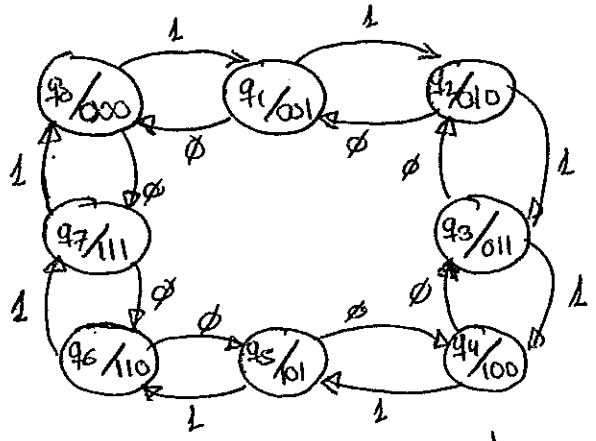
3.5 Asumiendo que la frecuencia del reloj que usamos es fina, modifique el diagrama de estados del apartado 3.1 para que el contador cuente a la mitad de velocidad. Suponga que la señal U/D permanece invariable.

3.1 Diagrama de estados:

ESTADO		DEFINICIÓN		CODIFICACIÓN
q_0	≡	contador a 0	≡	000
q_1	≡	contador a 1	≡	001
q_2	≡	" " 2	≡	010
q_3	≡	" " 3	≡	011
q_4	≡	" " 4	≡	100
q_5	≡	" " 5	≡	101
q_6	≡	" " 6	≡	110
q_7	≡	" " 7	≡	111

Leyenda:





3.2

las salidas coinciden directamente con el estado porque así estamos diseñando el contador

variables de estado				estado siguiente								
U/D	Q_2	Q_1	Q_0	Q_2^*	Q_1^*	Q_0^*	D_2	D_1	D_0	B_2	B_1	B_0
0	0	0	0	0	0	1	0	0	1	0	0	1
0	0	0	1	0	1	0	0	1	0	0	1	0
0	0	1	0	0	1	1	0	1	1	0	1	1
0	0	1	1	1	0	0	1	0	0	1	0	0
0	1	0	0	1	0	1	1	0	1	1	0	1
0	1	0	1	1	1	0	1	1	0	1	1	0
0	1	1	0	1	1	1	1	1	1	1	1	1
1	0	0	0	1	1	1	1	1	1	1	1	1
1	0	0	1	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	1	0	0	1	0	0	1
1	0	1	1	0	1	0	0	1	0	0	1	0
1	1	0	0	0	1	1	0	1	1	0	1	1
1	1	0	1	1	0	0	1	0	0	1	0	0
1	1	1	0	1	0	1	1	0	1	1	0	1
1	1	1	1	1	1	0	1	1	0	1	1	0

contando hacia arriba

contando hacia abajo

tabla de transiciones | tabla de excitaciones | tabla de salida

Nota: tabla biestable D

Q	Q*	D
0	0	0
0	1	1
1	0	0
1	1	1

Nota: es una máquina de Moore, cumple que las salidas sólo se asocian con el estado ACTUAL

3.3 Implementamos las funciones:

Algunas directamente: $B0 = Q0$ $\quad B2 = Q2$ $\quad B1 = Q1$ $\quad D0 = \overline{Q0}$ → estas 3 pq las salidas coinciden con el estado (ver tabla)

Las que nos quedan por Karnaugh:

D1

Q_1Q_0 \ $U/\bar{D}Q_2$	00	01	11	10
00	1	1	0	0
01	0	0	1	1
10	1	1	0	0
11	0	0	1	1

$$D_1 = \overline{U/\bar{D}}\,\bar{Q}_1Q_0 + U/\bar{D}\,\bar{Q}_1Q_0 + \overline{U/\bar{D}}\,Q_1Q_0 + U/\bar{D}\,Q_1\bar{Q}_0 =$$
$$= \overline{U/\bar{D}}\,(Q_1 \oplus Q_0) + U/\bar{D}\,(Q_1 \oplus Q_0) =$$
$$\boxed{D_1 = \overline{U/\bar{D} \oplus Q_1 \oplus Q_0}}$$

D2

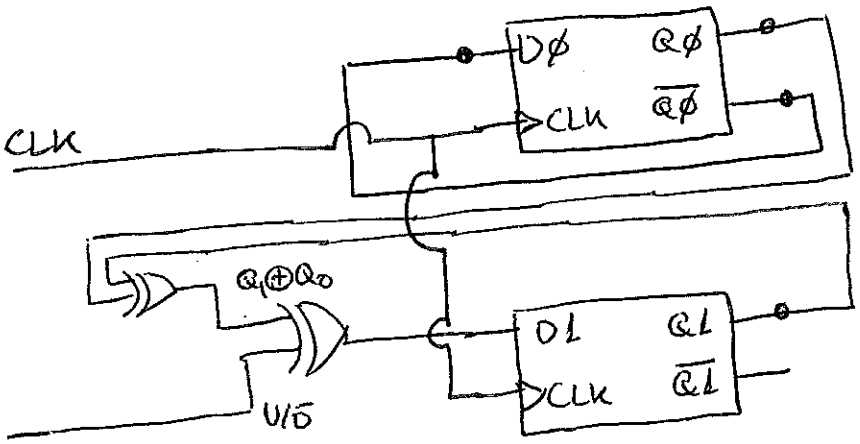
Q_1Q_0 \ $U/\bar{D}Q_2$	00	01	11	10
00	1	0	1	0
01	0	1	1	0
11	0	1	0	1
10	0	1	1	0

$$\boxed{D_2 = \overline{U/\bar{D}}\,\bar{Q}_2\bar{Q}_1Q_0 + U/\bar{D}\,Q_2\bar{Q}_1 + \overline{U/\bar{D}}\,\bar{Q}_2Q_1 + \overline{U/\bar{D}}\,Q_2Q_0 + Q_2Q_1\bar{Q}_0 + U/\bar{D}\,\bar{Q}_2Q_1Q_0}$$

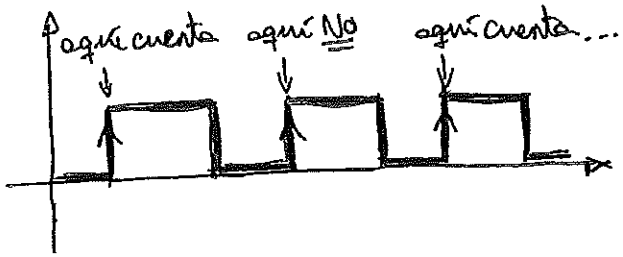
pusaco xD apenas se puede simplificar (no lo piden)

3.4 Implementación de D0 y D1

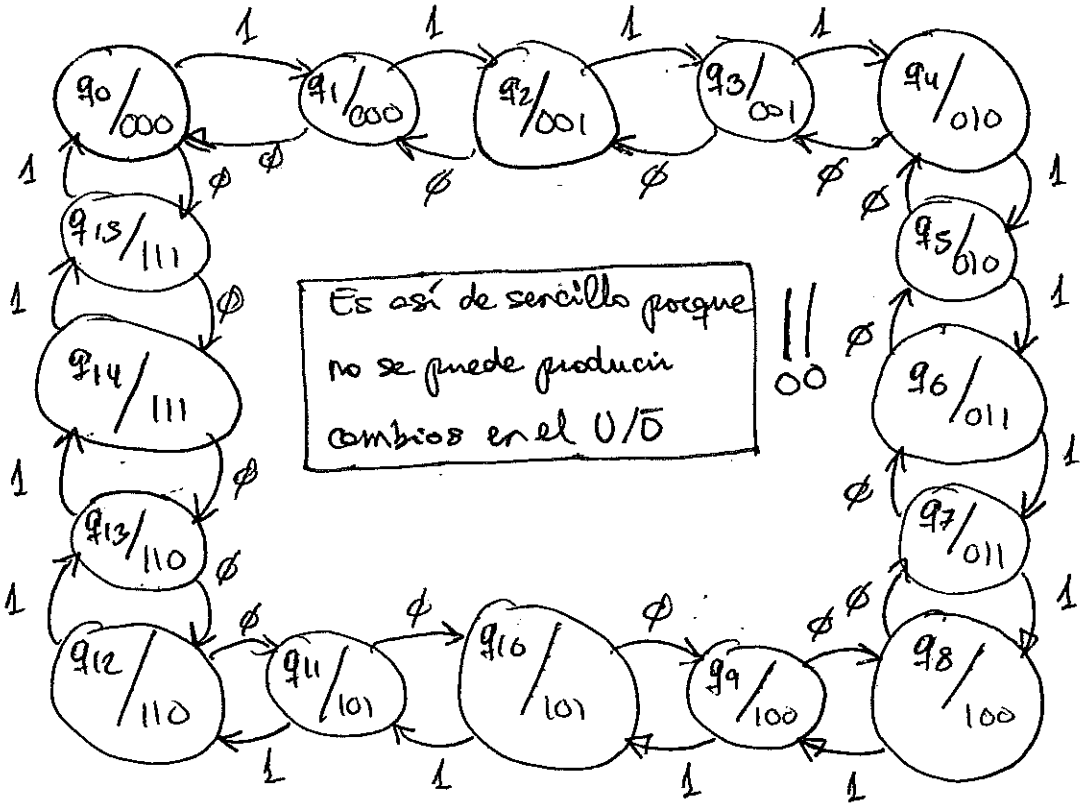
Nota: no dependen de Q2 así que no dibujamos ese biestable



3.5 Queremos que el contador cuente a la ½ de velocidad (sin modificar el CLK)



Vamos a añadir un estado intermedio entre cada dos estados originales, en el que NO variará la salida


q0/000
q1/000
q2/001
q3/001
q4/010
q5/010
q6/011
q7/011
q8/100
q9/100
q10/101
q11/101
q12/110
q13/110
q14/111
q15/111
1
ϕ
Es así de sencillo porque no se puede producir cambios en el U/D
11
00

SEPTIEMBRE 2005

Problema 4

4.1 Para la siguiente Máquina de Estados codificada en VHDL, obtenga el diagrama de estados indicando CLARAMENTE los estados, entradas y salidas.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY Automata1 IS
PORT (Clock, Resetn :IN  STD_LOGIC;
                w       :IN  STD_LOGIC;
                z       :OUT STD_LOGIC);
END Automata1;

ARCHITECTURE Behavior OF Automata1 IS
        TYPE State_type IS (A,B);
        SIGNAL y : State_type;
BEGIN
        PROCESS (Resetn, Clock)
        BEGIN
                IF Resetn = '0' THEN
                        y <= A;
                ELSIF (Clock'EVENT AND Clock = '1') THEN
                        CASE y IS
                                WHEN A =>
                                        IF w = '0' THEN
                                                y <= A;
                                                z <= '0';
                                        ELSE
                                                y <= B;
                                                z <= w;
                                        END IF;
                                WHEN B =>
                                        IF w = '0' THEN
                                                y <= A;
                                                z <= '0';
                                        ELSE
                                                y <= B;
                                                z <= w;
                                        END IF;
                        END CASE;
                END IF;
        END PROCESS;

END Behavior;
```

activo a nivel bajo significa la "n"

definimos un nuevo tipo de datos llamado State_type, que sólo tiene dos valores posibles: A ó B

creamos una variable auxiliar, de tipo SIGNAL (señal), que contendrá valores del tipo State_type (A ó B) (Y representará el estado en el que estamos. A ó B)

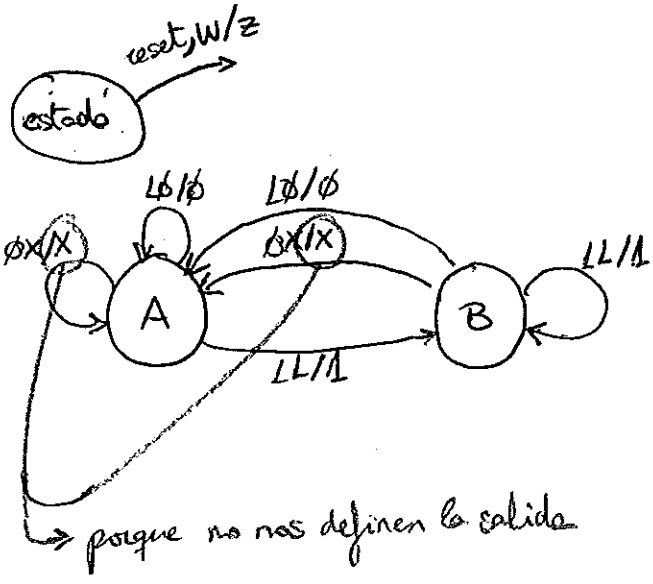
señales y activan el proceso

arquitectura (sólo piden diagrama de estados de)

cuando venga un flanco y sea el de subida

estado A

estado B



porque no nos definen la salida

Falta 4.2

FEBRERO 2006

PROBLEMA 2 (20 *PUNTOS*)

El sistema de apertura de una caja fuerte dispone de dos pulsadores: Pulsador BLANCO y pulsador NEGRO, que definen sendas señales **BLANCO** y **NEGRO** activas a nivel alto. La caja fuerte se abre mediante la generación de una señal de apertura **OP**, activa a nivel alto, que abre la puerta de la caja. Dicha apertura se produce si y solo si se efectúa una secuencia correcta de los pulsadores, que es: BLANCO-BLANCO-NEGRO; es decir pulsando BLANCO dos veces consecutivas y luego NEGRO.

El sistema también genera una señal de alarma, **AL** (activa a nivel alto), que se activa al finalizar la introducción de una secuencia de tres pulsaciones, si esta secuencia no fuera la correcta.

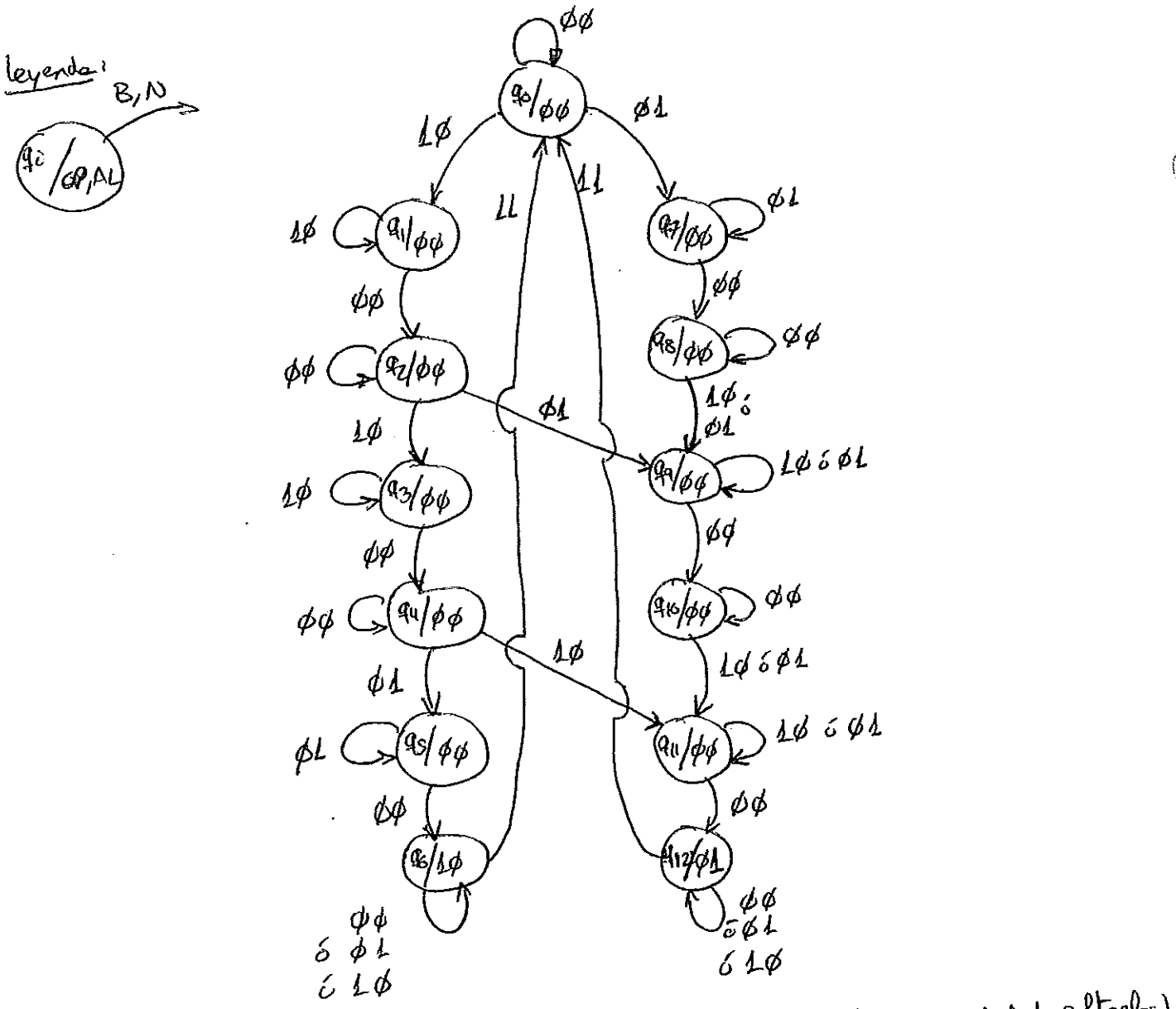
La operatividad del sistema es la siguiente:

- El sistema se encuentra inicialmente en un estado de reposo, con la puerta de la caja fuerte cerrada y ambas señales **OP** y **AL** desactivadas.
- Una vez introducida la secuencia correcta (BLANCO-BLANCO-NEGRO) se activa la señal **OP (OP=1)**, la caja se abre y permanece abierta hasta que se cierra manualmente.
- El sistema se **RESETEA**, volviendo al estado inicial de reposo, pulsando los dos pulsadores a la vez (BLANCO y NEGRO). Esta acción sólo se realiza al finalizar una secuencia de 3 pulsaciones (correcta ó no). No se considera que se puedan pulsar los dos pulsadores a la vez en ningún otro momento.
- La señal de alarma se debe activar (**AL=1**) si la secuencia introducida de pulsadores no es la correcta, pero **cuidado**: dicha activación se debe producir al finalizar de introducir el código de 3 pulsaciones. Esta decisión de diseño sirve para evitar dar pistas de cual de las tres pulsaciones ha sido la errónea.
- A la hora de dibujar el diagrama de estados tenga en cuenta:
  - que todas las pulsaciones duran varios ciclos de reloj,
  - que entre pulsación y pulsación pasan varios ciclos de reloj sin tener ningún pulsador pulsado,
  - que para considerar activo el pulsar BLANCO ó NEGRO, hace **falta pulsar dicho pulsador y soltarlo.**

2.1 Dibuje el diagrama de estados de dicho sistema de apertura. Indique **CLARAMENTE** el **tipo de Máquina** utilizada en su diagrama (Moore o Mealy), los **estados**, las **entradas** y las **salidas.** *(20 puntos)*

$q_0 \equiv$ reposo
$q_1 \equiv$ blanco pulsado (1ª pulsación correcta)
$q_2 \equiv$ nada pulsado tras 1ª pulsación correcta
$q_3 \equiv$ blanco pulsado (2ª pulsación correcta)
$q_4 \equiv$ nada pulsado tras 2ª pulsación correcta
$q_5 \equiv$ negro pulsado (3ª pulsación correcta)
$q_6 \equiv$ secuencia correcta (OP = 1)
(nada pulsado)

$q_7 \equiv$ primera pulsación errónea
$q_8 \equiv$ nada pulsado tras 1ª pulsación errónea
$q_9 \equiv$ segunda pulsación errónea o pulsación cualquiera en camino erróneo
$q_{10} \equiv$ nada pulsado tras 2ª pulsación en camino erróneo
$q_{11} \equiv$ tercera pulsación errónea o pulsación cualquiera en camino ya erróneo
$q_{12} \equiv$ secuencia incorrecta (AL = 1)



Nota: suponemos que para resetear basta con pulsar 11 (sin necesidad de soltarlos)

SEPTIEMBRE 2007

PROBLEMA 1

1.1 Se pretende calcular el valor de la capacidad de entrada de un inversor CMOS. Para ello se conectan dos inversores CMOS idénticos (U1 y U2) en cascada y se miden los tiempos de propagación, t_{pHL} y t_{pLH} del primero de ellos (U1). Se obtiene que el mayor de dichos tiempos es de 3 ns.

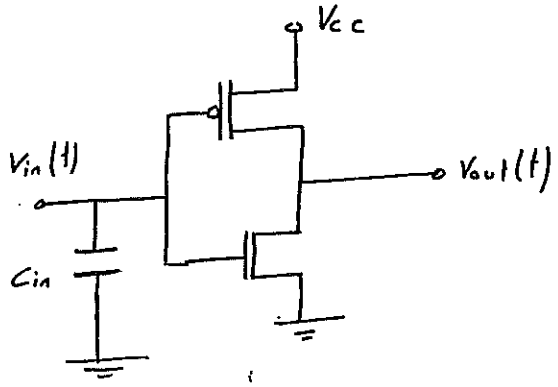
Considerando la tabla de valores que se adjunta, calcule el valor de la capacidad de entrada del inversor CMOS. **Justifique** su respuesta.

PMOS	NMOS
$R_{ON} = 200\ \Omega$	$R_{ON} = 100\ \Omega$
$R_{OFF} = 10\ M\Omega$	$R_{OFF} = 10\ M\Omega$

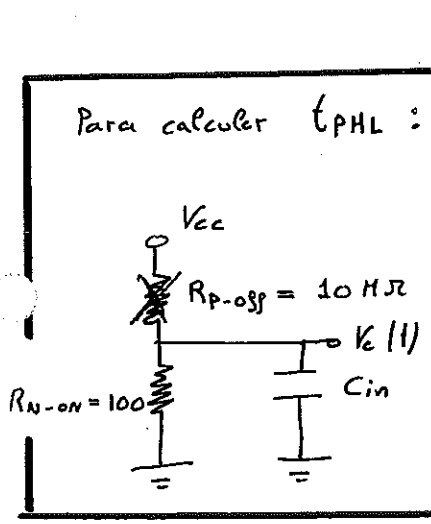
¿Cin?

Inversor CMOS :

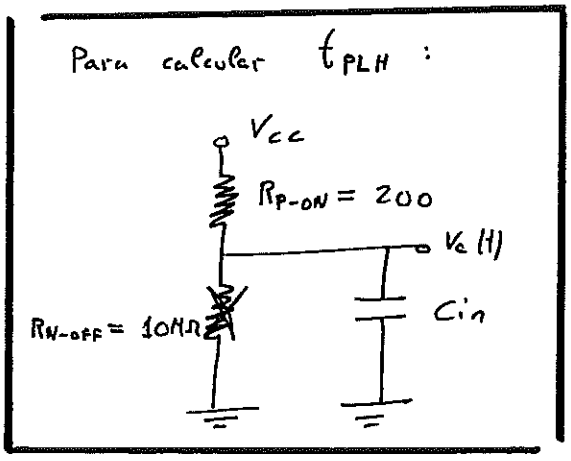
Al conectar dos en cascada, lo único que nos interesa del 2º es su capacidad Cin



Para calcular t_{PHL} :



Para calcular t_{PLH} :



Ambos son circuitos R-C, como siempre

Las resistencias R_{OFF} son despreciables en ambos casos (en un divisor de tensión, van a suponer un circuito abierto, casi).

A la vista de los circuitos, y recordando la gráfica de una exponencial negativa (más plana cuanto mayor sea $\tau \Rightarrow$ mayor tiempo de prop), el mayor de los tiempos de propagación será para el segundo circuito, t_{P-LH} (mayor resistencia, 200 Ω).

En ese circuito: $C_I = 0$

$C_F = V_{cc}$

$\tau = R_{P-ON} \cdot C_{IN}$

$$V_c(t) = (C_I - C_F)\, e^{\frac{-t}{R_{P-ON} \cdot C_{IN}}} + C_F = -V_{cc}\, e^{\frac{-t}{R_{P-ON} \cdot C_{IN}}} + V_{cc} =$$

$$= \boxed{V_{cc}\left(1 - e^{\frac{-t}{R_{P-ON} \cdot C_{IN}}}\right)}$$

Sabemos que el tiempo de propagación lo podemos modelar como el que tarda en reaccionar la salida (alcanzar $\frac{V_{cc}}{e}$) cuando cambia la entrada. Vamos a suponer un cambio a la entrada en $t = 0$ y por lo tanto el tiempo en que se llega a $\frac{V_{cc}}{2}$ es justo t_{p-LH}:

$$\cancel{V_{cc}}\left(1 - e^{\frac{-t_{p-LH}}{R_{P-ON} \cdot C_{IN}}}\right) = \frac{\cancel{V_{cc}}}{2}$$

$$e^{\frac{-t_{p-LH}}{R_{P-ON} \cdot C_{IN}}} = \frac{1}{2} \Rightarrow \boxed{C_{IN} = \frac{-t_{p-LH}}{R_{P-ON} \cdot \ln\left(\frac{1}{2}\right)}} = \frac{-3n}{200 \cdot \ln(0'5)} =$$

$$= \boxed{21'6 \text{ pF}}$$