VNIVERSITAT DE VALÈNCIA

UNIVERSITAT DE VALÈNCIA

TECNOLOGÍA Y DISEÑO DE SISTEMAS DIGITALES

Ingeniería Informática

Jose Antonio Boluda
Fernando Pardo

Prólogo

Los presentes apuntes nacen de la experiencia docente en la asignatura *Tecnología y Diseño de Sistemas Digitales* (3 créditos teóricos y 3 créditos prácticos) impartida en la **Ingeniería Informática** de la **Universitat de València**.

El objetivo de dicha asignatura es completar el conocimiento de la lógica digital impartida en *Estructura de Computadores I*, dotando simultáneamente al ingeniero informático de los conocimientos tecnológicos necesarios para afrontar con éxito el diseño de sistemas digitales. El temario de la signatura está diseñado con este objetivo, incluyendo los aspectos más modernos (como el capítulo de lógica programable) tan utilizados por los ingenieros de hoy en día.

En el temario inicialmente se complementa la formación en lógica digital (los estudiantes sólo conocen los sistemas combinacionales) con los sistemas secuenciales. Esto se hace en el **capítulo 1** con una descripción formal de las máquinas de estados y de los biestables que las implementan. Es aquí donde se introduce la distinción entre los sistemas secuenciales síncronos y asíncronos.

En el **capítulo 2** se expone, siguiendo dos ejemplos, la metodología para el análisis de sistemas secuenciales, tanto síncronos como asíncronos. En el **capítulo 3** se sigue el mismo método para exponer las técnicas de síntesis de sistemas secuenciales síncronos y asíncronos.

En el **capítulo 4** se introduce el lenguaje de descripción del hardware VHDL. Este método de descripción de sistemas digitales inicialmente estaba especficado para modelar y simular el comportamiento de hardware, pero actualmente este lenguaje ha pasado a ser el estándar de hecho hasta para incluso síntesis de hardware. En el capítulo se utiliza el lenguaje desde un punto de vista práctico para describir sistemas digitales sencillos, sin abordar las complejidades inherentes a este lenguaje de descripción del hardware.

De la misma manera que se diseñaban sistemas secuenciales con módulos MSI estándar, se puede afrontar el diseño de sistemas secuenciales con módulos MSI secuenciales. En el **capítulo 5** se presentan estos módulos secuenciales MSI y se muestra su utilización para diseño de sistemas secuenciales.

Una vez completada la descripción de la idealidad matemática, en el **capítulo 6** se afronta la necesidad de caracterizarlos mediante parámetros debido a su comportamiento no ideal en ciertos casos. Este comportamiento en el que la implementación física, bajo condiciones límite, no responde a la idealidad matemática ha de ser tenido en cuenta. Es importante conocer como los parámetros de los dispositivos lógicos los caracterizan y cual es su origen. De hecho, puede ser importante conocer de que manera una violación de un parámetro concreto influye en el correcto comportamiento de un sistema digital.

Con la idea de caracterizar los componentes lógicos discretos y conocer sus características se describen las dos grandes tecnologías que históricamente han dado lugar a los dos grupos de familias más populares en el diseño lógico: La lógica TTL y la lógica CMOS. Para ello en el **capítulo 7** se analizan los fundamentos de la familia lógica TTL

estándar a bajo nivel. Posteriormente, se revisan el resto de familias TTL avanzadas a las que ha dado lugar la lógica TTL estándar, comparándolas y caracterizándolas por su relación tiempo de propagación/potencia disipada. Análogamente se presenta la otra gran familia basada en transistores BJT: La lógica ECL.

En el **capítulo 8** se presenta el otro gran grupo de familias lógicas: Las familias lógicas CMOS. Para entender sus principales características se hace una rápida revisión de los transistores MOS en los que se basa esta lógica. En este capítulo se caracterizan las principales familias CMOS, de nuevo con su relación tiempo de propagación/potencia disipada. De manera adicional se presentan, justificando sus ventajas, las que están siendo las grandes líneas de desarrollo de lógica discreta de la actualidad: La lógica BiCMOS y la lógica de bajo voltaje.

Una vez se han descrito las principales tecnologías y familias lógicas es interesante introducir otra aproximación totalmente distinta para el diseño de sistemas digitales: La lógica programable. En el **capítulo 9** se describen los ASICs con sus ventajas y desventajas, y dentro de la taxonomía de los ASICs se presenta la lógica programable. En este tema se hace una descripción de las diversas tecnologías de programación y de las arquitecturas de los dispositivos programables.

Posteriormente, y como continuación lógica del capítulo anterior debido a que se utilizan las mismas tecnologías, se describe en el **capítulo 10** los diversos tipos de circuitos integrados de memoria.

Estos apuntes cubren un amplio espectro de materias a un nivel no excesivamente profundo. Para un desarrollo completo de los temas se recomienda la asistencia a las clases de teoría y prácticas, la resolución de los problemas propuestos en clase y el estudio de la bibliografía que se recomienda en el **apéndice A**.

Jose Antonio Boluda
Fernando Pardo

Jose.A.Boluda@uv.es
Fernando.Pardo@uv.es

Burjassot a 11 de febrero de 2004.
Universitat de València

Índice General

Índice de Tablas

Índice de Figuras

Parte I

Sistemas Secuenciales

Capítulo 1

Sistemas Secuenciales: Definiciones preliminares

1.1 Introducción

Sea el sistema digital de la figura 1.1. En principio el análisis del sistema no ofrece ninguna dificultad porque el único circuito que aparece es la puerta NOR cuya tabla de verdad es bien conocida. Observando con más detalle el circuito se ven realimentaciones que dificultan el análisis del mismo. Una **realimentación** aparece cuando la salida de un circuito se vuelve a conectar a la entrada del mismo circuito, tal como se muestra en la figura 1.1. La salida de las puertas NOR vuelve a servir de entrada a la puerta NOR vecina, de manera que la salida depende de la entrada R y S y también de la propia salida.

Es fácil advertir que la salida en un instante depende no sólo de la entrada al sistema (R y S), sino también de la salida en el momento anterior (X e Y), que a su vez dependerá de la salida en el instante anterior, etc. Para analizar el comportamiento del sistema digital hay que evaluar la tabla de verdad de las salidas Q y P. Esto se consigue aplicando la tabla de verdad en ambas puertas NOR (tabla 1.1).

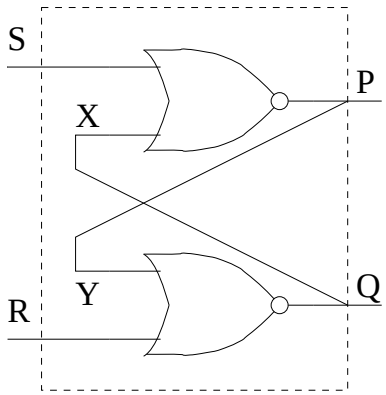


Figura 1.1: *Sistema con realimentación*

X/Y	S/R = 00	01	11	10
Q/P = 00	11	01	**00**	10
01	**01**	**01**	00	00
11	00	00	00	00
10	**10**	00	00	**10**

Tabla 1.1: *Tabla de verdad del sistema realimentado*

En el circuito se observa que se cumple: $Q = X$ y $P = Y$. Por tanto se observa que las únicas combinaciones posibles o estados estables del circuito serán los que cumplan la condición anterior. Éstos son las salidas Q/P de la tabla marcados con negrita. En cualquier otro caso el sistema evolucionará rápidamente hasta que el sistema sea estable dada la combinación de entrada. Si hay mas de un estado estable (caso en que $RS = 00$) podría evolucionar a cualquiera de ellos, dependiendo en este caso del estado anterior de las salidas.

La mejor manera de entender esto es con un ejemplo. Supóngase que se acaba de conectar el circuito y en ese instante, antes de que las puertas evalúen el estado de sus entradas, ambas salidas valen 0 con lo que $XY = 00$. Un instante después se hace $SR = 10$, con lo que unos cuantos nanosegundos después, (el tiempo de propagación de la señal en la puerta), la salida cambiará al valor que nos indica la tabla para dichas entradas, con lo que $QP = 10$, o equivalentemente $XY = 10$. Unos cuantos nanosegundos después se vuelven a evaluar las funciones NOR. Esta vez ya no hay cambio en la salida porque de nuevo $QP = 10$, con lo que $XY = 10$ y el sistema se dice que está en un **estado estable**.

Siguiendo el cronograma de la figura 1.2 que describe el comportamiento del sistema se puede observar que la salida no depende sólo de las entradas. Para un misma entrada SR=00 se tienen dos salidas distintas en dos intervalos de tiempo distinto. Por tanto no se trata de un sistema combinacional en el que las salidas dependan de las entradas. Las salidas también dependen de las salidas anteriores en el tiempo, es decir es necesario introducir conceptos como *estado del sistema*. La salida dependerá tanto de las entradas, como del estado del sistema en ese instante. En este caso hay 3 estados estables del sistema, que corresponden a las salidas 00, 01, 10. La salida 11 también es posible pero éste no es un estado estable.

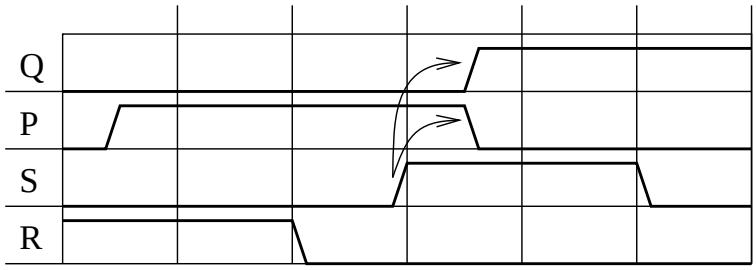


Figura 1.2: *Evolución del sistema realimentado. Inicialmente* $XY = 00$

Es necesario establecer un formalismo que tenga en cuenta, no sólo la dependencia con las *entradas presentes*, sino además la dependencia con la configuración del sistema en un instante dado [NNCI96], [GA95].

1.2 Definición de sistema secuencial

Se llamará **Sistema secuencial**, (en adelante S.S.), al conjunto formado por la quíntupla S.S.=$\{I, Q, Z, \delta, \lambda\}$ donde:

- $I = \{i_0, i_2...i_{n-1}\}$ es el conjunto de vectores de entrada (o alfabeto) del sistema. Por defecto, y salvo restricciones, habrán 2^n vectores, donde n es el número de entradas binarias. Pueden no estar permitidas todas las combinaciones binarias posibles de las entradas.
- $Q = \{q_0, q_2...q_{p-1}\}$ es el conjunto de p estados del S.S., es decir, el número de configuraciones internas posibles del S.S.
- $Z = \{z_0, z_2...z_{m-1}\}$ es el conjunto de m salidas del sistema. No tienen porqué darse todas las combinaciones binarias posibles en el caso de un vector de bits.
- δ es la **función de transición** o **función de estado siguiente** que calcula el estado siguiente $Q(t+1)$. Esta función depende del **estado presente** (estado en t) y de la entrada, de manera que se cumple:

$$\begin{aligned} &\delta: \ I \text{ x } Q \longrightarrow Q \\ &(i_i, q_j(t)) \longrightarrow q_{ij}(t+1) = \delta(i_i, q_j(t)) \end{aligned} \tag{1.1}$$

- λ es la función de salida para el estado presente. Esta salida dependerá del estado presente y de la entrada:

$$\begin{aligned} &\lambda: \ I \text{ x } Q \longrightarrow Z \\ &(i_i, q_j(t)) \longrightarrow z_{ij} = \lambda(i_i, q_j(t)) \end{aligned} \tag{1.2}$$

Para tener un S.S. totalmente especificado hay que tener perfectamente definidos tanto los conjuntos I, Q y Z, como las funciones δ y λ.

En la figura 1.3 se muestra cuál sería el esquema general de un S.S. Se puede observar cómo aparece de nuevo la realimentación vista en el caso anterior. El estado siguiente del sistema dependerá de la entrada y del estado presente (que se almacena en el módulo MEMORIA). La salida depende también del estado presente y de la entrada.

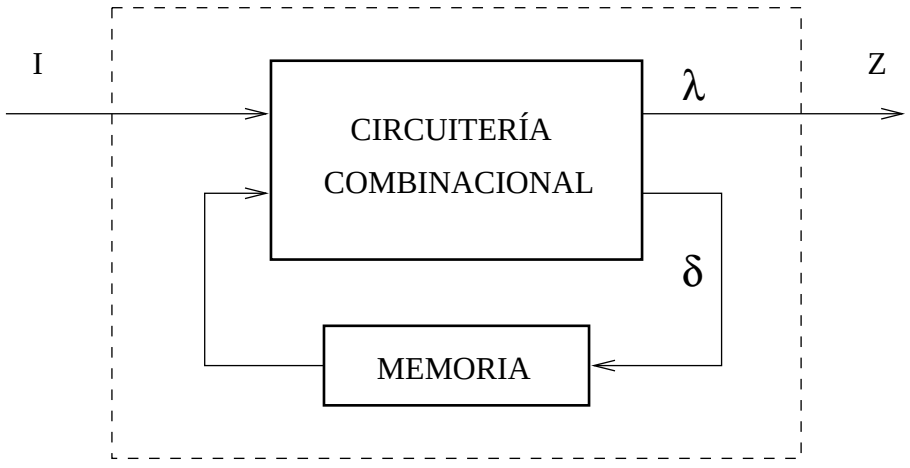


Figura 1.3: *Esquema general de un S.S.*

	i_0	i_1	...	i_{n-1}
q_0	$\delta(i_0,q_0)/\lambda(i_0,q_0)$	$\delta(i_1,q_0)/\lambda(i_1,q_0)$	...	$\delta(i_{n-1},q_0)/\lambda(i_{n-1},q_0)$
q_1	$\delta(i_0,q_1)/\lambda(i_0,q_1)$	$\delta(i_1,q_1)/\lambda(i_1,q_1)$	...	$\delta(i_{n-1},q_1)/\lambda(i_{n-1},q_1)$
...	...	...	...	...
q_{p-1}	$\delta(i_0,q_{p-1})/\lambda(i_0,q_{p-1})$	$\delta(i_1,q_{p-1})/\lambda(i_1,q_{p-1})$	...	$\delta(i_{n-1},q_{p-1})/\lambda(i_{n-1},q_{p-1})$

Tabla 1.2: *Representación tabular de un sistema secuencial genérico*

La idea más importante que cabe extraer de la definición de un S.S. es que para determinar su evolución en un instante no basta con conocer la quíntupla $\{I, Q, Z, \delta, \lambda\}$, sino que además es preciso fijar unas condiciones iniciales o **estado inicial**. Tanto la salida como el estado siguiente dependen del estado presente, que a su vez dependía del estado anterior, de manera que sin un estado inicial no se puede establecer la evolución de la máquina.

Para evitar este problema los circuitos secuenciales disponen de mecanismos que ponen la máquina en un estado inicial conocido. Estos mecanismos suelen consistir en fijar a cero todos los elementos de memoria que codifican los estados (**reset**), en fijarlos todos a uno (**preset**), o fijarlos en cualquier estado inicial conocido.

1.3 Representación de los Sistemas Secuenciales

A la hora de representar un S.S. se pretende especificar de manera completa la quíntupla $\{I, Q, Z, \delta, \lambda\}$ de manera que se pueda seguir la evolución del S.S. a partir de una cierta configuración inicial conocida.

1.3.1 Representación Tabular

Contiene una descripción completa del circuito secuencial organizada en una tabla de doble entrada. En la parte superior está el conjunto I de entradas posibles y en la izquierda el conjunto de estados Q. En las casillas intersección se representa el valor del estado siguiente y la salida para cada valor de entrada y estado presente. La tabla debe de estar completa para tener completamente especificado el S.S.

En la tabla 1.2 se puede observar una tabla de transición general y en la tabla 1.3 se puede ver un ejemplo de una máquina concreta. En el caso en que se tenga una máquina en la que la salida dependa solamente del estado presente, se puede poner una columna más a la derecha con el valor de la salida para cada estado. De esta manera se evita información redundante.

Es interesante hacer una simulación mental del caso particular propuesto y observar la evolución de los estados y de las salidas en el caso del ejemplo de la tabla 1.3. Para ello habría que suponer un estado inicial (por ejemplo q_0) y una secuencia en el único bit de entrada.

	0	1
q_0	$q_0/0$	$q_2/0$
q_1	$q_3/1$	$q_1/1$
q_2	$q_1/0$	$q_3/1$
q_3	$q_0/1$	$q_2/0$

Tabla 1.3: *Ejemplo de representación tabular de una máquina de estados*

1.3.2 Diagrama de Transiciones

La misma información que en una tabla de transiciones se puede tener en un grafo. En este caso cada nodo representa un estado y los arcos representan las transiciones. De cada nodo saldrán tantos arcos como palabras del alfabeto de entrada y cada arco llevará inscrito la condición en la entrada para que se produzca esta transición. Los arcos llevarán inscritos además el nuevo valor que tomará la salida. Como ejemplo se puede ver el S.S. de la tabla 1.3 representado en el grafo de la figura 1.4.

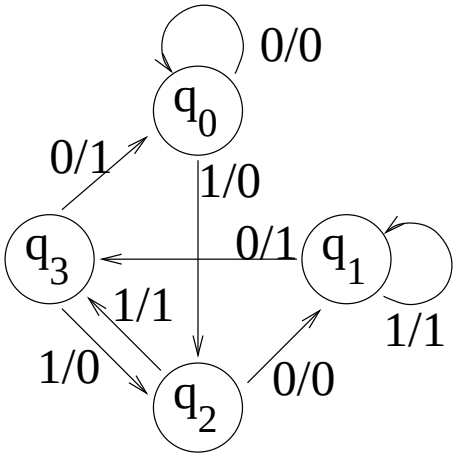


Figura 1.4: *Grafo equivalente al S.S. de la tabla 1.3*

En los arcos se da la condición de la entrada y después se indica el valor que tomará la salida. Así por ejemplo, cuando esté en el estado q_2, el estado futuro será el q_3 si la entrada vale 1. Por el contrario, si la entrada vale 0, el estado futuro será q_1.

En el caso de que se tenga una máquina de Moore, que es cuando la salida sólo depende del estado presente y no de la entrada, se podrá poner la salida dentro del nodo que representa el estado. De esta manera los arcos sólo indicarán la condición de la entrada para que se produzca esa transición.

1.4 Clasificación de los sistemas secuenciales

Los S.S. también son llamados máquinas de estados finitos o simplemente máquinas de estados. No es el objetivo de esta asignatura entrar en el formalismo matemático de las gramáticas y autómatas formales, conceptos que se estudiarán en la asignatura

Teoría de Autómatas y Lenguajes Formales independientemente de su uso en los sistemas digitales. Simplemente se tratarán en este tema los conceptos útiles desde el punto de vista hardware.

1.4.1 Máquina de Mealy y Máquina de Moore

Llamaremos **máquina de Mealy** a un S.S. donde los vectores de salida Z dependen tanto de las entradas I, como de los estados presentes Q. Es decir es el caso más general en el que $z_i = \lambda(I_j, q_s)$.

Por contra llamaremos **máquina de Moore** a un S.S. donde los vectores de salida Z sólo dependen del estado presente Q. Es decir en este caso $z_i = \lambda(q_j)$.

Las máquinas de Moore tienen la ventaja de que la salida sólo cambiará con los ciclos de reloj (un cierto tiempo después). Esto hace más fácil sincronizar la salida con otros circuitos digitales. Sin embargo, al depender la salida sólo de variables internas en la máquina de Moore, ésta puede tener más estados que su equivalente de Mealy.

Todo autómata de Moore tiene un autómata de Mealy equivalente y viceversa. Se puede probar que una máquina de Moore tiene como mínimo los estados de su equivalente de Mealy, (teniendo habitualmente más). La metodología para pasar de una máquina a otra equivalente no va a ser objeto de estudio en el presente curso. En los capítulos posteriores 2 y 3, se verán ejemplos de ambos tipos de máquinas.

1.4.2 Sistemas Secuenciales Síncronos y Asíncronos

Los cambios de estado en un S.S. son debidos a que la función estado siguiente δ genera un estado distinto del estado en ese momento presente en la memoria. Si este cambio de estado viene sincronizado por una señal global al S.S. se dice que el S.S. es síncrono. Esta señal suele ser periódica para así poder predecir el momento en el que se producirá el cambio de estado, y como la salida además de depender de la entrada depende del estado presente, también se puede controlar el momento en que se produce el cambio en las salidas.

Esta señal de sincronismo suele denominarse de reloj (clk). El parámetro que suele indicarse de un reloj es la frecuencia (inversa del periodo). Cuanto mayor sea la frecuencia de funcionamiento de un S.S. mayor será la rapidez con la que el S.S. evolucione, y por tanto, responderá a las entradas. El diseño de S.S. síncronos está muy extendido para unidades de control, CPUs y cualquier S.S. complejo. El motivo es que la señal de sincronismo externa simplifica el análisis de la propagación de las señales en el circuito, la predicción de los retrasos y en general todo el diseño del sistema digital.

El uso de sistemas secuenciales asíncronos es más delicado. El hecho de que no haya una señal que marque la evolución del sistema puede hacer que en la transición desde una configuración a otra se produzca un estado interno que sea estable con las entradas presentes. Esto puede ocurrir porque la función δ se genera con lógica que tiene un cierto retraso, si estos retrasos son diferentes entre los distintos bits, se puede generar la aparición de **carreras críticas**. Esto se estudiará con detalle en la síntesis de S.S. asíncronos en la sección 3.3. Los sistemas secuenciales asíncronos tienen como ventaja su rápida evolución (no hay que esperar la señal de sincronismo) nada más cambien las entradas hasta el próximo estado estable.

1.5 Los Biestables como Elementos de Memoria

El sistema con realimentación que ha sido analizado inicialmente en la figura 1.1 presenta la propiedad de *almacenar* la información. Para comprobar esta afirmación se va a analizar el comportamiento considerando sólo la salida Q como salida del S.S.

Si la entrada es $SR = 00$ el estado siguiente es el mismo que el estado presente, es decir, permanecerá invariable el 0 o 1 que se tenía en la salida. Si la entrada es $SR = 10$ el estado del circuito pasa a ser $Q = 1$ independientemente del estado anterior. Es la llamada puesta a uno o **set** del circuito, y es fácil recordar que se produce al activar la señal S de **S**et. Por el contrario la entrada $SR = 01$ es la puesta a 0 o **reset** del circuito, y se activa al poner un 1 en la señal R de **R**eset.

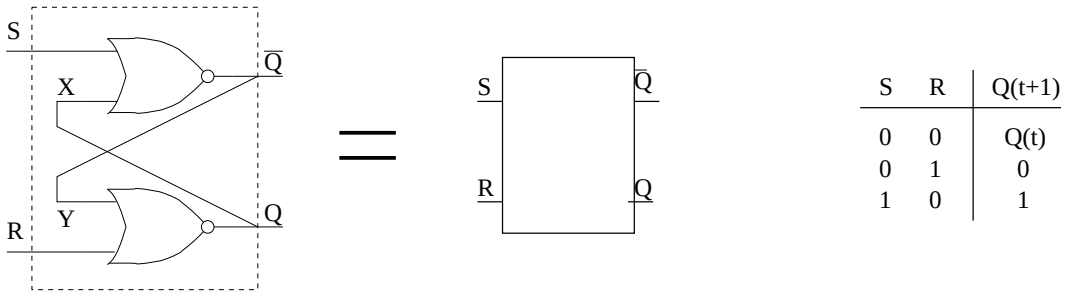
Una vez se deja el circuito con la salida a 1 o 0 y se desactivan las señales de control, es decir, se hace $SR = 00$, el *estado* permanece fijado hasta que se active de nuevo S o R. El circuito *recuerda* si la última acción fue una puesta a uno o una puesta a cero.

Este circuito con dos estados posibles, 0 y 1, se llama **biestable**. La entrada $SR = 11$ no se permite debido a que si después de esta entrada se aplica una entrada $SR = 00$, se produce una oscilación en la que el estado no es predecible. Por tanto nunca se producirá la pareja de salida $PQ = 00$ siendo siempre $P = \overline{Q}$.

1.5.1 Biestables asíncronos. El biestable SR

A partir de los circuitos realimentados se construyen los circuitos con dos estados estables posibles o **biestables**. La evolución del biestable se produce una vez cambia la señal de control, sin que exista otra señal de control (*reloj*) que marque la evolución del sistema.

La figura 1.5 corresponde a la implementación del biestable SR a partir de puertas lógicas, a su símbolo, y a su comportamiento lógico. La tabla que se adjunta representa el estado siguiente $Q(t+1)$ en función de las entradas y del estado presente $Q(t)$ (función δ).



S	R	Q(t+1)
0	0	Q(t)
0	1	0
1	0	1

Figura 1.5: *Implementación del biestable asíncrono SR, símbolo y función estado siguiente*

La función estado siguiente δ, también llamada **ecuación característica del biestable**, se puede obtener con un mapa de Karnaugh de la manera habitual, siendo para el biestable SR:

$$Q(t+1) = S + Q(t) \cdot \overline{R} \tag{1.3}$$

Estos biestables son utilizados para el diseño de S.S. asíncronos, teniendo siempre en cuenta la desventaja de que si la entrada es 11 y cambia a 00 el estado siguiente provoca una oscilación de resultados imprevisibles. Este problema hace que los biestables SR sean utilizados en aplicaciones muy concretas, como son los circuitos antirebote.

De manera análoga al diseño del biestable SR de la figura 1.3 se puede construir un biestable SR substituyendo las puertas NOR por puertas NAND. En este caso las entradas SR serán activas a nivel bajo, con lo que este biestable se suele llamar biestable $\overline{SR}$.

1.5.2 Biestables sensibles a nivel

El biestable anterior es asíncrono porque la evolución del sistema no viene sincronizada por ninguna señal externa o reloj. Por tanto la salida evolucionará hasta llegar al estado estable compatible con las entradas. Cada vez que haya un cambio de entrada el estado evolucionará sin más dilación.

Cuando se tiene un S.S. con bastantes biestables puede ser realmente complejo analizar la evolución de las señales con retrasos dentro de una circuitería asíncrona. Un error en la **temporización** prevista del circuito puede hacer que se violen los tiempos que deben permanecer los valores en los nodos del circuito. En definitiva, esto puede acarrear un mal funcionamiento del S.S.

Para evitar esto se han diseñado los biestables sensibles a nivel. Estos biestables tienen una señal adicional que habilitará el que el biestable responda a las entradas. Cuando esta señal esté en estado activo el biestable evolucionará pero si no lo está el biestable permanecerá fijado en el estado presente aunque cambien las entradas. Estos biestables sensibles a nivel también se suelen llamar *latches* o cerrojos.

El biestable SR tipo *latch*

En la figura 1.6 se puede observar un biestable SR sensible a nivel. Cuando la señal de enable está a nivel alto el biestable evolucionará pero si está a nivel bajo el biestable permanecerá fijado en el estado anterior. Esto es lo que se llama un biestable **cerrojo** o *latch* con compuertas, es decir hay una señal que permite o bien la evolución del sistema, o bien la *cierra*. El comportamiento de este biestable es idéntico al SR asíncrono teniendo en cuenta que el sistema sólo evoluciona cuando la señal de habilitación lo permite.

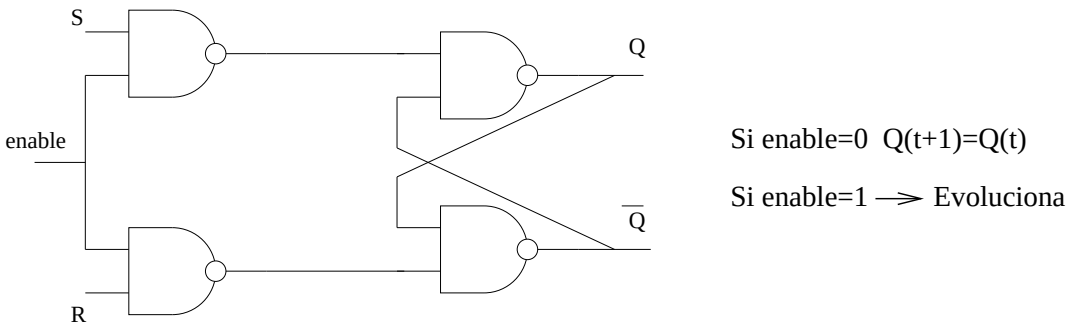


Figura 1.6: *Biestable SR síncrono sensible a nivel*

El biestable D tipo *latch*

Otro biestable sensible a nivel que se define es el biestable D que se construye a partir del biestable SR sensible a nivel tal como se indica en la figura 1.7. La ecuación característica que se obtiene trivialmente es:

$$Q(t+1) = D \tag{1.4}$$

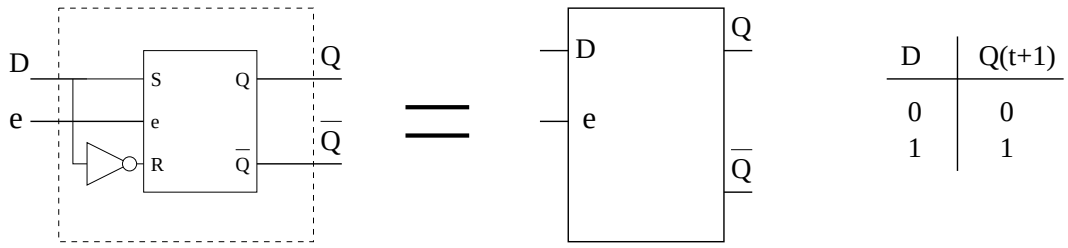


Figura 1.7: *Implementación del biestable D sensible a nivel*

Este biestable es de gran comodidad en su uso para diseño ya que para cambiar el biestable al estado deseado sólo se debe de poner en la entrada D al valor deseado. Esto no permite posteriores simplificaciones para las funciones de excitación de los biestables, con lo cual la comodidad en el diseño viene contrarrestada por la mayor complejidad de las funciones de excitación. Una ventaja de este biestable es que posee una sola entrada, con lo que habrá que realizar una sola función de excitación por biestable, al contrario que en los biestables SR y JK (este último se analizará en la siguiente sección).

1.5.3 Biestables síncronos sensibles a flanco: Biestables maestro esclavo

Con biestables tipo *latch* se consigue sincronizar la evolución de los S.S. de manera que sólo el sistema evolucionará cuando esté habilitado. Si las entradas cambian rápidamente mientras la señal de sincronismo está habilitada la evolución puede ser imprevisible. El sistema puede no estar estabilizado para una combinación de entrada cuando cambie de nuevo la entrada.

Para solucionar estos problemas de ambigüedad y realizar un sincronismo mejor definido se utilizan los biestables maestro-esclavo (biestable MS en adelante). En los biestables MS las salidas $Q(t)$ y $\overline{Q(t)}$ sólo cambian durante un flanco de reloj (o bien ascendente o bien descendente). En el biestable MS tipo SR las salidas cambian en el flanco, pero el biestable es sensible a la activación de las señales de control activas por pulso. Sin embargo el biestable D MS es sólo activo durante el flanco de reloj. Esto no significa que el biestable sólo evoluciona durante el flanco (de duración ínfima), sino que en ese instante es cuando se *capturan* las entradas evolucionando el sistema hasta su estado estable con la configuración de entrada que había en el momento de la captura. En algunos textos a los biestables MS se les denomina biestables síncronos y también *flip-flops*.

Se define así el **tiempo de establecimiento** o de *set-up* como el tiempo que debe permanecer la entrada estabilizada en los biestables antes del flanco de reloj. De la misma manera se define como **tiempo de mantenimiento** o de *hold* el tiempo que

se debe mantener la entrada estabilizada después del flanco de reloj. Si no se respetan estos tiempos el biestable puede tener una evolución imprevisible y el S.S. no funcionará adecuadamente.

Biestables SR maestro-esclavo

Este biestable de la figura 1.8 es del tipo *flip-flop* en el que las salidas cambian en el flanco de bajada. Este biestable se construye a partir de biestables sensibles a nivel alto tal como se muestra en la figura 1.8. Cabe hacer notar que las salidas sólo cambian en el flanco de bajada, pero la evolución será la indicada por el valor de las señales S y R durante el nivel alto de la señal de reloj.

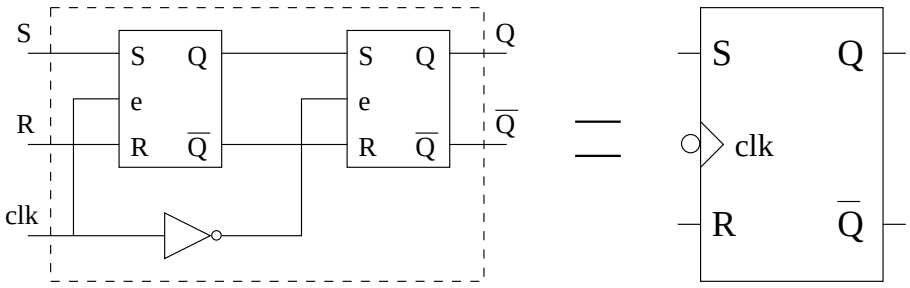


Figura 1.8: *Biestable SR síncrono sensible a flanco*

Este biestable tendrá un comportamiento idéntico (en cuanto a la función estado siguiente) a sus equivalentes asíncrono y sensible a nivel. Se deja como ejercicio al estudiante simular el comportamiento del biestable de la figura 1.8 y comprobar como sólo en el flanco de bajada se produce la evolución del sistema, "capturada" mientras la señal *clk* está a nivel alto.

Biestable D síncrono

Se puede implementar un biestable MS de tipo D a partir de biestables D sensibles a nivel, tal como se muestra en la figura 1.9. Este biestable es sensible a flanco de bajada con la misma tabla de verdad o función estado siguiente que un biestable D asíncrono o sensible a nivel.

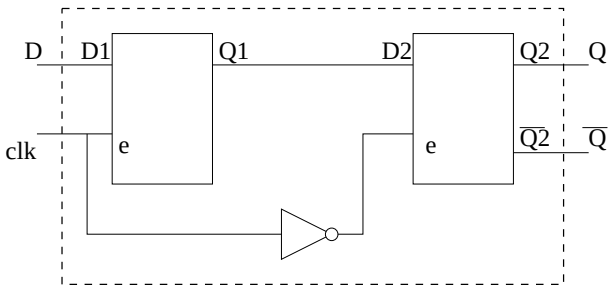


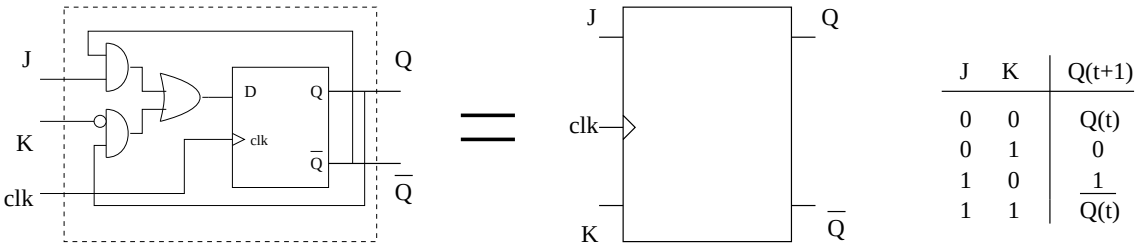
Figura 1.9: *Biestable D síncrono sensible a flanco*

En el esquema de la figura 1.9 se utilizan dos biestables sensibles a nivel, dependiendo el segundo de la evolución del primero. Es por esto que el primero se llama **biestable maestro** y el segundo **biestable esclavo**.

Biestable JK síncrono

De manera análoga se puede construir otro biestable síncrono a partir del biestable D, tal como se muestra en la figura 1.10. El comportamiento es similar al SR, sirviendo la J para la puesta a 1 y la K para la puesta a 0. La única diferencia con el SR es que este biestable tendrá la entrada 11 en su alfabeto. A partir de su esquema lógico es posible obtener la función estado siguiente de este S.S. o ecuación característica del biestable:

$$Q(t+1) = \overline{Q(t)} \cdot J + Q(t)\overline{K} \tag{1.5}$$



J	K	Q(t+1)
0	0	Q(t)
0	1	0
1	0	1
1	1	$\overline{Q(t)}$

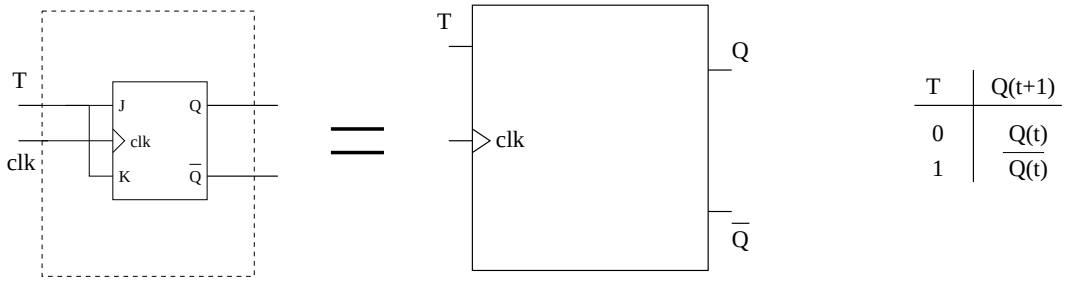
Figura 1.10: *Implementación del biestable síncrono JK, símbolo y tabla de verdad*

Este biestable se utiliza mucho porque para ponerlo a 1 o a 0 hay varias posibilidades. Por ejemplo, si el biestable está a 0 y se quiere poner a 1 se debería poner la combinación de entrada $JK = 10$ o bien la 11. Es decir la combinación de entrada será la 1ϕ, que equivale a decir que no importa cual sea el valor de K. Esta introducción de indeterminaciones será muy útil después para la simplificación de las funciones lógicas que van a excitar a los biestables, tal como se mostrará en el capítulo 3.

El biestable T síncrono

El último biestable estándar es el biestable T. El comportamiento de este biestable es útil básicamente para el diseño de contadores. Se construye a partir de un biestable JK, tal como se muestra en la figura 1.11 y tiene al igual que el D una sola entrada. La ecuación característica es:

$$Q(t+1) = T \cdot \overline{Q(t)} + \overline{T} \cdot Q(t) = T \oplus Q(t) \tag{1.6}$$



T	Q(t+1)
0	Q(t)
1	$\overline{Q(t)}$

Figura 1.11: *Implementación del biestable síncrono T, símbolo y tabla de verdad*

$Q(t)$	$Q(t+1)$	S	R
0	0	0	ϕ
0	1	1	0
1	0	0	1
1	1	ϕ	0

Tabla 1.4: *Tabla de excitación del biestable SR*

$Q(t)$	$Q(t+1)$	J	K
0	0	0	ϕ
0	1	1	ϕ
1	0	ϕ	1
1	1	ϕ	0

Tabla 1.5: *Tabla de excitación del biestable JK*

Los esquemas de las figuras 1.9 y 1.10 son dos ejemplos de las múltiples configuraciones existentes para implementar biestables D y JK maestro esclavo. Los biestables MS de tipo T se construyen a partir de los biestables JK, tal como se muestra en la figura 1.11.

Tablas de excitación de los biestables

A partir de las ecuaciones características de los biestables, es interesante construir unas tablas que indiquen qué valores se deben poner en las entradas de los biestables, en función del estado presente, para poner el biestable al valor deseado. Estas son las **tablas de excitación** y tendrán una gran utilidad para la síntesis de sistemas secuenciales, tal como se mostrará en en capítulo 3.

La tabla de excitación del biestable SR es la que se muestra en la tabla 1.4. En este caso sólo se introducen indeterminaciones cuando se pretende que $Q(t+1) = Q(t)$. Cabe hacer notar como en este biestable no se permite la entrada 11, tal como se puede observar en la tabla 1.4. Sin embargo, el biestable JK presenta un comportamiento predecible para la entrada 11 y por tanto esta entrada se permite. Esto hace que se puedan presentar más indeterminaciones, tal como se muestra en la tabla 1.5, que serán muy útiles para simplificar mediante mapas de Karnaugh las funciones de excitación.

Las tablas de excitación de los biestables D y T, mostradas en las tablas 1.6 y 1.7, no presentan la introducción de indeterminaciones, lo que no permitirá simplificar las funciones de excitación en los mapas de Karnaugh. Sin embargo en estos biestables sólo hay que generar una función de entrada, al contrario que en los biestables JK y SR donde hay que controlar dos entradas.

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

Tabla 1.6: *Tabla de excitación del biestable D*

$Q(t)$	$Q(t+1)$	T
0	0	0
0	1	1
1	0	1
1	1	0

Tabla 1.7: *Tabla de excitación del biestable T*

1.6 Conclusiones

El hecho de que existan realimentaciones en un S.S. hace que el sistema no sólo dependa de la entrada en ese instante, tal como ocurría en los sistemas combinacionales, sino que haya una dependencia con las entradas pasadas y la evolución del sistema. Esta dependencia de las entradas anteriores del circuito plantea la necesidad de la aparición de variables internas del circuito o estados. A partir de estas ideas se ha definido el formalismo de sistema secuencial o máquina de estados.

Se puede realizar una clasificación de los S.S. atendiendo a la dependencia de la salida de la entrada y del estado presente o sólo de los estados (máquina de Mealy y máquina de Moore). Siempre hay una máquina de Mealy equivalente a una máquina de Moore y viceversa. La ventaja de las máquinas de Moore es que las salidas evolucionan síncronamente con los estados, siendo la desventaja que como mínimo usan la misma lógica que su equivalente de Mealy y habitualmente más.

Otra clasificación de los S.S. atiende a la presencia de una señal de sincronismo que marque la evolución del sistema. De esta manera los sistemas secuenciales síncronos evolucionan controlados por una señal de reloj lo que facilita su diseño. Los sistemas secuenciales asíncronos serán más complejos de diseñar, teniendo como ventaja que responden rápidamente a las entradas debido a que el sistema no debe esperar la señal de sincronismo para evolucionar.

El almacenamiento de los estados en los S.S. se realiza en las realimentaciones. Para simplificar el diseño de S.S. se han definido unas unidades mínimas de almacenamiento de información que se llaman biestables. Hay diversos tipos de biestables, presentando cada tipo una serie de ventajas y desventajas lo cual hace que la elección de cada tipo dependa de las características del problema. En el apéndice A se pueden encontrar referencias para ampliar este tema de conceptos y definiciones preliminares.

Capítulo 2

Análisis de Sistemas Secuenciales

2.1 Introducción

El objetivo de realizar un análisis del S. S. consiste en tener totalmente especificadas las cinco cantidades de que consta para poder prever su comportamiento. Es decir hay que conocer totalmente:

1. El conjunto de vectores de entrada (o alfabeto) del sistema $I = \{I_1, I_2...I_n\}$.
2. El conjunto de estados del S. S. $Q = \{q_1, q_2...q_p\}$.
3. El conjunto de vectores de salida del sistema $Z = \{z_1, z_2...z_m\}$.
4. La función de transición o función estado siguiente δ.
5. La función de salida λ.

La especificación del circuito vendrá generalmente en forma de esquema o lista de nodos. A partir de esta entrada se deberá hallar la quíntupla anterior.

Generalmente, y salvo que no se especifique lo contrario, el conjunto de vectores de entrada será el formado por todas las combinaciones binarias posibles de las entradas. Es decir, si hay n entradas, 2^n vectores. Sin embargo los vectores de salida habitualmente no serán los formados por todas las combinaciones posibles de las salidas. Se deberá de explícitamente calcular λ y, en función de las entradas posibles y los estados presentes, calcular la salida.

La complejidad estriba en hallar el conjunto de estados e interpretar la evolución del sistema. Se va a proponer una metodología para, a partir de un esquema del circuito, hallar estas cantidades. Esta metodología será ligeramente distinta entre los sistemas con biestables y los que sólo tienen puertas realimentadas. El motivo es que en los biestables deben aplicarse las ecuaciones características para hallar la función estado siguiente, por el contrario con lógica combinacional el método será directo. Se va a ver la aplicación de este método en dos ejemplos paralelos, uno con biestables y el otro sólo con lógica combinacional con realimentaciones. Se recomienda que se consulte [NNCI96] y [GA95] como bibliografía para el análisis de sistemas secuenciales.

No debe perderse de vista que el objetivo de conocer las cinco cantidades que definen un S. S. tiene como misión realizar una interpretación de la *funcionalidad* del S. S. Para ello habrá que conocer el significado de las entradas si estas vienen de un sistema digital

en el que el S. S. está integrado. Análogamente habrá que conocer las señales de entrada conectadas a la salida del S. S.

2.2 Metodología

2.2.1 Análisis del circuito

En las figuras 2.1 y 2.2 se muestran dos circuitos secuenciales que se proponen para ser analizados. El de la figura 2.1 es un circuito síncrono ya que está realizado con biestables de tipo JK síncronos maestro-esclavo (sensibles a flanco). Por el contrario el de la figura 2.2 es un circuito realizado con puertas lógicas realimentadas y por tanto asíncrono.

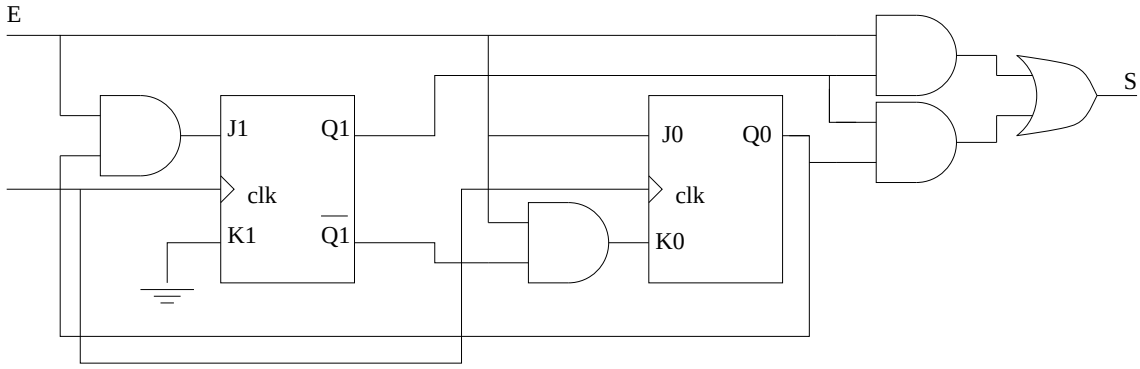


Figura 2.1: *Sistema secuencial síncrono ejemplo para el análisis*

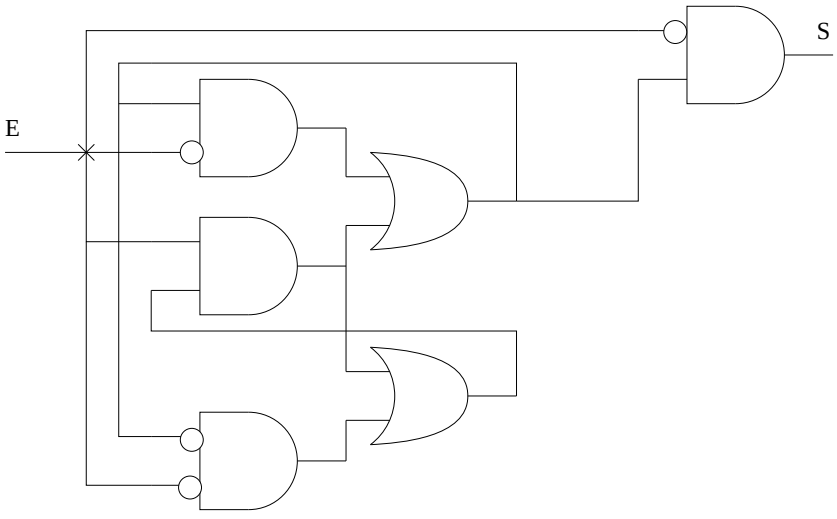


Figura 2.2: *Sistema secuencial asíncrono ejemplo para el análisis*

Ambos tendrán como alfabeto de entrada, al tener un sólo bit de entrada, el conjunto $I = \{0, 1\}$. De la misma manera ambos circuitos tienen una sola salida, con lo que a priori $Z = \{0, 1\}$, pero para ver si ambas salidas son posibles habrá que calcular la función de salida λ. Análogamente habrá que calcular la función estado siguiente o δ en ambos circuitos.

2.2.2 Obtención de la función estado siguiente

Una vez determinado el alfabeto de entrada del S.S. y el tipo de sistema del que se trata (síncrono o asíncrono), hay que hallar la función estado siguiente $Q(t+1) = \delta(Q(t), I)$.

En el caso del ejemplo de la figura 2.1 primero hay que obtener, a partir del esquema, las ecuaciones de excitación de los biestables:

$$\begin{cases} J_1 = Q_0 E \\ K_1 = 0 \end{cases} \qquad \begin{cases} J_0 = E \\ K_0 = \overline{Q_1} E \end{cases} \tag{2.1}$$

A partir de estas ecuaciones, y aplicando la función característica de los biestables JK expresada en la ecuación 1.5, se obtiene la función estado siguiente para cada biestable:

$$\begin{cases} Q_1(t+1) = \overline{Q_1(t)} Q_0(t) E + Q_1(t) \\ Q_0(t+1) = \overline{Q_0(t)} E + Q_1(t) Q_0(t) + Q_0(t) \overline{E} \end{cases} \tag{2.2}$$

La forma de obtener la función δ es la misma siempre que aparezcan biestables, independientemente de si son biestables asíncronos, síncronos sensibles a nivel o a flanco. Sin embargo cuando se trata de puertas lógicas realimentadas el método es ligeramente distinto. En primer lugar no se pueden aplicar las ecuaciones características de los biestables por no haberlos. Simplemente hay que identificar las **realimentaciones** del circuito que es donde tomaremos las variables que codifican el estado presente.

En el caso del circuito asíncrono propuesto en la figura 2.2 aparecen 2 realimentaciones, tal como están marcadas en la figura 2.3. Es ahí donde se van a tomar las variables $Q_1(t)$ y $Q_0(t)$ que identificarán el estado presente.

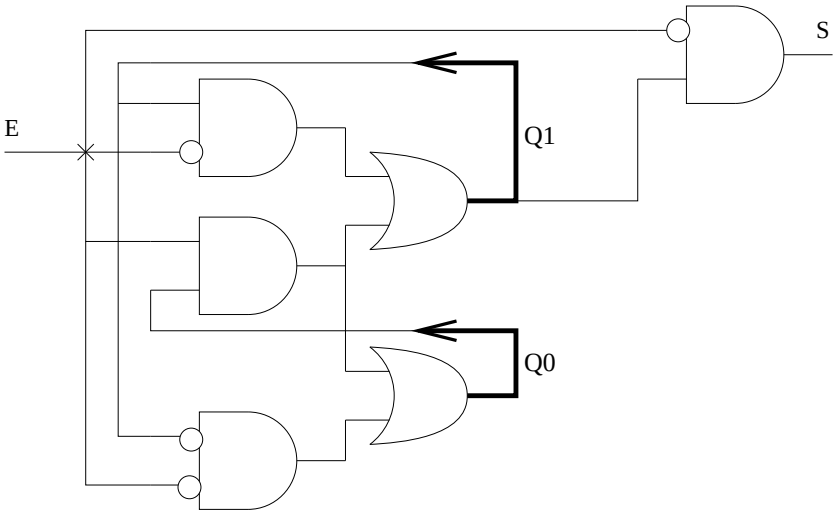


Figura 2.3: *Identificación de las variables internas (estado) del circuito asíncrono 2.2*

A partir de estas variables, y teniendo en cuenta el esquema de la figura 2.3 se puede hallar la dependencia del estado siguiente en función del presente y de la entrada:

		E	
	$Q_1Q_0(t+1)/\lambda$	0	1
$Q_1Q_0(t)$	00	00/0	01/0
	01	01/0	10/0
	10	10/0	11/1
	11	11/1	11/1

Tabla 2.1: *Tabla de transiciones del ejemplo de la figura 2.1*

$$\begin{cases} Q_1(t+1) = Q_1(t)\overline{E} + Q_0(t)E \\ Q_0(t+1) = \overline{Q_1}(t)\overline{E} + Q_0(t)E \end{cases} \tag{2.3}$$

2.2.3 Obtención de la función de salida

Trivialmente, a partir de las figuras 2.1 y 2.3 se puede obtener la función de salida $\delta(Q(t), I)$ sin mas que seguir de manera cuidadosa las conexiones de los esquemas. De esta manera para el circuito síncrono la función de salida será

$$S = Q_1(t)Q_0(t) + Q_1(t)E \tag{2.4}$$

Para el circuito asíncrono se obtiene

$$S = Q_1(t)\overline{E} \tag{2.5}$$

2.2.4 Análisis, Representación gráfica e interpretación

En el caso del S.S. síncrono, y a partir de la función de transición para ambos biestables, se puede construir la tabla 2.1 que es la tabla de transición del S.S. Esta tabla reune las cinco cantidades que expresan un S.S., permitiendo seguir de manera sencilla su evolución, tal como se explicó en la sección 1.3.1.

Con la tabla 2.1 se puede realizar una simulación del comportamiento del S.S. sólo con saber un estado inicial y el valor de la entrada en cada instante. A partir de esta tabla se puede construir un diagrama de transiciones que mostrará, de forma mucho más gráfica la evolución del S.S. Esto se hace con el grafo de la figura 2.4.

A partir de este grafo se puede interpretar lo que realiza este S.S. Si se supone que el estado inicial es el 00 la salida del sistema vale 0 hasta que en la entrada E se han presentado tres 1's. Entonces el sistema permanece con la salida a 1 hasta que se reinicialice.

Realizar el análisis del sistema secuencial asíncrono es similar. La única diferencia estriba en que, una vez se cambia la entrada, el sistema evoluciona de manera no controlada hasta llegar a un nuevo **estado estable**. La condición de estabilidad se cumplirá cuando, para una entrada dada, el estado siguiente sea igual al estado presente.

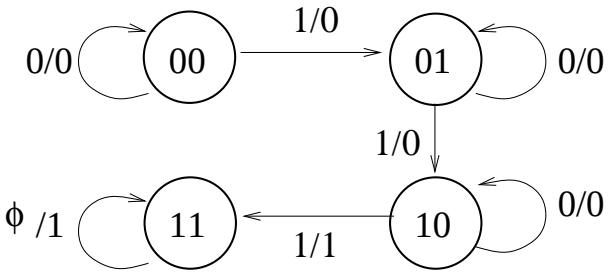


Figura 2.4: *Grafo equivalente a la tabla 2.1*

	E		
	$Q_1Q_0(t+1)/\lambda$	0	1
$Q_1Q_0(t)$	00	01/0	**00/0**
	01	**01/0**	11/0
	11	10/1	**11/0**
	10	**10/1**	00/0

Tabla 2.2: *Tabla de transiciones del ejemplo de la figura 2.2*

De esta manera se han marcado en negrita en la tabla de transición 2.2 los estados estables del sistema secuencial asíncrono de la figura 2.2.

Para interpretar el sistema secuencial asíncrono se puede, análogamente a como se ha hecho con el S.S. síncrono, realizar un grafo. En este caso, y a partir de la tabla 2.2 se obtiene la figura 2.5.

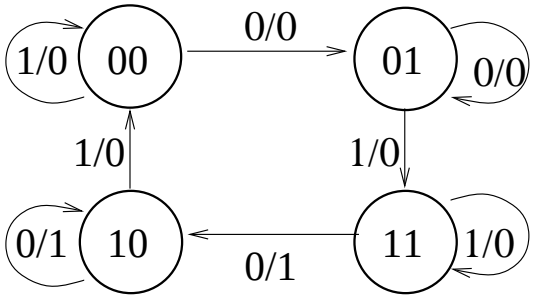


Figura 2.5: *Grafo equivalente a la tabla 2.2*

A partir de la figura 2.5 se observa como (suponiendo una entrada inicial 1) el sistema evolucionará si entra la secuencia 010 , pasando a valer entonces la salida 1. El sistema detectará 2 pulsos negativos consecutivos (supesta entrada estable 1) de cualquier ancho.

2.3 Conclusiones

La metodología para analizar el comportamiento de un S. S. es sencilla. Como datos del problema se tendrá el conjunto de entradas I y un esquema o lista de nodos del

circuito. A partir de estos datos se halla la ecuación estados siguiente o transición y la función de salida.

Para hallar la función estado siguiente en un sistema secuencial síncrono se deberán de aplicar las ecuaciones características de los biestables. En el caso de los sistemas secuenciales asíncronos también se aplicarán estas ecuaciones en el caso de que el sistema este implementado con biestables asíncronos. Si sólo se tienen puertas lógicas con realimentaciones se identificaran los nodos con realimentaciones como los nodos que almacenan los estados presentes. Una vez se tienen las ecuaciones del estado siguiente, si el sistema es síncrono, se obtendrá directamente la tabla de transición y el grafo. Sin embargo, si el sistema es asíncrono se deberán identificar los estados estables del sistema (estados en los que el estado siguiente es igual al presente) para cada entrada. Entonces ya se podrá construir la tabla de transición y el grafo para poder interpretarlo.

En el caso de que se trate de máquinas de estados que generen secuencias o que detecten secuencias será sencillo realizar la interpretación del S.S. Sin embargo en el caso de que la máquina de estados sea un subsistema de un sistema digital mayor habrá que conocer perfectamente la procedencia de las entradas el significado de las salidas.

Cabe hacer notar como en este capítulo no se han tenido en cuenta los parámetros temporales y sólo se ha hecho un análisis lógico de los S. S. Posteriormente, en el capítulo 6 se definirán los parámetros asociados a la lógica secuencial y se verá su influencia en su análisis y en su diseño. En el apéndice A se pueden encontrar referencias para profundizar en el análisis de los sistemas secuenciales.

Capítulo 3

Síntesis de Sistemas Secuenciales

3.1 Introducción

En el capítulo 1 se ha expuesto la necesidad de introducir un nuevo formalismo que, superando a la mera lógica combinacional, permite tener en cuenta la evolución temporal de los sistemas. Se han definido los conceptos de **estado presente** y **estado futuro**, así como la arquitectura de un sistema secuencial genérico. A partir de todo esto se han presentado las unidades mínimas de almacenamiento de la información o **biestables** que serán las piezas básicas de los S. S.

En el capítulo 2 se ha abordado la metodología de análisis de los S. S. tanto síncronos como asíncronos. En el análisis se partía de un esquema y se concluía cual era el comportamiento del S. S. En este capítulo se abordará la síntesis de sistemas secuenciales que será el proceso inverso. Es decir, se partirá de unas especificaciones y se llegará a un esquema de un circuito digital que cumpla estas especificaciones.

La parte mas compleja es entender bien las especificaciones del diseño. Una vez realizado ésto todo el proceso es bastante automático y sencillo. Entender bien las especificaciones significa que se realizará un grafo que tenga exactamente el mismo comportamiento que el S. S. propuesto. Puede ocurrir que alguna vez las especificaciones no sean completas y se tenga opción a elegir o matizar parte del comportamiento del autómata, pero siempre de forma razonable y cumpliendo todo lo que está definido. La mejor forma de entender la metodología de síntesis será la ejemplificación con un caso concreto. Se recomienda consultar [NNCI96] y [Hay96] para la síntesis de sistemas secuenciales síncronos, [GA95] para sistemas secuenciales asíncronos y [Wak00] para realizar ejercicios.

E	0	1	1	0	1	0	0	0	1	1	1	0
S	ϕ	ϕ	1	ϕ	ϕ	0	ϕ	ϕ	0	ϕ	ϕ	1

Tabla 3.1: *Simulación de la máquina que determina la paridad impar de tandas de 3 bits*

3.2 Metodología para sistemas secuenciales síncronos

3.2.1 Entender las especificaciones: Diagrama de flujo y autómata primitivo

Es interesante realizar una *simulación mental* para tener claro el funcionamiento del S. S. propuesto. Esto significa que se intentarán entender e interpretar las especificaciones previendo el comportamiento del S. S. ante cualquier estímulo de entrada. En el caso de que no se especifique el comportamiento ante ciertas entradas ésto será tenido en cuenta para posteriores simplificaciones de la máquina.

Una vez se tenga claro el funcionamiento se deberá realizar un grafo que cubra todos los posibles caminos que puede tomar el S. S. Este grafo se llama **autómata primitivo** porque cumplirá las especificaciones del problema pero no será el autómata más simple que lo haga. Posteriormente habrá que realizar la simplificación de éste. A continuación se propone el enunciado del S.S. que ejemplificará la metodología de síntesis.

Diseñar un S. S. síncrono con una entrada E y una salida S que determine la paridad impar de tandas de 3 bits. Es decir, si el número de unos es par al llegar el tercer bit de la serie la salida valdrá 1. Esto no se determinará de forma continua, de manera que cada nuevo bit de entrada no se evaluará la paridad impar de los tres últimos bits, sino que se evaluarán tandas de 3 bits. En los dos primeros bits de la entrada la salida puede tener cualquier valor

En la tabla 3.1 se realiza una simulación del comportamiento del sistema. En este caso se debe tomar alguna decisión ya que el problema no especifica el valor de la salida en los dos primeros bits. No se calcula la paridad para cada entrada de la acumulación de los últimos 3 bits. El sistema evalúa la paridad de los últimos 3 bits al recibir el tercero. La opción que se tomará, para después poder simplificar más aún el S.S., es que la salida sea 0 o 1 durante los 2 primeros bits de cada tanda. Es decir la salida será la indeterminación ϕ, que se arrastrará hasta que se haga una elección que simplifique la máquina.

El paso siguiente consiste en realizar un grafo o **autómata primitivo** (ver sección 1.3.2), que represente el funcionamiento del S.S. propuesto. Para ello se debe tener en cuenta todo lo que el autómata debe *recordar*, y de esta manera construir paso a paso el grafo de la figura 3.1. Partiendo de un estado inicial A se evolucionará a 2 estados diferentes (B y C) en función de si la entrada es 0 o 1, porque la contribución a la paridad impar será distinta. Lo mismo ocurrirá a partir de los estados B y C porque el bit que entrará será el segundo. De esta manera tendremos que se generarán 4 estados más, 2 como los sucesores de B y 2 como los sucesores de C.

En la siguiente transición se realizará la evaluación de la paridad de los 3 bits anteriores. El estado siguiente será común a todas las posibilidades y se volverá al inicio. Para saber el valor de la salida en esta última transición se deberá de seguir el camino hasta llegar a la tercera transición. De esta manera el autómata primitivo será

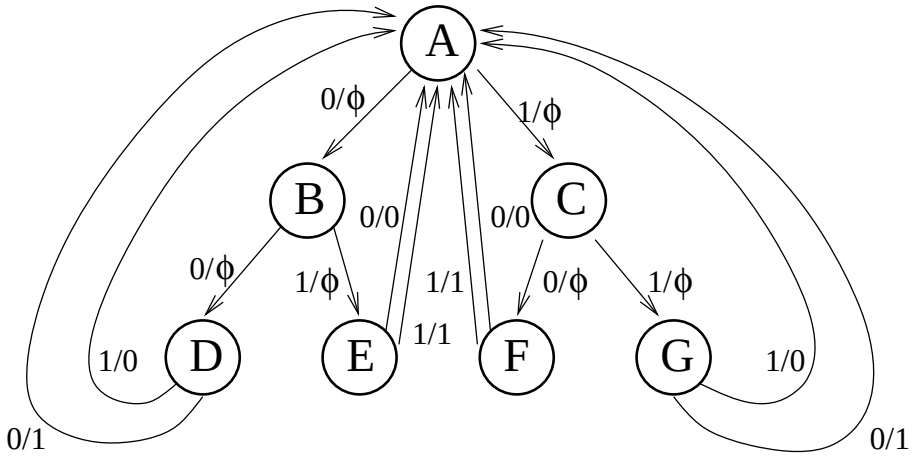


Figura 3.1: *Autómata primitivo del S.S. síncrono de ejemplo*

el que se especifica en la figura 3.1.

Una vez se llega a obtener el autómata primitivo se debe comprobar que el grafo cumple las especificaciones del problema. A partir de esta máquina primitiva se intentará llegar a una máquina más simple que también cumpla las especificaciones del problema, pero que utilizará menos lógica.

3.2.2 Simplificación del autómata: Estados equivalentes

Puede ocurrir que el autómata primitivo tenga **estados equivalentes** o redundantes. Es decir, estados que se pueden eliminar sin afectar al funcionamiento del S.S.

Definición: *Dos estados q_i y q_j son equivalentes (y se representará como $q_i \Leftrightarrow q_j$) si y sólo si son iguales o se cumple que:*

1. $\lambda(q_i, x) = \lambda(q_j, x)\ \forall\ x \in I$, es decir, las salidas son iguales para una entrada dada.
2. $\delta(q_i, x) \Leftrightarrow \delta(q_j, x)\ \forall\ x \in I$, es decir, los estados siguientes son a su vez equivalentes para una entrada dada.

Esto es, dos estados son equivalentes si sus salidas son las mismas y sus estados futuros también son equivalentes para toda entrada. Por tanto si dos estados tienen salidas distintas para una misma entrada (es decir no se cumple la primera condición) no son equivalentes.

Si se cumple esta primera condición deberá de comprobarse la segunda. Si los estados futuros son iguales entonces son equivalentes, con lo que la segunda condición será valida. Si los estados futuros son diferentes se deberá comprobar si son equivalentes a su vez. Como se puede advertir esta es un definición recursiva complicada de evaluar. Lo más cómodo es comprobar las equivalencias y no-equivalencias a través de una **tabla de fusión**. Para ello lo más simple es reescribir el S.S. como una tabla de transiciones, tal como se hace con la tabla 3.2.

En la tabla 3.2 ya se pueden detectar algunas no-equivalencias de forma directa. Los estados con salidas diferentes no cumplirán la primera condición de estados equivalentes y podrán ser eliminados de la tabla de transición. De esta manera por ejemplo, el estado D **no** es equivalente al E y el F por tener salidas diferentes, y **si** es equivalente al estado

Estado Presente	E=0	E=1
A	B/ϕ	C/ϕ
B	D/ϕ	E/ϕ
C	F/ϕ	G/ϕ
D	A/1	A/0
E	A/0	A/1
F	A/0	A/1
G	A/1	A/0

Tabla 3.2: *Tabla de transiciones del grafo 3.1*

G por ser la salida y los estados futuros iguales para todas las entradas, tal como se muestra en la figura 3.2. El resto de equivalencias, al estar la salida de los tres primeros estados indeterminada, no se pueden decidir sin elegir un valor concreto de la salida.

El hecho de fijar una salida para las indeterminaciones en esta fase del diseño, puede hacer que después no se puedan utilizar éstas para simplificar la salida. Por tanto es interesante mantenerlas hasta la fase final de implementación de la función de salida.

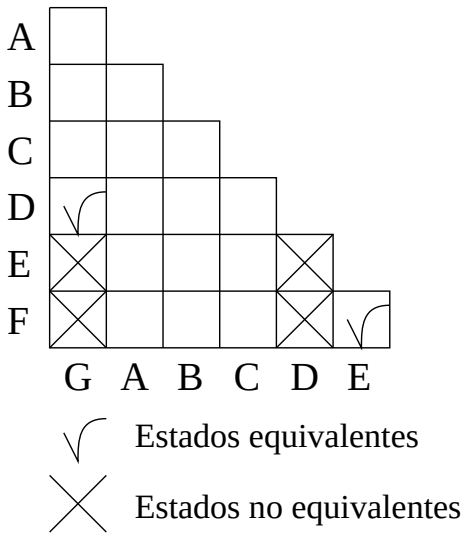


Figura 3.2: *Primera aproximación en la tabla de fusión de la tabla de transiciones 3.2*

A partir de la tabla de fusión de la figura 3.2 se puede construir la tabla de fusión que aparece en la figura 3.3. En ella aparece en cada casilla las condiciones de equivalencia que se deben de cumplir para que se cumpla esa equivalencia. Así por ejemplo para que B $\Leftrightarrow$ C se deberá cumplir que sus estados siguientes también lo sean, es decir D $\Leftrightarrow$ F y simultáneamente E $\Leftrightarrow$ G. Como no se cumplen ambas condiciones entonces de puede concluir que B y C no son equivalentes, tal como se muestra en la figura 3.3.

De la misma manera para que se cumpla que A $\Leftrightarrow$ B se deberá cumplir que sus estados siguientes también lo sean, es decir B $\Leftrightarrow$ D y simultáneamente C $\Leftrightarrow$ E. Al ser la propiedad de equivalencia una propiedad asociativa, se formarán clases de equivalencia de manera que se cumplirá que si A $\Leftrightarrow$ B y B $\Leftrightarrow$ D entonces A $\Leftrightarrow$ B $\Leftrightarrow$ D, formando

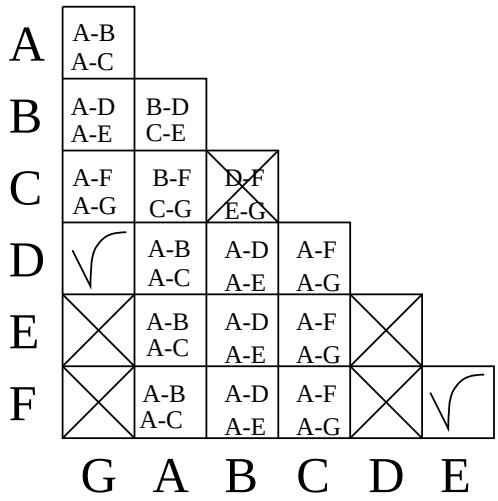


Figura 3.3: *Segunda aproximación en la tabla de fusión de la tabla de transiciones 3.2*

una clase de equivalencia estos tres estados. Para comprobar si B ⇔ D se debería de cumplir que A ⇔ E con lo que, como antes se tenia que C ⇔ E, se incorporan C y E a la clase de equivalencia de A. De esta manera, se debería de cumplir que A ⇔ B ⇔ D ⇔ C ⇔ E, pero como ya se tiene que B y C no son equivalentes entonces el supuesto inicial (A ⇔ B) no se cumplirá y por tanto A y B tampoco serán equivalentes. Utilizando estos razonamientos se puede llegar a la tabla de fusión final de la figura 3.4.

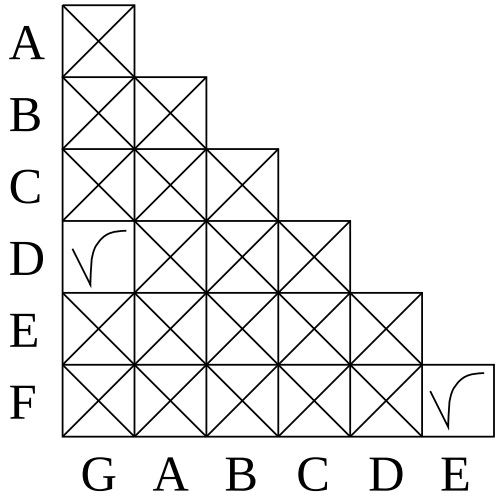


Figura 3.4: *Tercera aproximación en la tabla de fusión de la tabla de transiciones 3.2*

Los únicos estados equivalentes que se obtienen son D ⇔ G y E ⇔ F, por tanto se puede construir la tabla 3.3 que se denomina **tabla de estados reducida**.

Se debe comprobar que el S. S. simplificado cumple las especificaciones del problema igualmente. Una vez se asume que esta tabla representa al S. S. se debe realizar una codificación de estados con el propósito de obtener las funciones δ y λ.

Estado Presente	E=0	E=1
A	B/ϕ	C/ϕ
B	D/ϕ	E/ϕ
C	E/ϕ	D/ϕ
D	A/1	A/0
E	A/0	A/1

Tabla 3.3: *Tabla de transiciones reducida equivalente a la tabla 3.2*

Estado Presente	Q_2	Q_1	Q_0
A	0	0	0
B	0	0	1
C	0	1	0
D	0	1	1
E	1	0	0

Tabla 3.4: *Codificación de estados, minimizando número de biestables, de la tabla 3.3*

3.2.3 Codificación de estados

Sea N el número de estados del S. S. después de simplificar el autómata primitivo. El numero de biestables necesarios para codificar estos estados será como mínimo de $P = log_2 N$. En caso de que P no sea entero se redondeará al entero inmediatamente superior. En el caso del problema $P = log_2 5$, que como no es entero se redondea a 3. La tabla 3.4 muestra la codificación de **minimización de biestables** elegida para el problema del ejemplo.

La complejidad de las funciones δ y λ dependerá de la codificación realizada y de los biestables utilizados. Habitualmente se suele utilizar una codificación de manera que se conserve la contigüidad de estados lo máximo posible. Esto no garantiza una codificación óptima que reduzca la lógica de implementación, siendo éste un problema realmente complejo.

Puede ocurrir que no siempre sea óptima una codificación con el mínimo número de biestables ya que eso puede complicar la circuitería de la función de salida, de la función de excitación de los biestables o puede hacer que la velocidad de funcionamiento sea muy lenta.

Existe un tipo de codificación especial que se utiliza mucho en la síntesis automática de lógica programable. Es la codificación denominada **uno activo** (*one-hot encoding*). En esta codificación no se minimiza el número de biestables sino que en cada estado uno y sólo un biestable está activo (su estado presente es 1). De esta manera las funciones de excitación pueden ser más sencillas (menos minitérminos y más simples estos minitérminos), con menos lógica combinacional y menos retrasos. La desventaja de esta aproximación es que se utilizarán más biestables y habrá que generar más funciones de excitación (más biestables que excitar).

Este tipo de codificación es la que suelen utilizar los sintetizadores de lógica programable a partir de lenguajes de alto nivel. En las máquinas a implementar en FPGAs o

Estado Presente	Q_4	Q_3	Q_2	Q_1	Q_0
A	0	0	0	0	1
B	0	0	0	1	0
C	0	0	1	0	0
D	0	1	0	0	0
E	1	0	0	0	0

Tabla 3.5: *Codificación uno-activo de estados de la tabla 3.3*

CPLDs no importa usar unos pocos biestables más para codificar los estados siempre que se gane velocidad en el funcionamiento de la máquina. Un ejemplo de codificación uno activo para el S.S. de la tabla 3.3 se muestra en la tabla 3.5.

Se deja como ejercicio comprobar como a partir de la codificación uno-activo las funciones de excitación para biestables D son más sencillas, por tanto utilizan un árbol menos profundo de lógica combinacional y tendrán un funcionamiento más rápido.

3.2.4 Funciones de excitación de los biestables

Una vez se tiene la codificación de estados se tiene una función δ o estado siguiente implícita para cada biestable, que se puede calcular utilizando los métodos de simplificación de funciones habituales, pero no es el objetivo del problema. Lo que hay que conseguir es que las transiciones de los biestables se produzcan de la manera descrita en la tabla de transición. Para ello se utilizarán las tablas de excitación de los biestables indicados en el problema. Estas tablas están descritas en la sección 1.5.3.

En el capítulo 1 se indicó que la utilización de biestables JK permite una mayor simplificación de las funciones de excitación debido a la gran cantidad de indeterminaciones que introducen. Los biestables D son mas cómodos debido a su trivial tabla de excitación.

En este ejemplo se usarán biestables JK para resolver el S. S. del ejemplo. La tabla 3.5 muestra la función estado siguiente, que depende del estado presente y de la entrada. Así por ejemplo en el estado B (001) y con entrada 1 se pasará al estado E (100), por tanto Q2 pasará de 0 a 1 y las entradas de este biestable JK deberán provocar esta transición, que en este caso se conseguirá con JK=1ϕ. Realizando el mismo proceso para los tres biestables se obtendrán las 3 tablas que se muestran en la figura 3.5.

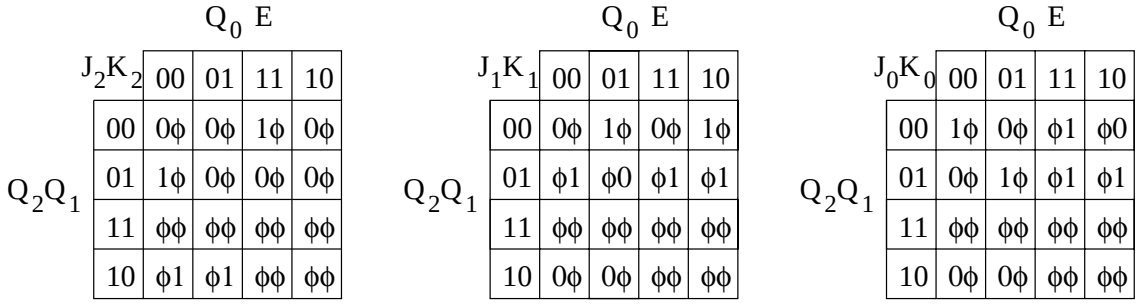


Figura 3.5: *Tablas de excitación de los biestables JK para la codificación de la tabla 3.4*

Q_2Q_1 \ Q_0 E	S	00	01	11	10
	00	ϕ	ϕ	ϕ	ϕ
	01	ϕ	ϕ	0	1
	11	ϕ	ϕ	ϕ	ϕ
	10	0	1	ϕ	ϕ

Tabla 3.6: *Mapa de Karnaugh de la función de salida del ejemplo de la tabla 3.3*

Los estados inexistentes (101, 110 y 111) se han rellenado de indeterminaciones para J y K. Así las funciones podrán simplificarse más aún. Se supone que nunca se llegará a estos estados así que se pueden utilizar las indeterminaciones como convenga para simplificar la lógica. Otra posibilidad sería rellenar estas casillas con transiciones a estados *seguros*. De esta manera si, debido al ruido electromagnético, se produjera una transición a un estado fuera de la máquina el resultado sería predecible y estaría bajo control.

A partir de los mapas de Karnaugh que se muestran en la figura 3.5 se obtiene las ecuaciones de excitación:

$$\left\{\begin{array}{l} J_2 = Q_1\overline{Q_0}\,\overline{E} + \overline{Q_1}Q_0E \\ K_2 = 1 \end{array}\right. \quad \left\{\begin{array}{l} J_1 = \overline{Q_2}\,\overline{Q_0}E + Q_0\overline{E} \\ K_1 = Q_0 + \overline{E} \end{array}\right. \quad \left\{\begin{array}{l} J_0 = \overline{Q_2}\,\overline{Q_1}\,\overline{E} + Q_1E \\ K_0 = Q_1 + E \end{array}\right. \tag{3.1}$$

Por tanto ya se tienen las funciones que excitarán a los biestables JK que almacenan el estado presente. Para realizar las funciones de excitación con otro tipo de biestable sólo hay que coger la función de excitación adecuada de la sección 1.5.3. Se deja como ejercicio resolver el mismo problema con la misma codificación de minimización de biestables pero con otros biestables (SR, D y T).

3.2.5 Función de salida y esquema

Finalmente ya sólo queda construir la función de salida a partir de su mapa de Karnaugh. En este caso el mapa de Karnaugh de la función de salida del S. S. de la tabla 3.3 se muestra en la tabla 3.6. A partir de esta tabla se obtiene la ecuación se salida S:

$$S = \overline{Q_0}E + Q_0\overline{E} = Q_0 \oplus E \tag{3.2}$$

Finalmente sólo queda realizar el esquema lógico del circuito con las puertas lógicas permitidas en el problema. El esquema que implementa las ecuaciones de excitación 3.1 y la función de salida 3.2 se muestra en la figura 3.6.

3.3 Metodología para sistemas secuenciales asíncronos

Los sistemas secuenciales síncronos tienen una señal de sincronismo o de reloj que marca la evolución del sistema, por tanto el sistema evoluciona de forma acompasada

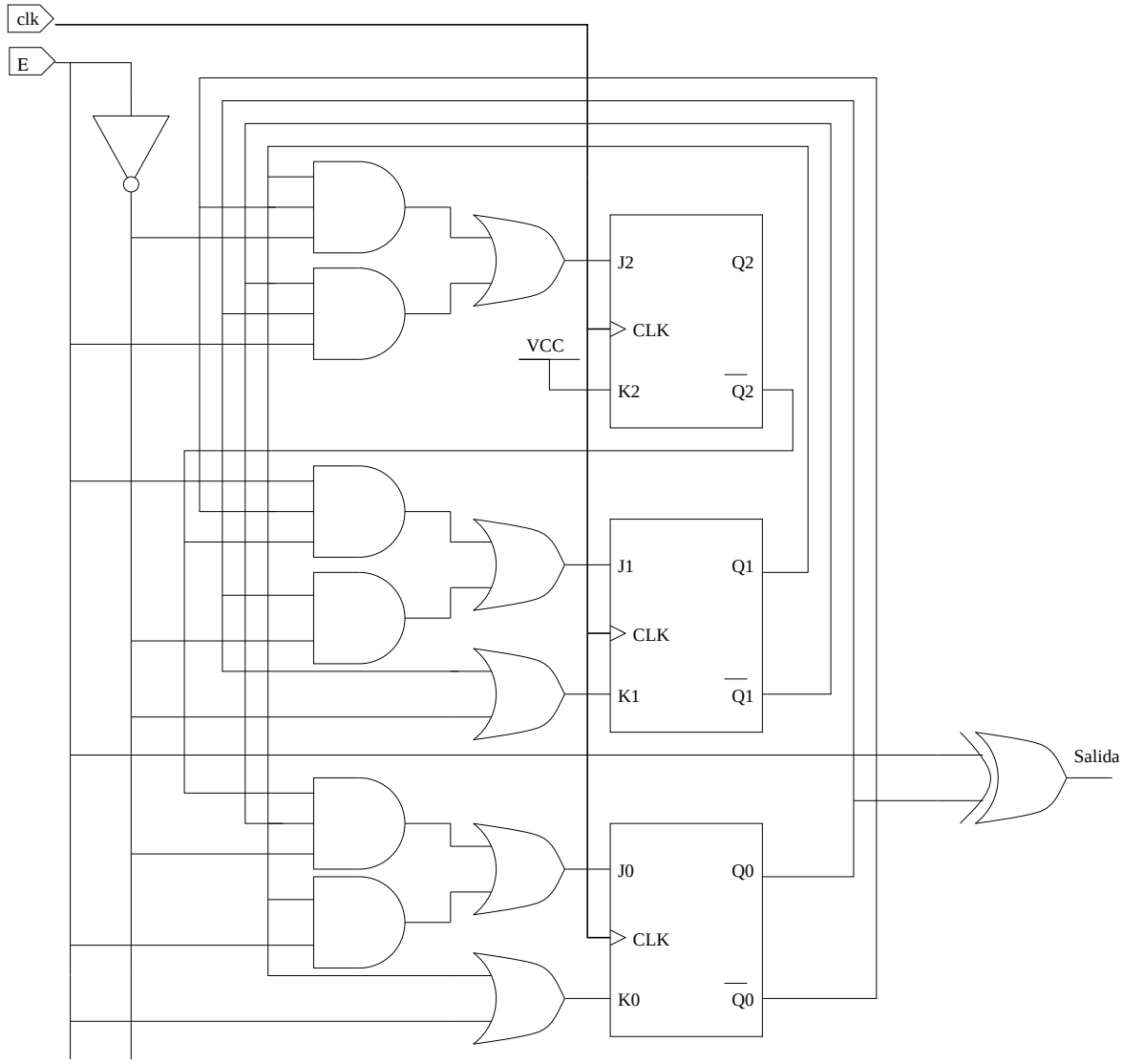


Figura 3.6: *Esquema del circuito secuencial síncrono del ejemplo*

a cada golpe de reloj. En un S. S. asíncrono el sistema evoluciona de forma automática al cambiar la entrada hasta llegar a un estado estable con la entrada presente. Los sistemas asíncronos tienen una respuesta más rápida frente a estímulos externos, pero presentan una mayor dificultad en su diseño.

3.3.1 Entender las especificaciones: Diagrama de flujo y autómata primitivo

Para observar las diferencias entre en la síntesis de máquinas asíncronas y síncronas se va a utilizar de nuevo un ejemplo:

Diseña un S. S. asíncrono con una entrada E y una salida S donde la salida vale siempre 0 hasta que se introduce la secuencia 101. Cuando esto ocurre el sistema genera un 1 en la salida que durará hasta que la entrada vuelva al valor 0 de nuevo. Una vez

Nombre	E=0	E=1
A	A/0	B/0
B	C/0	B/0
C	C/0	D/1
D	A/0	D/1

Tabla 3.7: *Tabla de transiciones del sistema secuencial asíncrono de la figura 3.7*

ocurra esto el sistema volverá a detectar la secuencia 101.

Este ejemplo corresponde claramente a una máquina asíncrona. En el caso de que el diseño fuera síncrono se debería fijar una duración mínima de pulso para evitar que no se detectase un pulso entre 2 flancos de reloj. Esta máquina además evolucionará lo más rápido posible, ya que no esperará a la señal de reloj para cambiar de estado.

Un grafo de una máquina de estados que cumple las especificaciones del problema se muestra en la figura 3.7.

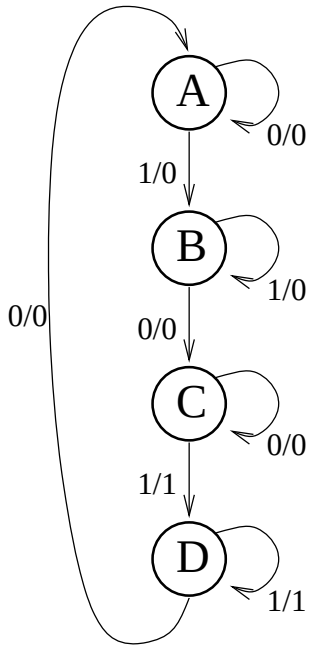


Figura 3.7: *Grafo del sistema secuencial asíncrono del ejemplo*

A continuación se debe construir la tabla de transiciones 3.7 para encontrar los estados equivalentes. En este caso trivialmente se encuentra que no hay ningún estado equivalente. Las únicas equivalencias permitidas por la salida serían A $\Leftrightarrow$ B y C $\Leftrightarrow$ D. En ambos casos para que se cumpla esa equivalencia se debe cumplir que A $\Leftrightarrow$ C, equivalencia que no se cumple al tener ambos estado una salida distinta para E=1. Por tanto la tabla 3.7 coincide con la tabla reducida.

$Q_1Q_0(t)$ \ $Q_1Q_0(t+1)$	E=0	E=1
00	**00/0**	01/0
01	10/0	**01/0**
11	00/0	**11/1**
10	**10/0**	11/1

Tabla 3.8: *Codificación inicial de la máquina asíncrona de la tabla 3.7*

3.3.2 Codificación de estados: Carreras críticas

Es en esta parte de la metodología de síntesis cuando aparece la diferencia fundamental con los sistemas síncronos. En primer lugar hay que marcar los **estados estables**, es decir, estados en los que se cumple que el estado siguiente coincide con el estado presente para una cierta entrada. El sistema secuencial asíncrono dejará de evolucionar quedándose *clavado* en ese estado hasta que cambie la entrada. Esta particularidad de evolución incontrolada hasta llegar a un estado estable presenta algunos problemas, tal como se mostrará a partir de la codificación que se muestra en la tabla 3.8. Los estados estables aparecen en este caso en negrita.

Supóngase que estando en el estado 01 con entrada E=1 se varía la entrada a E=0. El estado siguiente debe ser 10, y en este estado la entrada 0 estabiliza el estado (es decir estado presente igual a estado futuro). En este caso la transición se produce en dos bits simultáneamente ($01 \rightarrow 10$). Nada garantiza que el cambio en ambos bits sea simultáneo y que la transición no sea $01 \rightarrow 00 \rightarrow 10$ ó $01 \rightarrow 11 \rightarrow 10$.

Si se producen las transiciones en el orden $01 \rightarrow 00 \rightarrow 10$ cabe hacer notar que cuando el estado presente sea 00 para la entrada 0, que es la que ha hecho evolucionar el sistema, se tiene un estado estable. Esto significa que el estado siguiente será igual al presente y por tanto el sistema se estancará en ese estado hasta que cambie la salida. Se puede observar como se ha producido una transición no deseada ya que el estado final al que se pretendía llegar era el 10 no el 00.

Esta evolución de los sistemas secuenciales asíncronos que llevan a estados estables no deseados se denomina **carrera crítica**, y son el principal peligro en la síntesis de las máquinas asíncronas. Cabe hacer notar que esta transición no deseada no ha sido debida a problemas de ruido electromagnético, sino a la diferencia en la velocidad de transición de los dos bits que codifican los estados.

En el caso de que la evolución del sistema fuera $01 \rightarrow 11 \rightarrow 10$ se iría al estado 11 cuyo estado siguiente para la entrada 0 es 00, que a su vez desembocaría en el estado 00 que es estable para E=0. Se observa como de nuevo aparece otra carrera crítica.

Es fácil darse cuenta donde pueden aparecer las carreras críticas. Éstas aparecerán cuando el estado siguiente difiera en mas de un bit del estado presente. Entonces es cuando la transición puede ocurrir a través de un camino o carrera que desemboque en un estado estable incorrecto. Entonces la carrera será crítica. Con la codificación de la tabla 3.8 se puede ver que esto va a ocurrir en las transiciones comentadas anteriormente (del estado 01 al 10) y en la transición del estado 11 al 00 (ésta va a ocurrir cuando se esté en el estado 11 que es estable mientras E=1 y se pase a E=0).

Para solucionar este problema existen varias técnicas. La más obvia es cambiar la codificación de los estados de manera que siempre que se acceda a un estado vecino

$Q_1Q_0(t)$	$Q_1Q_0(t+1)$	E=0	E=1
	00	**00/0**	01/0
	01	11/0	**01/0**
	11	**11/0**	10/1
	10	00/0	**10/1**

Tabla 3.9: *Codificación sin carreras críticas de la máquina asíncrona de la tabla 3.7*

la diferencia sea de un solo bit. Esto puede ser imposible cuando se tienen muchos estados y además en el grafo de cada nodo salen muchos arcos (hay muchas transiciones posibles). Como ayuda para que la codificación evite carreras críticas se pueden observar los cubos adyacentes de la figura 3.8. En ellos se cumple siempre que de un estado a su vecino sólo varía un bit. Lo que se debe de hacer es tratar de adecuar esta regla a cada problema particular.

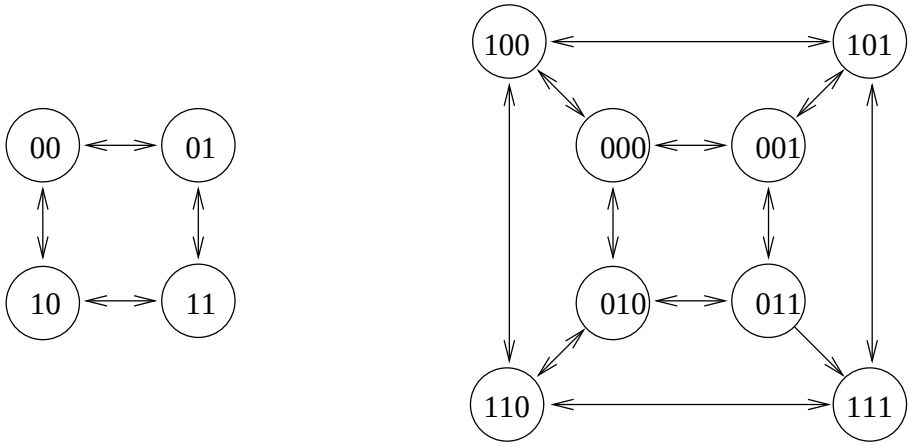


Figura 3.8: *Cubos adyacentes de orden 2 y 3 para evitar carreras críticas*

Teniendo en cuenta el cubo adyacente de orden 2 (número de bits) de la figura 3.8 se puede codificar de nuevo de forma correcta el S. S. de la tabla 3.7, evitando carreras críticas y obteniendo la tabla 3.9. El único cambio es que la codificación de estados C y D se permuta. De esta manera se consigue que siempre haya una diferencia de sólo un bit en cada transición.

Usar los cubos adyacentes es el método deseable para evitar carreras críticas, pero puede ocurrir que no sea posible *encajar* el grafo de una máquina de estados de n bits en su cubo adyacente. Un ejemplo puede ser el de la figura 3.9. En este caso se no es posible usar el cubo adyacente de orden 2 para evitar que hayan cambios de estado con transiciones de más de un bit. La solución adoptada consiste en añadir más bits a la codificación de estados, consiguiendo de esta manera evitar las carreras críticas. Esta solución a pesar de utilizar más lógica de la necesaria es más deseable que tener carreras críticas, que pueden hacer que el S. S. no funcione.

En el caso de que ni siquiera añadiendo bits se pueda ajustar la máquina de estados a la estructura de un cubo adyacente, se añadirán estados. De esta manera, a pesar de utilizar una máquina no simplificada que usará más lógica de la necesaria, se evitarán las carreras críticas. Cabe hacer notar que si se añaden estados hay que tener cuidado

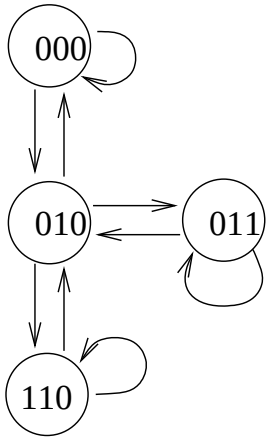


Figura 3.9: *Ejemplo de codificación con más bits de los necesarios para evitar carreras críticas*

de que la nueva máquina redundante cumplas las especificaciones originales.

Por último si tampoco se encuentra una máquina equivalente redundante que evite las carreras críticas habrá que recurrir al nada deseable método de analizar la implementación a bajo nivel del S. S. De esta manera se podrían prever los retrasos en los diversos bits y localizar caminos de retardo que introduzcan carreras críticas. Añadiendo elementos de retraso y escogiendo de forma adecuada las puertas se puede intentar eliminar la aparición de carreras críticas, pero este método es absolutamente inseguro y no deseable. Los tiempos de propagación de los circuitos integrados dependen de la temperatura y de la tensión de alimentación, por tanto si varían estas condiciones el sistema puede dejar de funcionar. Además el fabricante garantiza unos retrasos máximos, no unos retrasos exactos, lo cual hace más complicado aún este método.

3.3.3 Funciones estado siguiente, función de salida y esquema

Si se realiza el sistema con biestables asíncronos deberán realizarse las funciones de excitación de los biestables, de la misma manera a como se ha realizado en la sección 3.2.4. Si el sistema se va a realizar sólo con puertas entonces es necesario calcular previamente la función estado siguiente con un mapa de Karnaugh. En este caso se va a realizar una implementación sólo con puertas lógicas, para observar las diferencias con la implementación con biestables.

A partir de la codificación sin carreras críticas que se muestra en la tabla 3.9 se puede obtener la función estado siguiente y la función de salida:

$$\begin{cases} Q_1(t+1) = Q_0(t)\overline{E} + Q_1(t)E \\ Q_0 = Q_0(t)\overline{E} + \overline{Q_1(t)}E \end{cases} \qquad S = Q_1(t)E \tag{3.3}$$

A partir de estas ecuaciones construir el esquema es sencillo. Sólo debe tenerse en cuenta que se representa el estado siguiente. La dependencia con el estado presente se realiza realimentando el estado siguiente como una entrada más. En la figura 3.10 se muestra el esquema que correspondería a las ecuaciones 3.3.

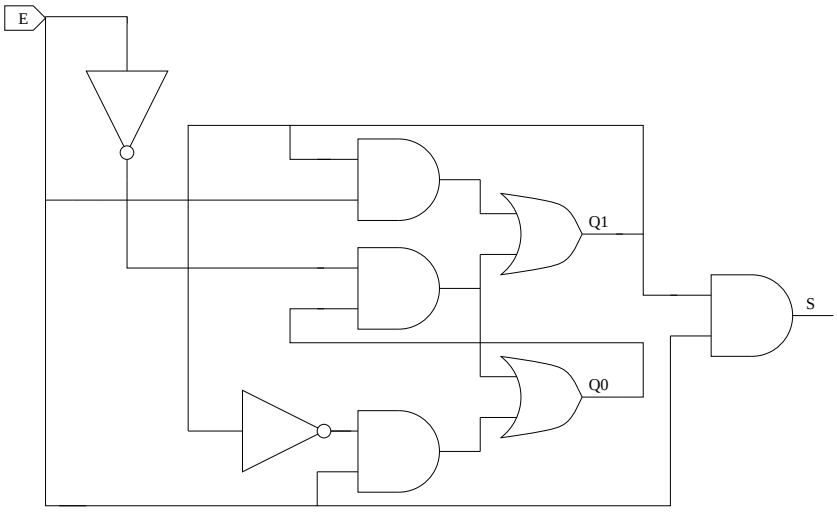


Figura 3.10: *Esquema sólo con puertas lógicas del S. S. asíncrono del ejemplo*

3.4 Conclusiones

La metodología presentada para la síntesis de sistemas secuenciales síncronos es sencilla y automática. En realidad lo más complicado es entender las especificaciones requeridas y plasmarlas en un grafo de una máquina primitiva. A partir del autómata primitivo hay que simplificar la máquina, siempre teniendo cuidado en que en realidad no se debe modificar el funcionamiento del autómata original.

Una vez comprobado que el autómata simplificado sigue cumpliendo las especificaciones del problema se codificarán los estados del S. S. Existen dos estrategias fundamentales de codificación: 1) La minimización de biestables, y 2) La codificación uno activo. La primera de ellas se utilizará preferentemente siempre que se usen componentes discretos y no haya que optimizar la velocidad de funcionamiento de la máquina. La codificación uno activo se utilizará cuando se usen PLDs complejas o la velocidad del sistema secuencial sea un requerimiento importante. A partir de la codificación realizada se implementará, con los biestables indicados, el esquema lógico del S. S. En el caso de que no haya una indicación explícita de los biestables a usar se utilizarán preferentemente los JK por introducir un gran número de indeterminaciones.

La metodología para S. S. asíncronos tiene unas diferencias fundamentales. La evolución de la máquina viene gobernada por el cambio en las entradas, quedando estabilizada cuando el estado futuro coincide con el estado presente. La codificación de estados en máquinas asíncronas presenta la particularidad de que una mala codificación puede hacer que aparezcan carreras críticas. Para evitarlas se utilizarán técnicas de codificación adecuadas, lógica redundante o (en último caso) se editará el circuito a bajo nivel. A partir de la codificación se construirá el circuito digital. En el caso de biestables asíncronos, se hallarán las funciones de excitación de manera análoga a como se hacía en los S. S. síncronos. En el caso de que se use lógica combinacional realimentada sólo hay que calcular la función δ o estado siguiente.

En este capítulo no se han tenido en cuenta los aspectos tecnológicos asociados al diseño de los S. S. Posteriormente, en el capítulo 6 se definirán los parámetros asociados a la lógica secuencial y combinacional y se verá su influencia y las limitaciones que imponen en el diseño de S. S.

Capítulo 4

Introducción al VHDL

4.1 Introducción

En los años 80 la complejidad en el diseño de sistemas digitales llegaba a ser tal que la metodología habitual de descripción del hardware mediante esquemas era pobre y limitada. Fue a mediados de los 80 cuando el IEEE[1] y el departamento de defensa de EEUU pusieron en marcha el proyecto de definir un lenguaje de descripción del hardware con las características:

- Permitiese el diseño jerárquico.
- Cada componente tuviese una interfaz y un comportamiento bien definidos.
- Debe permitirse la descripción del comportamiento de los elementos de diversas maneras; definiendo un conexionado interno o a partir de instrucciones de más alto nivel.
- Debe soportar concurrencia y un manejo integrado del tiempo para poder simular.

De esta manera, el lenguaje de descripción del hardware VHDL (**V**ery High Speed Integrated Circuit **H**ardware **D**escription **L**anguaje) se diseñó inicialmente para simular y modelar sistemas. Posteriormente, y con una limitación de instrucciones cada vez menor, se ha ido utilizando cada vez más para sintetizar sistemas digitales. Hoy en día ya ha desplazado a otros lenguajes de descripción del hardware que no gozaban del soporte de la estandarización de VHDL pero que eran muy utilizados por su simplicidad y síntesis eficiente (p. eje. Verilog, AHDL y ABEL).

La gran mayoría de ejemplos y estructura del capítulo han sido extraídos del [PB99] aunque también se ha usado información del [Wak01]. El primer libro hace una descripción muy didáctica del lenguaje VHDL, llegando a tocar temas complejos. El segundo texto introduce el VHDL como una mera herramienta para describir circuitos sin profundizar en exceso, aunque el nivel es más que suficiente para las necesidades del curso.

[1] Institute of Electrical and Electronics Engineers, institución encargada entre otras de definir estándares

4.2 Flujo de Diseño

Es interesante que previamente a la descripción del lenguaje se introduzcan los pasos que deben seguirse para obtener un sistema digital a partir de un lenguaje de descripción del hardware. La figura 4.1 describe las fases que se siguen y que a continuación se introducen con más detalle.

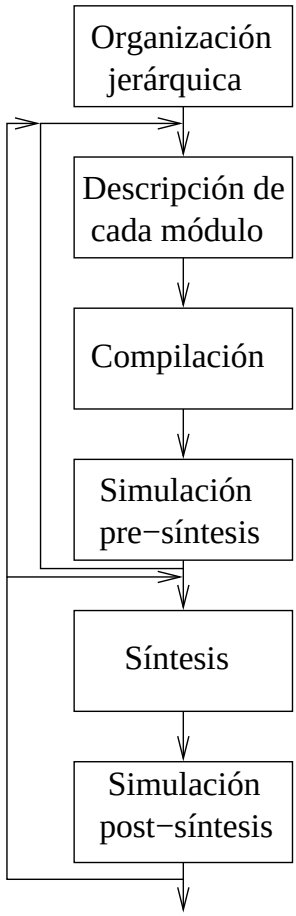


Figura 4.1: *Diversas fases de diseño digital con cualquier HDL*

1. La primera fase consiste en organizar el diseño con los módulos jerárquicos adecuados, de manera similar a la organización de los proyectos software. Esto incluye la utilización de librerías con componentes ya diseñados y de los que, conociendo su interfaz, se puede escoger el modelo de arquitectura que describe su comportamiento.
2. Posteriormente se describe el comportamiento de cada módulo utilizando alguno de los estilos y técnicas de descripción que incluye el lenguaje. Es importante conocer el destino final del proyecto: si es sólo para simulación o modelado o sí se va a incluir síntesis, para de esta manera escoger un estilo de descripción más fácilmente sintetizable.
3. La compilación analiza el código para comprobar que no hay errores de sintaxis, que todos los módulos están bien interconectados y con las dependencias bien definidas. Esta compilación también genera la información interna necesaria para que la etapa siguiente de simulación pueda procesar los estímulos y generar las salidas.

4. La simulación pre-síntesis tiene como objeto comprobar el comportamiento y la funcionalidad del sistema. Análogamente muchas veces en la literatura se denomina simulación comportamental. En esta simulación no se utilizan los retrasos que la implementación real introduce. Mayoritariamente la simulación se hace sin retrasos o con una estimación de retrasos que a priori no suele ser muy realista. Esta primera simulación es útil para detectar errores en el diseño lógico y en el comportamiento del diseño.
5. La síntesis es la creación de un circuito real que implemente el comportamiento descrito en el modelo ya simulado. Es posible sintetizar el diseño en un circuito programable, método mayoritario que se describirá en el capítulo 9, en un circuito totalmente hecho a medida o incluso utilizando componentes estándar conectados en un circuito impreso. Es posible utilizar diversas estrategias de síntesis orientadas a conseguir velocidad, reducir recursos o aumentar la fiabilidad. Cada tipo de síntesis tiene sus ventajas e inconvenientes, pero tienen en común que se trata de un proceso automatizado en el que al final se obtiene una lista de componentes y una descripción de su conexionado o *Netlist*. A partir de la lista de componentes básicos y nodos éstos deben implementarse con los recursos disponibles en el dispositivo escogido, con lo que se debe realizar una asignación o emplazado de componentes y su posterior interconexión o trazado de pistas. Una vez realizado este proceso ya es posible extraer un modelo de comportamiento temporal exacto (fichero de retrasos) que permitirá simular posteriormente el circuito con la máxima fiabilidad.
6. La simulación post-síntesis será la prueba final que dará la validez al diseño realizado. Si el comportamiento no es el esperado puede corregirse simplemente realizando una nueva síntesis en un dispositivo distinto o en el mismo con otra orientación (opciones de compilación). Puede ser incluso necesario redefinir el código, modificando ligeramente el comportamiento, para que sea posible una síntesis que cumpla las expectativas.

4.3 Estructura de un programa

VHDL integra los principios de la programación estructurada como los lenguajes de programación software ADA y PASCAL. La estructura de un componente VHDL se divide básicamente en dos partes: la entidad y la arquitectura. La entidad viene a definir la interfase que el componente presenta al exterior (sería como su encapsulado) y la arquitectura define su comportamiento. Adicionalmente, una arquitectura puede a su vez utilizar otras entidades para definir su comportamiento, estableciéndose niveles jerárquicos.

Una entidad puede cambiar su arquitectura y escoger la más conveniente en función del objeto del diseño. Como ejemplo, un sumador de n bits presenta una interfase única y bien definida (las entradas y salidas de este circuito). Esta interfase será su entidad. Sin embargo puede construirse de muchas maneras: encadenando sumadores completos de 1 bit, con acarreo adelantado, etc. Cada manera de especificar el comportamiento del sumador es una arquitectura distinta. Si se desea una simple descripción para hacer una simulación comportamental pueden ser útiles ciertos tipos de arquitectura. Sin embargo, la validez puede ser distinta si se desea hacer una síntesis en un circuito programable.

La descripción del circuito se realiza en un archivo de texto donde se incluye la declaración de entidad y la definición de la arquitectura. Es posible separar ambas partes en ficheros distintos, pero entonces en la declaración de entidad debe incluirse una referencia al fichero o la librería donde se encuentra la arquitectura. Esta idea es similar a la utilización de librerías en lenguajes de programación software.

Como otros lenguajes de programación, el VHDL define **palabras clave** o cadenas de caracteres reservadas. Generalmente se ignoran los saltos de línea y los espacios y se pueden añadir para mejorar la legibilidad del texto. No hay distinción entra mayúsculas y minúsculas y los comentarios empiezan con dos guiones (–) y finalizan al terminar la línea.

El ejemplo define una puerta AND en VHDL, aunque se puede realizar la descripción de esta puerta de muchas otras maneras, tal como se indicará en la sección siguiente. En el texto se escriben en mayúscula las palabras clave y se añaden comentarios.

```
LIBRARY ieee; -- Esta librería incluye los tipos STD_LOGIC
  USE ieee.std_logic_1164.all; -- El paquete concreto de la librería

ENTITY mi_puerta IS
  PORT (a,b: IN STD_LOGIC; -- La puerta tiene 2 entradas
        y: OUT STD_LOGIC); -- y una salida
END;

ARCHITECTURE comportamiento OF mi_puerta IS
BEGIN
   y<='1' WHEN (a='1' AND b='1') ELSE '0';  -- Una de las muchas descripciones posibles
END comportamiento;
```

La estructura es bastante auto-explicativa. Con la `ENTITY` se define el nombre del componente u objeto. Con la palabra clave `PORT` se definen las señales del componente. `IN` indica que la señal es una entrada y `OUT` que es una salida. Análogamente existe el nombre `INOUT` para indicar señales bidireccionales y `BUFFER` para señales de salida que pueden ser leídas dentro de la arquitectura.

Las instrucciones dentro del bloque de la arquitectura se ejecutan concurrentemente, es decir, todas simultáneamente al igual que un circuito físico donde todos los componentes realizan su función en paralelo. Cabe hacer notar que el `PROCESS` es un bloque o instrucción especial donde las instrucciones que contiene se ejecutan secuencialmente. El propio bloque `PROCESS` se ejecuta concurrentemente con el resto de instrucciones de la arquitectura o con otros bloques `PROCESS`. A continuación, se introducen los diversos estilos de descripción VHDL dentro de la arquitectura.

4.4 Estilos de descripción en VHDL

Existen tres estilos de descripción de circuitos en VHDL, dependiendo del nivel de abstracción. El menos abstracto es una descripción puramente estructural, similar a un esquema o a una lista de nodos y componentes (*Netlist*). Los otros dos estilos representan una descripción comportamental o funcional y la diferencia estriba de la utilización o no de la ejecución serie.

Mediante un sencillo ejemplo se van a mostrar estos tres estilos de descripción. Para ello se va describir el multiplexor de la figura 4.2 utilizando estos tres estilos.

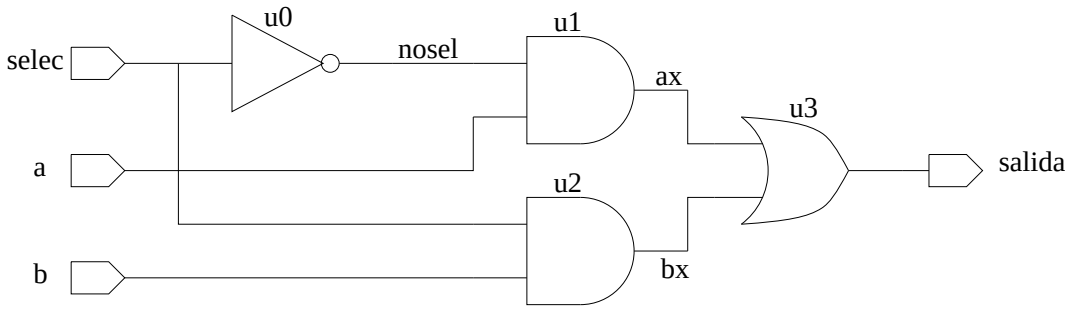


Figura 4.2: *Multiplexor de 1 bit construido con puertas lógicas*

4.4.1 Descripción algorítmica

A continuación se va a realizar la descripción comportamental algorítmica del circuito de la figura 4.2.

En primer lugar, sea el tipo de descripción que sea, hay que definir la interfase externa o **entidad** del circuito. De esta manera se deben definir las entradas y salidas del circuito, es decir, la caja negra que lo define. Se le llama entidad porque en la sintaxis de VHDL esta parte se declara con la palabra clave `ENTITY`. Esta definición de entidad, que suele ser la primera parte de toda descripción VHDL, se expone a continuación:

```
-- Los comentarios empiezan por dos guiones
ENTITY mux IS
PORT ( a:      IN bit;
       b:      IN bit;
       selec:  IN bit;
       salida: OUT bit);
END mux;
```

Este fragmento del lenguaje indica que la entidad `mux` (que es el nombre que se le ha dado al circuito) tiene tres entradas de tipo `bit`, y una salida también del tipo `bit`. Los tipos de las entradas y salidas se describirán en la sección 4.5. El tipo `bit` simplemente indica que puede tomar los valores `'0'` o `'1'`.

La entidad de un circuito es única, sin embargo un mismo símbolo, en este caso entidad, puede tener varias **arquitecturas**. Cada bloque de arquitectura, que es donde se describe el circuito, puede ser una representación diferente del mismo circuito. Por ejemplo, puede haber una descripción estructural y otra comportamental, ambas son descripciones diferentes, pero ambas descripciones corresponden al mismo circuito, símbolo, o entidad. En este caso la descripción comportamental propuesta es:

```
ARCHITECTURE comportamental OF mux IS
BEGIN
   PROCESS(a,b,selec)
   BEGIN
      IF (selec='0') THEN
         salida<=a;
      ELSE
         salida<=b;
      END IF;
   END PROCESS;
END comportamental;
```

Como primera aproximación se considerará que un bloque `PROCESS` es una especie de subrutina cuyas instrucciones se ejecutan secuencialmente cada vez que alguna de las señales de la *lista sensible* cambia. Esta lista sensible es una lista de señales que se suele poner junto a la palabra clave `PROCESS`, y en el caso del ejemplo es `(a,b,selec)`.

Esta descripción comportamental es sencilla conceptualmente ya que sigue una estructura parecida a los lenguajes de programación convencionales. Por este motivo se dice que se trata de una descripción comportamental algorítmica. Simplemente se está indicando que si la señal `selec` es cero, entonces la salida es la entrada `a`, y si `selec` es uno, entonces la salida es la entrada `b`. Esta forma tan simple de describir el circuito permite a ciertas herramientas sintetizar un circuito a partir de una descripción comportamental. La diferencia con una lista de nodos es directa: en una descripción comportamental no se están indicando ni los componentes ni sus interconexiones, sino simplemente se especifica su comportamiento o funcionamiento.

4.4.2 Descripción flujo de datos

La descripción anterior era puramente comportamental, de manera que con una secuencia sencilla de instrucciones era posible definir el circuito. Naturalmente, en algunas ocasiones resulta más interesante describir el circuito de forma que esté más cercano a una posible realización física del mismo. En ese sentido en VHDL hay una forma de describir circuitos que además permite la paralelización de instrucciones y que se encuentra más cercana a una descripción estructural, pero siendo todavía una descripción funcional. A continuación se muestran dos ejemplos de una descripción concurrente o también llamada **flujo de datos** o de **transferencia entre registros**:

```
ARCHITECTURE transferencia OF mux IS
SIGNAL nosel,ax,bx: bit;
BEGIN
  nosel<=NOT selec;
  ax<=a AND nosel;
  bx<=b AND selec;
  salida<=ax OR bx;
END transferencia;
```

```
ARCHITECTURE transferencia OF mux IS
BEGIN
  salida<=a WHEN selec='0' ELSE b;
END transferencia;
```

En la descripción de la izquierda hay varias instrucciones todas ellas concurrentes, es decir, se ejecutan cada vez que cambia alguna de las señales que intervienen en la asignación. Este primer ejemplo es casi una descripción estructural ya que de alguna manera se están definiendo las señales (cables) y los componentes que la definen, aunque no es el caso ya que en realidad se trata de asignaciones a señales y no una descripción de componentes y conexiones. El segundo ejemplo (derecha) es también una descripción de transferencia aunque es suficiente una única instrucción de asignación para definir el circuito.

4.4.3 Descripción estructural

El lenguaje VHDL también permite ser usado como *Netlist* o lenguaje de descripción de estructura, aunque no es la característica más interesante del VHDL. En este caso, esta estructura también estaría indicada dentro de un bloque de arquitectura, aunque la sintaxis interna es completamente diferente:

```
ARCHITECTURE estructura OF mux IS
SIGNAL ax,bx,nosel: bit;
BEGIN
  u0: ENTITY inv PORT MAP(e=>selec,y=>nosel);
  u1: ENTITY and2 PORT MAP(e1=>a,e2=>nosel,y=>ax);
  u2: ENTITY and2 PORT MAP(b,sel,bx);
  u3: ENTITY or2  PORT MAP(e1=>ax,e2=>bx,y=>salida);
END estructura;
```

Es posible darse cuenta de que esta descripción es un poco más larga y mucho menos clara que las anteriores. En el cuerpo de la arquitectura se construye una lista de nodos normal, es decir, se especifican los componentes y sus interconexiones. Para los componentes se utilizan entidades que estarán definidas en algún lugar del fichero o en alguna librería, y para la conexiones se utilizarán señales que se declaran al principio de la arquitectura.

Al igual que ocurre en cualquier lista de nodos, las señales o conexiones deben tener un nombre. En el esquema se han nombrado las líneas de conexión internas al circuito. Estas líneas hay que declararlas como `SIGNAL` en el cuerpo de la arquitectura delante del `BEGIN`. Una vez declaradas las señales que intervienen se procede a conectar entre sí las señales y entidades que representan componentes. Para ello la sintaxis es muy simple. En primer lugar se debe identificar y escoger cada componente. Este concepto comúnmente se conoce como *replicación*, es decir, asignarle a cada componente concreto un símbolo. En este ejemplo se ha llamado `u` a cada componente y se ha añadido un número para distinguirlos. En principio el nombre puede ser cualquier identificador válido y la única condición es que no haya dos nombres iguales. A continuación del nombre viene la entidad correspondiente al componente que, en este caso, puede ser una `and2`, una `or2`, o una puerta inversora `inv`. Después se realizan las conexiones situando cada señal en su lugar correspondiente con las palabras `PORT MAP`. De esta manera, los dos primeros argumentos en el caso de la puerta `and2` son las entradas, y el último es la salida. Con esta idea básica tan sencilla se va creando la lista de nodos o definición de la estructura. En este capitulo de introducción no se va a utilizar este método de descripción por ser más lejano a la descripción de alto nivel más cómoda y natural.

Aunque los dos primeros ejemplos se ajustan al VHDL'87, el último sólo es válido para el VHDL'93. Estas dos diversas especificaciones históricas de VHDL son muy similares y el VHDL'93 se puede decir que casi engloba completamente al VHDL'87, aunque hay alguna excepción. En cualquier caso se remite a la bibliografía para una especificación de las diferencias entre ambas versiones.

4.5 Elementos sintácticos

Realizar una descripción completa de todas las instrucciones y estructuras VHDL es objeto de una asignatura completa y no de este capítulo que es una mera introducción al VHDL. A pesar de este problema, se va a realizar una descripción de los conceptos fundamentales y se van a enumerar las palabras claves para que este capítulo sirva de referencia básica.

4.5.1 Señales, variables, constantes y tipos

El concepto de **señal** es similar al concepto de nodo en un circuito eléctrico. Pueden definirse cero o más señales dentro de la arquitectura que se corresponden a los cables de su esquema equivalente. Estas señales pueden leerse y escribirse solamente dentro de la arquitectura donde han sido definidos. La declaración de señales se realiza dentro de la arquitectura, pero antes del cuerpo de la arquitectura, indicando el nombre de la señal y el tipo.

```
ARCHITECTURE nombre_arquitectura OF nombre_entidad IS
  SIGNAL nombre_señal1 : tipo_señal_1 ;
  SIGNAL nombre_señal2 : tipo_señal_2 ;
  ...
BEGIN
  ...
END nombre_arquitectura;
```

Las **variables** VHDL son semejantes a las variables dentro de lenguajes software y similares a las señales, excepto que no tienen significado físico en el circuito. Por este motivo no se definen en el cuerpo de la arquitectura, tal como se hace con las señales. Por el contrario se definen y utilizan dentro de procedimientos, funciones y procesos VHDL.

Una **constante** por contra es un elemento que se inicializa a un determinado valor y no puede ser cambiado una vez inicializado.

La diferencia principal entre variables y señales es que una asignación a una variable se realiza de forma inmediata, es decir, la variable toma el valor que se le asigna en el mismo momento de la asignación. La señal, en cambio, no recibe el valor que se le está asignando hasta el siguiente paso de simulación, es decir, cuando el proceso se acaba al encontrar una sentencia `WAIT` dentro de un proceso o al final de éste si no tiene sentencias de espera.

Todas las señales, variables y constantes en un programa VHDL tienen asociado un **tipo**. Éste especifica el intervalo de valores que el objeto puede tomar. Adicionalmente hay un conjunto de operadores asociados con un tipo dado (como suma, AND, etc.) VHDL no define tipos pero sí permite definirlos con facilidad. Hay unos cuantos tipos predefinidos que están en cualquier librería VHDL y son estándares de hecho. De manera similar a otros lenguajes los tipos pueden ser **escalares** (tipos que expresan alguna magnitud) o **compuestos** (formados por la agrupación de tipos escalares). A continuación se muestran ejemplos de definición de tipos, indicándose los ya predefinidos en VHDL e introduciéndose la utilización de la palabra clave RANGE.

```
--Todos los ejemplos son de tipos escalares:

--Ejemplos de tipos enteros
TYPE byte IS RANGE 0 TO 255;
TYPE integer IS RANGE -2147483648 TO 2147483647;--Tipo predefinido

--Ejemplos de tipos reales o de coma flotante
TYPE consumo_de_mi_coche IS RANGE 0.0 TO 13.5;
TYPE real IS RANGE -1.0E38 TO 1.0E38;  --Tipo predefinido

--Ejemplo de tipo físico (tienen valor y unidades)
TYPE longitud IS RANGE 0 TO 1.0E9
UNITS
um;
```

```
mm=1000um;
m=1000mm;
inch=25.4 mm;
END UNITS;

--Ejemplos de tipos enumerados
TYPE valor_lógico IS (indefinido,alto,bajo,Z);
TYPE bit IS (0,1); --Tipo predefinido
```

Básicamente existen dos grupos de tipos compuestos: las matrices y los registros. A continuación se muestran varios ejemplos de estos tipos.

```
--Todos los ejemplos son de tipos compuestos:

--Ejemplos de tipos matrices
TYPE dword IS ARRAY(31 DOWNTO 0) OF bit;
TYPE matriz_transformada IS ARRAY(1 TO 4, 1 TO 4) OF real;
TYPE string IS ARRAY (positive RANGE <>) OF character; --Tipo predefinido
TYPE bit_vector IS ARRAY(natural RANGE <>) OF real;
```

En estos dos últimos casos cabe hacer notar como los rangos no están definidos en la declaración del tipo. Mas tarde en la declaración del dato que utiliza el tipo se deberá incluir la especificación de los límites:

```
SIGNAL password : string(1 TO 8);
```

A los elementos de la matriz se accede mediante el índice, de manera individualizada o mediante bloques de la misma dimensión entre el origen y el destino de la información.

4.5.2 Estructuras típicas en la arquitectura

Las estructuras descritas en lenguaje VHDL deben utilizar alguno de los estilos de descripción enumerados anteriormente o incluso una combinación de ellos. En esta sección simplemente se van a describir las características más importantes a tener en cuenta, ilustrándolas con ejemplos.

Diseño flujo de datos

En este estilo de descripción se utiliza una asignación de señales de manera concurrente y la estructura habitual utiliza la instrucción `WHEN .. ELSE` o la instrucción `WITH .. SELECT .. WHEN`. La estructura genérica de ambas instrucciones es similar. Para el caso de la instrucción `WHEN .. ELSE`:

```
[id_instr:]
señal<=[opciones] {forma_de_onda WHEN condición ELSE}
       forma_de_onda [WHEN condición];
```

Un ejemplo de utilización es el comparador de 2 vectores de 11 bits. Las 3 señales de salida deben estar definidas para todas las posibilidades de las condiciones.

```
ENTITY comparador IS
  PORT(a,b: IN bit_vector(10 DOWNTO 0);
       a_mayor_que_b,a_menor_que_b,a_igual_b: OUT bit);
END comparador;

ARCHITECTURE flujo_de_datos OF comparador IS
BEGIN
  a_mayor_que_b<='1' WHEN a>b ELSE '0';
  a_menor_que_b<='1' WHEN a<b ELSE '0';
  a_igual_b<='1' WHEN a=b ELSE '0';
END flujo_de_datos;
```

La otra estructura comúnmente utilizada para realizar descripciones flujo de datos es la instrucción `WITH .. SELECT .. WHEN`, que tiene la forma genérica:

```
WITH expresión SELECT
     señal <= [opciones] {forma_onda WHEN condición,} forma_onda WHEN condición;
```

En este caso todas las posibles condiciones deben cubrirse con la palabra clave `OTHERS`. El ejemplo siguiente muestra la utilización de la instrucción para definir el comportamiento de un decodificador BCD a 7 segmentos. En el caso de que el código de entrada no sea BCD no se activará ningún segmento del *display*.

```
WITH bcd SELECT
   abcdefg<="1111110"  WHEN "0000",
            "0110000"  WHEN "0001",
            "1101101"  WHEN "0010",
            "1111001"  WHEN "0011",
            "0110011"  WHEN "0100",
            "1011011"  WHEN "0101",
            "1011111"  WHEN "0110",
            "1110000"  WHEN "0111",
            "1111111"  WHEN "1000",
            "1110011"  WHEN "1001",
            "0000000"  WHEN OTHERS;
```

Diseño comportamental

La descripción flujo de datos es muy similar al hardware real. De hecho, también se conoce como descripción a nivel de transferencia de registros o RTL por el tipo de asignaciones que se producen. Este tipo de descripción es siempre concurrente y es similar a la descripción que se realiza con lenguajes software de alto nivel declarativos como el prolog. Sin embargo, esta forma de descripción esta todavía muy cerca del hardware final y no es realmente una descripción de alto nivel, lejana de los detalles de funcionamiento e implementación.

Los lenguajes de programación software son un ejemplo de lenguajes de descripción comportamental de alto nivel. La diferencia fundamental entre el estilo flujo de datos VHDL y un lenguajes software como el *C*, es el modo de ejecución. En VHDL se han visto estructuras que se ejecutan de forma concurrente, los lenguajes software se ejecutan de manera secuencial, permitiendo la utilización de estructuras como bucles que no son posibles en una ejecución concurrente.

La programación concurrente no es siempre la más cómoda para la descripción de sistemas, por lo que VHDL también permite la programación secuencial o en serie. En VHDL esta programación serie se define dentro de bloques `PROCESS`. Por tanto, siempre

que en VHDL se precise de una descripción en serie, se deberá utilizar un bloque de tipo `PROCESS`. Pueden haber múltiples bloques `PROCESS` en un mismo programa. En el caso de haber varios de estos bloques, cada uno de ellos equivale a una instrucción concurrente. Es decir, internamente a los `PROCESS` la ejecución de las instrucciones es secuencial, pero entre los propios bloques `PROCESS`, que coexisten con otras instrucciones concurrentes, la ejecución es en paralelo.

A continuación se muestra una versión de arquitectura comportamental del mismo comparador que se ha utilizado para ejemplificar la descripción flujo de datos.

```
ARCHITECTURE comportamental OF comparador IS
BEGIN
  PROCESS(a,b)  -- Este bloque se ejecuta cuando a o b cambian
  BEGIN
     IF a>b THEN
        a_mayor_que_b<='1';
        a_menor_que_b<='0';
        a_igual_b<='0';
     ELSIF a<b THEN
        a_mayor_que_b<='0';
        a_menor_que_b<='1';
        a_igual_b<='0';
     ELSE
        a_mayor_que_b<='0';
        a_menor_que_b<='0';
        a_igual_b<='1';
     END IF;
  END PROCESS;
END comportamental;
```

Es posible utilizar estructuras similares dentro de un `PROCESS` con otras instrucciones. Una de las más útiles con este estilo de descripción es la sentencia `CASE`, cuya estructura genérica tiene la forma:

```
CASE expresion IS
   WHEN caso1 =>
            instrucciones
   WHEN caso2 =>
            instrucciones
   ...
   WHEN OTHERS =>
            instrucciones
END CASE;
```

Unos ejemplos alternativos de descripción comportamental son los sistemas secuenciales. De esta manera, un simple biestable D sensible a flanco con señal de puesta a cero o *reset* asíncrona se podría describir:

```
PROCESS(clk,reset)
BEGIN
  IF reset='1' THEN q<='0';
  ELSIF clk'EVENT AND clk='1' THEN q<=d;
  END IF;
END PROCESS;
```

Es posible realizar la descripción de máquinas de estados de varias formas. En cualquier caso hay que tener en cuenta si el objeto del diseño es realizar una simulación o si además se va a sintetizar el sistema secuencial. En este segundo caso es aconsejable utilizar la referencia del manual de VHDL particular de la herramienta de síntesis y aplicar

el estilo recomendado. En cualquier caso es siempre aconsejable separar el almacenamiento de estados y cálculo del estado siguiente de la parte meramente combinacional de cálculo de la salida en función del estado presente y de la entrada.

```
ENTITY maquina IS
PORT (entrada: IN tipoin; clk,reset: IN bit; salida: OUT tipout);
END maquina;

ARCHITECTURE mealy OF maquina IS
TYPE estado IS (est1,est2,...,estN);
SIGNAL presente: estado:=est1;         -- el inicial
BEGIN

--Bloque de almacenamiento del estado y generación del estado siguiente
estados:
PROCESS(clk,reset)    -- reset asíncrono
BEGIN
   IF reset='1' THEN
     presente<=est1;    -- estado inicial
   ELSIF clk='1' AND clk'EVENT THEN
     CASE presente IS
       WHEN est1=>
          presente<=f1(entrada);
       WHEN est2=>
          presente<=f2(entrada);
       .
       .
       WHEN estN=>
          presente<=fN(entrada);
     END CASE;
   END IF;
END PROCESS estados;

-- Bloque combinacional de la función de salida
salida:
PROCESS(entrada,presente)
BEGIN
   CASE presente IS
     WHEN est1=>
        salida<=g1(entrada);
     WHEN est2=>
        salida<=g2(entrada);
     .
     .
     WHEN estN=>
        salida<=gN(entrada);
   END CASE;
END PROCESS salida;

END mealy;
```

Como ejemplo concreto se va a incluir la especificación VHDL de la máquina de estados que ha servido de ejemplo de síntesis de máquinas síncronas en la sección 3.2. La tabla 4.1 muestra la tabla de transiciones simplificada del sistema secuencial con una entrada `e` y una salida `s` que se describe a continuación.

```
ENTITY ejemplo_de_SS IS
 PORT (e, clk, reset:IN BIT; s: OUT BIT);
END ejemplo_de_SS;

ARCHITECTURE comportamiento OF ejemplo_de_SS IS
TYPE estado IS (A, B, C, D, E);
SIGNAL presente: estado:=A;
BEGIN
```

E.P.	E.S. / s	e=0	e=1
	A	B/0	C/0
	B	D/0	E/0
	C	E/0	D/0
	D	A/1	A/0
	E	A/0	A/1

Tabla 4.1: *Tabla de transiciones*

```
estados:
PROCESS(clk,reset)
BEGIN
   IF reset='1' THEN
      presente<=A;
   ELSIF clk='1' AND clk'EVENT THEN
     CASE presente IS
       WHEN A=>
        IF e='0' THEN presente<=B; ELSE presente<= C; END IF;
       WHEN B=>
        IF e='0' THEN presente<=D; ELSE presente<= E; END IF;
       WHEN C=>
        IF e='0' THEN presente<=E; ELSE presente<= A; END IF;
       WHEN D=>
        presente<=A;
       WHEN E=>
        presente<=A;
     END CASE;
   END IF;
END PROCESS estados;

salida:
PROCESS(e, presente)
BEGIN
   CASE presente IS
       WHEN A=>
        s<='0';
       WHEN B=>
        s<='0';
       WHEN C=>
        s<='0';
       WHEN D=>
        IF e='0' THEN s<='1'; ELSE s<='0'; END IF;
       WHEN E=>
        IF e='0' THEN s<='0'; ELSE s<='1'; END IF;
     END CASE;
END PROCESS salida;
END comportamiento;
```

4.5.3 Palabras clave y operadores

El VHDL'87 presenta las siguientes palabras clave:

```
ABS             ELSE            NAND            RETURN
ACCESS          ELSIF           NEW             SELECT
AFTER           END             NEXT            SEVERITY
ALIAS           ENTITY          NOR             SIGNAL
ALL             EXIT            NOT             SUBTYPE
AND             FILE            NULL            THEN
ARCHITECTURE    FOR             OF              TO
ARRAY           FUNCTION        ON              TRANSPORT
ASSERT          GENERATE        OPEN            TYPE
ATTRIBUTE       GENERIC         OR              UNITS
BEGIN           GUARDED         OTHER           UNTIL
BLOCK           IF              OUT             USE
BODY            IN              PACKAGE         VARIABLE
BUFFER          INOUT           PORT            WAIT
BUS             IS              PROCEDURE       WHEN
CASE            LABEL           PROCESS         WHILE
COMPONENT       LIBRARY         RANGE           WITH
CONFIGURATION   LINKAGE         RECORD          XOR
CONSTANT        LOOP            REGISTER
DISCONNECT      MAP             REM
DOWNTO          MOD             REPORT
```

El VHDL'93 completa el lenguaje con las siguientes palabras clave:

```
GROUP           REJECT          SRA
IMPURE          ROL             SRL
INERTIAL        ROR             UNAFFECTED
LITERAL         SHARED          XNOR
POSTPONED       SLA
PURE            SLL
```

Operador de concatenación

& Concatena matrices de manera que la dimensión de la matriz resultante es la suma de las dimensiones de las matrices sobre las que opera: `punto<=x&y` encaja en la matriz `punto` la matriz `x` en las primeras posiciones, y la matriz `y` en las últimas.

Operadores aritméticos

****** Sirve para elevar un número a una potencia: `4**2` es 4^2. El operador de la de la derecha sólo puede ser entero pero el operador de la izquierda puede ser entero o real.

ABS() Esta función devuelve el valor absoluto de su argumento que puede ser de cualquier tipo numérico.

***** Multiplica dos números de cualquier tipo (los tipos `bit` o `bit_vector` no son numéricos).

/ Divide datos de tipo numérico.

MOD Calcula el módulo de dos números, siendo el módulo el número entero que cumple que `a=b*N+(a MOD b)` donde **N** es un entero. Los operandos sólo pueden ser enteros. El resultado toma el signo de `b`.

REM Calcula el resto de la división entera y se define como el operador que cumple: `a=(a/b)*b+(a REM b)`, siendo la división entera. Los operandos sólo pueden ser enteros. El resultado toma el signo de `a`.

\+ Este operador sirve para indicar suma, si va entre dos operandos, o signo si va al principio de una expresión. La precedencia es diferente en cada caso. Opera sobre valores numéricos de cualquier tipo.

− Indica sustracción cuando va entre dos operandos y cambio de signo si va delante de una expresión.

Operadores de desplazamiento

SLL, SRL Desplaza un vector un número de bits hacia la izquierda (`SLL`)o hacia la derecha (`SRL`) rellenando con ceros los huecos libres. Se utiliza en notación infija de manera que la señal a la izquierda del operador es el vector que se quiere desplazar y el de la derecha es un valor que indica el número de bits a desplazar. Por ejemplo `dato SLL 3` desplaza a izquierda el vector `dato` 3 posiciones, es decir, lo multiplica por 8.

SLA, SRA En este caso el desplazamiento conserva el signo, es decir, conserva el valor que tuviera el bit más significativo del vector.

ROL, ROR Se realiza una rotación de los bits del vector.

Operadores relacionales

`=, /=` El primero devuelve `true` si los operandos son iguales y `false` en caso contrario. El segundo indica desigualdad, de esta manera su funcionamiento es el inverso. Los operandos deben ser del mismo tipo, sin importar cual sea éste.

`<,<=,>,>=` Tienen el significado habitual. La diferencia con los anteriores es que los tipos de datos que pueden manejar son siempre de tipo escalar o matrices de una sola dimensión de tipos discretos.

Operadores lógicos

Los operadores lógicos en VHDL son los habituales: `NOT, AND, NAND, OR, NOR, XOR` en VHDL'87 y `XOR` en VHDL'93. Actúan sobre los tipos `bit`, `bit_vector` y `boolean`. En el caso de realizarse estas operaciones sobre un vector, la operación se realiza bit a bit.

4.6 Conceptos de simulación

El lenguaje VHDL puede ser utilizado para modelar el comportamiento de sistemas, incluyendo la simulación como parte crucial para comprobar el correcto funcionamiento. Cabe hacer notar como el estilo de descripción utilizado no es importante si el simulador está bien construido. El simulador interpretará las instrucciones VHDL que definen el comportamiento del circuito y responderá a los estímulos propuestos en la simulación de la misma manera que lo haría el sistema.

El elemento más importante en una simulación temporal es el retraso que imponen sus componentes en la respuesta a los estímulos. En todo circuito digital además de la funcionalidad inherente a cada componente debe especificarse el retraso que éste tendrá en la respuesta si se quiere una simulación temporal fiable. De esta manera, la asignación de señales puede incorporar la especificación de retrasos. La salida `O` de un inversor con un tiempo de propagación de 50 ns con respecto a un cambio en la entrada `I` se podría especificar como:

```
O<=NOT I AFTER 50 ns;
```

La palabra clave `AFTER` indica al simulador que cuando cambie el valor de la señal de entrada `I` deberá cambiar la salida después de 50 ns.

En realidad el simulador VHDL realiza simulaciones guiadas por **eventos**. Intuitivamente es posible entender el funcionamiento de la simulación imaginando una lista de eventos temporales donde se van ordenando los cambios que se van a producir en las señales. La forma de gestionar la lista variará en función de si se definen **retrasos inerciales** o **retrasos transportados**. En el primer caso (caso por defecto) el simulador introduce en la lista de eventos de la salida `O` cuando debe cambiar la señal. Si se produce un segundo cambio previamente a que se haya reflejado el primer cambio en la salida éste segundo cambio substituye al primero. En el caso de que simplemente se añada el segundo cambio a la lista sin ninguna substitución se tratará de un retraso transportado. La figura 4.3 muestra la diferencia en el comportamiento de ambos tipos de retrasos.

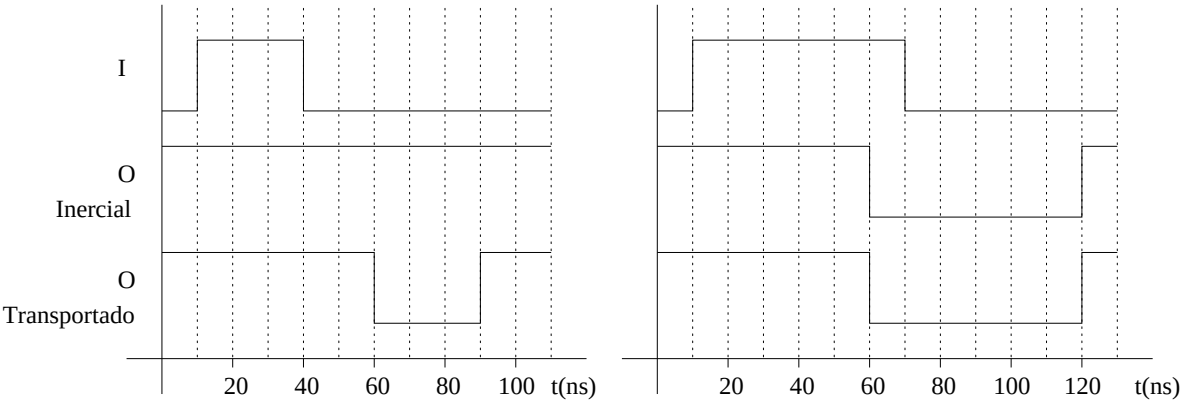


Figura 4.3: *Comportamiento de un inversor con retraso inercial (superior) y transportado (inferior)*

Ambos conceptos se revisarán en el capítulo 6 cuando se introduzcan los los parámetros dinámicos, incidiendo en el significado físico de cada tipo de retraso.

Una forma de verificar el correcto funcionamiento de un modelo VHDL mediante el uso de una herramienta de simulación consiste en cambiar las entradas y observar como evolucionan las salidas. Esta metodología de simulación interactiva puede ser útil para diseños sencillos, pero en el caso de modelos más complejos es mejor definir un subconjunto de entradas llamadas *patrones de verificación* o patrones de test, con las que comprobar el circuito o modelo VHDL. El conjunto de patrones de test se denomina globalmente **banco de pruebas**.

Con el lenguaje de descripción del hardware VHDL se puede modelar el banco de pruebas independientemente de la herramienta de simulación. Además, cabe añadir que el mismo modelo de banco de pruebas puede ser utilizado en cualquier fase del diseño con VHDL, permitiendo un importante ahorro de tiempo y esfuerzo.

El banco de pruebas es una entidad sin entradas ni salidas en su tipo más simple. La arquitectura del banco de pruebas, de tipo estructural, tiene como señales internas las entradas y salidas del circuito. El único componente es el correspondiente a la entidad que se desea simular. Si se realiza una descripción de un modelo VHDL se puede simultáneamente describir un banco de pruebas para éste. Si posteriormente se sintetiza el modelo, el mismo banco de pruebas utilizado para verificar la descripción pre-síntesis, puede ser utilizado para simular el modelo VHDL post-síntesis generado por la herramienta. La construcción de un banco de pruebas para un modelo determinado puede abordarse de varias formas según la generación de los vectores de test.

4.7 Conceptos de síntesis

La **síntesis** es el proceso mediante el cual a partir de una descripción VHDL se construye un circuito que se comporta exactamente como la descripción especifica. De esta manera se reduce el nivel de abstracción de la descripción de un circuito hasta convertirlo en una definición puramente estructural (lista de nodos) cuyos componentes son los elementos de una determinada biblioteca, que dependerá del circuito que se quiera realizar, la herramienta de síntesis, etc.

Inicialmente, cualquier descripción en VHDL es sintetizable, no importa el nivel de abstracción que la descripción pueda tener ya que cualquier descripción en VHDL se puede simular y por tanto corresponde a un circuito, que en última instancia es el propio simulador (un ordenador ejecutando un programa).

La complejidad del circuito resultante, y también incluso la posibilidad o no de realizar el circuito, va a depender sobre todo del nivel de abstracción inicial que la descripción tenga. La aproximación de las herramientas de síntesis consiste en, partiendo de la descripción original, reducir el nivel de abstracción hasta llegar a un nivel de descripción estructural. La síntesis es por tanto una tarea vertical entre los niveles de abstracción de un circuito. De esta manera, una herramienta de síntesis comenzará por la descripción comportamental abstracta y secuencial e intentará traducirla a un nivel de transferencia entre registros descrita con ecuaciones de conmutación. A partir de esta descripción se intentará transformar a una descripción estructural donde se realiza la asignación con los componentes de una biblioteca especial que depende de la tecnología con la cual se quiera realizar el circuito.

Las herramientas de síntesis actuales cubren a la perfección la síntesis a partir de descripciones RTL y estructurales, pero muchas veces no funcionan si el diseño se encuentra descrito en un nivel de abstracción más alto. Es por tanto interesante seguir una serie de consejos que harán más sintetizable el circuito. Estas líneas de actuación serán siempre dependientes de la herramienta concreta de síntesis y pueden ir desde la sugerencia de un estilo para describir máquinas de estados a imponer una serie de restricciones en el juego de instrucciones. En cualquier caso sí que es posible identificar unas actuaciones típicas que simplificarán la tarea de cualquier situación y que a continuación se enumeran.

1. **Evitar las cláusulas temporales y de espera** como `AFTER`. Útiles en modelos de simulación pero complicadas o imposibles de sintetizar.
2. **Identificar las puertas lógicas con claridad** ya que generalmente tienen una estructura clara e incluso se pueden utilizar comandos directos que realizan estas funciones. No es conveniente en estos casos realizar descripciones comportamentales complejas.
3. **Vigilar las listas sensibles** ya que la mayoría de sintetizadores admiten la lista sensible en los `PROCESS` , pero no siempre la interpretan como lo haría un simulador (en determinadas ocasiones el proceso se ejecutará cuando cambia una señal que se encuentra en el proceso pero no en la lista sensible.)
4. **Limitar las señales de reloj** ya que normalmente sólo se permite una señal de reloj por proceso, y además debe especificarse claramente el flanco de subida del reloj mediante la condición `clk='1' AND clk'EVENT`. En general sólo puede indicarse esta condición una vez por proceso y en ningún caso se puede añadir `ELSE` en el `IF` en el que se usó la condición.
5. **Realizar asignaciones únicas** ya que aunque en simulación es bastante corriente que a una señal se le asignen varios valores a lo largo de un mismo proceso, en síntesis esto resulta difícil de interpretar.
6. **Evitar `IF`s anidados.** Normalmente las herramientas tienden a no sintetizar de manera óptima varios condicionales anidados entre sí. De esta manera, es preferible utilizar `CASE` antes que varios `IF`s. Las estructuras `CASE` tienen para los sintetizadores un modelo optimizado de síntesis.
7. **Utilizar el estilo indicado para las máquinas de estado.** En VHDL hay muchas maneras para poder describir máquinas de estados, pero no todas son reconocibles por un sintetizador. En los manuales de la herramienta específica se indica habitualmente el estilo a seguir para obtener un resultado final óptimo.

4.8 Conclusiones

La descripción de circuitos mediante VHDL tiene varias ventajas respecto a la descripción esquemática (además de su estandarización.) De hecho, la descripción esquemática se puede considerar una representación gráfica de la descripción estructural VHDL, con lo que quedaría englobada dentro de las posibilidades del lenguaje.

Con VHDL se pueden realizar descripciones de la funcionalidad desde un alto nivel, sin detallar la implementación concreta del sistema, lo cual aumenta la portabilidad de los diseños. Con la funcionalidad así descrita se puede completar todo el ciclo de diseño de manera cómoda (simulación en cualquiera de las fases y posterior síntesis.)

El lenguaje incluye todas las estructuras necesarias para la simulación de cualquier sistema digital sea cual sea el estilo de descripción. Para realizar la síntesis a partir de la especificación simulada sí que deben seguirse una serie de consejos que faciliten la labor de los sintetizadores.

Debido a todas las ventajas que VHDL aporta al diseño hardware éste se ha convertido en una herramienta básica para la ingeniería de sistemas.

Capítulo 5

Módulos Secuenciales

5.1 Introducción

Para realizar el diseño de lógica combinacional no siempre se utilizan puertas lógicas. Se definen una serie de módulos con funciones comunes, a partir de los cuales se puede abordar de manera más sencilla ciertas funciones lógicas. Estos módulos MSI[1] son los multiplexores, codificadores y decodificadores. Los módulos combinacionales incorporan una serie de funciones lógicas que pueden ser utilizadas para realizar funciones combinacionales.

Para la lógica secuencial también se puede hacer lo mismo, definiendo módulos secuenciales MSI síncronos. Estos módulos secuenciales simplificarán el diseño de los S. S. en ciertos casos, almacenando el estado presente y parte (o la totalidad) de la lógica combinacional que genera el estado siguiente. De esta manera no habrá que recurrir siempre al **diseño canónico** con biestables, metodología descrita en el capítulo 3.

Una buena revisión de módulos secuenciales y diseño de sistemas secuenciales se puede encontrar en [Wak00], [GA95] y [Toc93].

5.2 Registros

Se llamará registro de n bits a un circuito secuencial síncrono que puede almacenar n bits. Un registro es un circuito de memoria temporal de pequeña capacidad y alta velocidad. La forma de entrada y salida de los datos (serie o paralelo) y el número de bits que almacene distinguirá a un registro de otro.

5.2.1 Registros paralelos

Un registro paralelo se puede conseguir sin más que conectar en paralelo n biestables de tipo D, como es el caso del registro de 8 bits de TEXAS SN74AS374, que además incorpora salida triestado. Los registros pueden tener además una señal de carga síncrona

[1] Las siglas MSI corresponden a las iniciales de *Medium Scale of Integration*: Media escala de integración

L (*Load*), y una puesta a cero (*clear*) que suele ser asíncrona. El comportamiento y símbolo de un registro de n bits con señal de carga L y $\overline{CLEAR}$ se muestra en la figura 5.1. Un ejemplo de registro paralelo con carga síncrona es el SN74AS195, que además incorpora la posibilidad de hacer desplazamientos.

I	CLK	LOAD	$\overline{\text{CLEAR}}$	Q_{t+1}
A	↑	1	1	A
X	X	X	0	0
X	X	0	1	Q_t

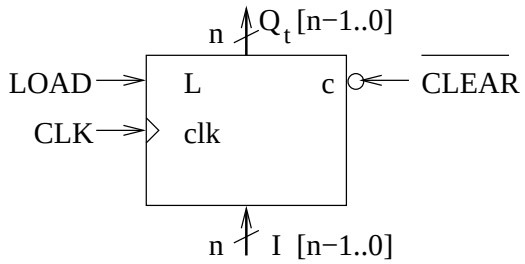


Figura 5.1: *Tabla de verdad y símbolo de un registro paralelo de n bits con señal de carga síncrona*

En las señales de control de un registro paralelo pueden darse ligeras variaciones, como que no se disponga de una señal de carga síncrona L, que la puesta a cero sea síncrona o que también haya una puesta a uno o *preset*. Análogamente los registros pueden ser sensibles a flanco de bajada en vez de a flanco de subida.

Un caso particular de los registros de n bits son los **cerrojos** (*latches*). Un cerrojo es un biestable sensible a nivel que no tiene señal de reloj CLK y carga L pero tiene una señal de *enable* que habilita el que el biestable siga a sus entradas. Los cerrojos suelen ser construidos con registros paralelos de n bits de tipo D, pero también los hay de otros tipos como latches SR (p. ej. SN74LS279).

Los registros de n bits se pueden implementar a partir de los diversos tipos de biestables. En la figura 5.2 se muestra como se ha realizado con biestables JK maestro-esclavo un registro de n bits.

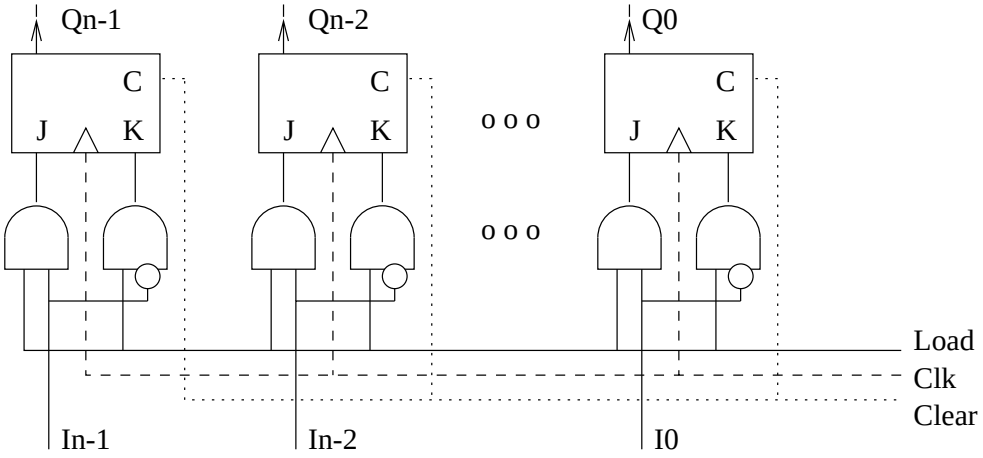


Figura 5.2: *Registro de n bits realizado con biestables JK*

La ampliación de registros es sencilla. A partir de registros de n bits se pueden construir registros de $2n$ bits, simplemente poniéndolos en paralelo. Añadir bits de forma individual es también sencillo a partir de biestables D o JK, tal como se muestra en la figura 5.2.

5.2.2 Registros de desplazamiento

Los registros de desplazamiento de n bits son registros capaces de transferir datos a los elementos de memoria vecinos. Los hay de desplazamiento a la derecha, desplazamiento a la izquierda y bidireccionales. La carga puede ser en paralelo o serie, definiéndose a partir del tipo de carga/salida los registros:

- **PIPO**: *Parallel Input Parallel Output.* Son los registros con entrada y salida en paralelo descritos en la sección anterior.
- **SIPO**: *Serial Input Parallel Output.* Son registros de entrada serie y salida en paralelo un ejemplo de como se pueden implementar estos registros se muestra en la figura 5.3.

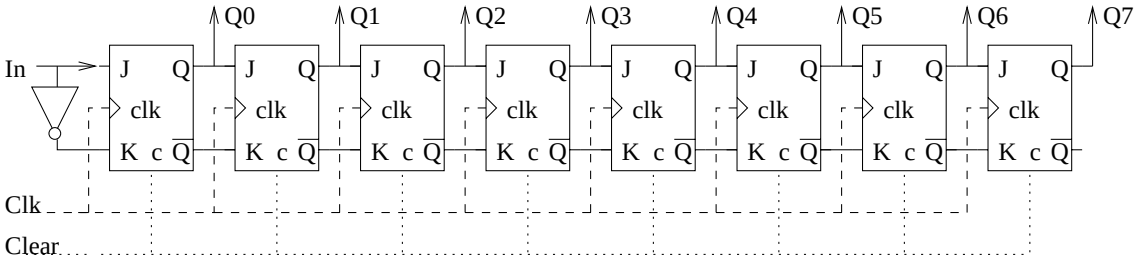


Figura 5.3: *Registro de 8 bits SIPO realizado con biestables JK*

- **PISO**: *Parallel Input Serial Output.* Este registro tiene entrada en paralelo y salida en serie. El registro SN74165 es un ejemplo de este tipo de dispositivos que se suelen utilizar, combinados con los registros SIPO, para realizar transmisión de datos serie.
- **SISO**: *Serial Input Serial Output.* Este registro tiene la entrada y salida en serie. Estos dispositivos se utilizan como memoria intermedia en periféricos o para introducir ciclos de retraso en sistemas secuenciales.

De todas maneras lo más habitual es utilizar **registros universales** que permiten las cuatro posibilidades de transmisión de datos. Para configurar un registro universal será necesario especificar el tipo de desplazamiento de los datos mediante señales de control. Un ejemplo de un registro universal es el SN74198, cuya tabla de verdad se muestra en la tabla 5.1.

El registro SN74198 tiene un *clear* asíncrono $\overline{C}$, dos entradas serie S_{left} y S_{right}, 8 entradas en paralelo señaladas como P=a..h, y dos entradas de control S_1 y S_0. Con las entradas de control se selecciona el modo de funcionamiento del registro universal. Así si S_1S_0=00 el registro mantiene el estado presente, si S_1S_0=01 se produce un desplazamiento a la izquierda pasando la entrada en S_{right} a ser almacenada en el primer bit Q_A. Si por el contrario S_1S_0=10 se produce un desplazamiento a la derecha pasando la entrada en S_{left} al bit más alto Q_H. Finalmente si la entrada es S_1S_0=11 se produce una carga en paralelo por los 8 bits de la señal P.

Se puede construir un registro universal a partir de biestables y lógica combinacional. Como ejemplo se muestra el diseño realizado con biestables D y multiplexores de la figura 5.4. El comportamiento de este registro, gobernado por las señales de control C_1 y C_0, es idéntico al SN74198. De esta manera con C_1C_0=00 se mantiene el estado presente, con C_1C_0=10 se produce un desplazamiento a la derecha, con C_1C_0=01 se desplaza a la izquierda, y con C_1C_0=11 se realiza una carga en paralelo. En el esquema

Entradas							Salidas			
$\overline{C}$	S_1	S_0	clk	S_{left}	S_{right}	P	$Q_H(t+1)$	$Q_G(t+1)..$	$..Q_B(t+1)$	$Q_A(t+1)$
L	X	X	X	X	X	X	0	0	0	0
H	X	X	L	X	X	X	$Q_H(t)$	$Q_G(t)..$	$..Q_B(t)$	$Q_A(t)$
H	H	H	↑	X	X	h..a	h	g..	..b	a
H	L	H	↑	X	H	X	$Q_G(t)$	$Q_F(t)..$	$..Q_A(t)$	H
H	L	H	↑	X	L	X	$Q_G(t)$	$Q_F(t)..$	$..Q_A(t)$	L
H	H	L	↑	H	X	A..H	H	$Q_H(t)..$	$..Q_C(t)$	$Q_B(t)$
H	H	L	↑	L	X	A..H	L	$Q_H(t)..$	$..Q_C(t)$	$Q_B(t)$
H	L	L	X	X	X	X	$Q_H(t)$	$Q_G(t)..$	$..Q_B(t)$	$Q_A(t)$

Tabla 5.1: *Tabla de verdad del registro universal SN74198*

de la figura 5.4 se han omitido las líneas C_1C_0 y clk, que irían en paralelo, para aumentar la claridad del esquema.

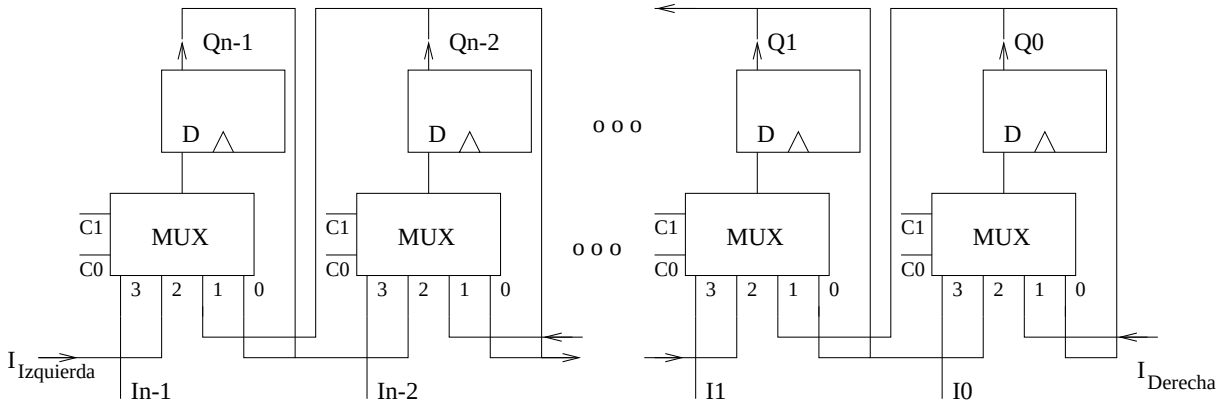


Figura 5.4: *Diseño de un registro universal*

5.2.3 Descripción VHDL de registros

La descripción de cualquier registro es posible en cualquiera de los niveles de abstracción que soporta VHDL. A modo de ejemplo se incluye una descripción VHDL de un registro universal cuyo comportamiento se especifica a continuación. La descripción sería comportamental por introducir la estructura `PROCESS` pero la definición del comportamiento es muy próxima al flujo de datos. Cabe hacer notar que la descripción es claramente mejorable utilizando los operadores lógicos de desplazamiento. La señal `c` es la señal de puesta a cero asíncrona activa a nivel bajo, `e[7:0]` son las 8 entradas en paralelo, `ed` y `ei` son las entradas en los bordes, útiles cuando hay desplazamientos, `q[7:0]` son las 8 salidas o estado presente y las señales `m[1:0]` indican el modo de funcionamiento. Cuando `m[1:0]=10` se realiza un desplazamiento a la derecha, pasando la entrada `ed` a ser el bit más significativo q(7).

```
LIBRARY IEEE;
USE ieee.std_logic_1164.all; -- Se va a usar STD_LOGIC

ENTITY desplazamiento IS
PORT(c, clk, ed, ei : IN std_logic;
     m : IN std_logic_vector(1 DOWNTO 0);
     e : IN std_logic_vector(7 DOWNTO 0);
     q : BUFFER std_logic_vector(7 DOWNTO 0));
END desplazamiento;

ARCHITECTURE comportamiento OF desplazamiento IS
BEGIN
  PROCESS (clk, c)
  BEGIN
    IF c='0' THEN q<="00000000";
    ELSIF (clk'EVENT AND clk='1') THEN
      CASE m IS
        WHEN "10" => q(0)<=q(1); q(1)<=q(2); q(2)<=q(3); q(3)<=q(4); --Desplazamiento a
                     q(4)<=q(5); q(5)<=q(6); q(6)<=q(7); q(7)<=ed;   --la derecha
        WHEN "01" => q(7)<=q(6); q(6)<=q(5); q(5)<=q(4); q(4)<=q(3); --Desplazamiento a
                     q(3)<=q(2); q(2)<=q(1); q(1)<=q(0); q(0)<=ei;   --la izquierda
        WHEN "11" => q<=e;   --Carga en paralelo
        WHEN OTHERS => q<=q; --El estado presente no cambia
      END CASE;
    END IF;
  END PROCESS;
END comportamiento;
```

5.3 Contadores

Otros módulos secuenciales muy útiles que se pueden utilizar en el diseño de sistemas secuenciales son los contadores. Los contadores como su propio nombre indica van a seguir una secuencia de cuenta. El código de la secuencia podrá ser de diferentes tipos y distinguirá los diversos tipos de contadores.

Los contadores son sistemas secuenciales síncronos, desde el punto de vista que usan biestables síncronos maestro-esclavo. Pero hay una familia de contadores que, al no compartir todos los biestables la señal de reloj en paralelo, se llaman **contadores asíncronos** aunque utilicen biestables síncronos.

5.3.1 Contadores asíncronos

Estos contadores se caracterizan por tener un tipo de conexionado donde la señal de reloj para los diversos biestables se va generando de manera secuencial. Un estructura típica de contador asíncrono es la que se muestra en la figura 5.5. Los biestables tipo T tiene su entrada siempre conectada a '1', con lo que en cada flanco de reloj invertirán su estado (tal como se indicó en la sección 1.5.3). Este tipo de contadores también se conocen por el nombre de **contadores *Ripple*** o de rizado.

Una característica fundamental de estos dispositivos es que su frecuencia de funcionamiento viene limitada por el número de bits del contador. Esto se debe a que la señal de reloj no va en paralelo a todos los biestables y que desde que se produce el flanco de reloj en el primer biestable, hasta que llega el flaco de reloj al último biestable se acumulan los retrasos de todos los biestables.

También cabe hacer notar como las señales Q_i en realidad están dividiendo la fre-

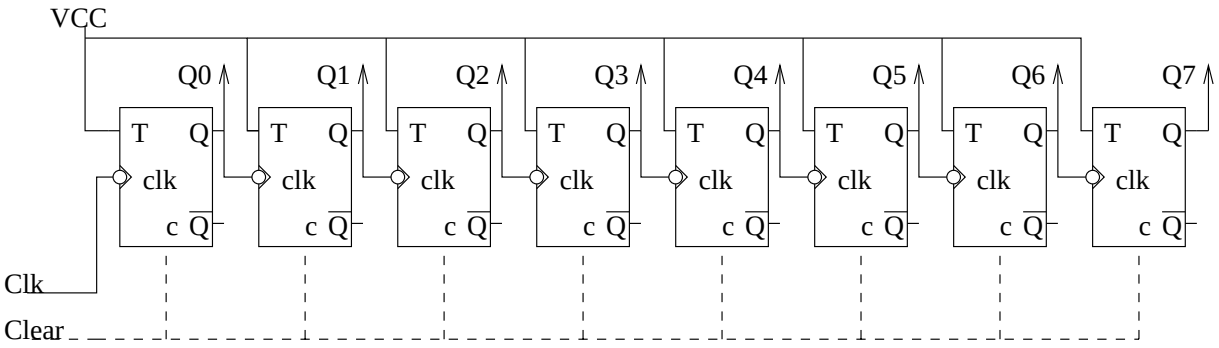


Figura 5.5: *Contador asíncrono de 8 bits*

cuencia de funcionamiento de la señal clk. De hecho, si se realiza una simulación del sistema de la figura 5.5, se observará como la señal Q_0 tiene un periodo que es el doble de clk (la mitad de la frecuencia), la señal Q_1 tendrá la frecuencia de clk dividida por 4, etc. Se puede concluir que en un contador asíncrono de n bits, se cumplirá que la frecuencia ν_n a la que cambia la última señal Q_n será:

$$\nu_n = \frac{\nu}{2^n} \tag{5.1}$$

Donde ν es la frecuencia de clk.

5.3.2 Contadores síncronos

Este tipo de contadores son los de utilización más normal y se caracterizan por que la señal de reloj llega en paralelo a todos los biestables, y su diseño se aborda con la metodología normal descrita en la sección 3.2.

Un contador síncrono típicamente tendrá el símbolo que se muestra en la figura 5.6 y las señales externas:

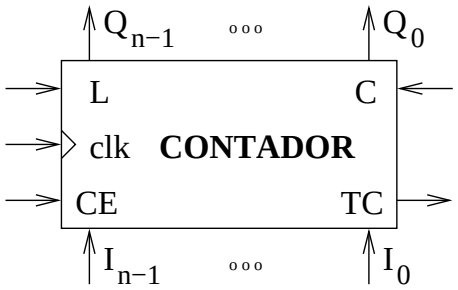


Figura 5.6: *Señales asociadas a un contador típico*

- clk: Señal de reloj. El contador será siempre sensible a flanco para tener una evolución controlada.
- L: Señal de carga síncrona($Load$). Cuando sea activa esta señal y llegue un flanco de reloj se carga el dato presente en la entrada paralelo y $Q_i(t+1) = I_i$.
- CE: Habilitación de cuenta ($Count\ Enable$). Cuando esta señal está habilitada el estado siguiente será el que siga al estado presente, según el tipo de cuenta

- I_i: Bits de entrada al sistema. Se cargan en el registro del contador cuando se habilita la señal L y llega un flanco de reloj.
- Q_i: Estado presente del contador. Evoluciona siguiendo la cuenta marcada si la señal CE está habilitada.
- TC: Señal de final de cuenta (*Terminal Count*). Esta salida (opcional) indica el final de la cuenta.

Muchos contadores tienen además una señal adicional de control con la que se indica el *sentido* de la cuenta. Estos contadores se suelen llamar **contadores reversibles** (*up-down*). La señal de TC no es muy habitual ya que en general los contadores, una vez llegados al último estado de la cuenta, empiezan por el primero de nuevo.

Un contador se dice que es **módulo N** cuando la secuencia total tiene N estados. Así por ejemplo un contador binario de 4 bits será módulo 16 porque hay 16 estados distintos.

Un contador se dirá que es **cíclico** cuando el contador después del último estado de la secuencia de cuenta, sigue con el primero. Así por ejemplo un contador binario módulo 16 cíclico realizará la cuenta ... 1110 $\rightarrow$ 1111 $\rightarrow$ 0000 $\rightarrow$ 0001 ... etc.

Los sistemas secuenciales se podrán realizar almacenando el estado presente en los biestables del contador y utilizando lógica combinacional para controlar la secuencia de cuenta. Estas ideas se describirán con más detalle en la sección 5.4.2.

Análogamente la descripción VHDL de contadores es inmediata a partir de la descripción VHDL de máquinas de estados realizada en la sección 4.5.2. Un contador no es más que una máquina de Moore donde en función de la entrada se tendrá la evolución del sistema. Se deja como ejercicio completar la descripción comportamental de un contador genérico.

Las señales L y CE hay veces que se funden en una sola señal teniendo la señal $L/\overline{CE}$. Esto tiene ventajas en algunos diseños de S. S., adicionalmente a la ventaja de que se tendrá que implementar una función menos que controle el contador.

Los contadores se pueden clasificar por el tipo de cuenta que realizan. No todos realizan el tipo de cuenta binaria de números naturales. Hay códigos de cuentas un poco más extraños que tienen cierta utilidad en algunas aplicaciones. Los más habituales son:

- **Contadores binarios**: La secuencia generada es la binaria habitual. Un ejemplo de secuencia de un contador ascendente de 4 bits sería: ... 1100, 1101, 1110, 1111, 0000, 0001...
- **Contadores de anillo**: Hay tantos estados como biestables y la secuencia de cuenta consiste en el desplazamiento de un solo 1 a través éstos de forma indefinida: 0001, 0010, 0100, 1000, 0001, 0010... Estos contadores siguen la codificación uno-activo vista en la sección 3.2.3.
- **Contadores BCD**: Se sigue una secuencia BCD, útil típicamente para observar el estado con un display de siete segmentos.
- **Contadores de código Gray**: Son códigos donde un estado difiere del anterior sólo en un bit. Un ejemplo sería 000, 001, 011, 010. La utilidad de esta codificación está en transmisiones, tolerancia a fallos y codificación en discos ópticos.
- **Contadores Jonhson**: El código consiste en un desplazamiento de unos y ceros de manera que nunca hay unos o ceros sueltos. Un ejemplo sería el código Jonhson de 4 bits (módulo 8) cuya secuencia sería: 0000, 0001, 0011, 0111, 1111, 1110, 1100 y 1000.

5.3.3 Modificación de la secuencia de cuenta

Puede ocurrir que se necesite adaptar un contador para que haga una cuenta más adecuada, o bien acortando la cuenta en los estados superiores. o bien empezando la cuenta en estados intermedios. A continuación se describen algunas técnicas para ello.

Reducción de la secuencia de cuenta

Esto se puede realizar cuando el contador tiene una señal de carga síncrona L. Se debe detectar el último estado de la cuenta con lógica combinacional para activar la señal L y así cargar el nuevo estado inicial.

En la figura 5.7 se muestra un contador módulo 16 que se ha adaptado para que haga una cuenta de 0010 a 1011, es decir se ha reducido la secuencia de cuenta para hacer un contador módulo 10.

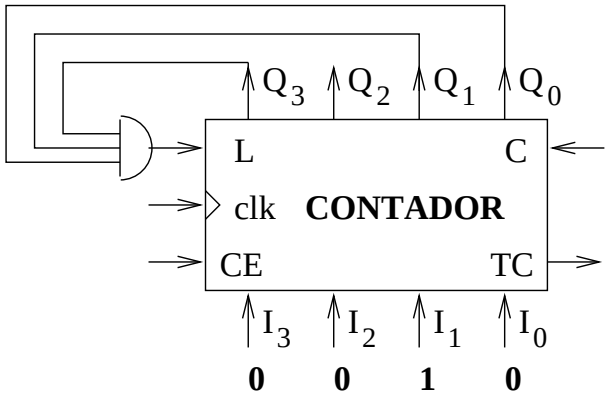


Figura 5.7: *Contador módulo 10 realizado a partir de un contador módulo 16*

Se llama **divisor módulo n** al contador módulo n cuando se permite la cuenta hasta el final de la secuencia cargándose un estado inicial intermedio. Esto se podría hacer con TC directamente conectada a L. De esta manera a partir de un contador módulo 16 con TC conectada a L, y con la entrada 0111 se realiza un divisor módulo 9 (16-7). El nombre de divisor viene porque los bits de la cuenta binaria dividen la frecuencia de funcionamiento del reloj.

Ampliación de contadores

Puede darse el caso de que se necesite realizar una cuenta más larga que la que se pueda realizar con los contadores disponibles. Habitualmente se requerirá controlar las señales de habilitación de cuenta CE mediante lógica combinacional. Esto se podrá realizar de manera muy sencilla en el caso de que los contadores dispongan de señal de final de cuenta TC y continúen la cuenta de manera indefinida.

La figura 5.8 nuestra este tipo de ampliación. Se partirá de una puesta a cero general, con lo que el estado de ambos contadores será 0000 0000. El contador menos significativo seguirá su cuenta normal al tener habilitada su señal CE. Sin embargo el contador más significativo no contará al estar su señal CE conectada a TC del otro contador y ser ésta 0. Cuando el contador menos significativo llegue a 1111 entonces su

señal TC será igual a 1, habilitando la cuenta del otro contador en el siguiente flanco, por tanto en el siguiente flanco se tendrá 0001 0000. Esto hará que de nuevo se deshabilite la cuenta del contador más significativo, por tanto la secuencia seguirá la secuencia correcta 0000 1111 → 0001 0000 → 0001 0001 → 0001 0010 → 0001 0011, etc. En el esquema de la figura 5.8 no se han dibujado las líneas de carga L, puesta a cero C y reloj clk para una mayor claridad. Todas estas líneas irían en paralelo a ambos contadores.

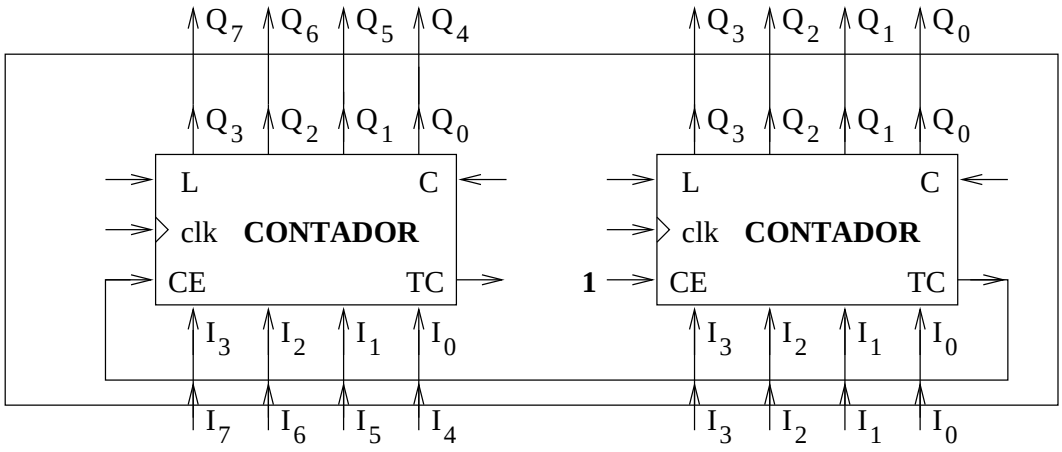


Figura 5.8: *Ampliación de la secuencia de cuenta*

5.4 Diseño de S. S. con módulos estándar

Los módulos estándar secuenciales se utilizarán para la implementación de S.S., de la misma manera que los módulos MSI combinacionales también eran útiles para implementar sistemas combinacionales. Así existirán ciertos tipos de S.S. cuya implementación será mucho más sencilla y rápida utilizando contadores o registros. A continuación se muestran algunas técnicas para sintetizar S.S. con módulos secuenciales.

5.4.1 Diseño con registro de estado y ROM

Sea un S. S. definido por los siguientes parámetros:

- n: número de bits de entrada.
- m: número de bits de salida.
- k: siendo $k = \lceil log_2 N \rceil$ donde N es el número de estados. Es decir k será el número entero igual o inmediatamente superior que $log_2 N$.

Con estos parámetros se puede construir un S. S. (máquina de Mealy) con un registro de estado de k bits para almacenar el estado presente, y una ROM de 2^{n+k} `x` $(k+m)$. Es decir una ROM con un ancho de bus de direcciones de $n+k$ bits y un ancho de bus de datos de $k+m$ bits.

En la figura 5.9 se muestra la arquitectura de una máquina de Mealy genérica construida a partir de los parámetros descritos anteriormente. Cabe hacer notar como en el registro de estado se almacena el estado presente, que va cambiando en cada pulso

de reloj. La ROM actúa como un mero circuito combinacional, ya que genera el estado siguiente y la salida en función de la entrada y del estado presente. Es fácil darse cuenta como en cada palabra de la ROM se almacena el estado siguiente y la salida. Las direcciones de la palabras están formadas por la entrada y el estado presente que se realimenta, valores de los que depende el estado siguiente y la salida.

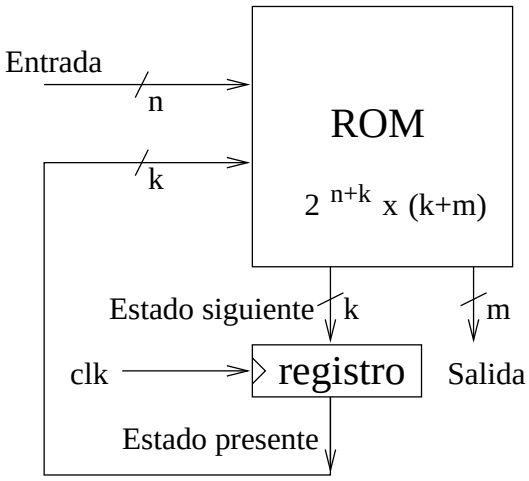


Figura 5.9: *Arquitectura genérica de un S. S. implementado con ROM + registro de estado*

Cuando llega un flanco al registro de estado entonces la entrada del registro (estado siguiente) pasará a almacenarse (estado presente). Una vez cambia el estado presente cambiará también la dirección de la ROM, por tanto cambiará la palabra que aparece en el bus de datos, que no es más que la salida y el estado siguiente.

Como ejemplo se va a realizar siguiendo este método el S. S. representado en el grafo de la figura 5.10. Es conveniente representar previamente el sistema secuencial con una tabla, de esta manera se rellenará la ROM de una manera mucho más directa. La tabla 5.2 representa la misma máquina de estados.

Esta máquina de estados tiene una sola entrada E y una sola salida S, con lo que $n = m = 1$. El número de estados es 5 con lo que $k = \lceil log_2 5 \rceil = 3$. En la figura 5.11 se muestra el esquema de conexionado y en la tabla 5.3 el contenido de la ROM correspondiente. Cabe hacer notar que las celdas de memoria rellenadas con indeterminaciones corresponden a estados a los que no se va a acceder. Es conveniente tener en cuenta que en ciertos casos (máquinas de estados que controlen sistemas peligrosos) si se producen estas transiciones no previstas por problemas de ruido se deberá acceder a un estado seguro o un estado donde se activará una señal de error.

Este tipo de diseños puede llevar, si se tiene un gran número de entradas, a utilizar memorias de gran tamaño (estando la mayoría de posiciones vacías). Esto se puede evitar utilizando un registro de estado, un multiplexor y una ROM en el caso de que: **1)** desde cada estado se puede acceder a un máximo de 2 estados sucesores y **2)** La transición de un estado a otro dependa como mucho de una entrada (pudiendo ser ésta incondicional).

$Q_2Q_1Q_0(t)$	$Q_2Q_1Q_0(t+1)/S$	E=0	E=1
	000	000/0	001/0
	001	001/0	010/0
	010	010/0	011/0
	011	010/0	100/1
	100	000/1	000/1

Tabla 5.2: *Tabla de transiciones del sistema secuencial de la figura 5.10*

Dirección		Dato	
$A_3A_2A_1$	A_0	$D_3D_2D_1$	D_0
$Q_2Q_1Q_0(t)$	E	$Q_2Q_1Q_0(t+1)$	S
000	0	000	0
000	1	001	0
001	0	001	0
001	1	010	0
010	0	010	0
010	1	011	0
011	0	010	0
011	1	100	1
100	0	000	1
100	1	000	1
101	0	$\phi\phi\phi$	ϕ
101	1	$\phi\phi\phi$	ϕ
110	0	$\phi\phi\phi$	ϕ
110	1	$\phi\phi\phi$	ϕ
111	0	$\phi\phi\phi$	ϕ
111	1	$\phi\phi\phi$	ϕ

Tabla 5.3: *Contenido de la ROM de la figura 5.11*

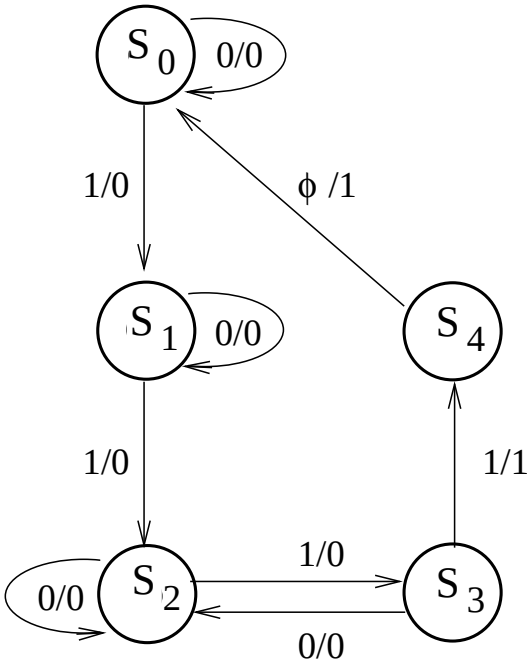


Figura 5.10: *Grafo del sistema secuencial ejemplo de diseño con ROM*

5.4.2 Diseño con contador y lógica combinacional

Existen otras metodologías para la síntesis de sistemas secuenciales basándose en módulos estándar secuenciales. De la misma manera que se ha mostrado en la sección anterior la estrategia a seguir en diseños con registros y ROM, se puede realizar lo mismo utilizando contadores. En este caso el método consistirá en controlar las entradas del contador para que sigan la secuencia deseada en vez de la secuencia de cuenta normal.

Claramente los sistemas secuenciales más fáciles de implementar serán los que presenten un grafo similar a la secuencia de cuenta original del contador. De esta manera, se podrá realizar el grafo sin tener que realizar funciones combinacionales complejas que controlen el contador.

Cuando un sistema secuencial siga la cuenta binaria habitual y, llegado a un punto y cumpliéndose una cierta condición se modifique esta secuencia, se deberá hacer que el estado *salte* a este estado. La manera habitual será detectar el estado y la condición de ruptura de la secuencia, y mediante lógica combinacional generar la señal de carga L y deshabilitar la cuenta CE. Simultáneamente en la entrada paralela del contador se generará el estado a cargar.

Análogamente si lo que se produce es una vuelta atrás en la secuencia de cuenta y el contador es reversible, se puede modificar la señal que indica el sentido de la cuenta con la señal *up/down*.

La circuitería combinacional que controle las entradas del contador se diseñarán de la manera habitual (mapas de Karnaugh) en función del estado presente y de las entradas. De la misma manera se puede utilizar una ROM, en el lugar de puertas lógicas, para implementar las funciones combinacionales de control del contador. Es más, se pueden simplemente utilizar los biestables del contador teniendo siempre habilitada la señal de carga, y generar el estado siguiente en función de la entrada y del estado presente. Esta

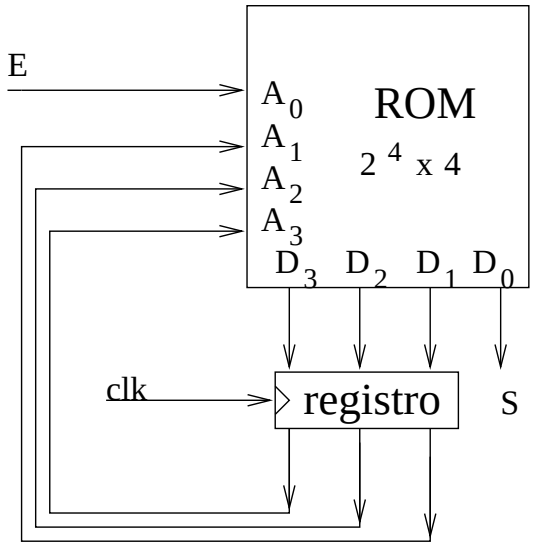


Figura 5.11: *Esquema de conexionado del ejemplo de la tabla 5.2*

última posibilidad no es más que el diseño canónico del S.S. realizado con biestables D.

El esquema general es como el que se describe en la figura 5.12. El estado presente se almacena en el contador. Los módulos CUENTA, CARGA, E.S. (Estado Siguiente) y SALIDA generarán las correspondientes señales de control del contador y la salida en función del estado presente y de la entrada.

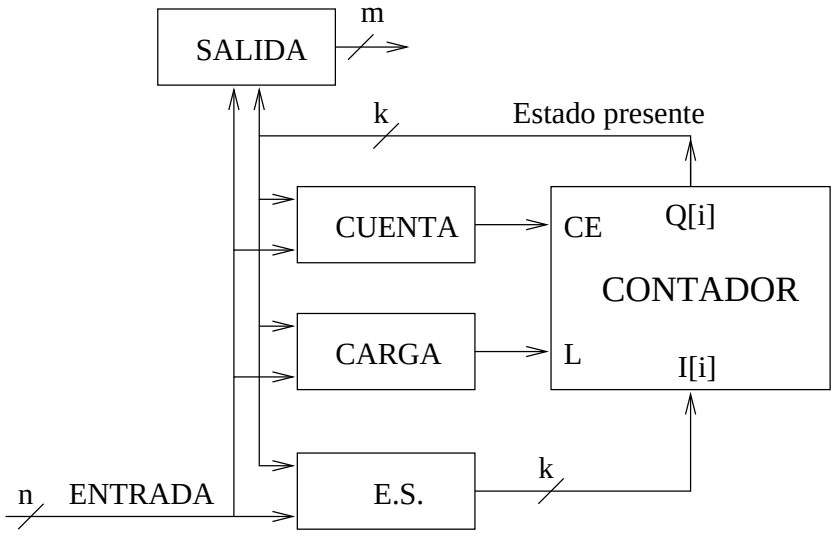


Figura 5.12: *Esquema de conexionado genérico de un S. S. implementado con contador y lógica combinacional*

Como ejemplo se propone el diseño del mismo S. S. de la figura 5.13 con un contador binario con las señales de carga síncrona L y habilitación de cuenta CE. Claramente el contador necesario será como mínimo módulo 5, al tener el S.S. 5 estados. El diseño se realizará con un contador módulo la potencia de 2 inmediatamente superior, en este caso 8.

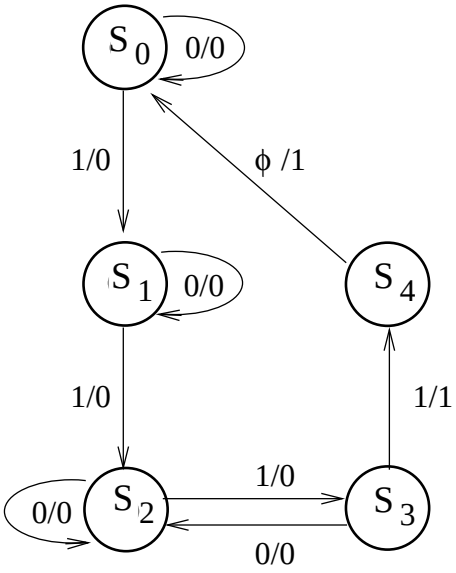


Figura 5.13: *Grafo del sistema secuencial ejemplo de diseño con contador*

$Q_2Q_1(t)$ \ $Q_0(t)E$	00	01	11	10
00	0	0	0	0
01	0	0	0	1
11	ϕ	ϕ	ϕ	ϕ
10	1	1	ϕ	ϕ

Tabla 5.4: *Señal de carga L para la implementación con contador + lógica del S. S. de la figura 5.13*

Observando el grafo de la figura 5.13 de manera cuidadosa se descubre como en este S. S. *casi* se sigue la secuencia de cuenta de un contador binario. Se romperá la secuencia de cuenta en el estado S_3 ya que si la entrada vale 0 la máquina volverá al estado S_2. De la misma manera se romperá la secuencia de cuenta al realizarse la transición de S_4 a S_0, ya que después del 4 iría el 5 binario. La máquina no cambia de estado cuando está en los estado S_0, S_1 y S_2 y entra un 0. En estos casos la señal de habilitación de cuenta CE y carga L no deben estar habilitadas.

Partiendo de este análisis preliminar se pueden realizar los mapas de Karnaugh de las funciones de carga L, habilitación de cuenta CE, estado siguiente $I_2I_1I_0$ y salida S que se muestran en las tablas 5.4, 5.5, 5.6 y 5.7.

Los estados inaccesibles se rellenan con indeterminaciones para simplificar más el S. S. De nuevo podría ser interesante prever transiciones erróneas debido al ruido electromagnético y forzar transiciones a un estado seguro. En este caso simplemente se aprovecharán más para implementar funciones más sencillas.

A partir de las tablas 5.4, 5.5, 5.6 y 5.7 se obtienen las ecuaciones:

	$Q_0(t)E$			
$Q_2Q_1(t)$	CE	00	01	11	10
	00	0	1	1	0
	01	0	1	1	0
	11	ϕ	ϕ	ϕ	ϕ
	10	0	0	ϕ	ϕ

Tabla 5.5: *Señal de habilitación de cuenta CE para la implementación con contador + lógica del S. S. de la figura 5.13*

	$Q_0(t)E$			
$Q_2Q_1(t)$	$I_2I_1I_0$	00	01	11	10
	00	$\phi\phi\phi$	$\phi\phi\phi$	$\phi\phi\phi$	$\phi\phi\phi$
	01	$\phi\phi\phi$	$\phi\phi\phi$	$\phi\phi\phi$	010
	11	$\phi\phi\phi$	$\phi\phi\phi$	$\phi\phi\phi$	$\phi\phi\phi$
	10	000	000	$\phi\phi\phi$	$\phi\phi\phi$

Tabla 5.6: *Entrada paralela $I_2I_1I_0$ para la implementación con contador + lógica del S. S. de la figura 5.13*

	$Q_0(t)E$			
$Q_2Q_1(t)$	S	00	01	11	10
	00	0	0	0	0
	01	0	0	1	0
	11	ϕ	ϕ	ϕ	ϕ
	10	1	1	ϕ	ϕ

Tabla 5.7: *Salida del S. S. de la figura 5.13*

$$\begin{aligned} L &= Q_2 + Q_1 Q_0 \overline{E} \\ CE &= \overline{Q_2} E \\ I_2 &= I_0 = 0 \\ I_1 &= \overline{Q_2} \\ S &= Q_2 + Q_1 Q_0 E \end{aligned} \tag{5.2}$$

5.4.3 Diseño con registro de desplazamiento SIPO y lógica combinacional: Generadores de secuencia

Es sencillo realizar generadores de secuencia utilizando un registro de desplazamiento SIPO y lógica combinacional. La secuencia generada se puede realimentar como entrada serie al SIPO, marcando de esta manera la evolución de los estados. De esta manera si se desea realizar la secuencia de n bits S_0, S_1, ... S_{n-2}, S_{n-1} se tomará ésta como la salida de uno de los biestables del SIPO Q_i. La función f que generará la secuencia que se realimentará al SIPO dependerá de los m biestables del registro SIPO, con lo que $f = f(Q_0, ..., Q_{m-1})$. La única condición será que no se repitan estados al producirse la realimentación, ya que la función que genera la secuencia depende únicamente del estado presente.

Claramente la función no podrá tomar los valores $0 = f(0, 0...0, 0)$ ni $1 = f(1, 1...1, 1)$, ya que la realimentación no hará que evolucione el estado. De esta manera, si se parte de un estado inicial con todos los biestables a 0 (*reset*) el primer bit de la secuencia a generar deberá ser un 1. Esto no quiere decir que el primer bit de la secuencia tenga que ser necesariamente 1, ya que se escogerá el biestable Q_i como salida de manera que el número de ceros antes del primer 1 sea el adecuado.

A partir de estas premisas se puede generar la secuencia de manera sencilla. Lo único que hay que hacer es organizar por columnas la secuencia inicial a partir del primer 1 y realimentarla en la entrada serie. El primer estado será todo ceros (reset) que generará el primer bit de la secuencia S_0. El siguiente estado será S_0, 0, 0...0, que a su vez generará el siguiente estado de la secuencia S_1. Como hay que generar hasta S_{n-1} (n bits de secuencia) serán necesarios n estados distintos generados al realimentar f. Inicialmente se puede probar con $n-1$ bits, añadiendo un bit si es necesario (si para un mismo estado en el SIPO la salida es distinta).

En la figura 5.14 se ha supuesto que $S_0 = 1$, cosa que no le resta generalidad al método. Tal como se muestra en la figura 5.14 (A) se ordena la secuencia y posteriormente se añade el estado $Q_{n-1}...Q_0 = 0..0$ que generará este primer uno de la secuencia (ver figura 5.14 (B)). La función f se realimenta para que la máquina evolucione y una vez se ha generado la secuencia completa se sigue generando la secuencia hasta que se llegue a cerrar el ciclo.

Para observar esta metodología con mayor claridad supóngase que se quiere generar la secuencia 00101 de forma cíclica. Esta secuencia se puede reordenar para que el primer bit sea un 1, de manera que se tendría 10100. Al ser n=5, inicialmente sería necesario un registro SIPO de 5-1=4 bits, aunque como se justificará con posterioridad con un SIPO de 3 bits y más lógica combinacional sería suficiente.

La función f generaría la secuencia 10100 que una vez realimentada generaría la secuencia de estados especificados en la figura 5.15. Cabe hacer notar como a partir del

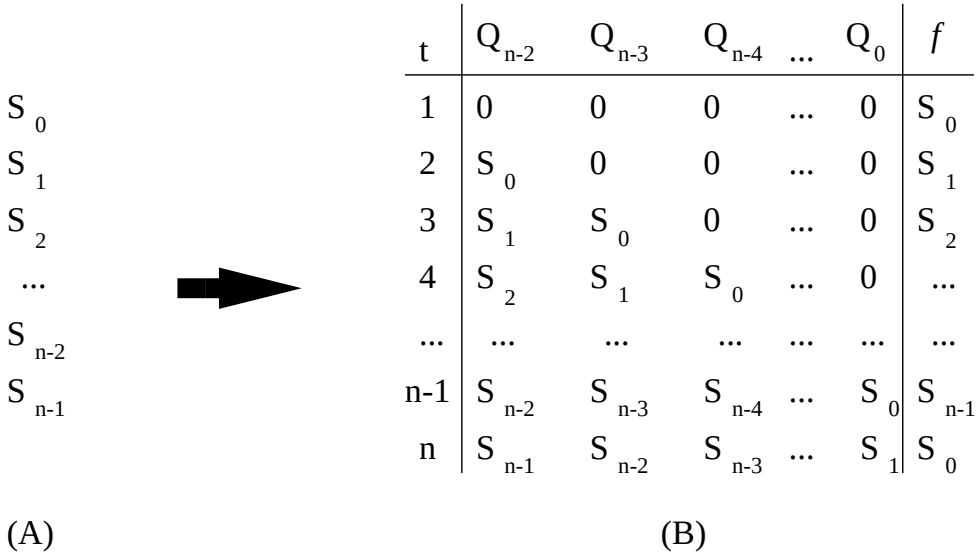


Figura 5.14: *Caso genérico de un generador de secuencia con un registro SIPO*

estado 6 se debe seguir generando la secuencia 10100 que una vez realimentada generaría los 5 estados siguientes. Al realimentarse la misma secuencia, llega un momento en que el estado presente se repite, con lo que el generador de secuencia se mantiene en un ciclo indefinido. La secuencia deseada original 00101 se obtiene en el bit Q_1. Se deja como ejercicio al estudiante la obtención de f en función del estado presente.

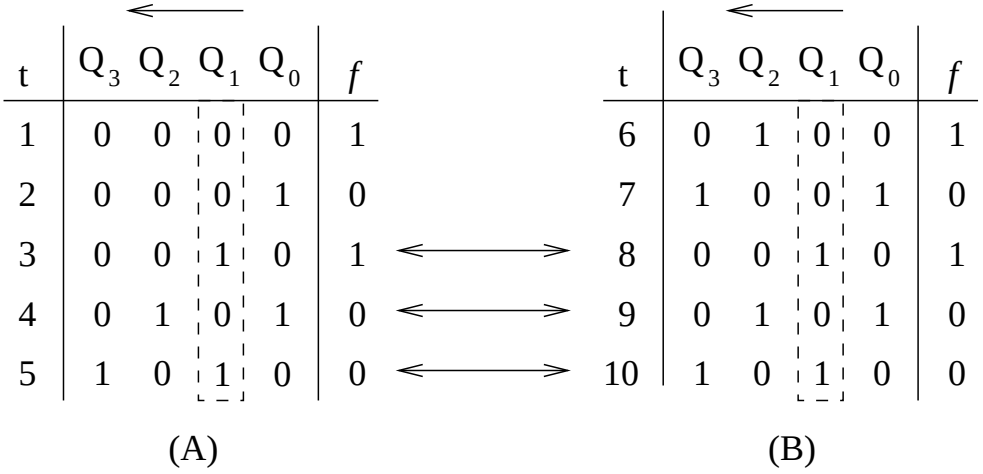


Figura 5.15: *Ejemplo de un generador de secuencia con un registro SIPO*

Esta es la aproximación más sencilla, pero la utilización de un SIPO de un tamaño igual a la secuencia generada es innecesaria. De hecho en el caso anterior serían solamente necesarios 3 bits para generar un ciclo de 5 estados. El problema en este caso estriba en generar una función que genere los estados necesarios como un desplazamiento de manera cíclica, problema resuelto por la teoría de campos finitos de Évariste Galois.

Estas secuencias especiales se generan mediante ecuaciones de retroalimentación generadas con funciones que utilizan puertas lógicas XOR que implementan unos polinomios primitivos [Wak00] [LP96].

5.5 Conclusiones

Se han definido los registros y contadores como módulos secuenciales estándar que pueden ser útiles para diseñar sistemas secuenciales.

El diseño con ROM y registro de estado es extremadamente sencillo, siendo el concepto similar a la microprogramación de la unidad de control. Es posible en sistemas secuenciales que cumplan ciertas condiciones reducir el tamaño de la memoria que se utiliza mediante un multiplexor. Esto será útil especialmente si el S. S. presenta muchas entradas.

Análogamente se ha presentado una metodología para diseñar S. S. con contadores y lógica combinacional. Este método será ventajoso en los S. S. que tengan una secuencia de estados lo más parecidos posible a la secuencia de cuenta del contador.

La utilización de señales asíncronas para el diseño de S. S. con contadores puede ser peligrosa, planteando la aparición de transitorios en las señales de salida. Cuando se utilicen estas señales, se deberán de analizar de manera cuidadosa los retrasos asociados. Si es necesario se ajustarán las simplificaciones de la salida para evitarlos.

En estos apuntes se ha tratado explícitamente el diseño de sistemas secuenciales mediante registros de desplazamiento SIPO. La generación de secuencias realimentando la salida a la entrada serie para generar el estado siguiente es intuitivamente sencilla. En este caso será necesario un SIPO de n bits, donde n es el tamaño de la secuencia. El problema es más complejo cuando se quiere utilizar un SIPO de tamaño $k = \lceil log_2 n \rceil$. Entonces se tiene que generar una función de realimentación que recorra todas las combinaciones binarias posibles mediante desplazamientos. A partir de esta secuencia de estados se generará la salida que dependerá del estado presente. Estas secuencias de estados, en las que el estado siguiente es el estado presente con un desplazamiento de un bit, son útiles para generar secuencias de cuenta *pseudo-aleatorias* completas.

Estas secuencias especiales se generan mediante ecuaciones de retroalimentación generadas con funciones que utilizan puertas lógicas XOR que implementan unos polinomios primitivos [Wak00] [LP96].

Parte II

Tecnologías Digitales

Capítulo 6

Parámetros de las Familias Lógicas

6.1 Introducción

La lógica Booleana es bastante sencilla y sistemática. Se puede llegar a entender como un formalismo matemático que opera con dos elementos (unos y ceros), mediante una serie de operadores lógicos, como son el producto lógico AND, la negación NOT, y la suma OR. A partir de estos operadores mínimos, y representando la información de forma adecuada, se construye toda la lógica necesaria para implementar sistemas complejos, como son los computadores.

Sin embargo el mundo físico es muy diferente a la idealidad de las matemáticas. Si se desea procesar información con el formalismo Booleano, se deben tener en cuenta una serie de parámetros tecnológicos que no tienen ninguna relación con las matemáticas. Estos parámetros, heredados de la implementación física de los operadores Booleanos, deberán ser respetados si se desea un correcto funcionamiento lógico del circuito. De esta manera, aparecen conceptos como la carga máxima de un puerta lógica, la frecuencia máxima de funcionamiento, y la tensión más baja de salida permitida a nivel alto.

Una primera clasificación de los parámetros los agrupará en estáticos o dinámicos, en función de su relación con el tiempo. En este capitulo se presentarán los más relevantes, y las consideraciones de diseño que hay que tener en cuenta para garantizar un correcto funcionamiento lógico de un circuito digital. Se recomienda la consulta de [CGT93] y [Toc93] para seguir el presente tema.

6.2 Parámetros estáticos

Para construir circuitos que se comporten como los operadores ideales matemáticos hay que utilizar dispositivos físicos que tengan un comportamiento global similar. Por tanto hay que realizar un análisis de las condiciones físicas en las que los dispositivos se comportarán como su equivalente ideal matemático.

Como ejemplo, debido a su simplicidad pero sin pérdida de generalidad en el modelo, se va a utilizar el **inversor mínimo** de la figura 6.1. En este caso se van a suponer unos niveles de tensión ideales TTL (0 voltios para el 0 lógico y 5 voltios para el 1 lógico). Con esta codificación y sólo con un transistor BJT y un par de resistencias se

puede construir un circuito electrónico que se comporte como un inversor lógico. El dispositivo de la figura 6.1 no se utiliza nunca como inversor debido a las limitaciones que se van a exponer a continuación (se va suponer al estudiante familiarizado con las bases de la electrónica digital).

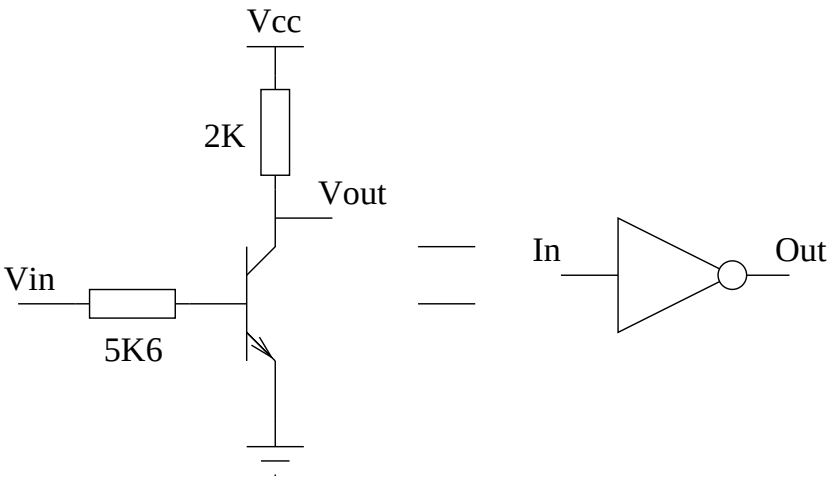


Figura 6.1: *El inversor mínimo*

Si se tiene un nivel lógico alto (que corresponde a 5 voltios) en la entrada del circuito V_{in}, el diodo base-emisor del transistor BJT se polarizará directamente, pasando el transistor a saturación. De esta manera si $V_{CE} \simeq 0$ entonces en V_{out} se tendrán 0 voltios, que corresponden a un 0 lógico.

Por el contrario, si el nivel de entrada es un 0 lógico (0 voltios), el diodo base-emisor del transistor BJT estará cortado, con lo que que no pasará corriente de colector a emisor y no caerá tensión en la resistencia de colector, con lo que la salida será 5 voltios, que corresponde a un uno lógico.

Tras el análisis realizado parece que se ha encontrado un circuito que cumple perfectamente las especificaciones para implementar un inversor lógico. Esta conclusión se probará que es incorrecta en el análisis que se va a realizar a continuación. Supóngase que a la salida de este primer inversor se quiere conectar otro dispositivo lógico para implementar funciones más complejas. De nuevo, por simplicidad, se va a conectar otro inversor a la salida del primero tal como se muestra en la figura 6.2.

De nuevo si el nivel de entrada es un 0 lógico (0 voltios), el diodo base-emisor del transistor BJT estará cortado. Pero ahora, al haber una camino que conecta V_{CC} con tierra, si que bajará corriente por la resistencia de colector, polarizando el diodo base emisor del segundo transistor directamente. Si se calcula el nivel de tensión del nodo intermedio $V_{intermedio}$ se obtendrá un valor de 3'5V debido a la caída de tensión producida en la resistencia de colector. Se observa por tanto como la tensión correspondiente al nivel lógico alto ha bajado de manera considerable.

Si se amplia el experimento conectado más de un inversor en paralelo a la salida del primer inversor se observará como, al bajar más corriente por la resistencia de colector, cae más la tensión del nodo intermedio, pudiendo llegar a no polarizar a los transistores BJT que estan conectados a la salida del primero.

Se observa como el nivel de tensión del nodo intermedio (y por tanto el correcto funcionamiento lógico del dispositivo) depende de la **carga** de la salida. Obviamente la

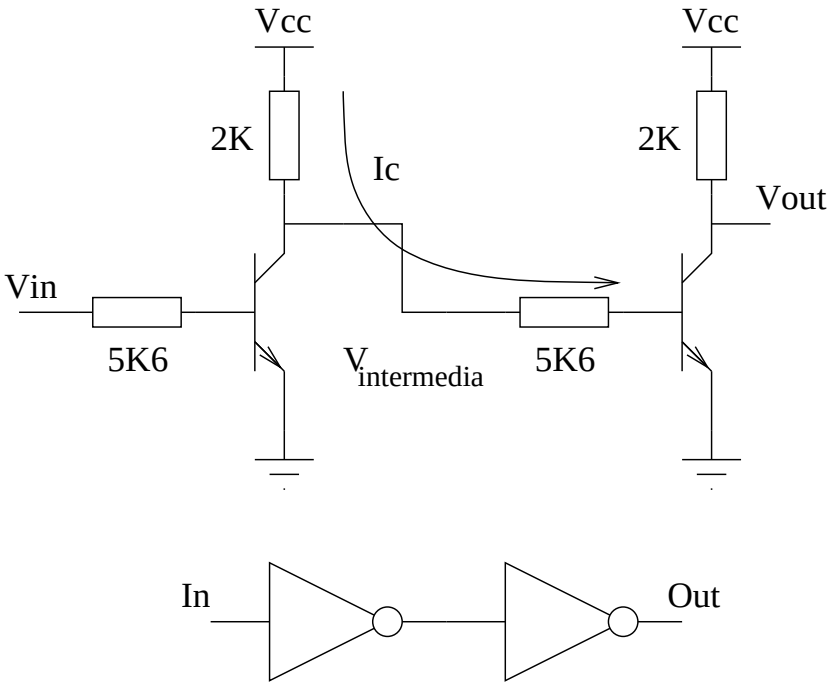


Figura 6.2: *Efecto de la carga en la salida*

definición de equivalencia entre valores lógicos y niveles eléctricos se debe realizar entre márgenes de tensión. De todas maneras, a pesar de la existencia de estos márgenes, el funcionamiento correcto vendrá limitado por las características físicas de los circuitos.

6.2.1 Niveles lógicos

A partir de la definición de los valores lógicos como niveles de tensión cabe la pregunta de cuales serán los margenes adecuados para los niveles lógicos. Se ha comprobado como se deben definir unos niveles lógicos dentro de unos márgenes para asegurar el correcto funcionamiento lógico. Para definir estos márgenes habrá que tener en cuenta las características tecnológicas del circuito, tal como se va a razonar a continuación.

La figura 6.3 muestra la función de transferencia de un inversor genérico. Se observa como para valores bajos de tensión de entrada (un cero lógico), se obtienen valores altos de tensión en la salida (un uno lógico). Este comportamiento ideal ya no es tan claro si la tensión de entrada se aleja de los valores extremos y se acerca al centro.

Un comportamiento ideal para la función de transferencia sería una función escalón. Lamentablemente la función de transferencia nunca es una función escalón perfecta y presenta la forma de la figura 6.3. El comportamiento ideal se cumple para los extremos pero no en la zona intermedia de la función. Se pueden definir, a partir de la gráfica de transferencia, unos valores mínimos a partir de los cuales se entenderá en las entradas que se tiene un uno lógico o un cero lógico. Estos valores serán $V_{ILumbral}$ y $V_{IHumbral}$, y vendrán definidos como los valores de entrada en los cuales la función de transferencia tiene una pendiente negativa $m = -1$.

Claramente los valores de entrada menores que $V_{ILumbral}$ se pueden interpretar como valores bajos, ya que la salida es un nivel alto, y los valores mayores que $V_{IHumbral}$ se

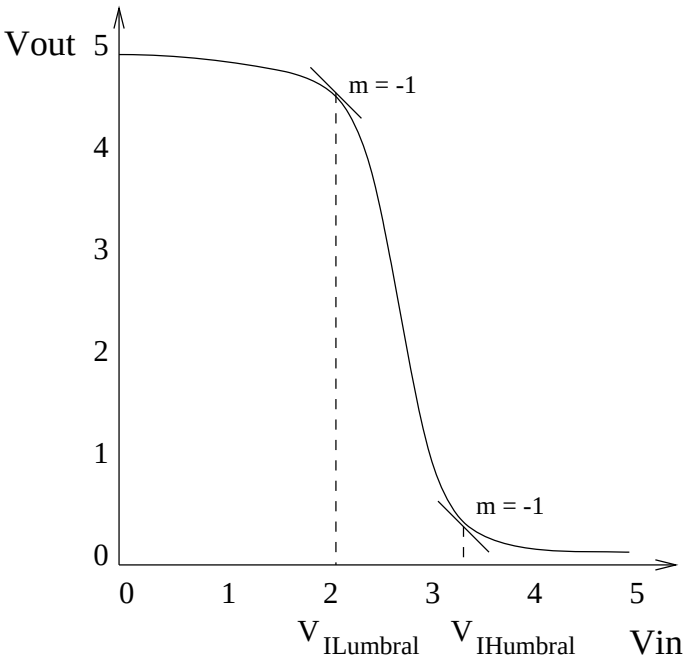


Figura 6.3: *Función de transferencia de una puerta inversora*

pueden interpretar como valores de entrada altos, ya que la salida es un nivel bajo. La definición de los márgenes de entrada para los niveles lógicos debe tener esto en cuenta.

El nivel de tensión más bajo que corresponda al nivel lógico 0 (V_{ILmin}) será el límite inferior del **margen del cero** (límite inferior totalmente definido por la tecnología), y el nivel de tensión de entrada más alto para el cero lógico (V_{ILmax}) será el límite superior de este margen. Para el límite superior una primera propuesta podría ser hacer coincidir V_{ILmax} con $V_{ILumbral}$, pero no se suele optar por esta elección ya que se desea tener un cierto margen de ruido en la entrada. Si se define el **margen de ruido a nivel bajo** (V_{ML}) en la entrada como el margen de tensión ruido tolerable para la entrada a nivel bajo se tendrá que:

$$V_{ILmax} = V_{ILumbral} - V_{ML} \tag{6.1}$$

Análogamente se pueden definir las tensiones mas bajas y más altas que se reconocerán como un 1 lógico que definirán el **margen del uno**. Estas tensiones serán V_{IHmin} y V_{IHmax} respectivamente. La tensión V_{IHmax} de nuevo vendrá definida por la tecnología de la puerta, y la tensión V_{IHmin} se definirá a partir del **margen de ruido a nivel alto** (V_{MH}) en la entrada, de manera que se cumple que:

$$V_{IHmin} = V_{IHumbral} + V_{MH} \tag{6.2}$$

De la misma manera se definirán los niveles de tensión de salida a nivel alto y bajo máximos y mínimos:

- V_{OHmin}: Tensión mínima de salida que saldrá de una puerta lógica y que se entenderá como un 1 lógico.
- V_{OHmax}: Tensión máxima de salida que saldrá de una puerta lógica que se entenderá como un 1 lógico.

- V_{OLmin}: Tensión mínima de salida que saldrá de una puerta lógica y que se entenderá como un 0 lógico.
- V_{OLmax}: Tensión máxima de salida que saldrá de una puerta lógica que se entenderá como un 0 lógico.

Se podría hacer coincidir las tensiones de salida V_{OHmin} y V_{OLmax} exactamente con V_{IHmin} y V_{ILmax} respectivamente, pero esta definición presentaría un comportamiento muy peligroso en el caso de que se tuviera ruido electromagnético en la salida. Efectivamente, si se supone que a el nivel de tensión de salida a nivel bajo más alto V_{OLmax} se le acopla un margen de ruido en forma de tensión positiva, se producirá un nivel de tensión a nivel bajo más alto que V_{OLmax}. Este nivel deberá de ser entendido como un nivel bajo, con lo que no se puede superar V_{ILmax}. Estas ideas llevan de nuevo a la aplicación de unos márgenes de ruido para las salidas, de manera que se tiene que:

$$V_{OLmax} = V_{ILmax} - V_{ML} \tag{6.3}$$

Y para el nivel alto mínimo de salida:

$$V_{OHmin} = V_{IHmin} + V_{MH} \tag{6.4}$$

De nuevo los valores de salida V_{OLmin} y V_{OHmax} son puramente tecnológicos. En la figura 6.4 se pueden observar estas definiciones.

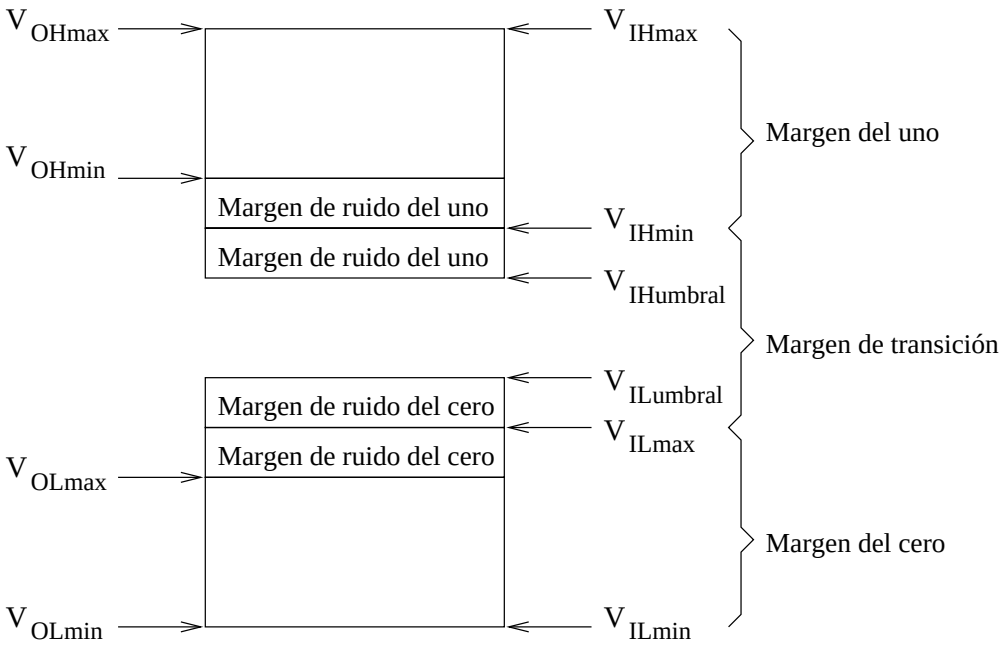


Figura 6.4: *Definición de los márgenes de ruido*

De la misma manera se definirá el **margen de transición** como los niveles de tensión de entrada que no se reconocen ni como nivel bajo ni como nivel alto. Esta zona será $V_{IHmin} - V_{ILmax}$, tal como se muestra en la figura 6.4.

El fabricante suministra sus circuitos con unos valores de tensión de entrada y salida máximos, mínimos y típicos. Se garantiza el funcionamiento dentro de estos

parámetros de tensión siempre que se cumplan las condiciones de temperatura y tensión de alimentación especificados.

Típicamente no se cumplirán los valores de la tensión de salida V_{OLmax} y V_{OHmin} cuando las impedancias de los dispositivos conectados a la salida consuman más corriente de la permitida. El consumo de corriente por las entradas y salidas también debe estar parametrizado con unos valores máximos. Si no se cumplen estos niveles de corriente también se violarán los parámetros de tensión, con lo que el dispositivo lógico no funcionará.

6.2.2 Corrientes eléctricas

La definición de los parámetros referentes a las corrientes es similar a la definición de los parámetros de tensión. De esta manera se tendrán:

- I_{ILmax}: Corriente máxima de entrada para el nivel bajo
- I_{IHmax}: Corriente máxima de entrada para el nivel alto
- I_{ILmin}: Corriente mínima de entrada para el nivel bajo
- I_{IHmin}: Corriente mínima de entrada para el nivel alto
- I_{OLmax}: Corriente máxima de salida para el nivel bajo
- I_{OHmax}: Corriente máxima de salida para el nivel alto
- I_{OLmin}: Corriente mínima de salida para el nivel bajo
- I_{OHmin}: Corriente mínima de salida para el nivel alto

Siendo de los parámetros anteriores los que que hacen referencia a valores máximos de la corriente los que tienen más sentido, aunque pueda haber familias lógicas que necesiten de una corriente mínima de polarización para las entradas o salidas.

Cuando la corriente sea absorbida o entre en el circuito se considerará positiva y cuando salga del circuito se considerará negativa.

Se definirá la carga máxima de una puerta o **fan-out** como el número máximo de puertas lógicas que se pueden conectar a la salida de esta puerta. Habitualmente esta cantidad se refiere a número de puertas del mismo tipo o de la misma familia, aunque a veces se den cifras respecto a familias lógicas compatibles.

Es sencillo ver como existe una relación directa entre las corrientes eléctricas y el concepto de carga anteriormente definido. El hecho de conectar a una salida una entrada hace que haya un flujo de corriente en esta conexión (esto es cierto en TTL pero no en CMOS), y por tanto se produzca un cambio de tensión.

A partir de las corrientes eléctricas anteriormente definidas se puede hallar la expresión para el *fan-out*:

$$Fan-out = min\{\lfloor \frac{I_{OHmax}}{I_{IH}} \rfloor, \lfloor \frac{I_{OLmax}}{I_{IL}} \rfloor\} \tag{6.5}$$

Números típicos para familias TTL de *fan-out* pueden estar alrededor de 10-20 puertas lógicas. para familias CMOS se justificará en el capítulo 8 que esta cifra, si se tienen en cuenta sólo las características estáticas, puede llegar a ser casi infinita.

6.3 Parámetros dinámicos

Por parámetros dinámicos se entenderán aquellos relacionados con el tiempo. Estos parámetros son tanto o más importantes que los parámetros estáticos, y de nuevo un desconocimiento de éstos puede llevar a un mal funcionamiento del circuito lógico.

6.3.1 Capacidades parásitas

Hasta ahora no se ha presentado el problema de la propagación de la señal. Existen retrasos entre la presentación de unos valores lógicos en la entrada de una puerta y la respuesta de ésta evaluando el resultado. Estos retrasos son inherentes a los circuitos electrónicos y tienen una relación directa con las **capacidades parásitas**.

Las capacidades parásitas son los condensadores que aparecen de forma inevitable en los circuitos electrónicos que implementan las puertas lógicas. El objetivo de los tecnólogos que diseñan procesos de fabricación de circuitos integrados es siempre minimizar la aparición de estas capacidades parásitas, y el objetivo de los diseñadores de circuitos es minimizar su efecto. Además de los retrasos debidos a las capacidades parásitas inherentes a los circuitos electrónicos, también se tiene un cierto retraso debido a las líneas de conexión de los circuitos impresos. Se va a estudiar como las capacidades parásitas hacen inevitable la aparición de retrasos.

Como ejemplo de análisis se considerará de nuevo el ejemplo de 2 inversores mínimos interconectados de la figura 6.5. Este ejemplo es suficientemente sencillo para entenderse sin dificultad y a la vez es muy ilustrativo. En este caso se ha dibujado adicionalmente la capacidad parásita C que podría entenderse como la suma de las capacidades parásitas de salida de la primera puerta y la de entrada de la segunda puerta.

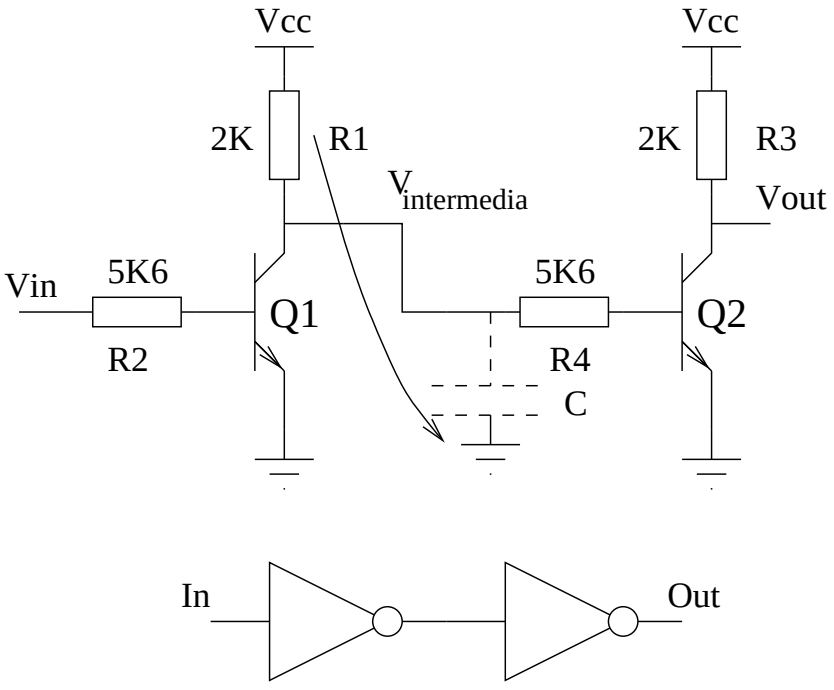


Figura 6.5: *Efecto de la capacidad parásita en la transmisión de la señal*

Supóngase el sistema estabilizado y que inicialmente la entrada del primer inversor mínimo es un nivel alto, con lo que V_{in}=5V. Si la tensión de saturación $V_{CE}(sat) = V_{intermedia}$ = 0'2V Q2 estará cortado y por tanto V_{out}=5V (un nivel alto).

Supóngase ahora que en $t = 0$ V_{in}=0V, por tanto Q1 se cortará y el condensador C se empezará a cargar con la corriente de colector dibujada en la figura 6.5 hacia 5 voltios. Se pretende saber cuando en V_{out} también se tendrá un cero lógico. Hay que distinguir dos situaciones diferentes:

1. Por una parte el condensador pasa a cargarse hacia 5V con una carga inicial de 0'2V hasta que $V_{intermedia}$=0'7V. En este momento Q2 empezará a conducir y las condiciones de carga del condensador serán diferentes.
2. Cuando Q2 empieza a conducir se tendrá que C se cargará hacia la tensión que habría en el nodo intermedio si no estuviera el condensador. Esta tensión vendrá fijada por el divisor de tensión R1-R4.

Mientras $V_{intermedia}$ vaya creciendo (el condensador se va cargando) aumentará la corriente de base de Q2 y por tanto la corriente de colector de Q2 también aumentará. Esto hará que V_{out} disminuya, hasta que se reconozca un nivel bajo en V_{out}. Es decir cuando se cumpla que V_{out}=V_{OLmax} ya se tendrá un cero en la salida y se considerará que se ha propagado la señal.

Se puede calcular el tiempo t que ha tardado en propagarse la señal a través de estos dos inversores con la ecuación de carga/descarga de un condensador a través de una resistencia:

$$V_C = V_f - (V_f - V_i)e^{\frac{-t}{RC}} \tag{6.6}$$

Donde cada valor es:

- V_C: Tensión en el condensador en el borne conectado a $V_{intermedia}$ en función de t.
- V_f: Tensión final hacia la que tiende a cargarse asintóticamente el condensador.
- V_i: Tensión inicial del condensador en $t = 0$.
- R: Resistencia a través de la que se carga el condensador (expresada en Ω).
- C: Valor de la capacidad del condensador en Faradios.

De una manera sencilla se puede obtener que el tiempo de retardo τ es proporcional a RC, por lo que a mayor capacidad parásita se tendrá un mayor retraso. También se puede concluir que una manera de combatir el retraso, al no poder evitar los valores de C, será disminuyendo los valores de R, con lo que disminuirá el producto RC. Esta conclusión interesante es parte de la estrategia seguida por la familia TTL-Schottky que se analizará en el capítulo 7, pero plantea otra serie de desventajas, como es el consumo excesivo.

6.3.2 Tiempo de respuesta

A partir de las consideraciones analizadas en el ejemplo de la sección anterior es sencillo entender que las puertas no responden de manera automática a la aplicación de las entradas. Será necesario un cierto tiempo de respuesta. Para caracterizar de manera inequívoca estos tiempos de respuesta hay que hacer unas definiciones exactas, de esta manera se definirán los tiempos que se observan en la figura 6.6:

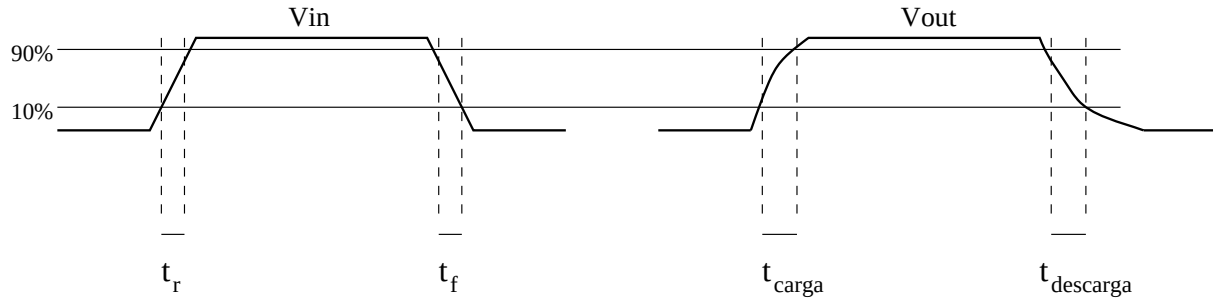


Figura 6.6: *Curvas de carga y descarga de las entradas y salidas*

- t_r: Tiempo de crecida (*rise time*). Es el tiempo necesario para que una señal de **entrada** pase del 10% hasta el 90% del valor final.
- t_f: Tiempo de caída (*fall time*). Es el tiempo necesario para que una señal de **entrada** pase del 90% hasta el 10% del valor final.
- t_{carga}: Tiempo de carga. Es el tiempo necesario para que una señal de **salida** pase del 10% hasta el 90% del valor final.
- $t_{descarga}$: Tiempo de descarga. Es el tiempo necesario para que una señal de **salida** pase del 90% hasta el 10% del valor final.

Los parámetros relacionados directamente con la propagación de la señal, que se muestran en la figura 6.7 son:

- t_{PLH}: Tiempo de propagación de nivel bajo a nivel alto. Es el tiempo necesario para que la señal suba hasta llegar al 50% del nivel alto medido desde el instante en el que se aplica el 50% de la señal que genera ese cambio.
- t_{PHL}: Tiempo de propagación de nivel alto a nivel bajo. Es el tiempo necesario para que la señal baje hasta llegar al 50% del nivel alto medido desde el instante en el que se aplica el 50% de la señal que genera ese cambio.
- t_P: Es la media entre t_{PLH} y t_{PHL}, es decir $t_P = \frac{t_{PLH}+t_{PHL}}{2}$.

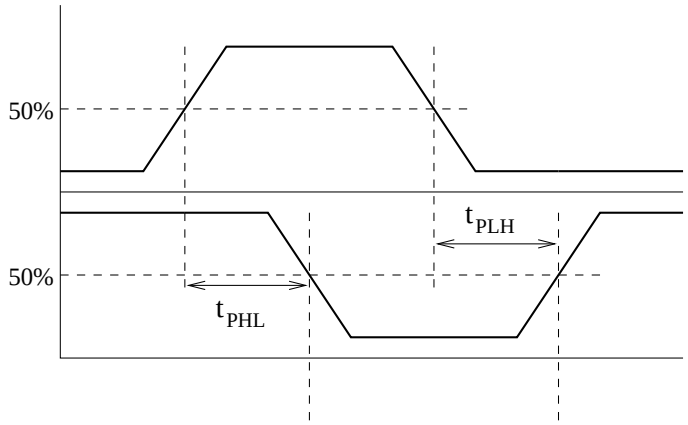


Figura 6.7: *Tiempos de propagación en un inversor*

De manera análoga se pueden definir los tiempos necesarios para pasar a un estado de alta impedancia en la salida desde un estado bajo (t_{PLZ}) o alto (t_{PHZ}), o para pasar desde un estado de alta impedancia a un estado bajo (t_{PZL}) o alto (t_{PZH}).

Estos tiempos de propagación suelen tener unos nombre diferentes en el caso de los biestables sensibles a flanco, aunque conceptualmente son idénticos. Estos tiempos son:

- $t_{CLK\ to\ Output\ LH}$: Tiempo de propagación de nivel bajo a nivel alto. Es el tiempo necesario para que la señal suba hasta llegar al 50% del nivel alto medido desde el flanco de reloj que origina ese cambio.
- $t_{CLK\ to\ Output\ HL}$: Tiempo de propagación de nivel alto a nivel bajo. Es el tiempo necesario para que la señal baje hasta llegar al 50% del nivel alto medido desde el flanco de reloj que origina ese cambio.

Los biestables en particular y en general cualquier circuito secuencial tienen, además de los parámetros de propagación descritos, unos parámetros temporales que es necesario respetar para conseguir un correcto funcionamiento del circuito. De esta manera se tendrá:

- t_{su}: Tiempo de establecimiento, o de *set-up*. Es el tiempo que deben de permanecer las entradas estables antes de la señal de sincronismo. En la figura 6.8 se ejemplifica este tiempo para un biestable maestro-esclavo de tipo D.
- t_h: Tiempo de mantenimiento, o de *hold*. Es el tiempo que deben de permanecer las entradas estables después de la señal de sincronismo. En la figura 6.8 también se muestra este tiempo.

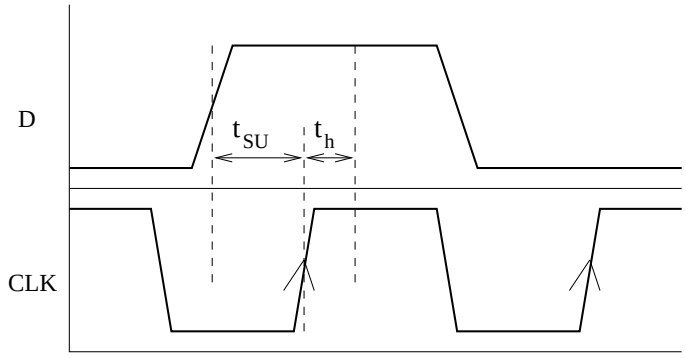


Figura 6.8: *Tiempo de establecimiento y de mantenimiento en un biestable maestro-esclavo*

El respeto de estos parámetros dinámicos es fundamental para el correcto funcionamiento del sistema secuencial. Si la entrada no permanece estable un cierto tiempo antes y después de su captura con el flanco de reloj el estado del biestable es impredecible. En la sección 6.5 se mostrará como el respeto de estos parámetros, unido al retraso de la lógica de las funciones de excitación de los biestables, limitará la frecuencia de reloj del circuito secuencial.

6.4 Otros parámetros

Existen otros parámetros dinámicos que tienen también una gran importancia. De esta manera en los circuitos secuenciales se suele indicar la frecuencia máxima de funcionamiento (que será mayor que la impuesta por las realimentaciones del diseño), y otros parámetros dinámicos que se adjuntan en la tabla 6.1.

Parámetro	Descripción	Comentarios
F_{MAX}	Frecuencia máxima de reloj	Frecuencia máxima de reloj a la que puede funcionar el S.S.
I_{CC}	Corriente de alimentación	Corriente de alimentación global
I_{CCH}	Corriente de alimentación con las salidas altas	Corriente de alimentación con las salidas indicadas a nivel alto
I_{CCL}	Corriente de alimentación con las salidas bajas	Corriente de alimentación con las salidas indicadas a nivel bajo
P_D	Potencia disipada	Potencia que puede ser fija o función de la frecuencia
t_{ACC}	Tiempo de acceso *(Access time)*	Intervalo de tiempo necesario para acceder a un dato de memoria
t_{dis}	Tiempo de deshabilitación *(Disable time)*	Tiempo necesario para que todas las salidas pasen a alta impedancia (Z)
t_{en}	Tiempo de habilitación *(Enable time)*	Tiempo necesario para que todas las salidas pasen de Z a L o H.
t_{SR}	Tiempo de recuperación *(Sense recovery time)*	Tiempo necesario para cambiar el modo de acceso a una memoria
t_W	Ancho de pulso *(Width pulse time)*	Ancho mínimo de pulso de aplicación de una señal

Tabla 6.1: *Otros parámetros lógicos habituales*

Otro grupo de parámetros de gran importancia es el relacionado con el consumo. De esta manera se suelen indicar las corrientes máximas de alimentación o directamente el consumo del circuito integrado que podrá ser estático (lógica TTL) o función de la frecuencia (lógica CMOS). Estos parámetros de consumo son útiles para calcular la autonomía de un diseño que vaya a funcionar con baterías, o para establecer la necesidad de disipadores de calor para evitar el calentamiento excesivo de los circuitos.

6.5 Consideraciones de diseño

Hay que conocer de qué manera pueden influir en el diseño lógico los parámetros de funcionamiento de un circuito digital. Todos los parámetros se deben tener en cuenta para garantizar el correcto funcionamiento lógico del circuito.

Los parámetros estáticos se deben tener en cuenta para garantizar la conectividad entre los diversos circuitos lógicos que implementan el diseño (biestables, puertas lógicas, etc). Claramente si todos los circuitos son de la misma familia, el fabricante garantiza la conectividad entre ellos. Los niveles de salida y entrada están ajustados para que, si se mantiene un grado de tolerancia al ruido dentro de los márgenes especificados, los dispositivos funcionen correctamente. Lo único a tener en cuenta dentro de una misma familia será la carga máxima de las salidas o *fan-out*.

La consideración de los parámetros estáticos entre diversas familias debe garantizar la conectividad en lo referente a las tensiones (niveles lógicos) y a las corrientes eléctricas. Cualquier valor de la tensión de salida dentro de la ventana de un nivel lógico, debe ajustarse a las tensiones de la ventana de entrada del mismo nivel lógico. Para conseguir esto puede bastar con la utilización de resistencias de *pull-up* o *pull-down*, o puede ser necesaria la utilización de circuitos traductores de nivel de tensión o *transceivers*.

Los parámetros dinámicos también se deben tener en cuenta para garantizar el correcto funcionamiento del diseño digital. Los retrasos inherentes a la propagación de la señal, y los tiempos de establecimiento y de mantenimiento, imponen restricciones a las prestaciones del circuito. Como ejemplo se va a analizar la frecuencia máxima de funcionamiento del S. S. genérico implementado con biestables de la figura 6.9.

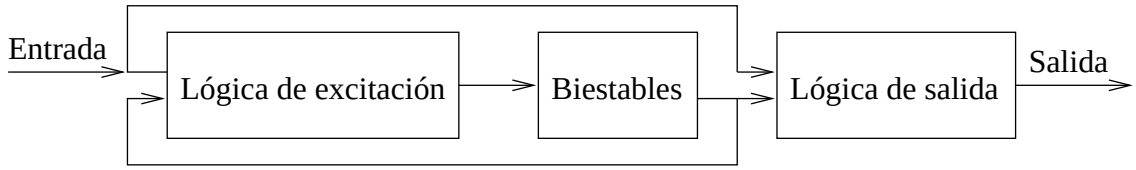


Figura 6.9: *Esquema de un sistema secuencial genérico diseñado con biestables*

En el sistema secuencial de la figura 6.9 se pueden distinguir los siguientes tiempos:

- t_{P1}: Tiempo de propagación de la lógica de excitación.
- t_{P2}: Tiempo de propagación en los biestables.
- t_{P3}: Tiempo de propagación en la lógica de salida.
- t_{su}: Tiempo de establecimiento para los biestables.
- t_h: Tiempo de mantenimiento para los biestables.

Los tiempos de propagación se deberán calcular teniendo en cuenta el peor camino de propagación de la señal. Así por ejemplo, si una función lógica se calcula con diversas entradas que atraviesan diversas ramas de un árbol, se tendrá en cuenta el camino más largo o *profundo* para calcular el tiempo de propagación, tal como se muestra en la figura 6.10. De la misma manera, para seleccionar el tiempo de propagación de una puerta lógica se deberá tener en cuenta las transiciones posibles para escoger la que dé la combinación posible de mayor retraso.

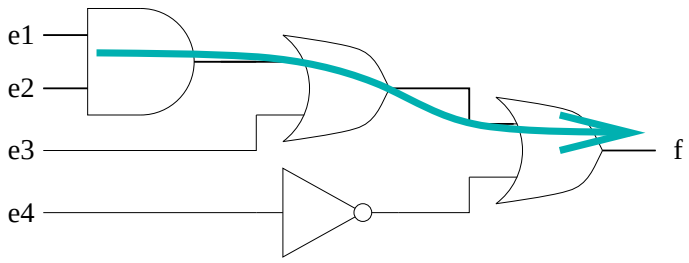


Figura 6.10: *Ramas más profundas en el árbol de una función*

Con los parámetros descritos anteriormente para el S. S. genérico de la figura 6.9 es posible darse cuenta que el periodo mínimo de reloj viene impuesto por el bucle de realimentación. De la figura 6.11 se puede deducir que el periodo mínimo vendrá dado por la relación:

$$T_{minimo} = t_{P1} + t_{P2} + t_{su} \tag{6.7}$$

Esto se cumplirá siempre que $t_h < t_{P1} + t_{P2}$, cosa que será cierta en la gran mayoría de casos. Si no se cumple esta condición el periodo mínimo (y por tanto la frecuencia máxima) vendrá dada por la relación:

$$T_{minimo} = t_h + t_{su} \tag{6.8}$$

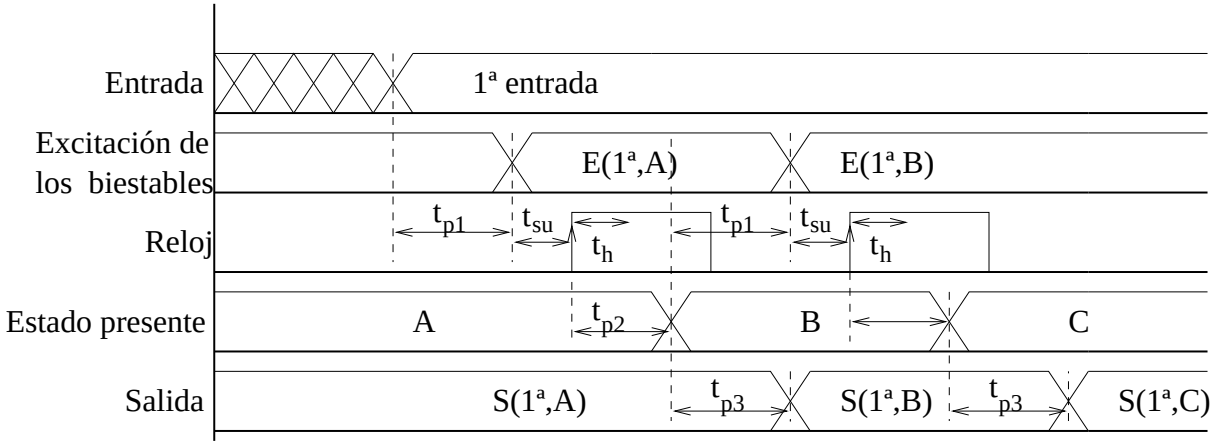


Figura 6.11: *Cronograma genérico del S.S. de la figura 6.9*

Los tiempos t_{P1}, t_{P2} y t_{P3} que aparecen varias veces en la figura pueden ser distintos porque en cada transición puede variar una señal distinta que recorrerá una rama distinta del árbol lógico. Siempre para calcular la frecuencia máxima se tendrá en cuenta el peor caso o la transición más lenta de las posibles, de esta manera se garantizará el periodo mínimo y la frecuencia máxima de funcionamiento del S. S.

De la misma manera se eliminan la influencia de los riesgos en las funciones de excitación. Si el periodo del reloj es mayor que el periodo mínimo apuntado las señales ya se habrán propagado totalmente antes del flanco de reloj. Sin embargo para la función de salida (que es una función combinacional de la entrada y del estado presente) se pueden presentar estos fenómenos aleatorios tanto estáticos como dinámicos. Para evitarlos simplemente hay que aplicar las técnicas de diseño de sistemas combinacionales que evitan las apariciones de riesgos.

Cabe hacer notar como además las entradas deben estar sincronizadas con el reloj para evitar violaciones en el tiempo de establecimiento y mantenimiento. Para conseguir esto es necesario que haya un biestable en las entradas que las capture y las mantenga invariables entre ciclos de reloj. Esto a su vez plantea el problema de que, al ser la entrada de éste biestable asíncrona, se puedan infringir t_{su} o t_h, con lo que el biestable tendría un estado no definido o **metaestable**. Después de un cierto tiempo t_r o **tiempo de estabilización** el estado del biestable estará definido y no presentará ningún problema. La dificultad estriba en que durante unos instantes la circuitería combinacional que excita los biestables tendrá entradas mal definidas, (la salida del biestable será metaestable), y el comportamiento será imprevisible. Para evitar esto se pueden conectar en la entrada asíncrona 2 biestables en serie como los de la figura 6.12.

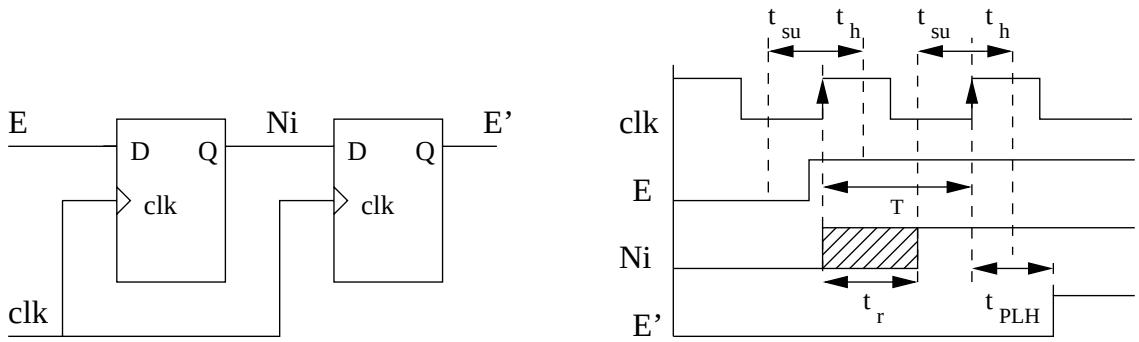


Figura 6.12: *A) Configuración para entradas asíncronas y cronograma correspondiente*

Se puede observar como el tiempo de establecimiento t_{su} más el tiempo de mantenimiento t_h imponen una ventana en la que la entrada D del biestable no debe cambiar. En el primer biestable la entrada asíncrona E viola t_{su}, con lo que la salida del primer biestable Ni permanece un tiempo t_r en estado metaestable, tras el cual adopta una salida binaria (en este caso un 1 pero podía ser un 0). Es entonces cuando el segundo biestable puede capturar la señal Ni y ofrecer en su salida E' un valor estable. La condición que se debe cumplir es:

$$T_{minimo} > t_r + t_{su} \tag{6.9}$$

6.6 Conclusiones

En el presente capítulo se ha mostrado como la construcción física de las puertas lógicas hace que no se cumpla la idealidad de la lógica booleana en ciertas condiciones. Aparecen conceptos como cargabilidad máxima de una puerta o margen de los niveles lógicos. Para evitar tener que hacer un análisis eléctrico de cada circuito lógico se parametrizan las prestaciones y de los dispositivos lógicos con los parámetros lógicos.

Dentro de los parámetros estáticos están los niveles de tensión que definen los valores lógicos. Estos niveles en las salidas varían con la conexión de dispositivos al cambiar el paso de la corriente eléctrica. Para evitar estos problemas y tener una cierta tolerancia al ruido electromagnético, se definen los márgenes del 0 y del 1 y los niveles máximos y mínimos de tensión y corriente de las entradas y salidas. Violar estos parámetros provocará que no se reconozcan los valores lógicos y pueda no funcionar el circuito.

De la misma manera, la existencia de las inevitables capacidades parásitas, obliga a caracterizar a los circuitos digitales con parámetros de tiempo de propagación de la señal. Esta característica, unida a la existencia del tiempo de establecimiento y de mantenimiento en los circuitos secuenciales, hace que el bucle de realimentación en la excitación de los biestables limite la frecuencia máxima de funcionamiento de los S. S.

Como apunte final, cabe hacer notar que dentro de una familia lógica pueden coexistir diversas subfamilias con rangos de parámetros distintos. De esta manera puede estar el rango militar y el comercial. Dentro del rango militar se tendrán parámetros más óptimos garantizados en condiciones de temperatura muy adversas (p. eje. desde -55^o hasta 125^o). En el rango comercial las temperaturas en las que se garantizan los parámetros de funcionamiento están típicamente dentro del rango de 0^o hasta 70^o.

Capítulo 7

Lógica TTL y ECL

7.1 Introducción

La lógica booleana aparecía como un sencillo formalismo matemático. Posteriormente, a partir de la implementación de estas puertas ideales en dispositivos físicos, se observa como la idealidad matemática viene limitada por unos parámetros tecnológicos. A pesar de que los parámetros lógicos describen el comportamiento de la lógica booleana con la suficiente precisión para realizar diseños digitales correctos, es también interesante conocer el funcionamiento interno y la constitución de las puertas lógicas. De esta manera se conocerán las limitaciones de cada familia y se podrá escoger una u otra familia en función de las características del diseño.

Una de las primeras lógicas desarrolladas es la lógica transistor-transistor o lógica TTL (*Transistor Transistor Logic*). Esta lógica se caracteriza por que los transistores BJT que incorpora funcionan básicamente en corte/saturación.

La lógica TTL estándar inicialmente competía con las primeras familias CMOS, siendo su principal ventaja frente a éstas su velocidad y su principal desventaja su consumo. La lógica TTL estándar ha sido ampliamente superada en prestaciones por las familias TTL y CMOS avanzadas, pero el conocimiento de su funcionamiento y características es fundamental para la comprensión de sus evoluciones posteriores.

La familia TTL estándar, al igual que el resto de familias TTL, se caracteriza por estar basada en la tecnología bipolar. Todas las puertas TTL estándar están construidas a partir de transistores BJT, cuyo funcionamiento se supondrá que es conocido en el presente capítulo.

Los grandes fabricantes mantienen la línea de producción de la familia TTL estándar debido a su bajo coste, pero sugiriendo simultáneamente al diseñador la migración a familias más avanzadas.

La notación de los dispositivos de las familias TTL sigue una convención que indica el fabricante, la familia y la función del dispositivo. De esta manera, por ejemplo el dispositivo SN74S00N las iniciales SN indican el fabricante (en este caso *Texas Instruments*, el 74 indica que el dispositivo es de niveles TTL pero de rango comercial, la S indica que es de la familia TTL-Schottky (que se analizará en el capítulo 7), el 00 indica que se trata de un circuito integrado con 4 puertas NAND de 2 entradas, y la N final

indica el encapsulado, en este caso un DIP plástico.

Todas las familias TTL se pueden encontrar en rango comercial (indicado por el 74) y en rango militar (indicado por un 54). Para la familia TTL estándar no hay una letra que indique la familia al ser ésta la primera que se fabricó.

La notación de la función de los circuitos depende de cada fabricante y no es estándar. De hecho esta notación sólo la siguen los fabricantes para las familias TTL. Así por ejemplo, el nombre MC7400 corresponde a un circuito integrado de *Motorola* con 4 puertas lógicas NAND de la familia TTL. La funcionalidad viene de nuevo indicada por el 00 del final del nombre, pero en el caso del circuito MC14000 el final 00 no indica puertas NAND al ser una familia CMOS. Se puede encontrar en [CGT93], [Mar96] y en [Toc93]un profundo análisis de las diversas familias lógicas TTL y ECL.

7.2 Puerta TTL básica: El transistor multiemisor

La mejor manera de entender el funcionamiento de la lógica TTL será empezar analizando el comportamiento de la puerta lógica TTL básica de la figura 7.1. Esta puerta NAND no es la que realmente está implementada en la lógica TTL estándar, pero servirá para ilustrar el funcionamiento del transistor multiemisor que se utiliza en las familias TTL. El presente ejemplo se basa en el excelente libro **Tecnologías digitales: De la teoría a la práctica** de *Pedro Casanova et alt.*, editado por **Ed. Paraninfo**.

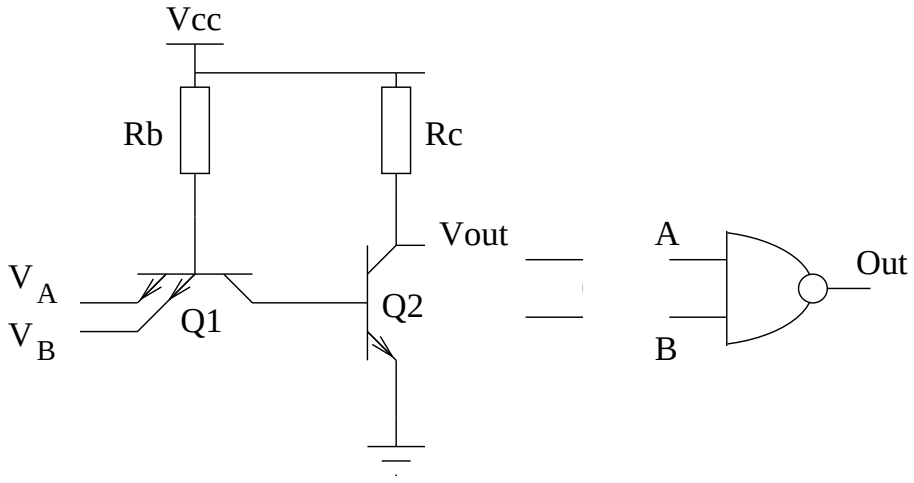


Figura 7.1: *Puerta NAND TTL básica*

Para el análisis del circuito se van a suponer niveles lógicos TTL ideales, con lo que un cero lógico serán 0 voltios y un uno lógico serán 5 voltios.

El transistor multiemisor de la figura 7.1 tiene dos emisores y un funcionamiento similar al transistor BJT normal. Si se polariza directamente uno de los dos diodos base-emisor del transistor la corriente de base atravesará este diodo (o los dos si ambos están polarizados).

Si en V_A y en V_B se tiene un nivel alto, entonces los dos diodos base-emisor de Q1 no se pueden polarizar directamente, polarizándose el diodo base-colector. Esta corriente que baja por el colector de Q1 polariza directamente el diodo base-emisor de Q2, saturando a este transistor (aunque esto dependerá del valor de la β de Q2 y de la

relación entre las resistencias Rb y Rc. En definitiva para la entrada $V_A = V_B = 5\ V$ (A=B=1) la salida será $V_{CEsat}(Q2) \approx 0\ V$ (Out=0).

Sin embargo si en V_A o en V_B se tiene un nivel bajo, entonces se polariza el diodo base-emisor correspondiente (o los dos si ambos están a nivel bajo). Ya no hay corriente de base en Q2 y por tanto éste estará cortado, presentándose en V_{OUT} un nivel alto al no caer tensión en Rc.

El comportamiento de la puerta analizada es por tanto como sigue: Si ambas entradas son altas la salida es baja, y si alguna de las entradas es baja la salida es alta. Este comportamiento corresponde a una puerta lógica NAND. La estructura analizada de dos transistores de la puerta básica de la figura 7.1 no se utiliza en realidad para implementar puertas TTL por los problemas de *fan-out* analizados en la sección 6.2. Para construir una puerta lógica con un funcionamiento más correcto cuando se carga la salida se ha diseñado un circuito algo más complejo.

7.3 Puerta TTL estándar

La puerta NAND TTL estándar se muestra en la figura 7.2. En ésta se observa como en la entrada está el transistor multiemisor Q1, cuyo funcionamiento ha sido analizado anteriormente. Este transistor realiza la función lógica AND, es decir, si alguno de sus emisores (que son las entradas de la puerta NAND) está a 0 la corriente de la base deriva por ese emisor.

El transistor Q2 se denomina *separador de fase* o etapa inversora ya que si Q1 conduce Q2 está cortado y viceversa. La etapa de salida, también llamada *totem pole*, está formada por el transistor Q3, que realiza la carga de la capacidad de salida o puesta a uno por conducción, el transistor Q4, que realiza la descarga de la capacidad de salida o puesta a cero por saturación, y el diodo D_1 que limita los picos de corriente cuando Q3 y Q4 conducen simultáneamente. Los diodos D_A y D_B protegen al transistor Q1 cuando las entradas sean tensiones negativas.

7.3.1 Entradas a nivel alto

Cuando ambas entradas están a nivel alto ambos diodos base-emisor no se pueden polarizar directamente, con lo que la corriente de base de Q1 polariza el diodo base-colector. Ésta es la corriente I_1 que aparece en la figura 7.3. La corriente I_1 polariza directamente el diodo base-emisor de Q2, con lo que Q2 estará en saturación.[1]

El hecho de que Q2 esté en saturación hace que aparezca la corriente I_4 que satura a su vez a Q4, y hace que Q3 esté cortado, con lo que en la salida se tiene que $V_{OUT} = V_{OL} = V_{CEsat} = 0'2\ V$, que es un nivel bajo. Por tanto para ambas entradas a nivel alto se tiene como salida un cero lógico, tal como cabía esperar de una puerta NAND.

Para justificar que Q3 estará cortado sólo hay que tener en cuenta que la tensión en la base de Q3 debería ser 0'7 voltios mayor que en el emisor para que se polarizara directamente su diodo. Como Q4 está saturado se tiene que $V_C(Q4) = 0'2\ V$. Si se le suma la caída de tensión del diodo D_1 $V_D = 0'7\ V$ se tiene que $V_E(Q3) = 0'9V$, con lo

[1] Se deja al estudiante la comprobación de que el transistor Q2 está en saturación y no en conducción. Para ello se supone una $\beta \approx 100$.

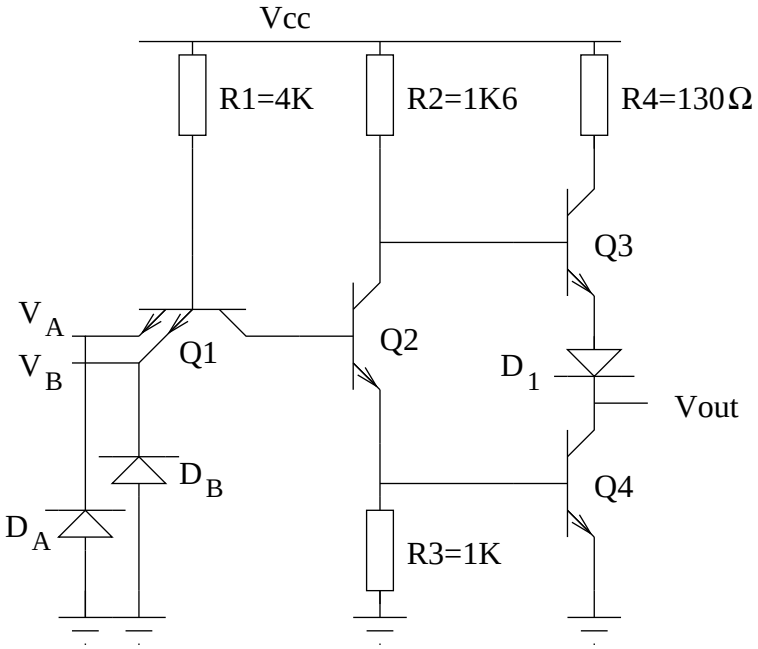


Figura 7.2: *Puerta NAND TTL estándar*

que la tensión en la base de Q3 debería ser $V_B(Q3) = 0'9 + 0'7 = 1'6\ V$, cosa que no es cierta al ser $V_B(Q4) = 0'7 = V_E(Q2)$ y al estar saturado Q2 $V_C(Q2) = V_E(Q2) + 0'2 = 0'9\ V = V_B(Q3)$.

El cálculo de las corrientes y tensiones de los nodos de la puerta NAND con entradas a nivel alto es sencillo.

$$I_1 = \frac{V_{CC} - V_{BC}(Q1) - V_{BE}(Q2) - V_{BE}(Q4)}{4K} = \frac{5 - 0'7 - 0'7 - 0'7}{4K} \approx 0'73mA \quad (7.1)$$

$$I_2 = \frac{V_{CC} - V_{CE}(Q2) - V_{BE}(Q4)}{1K6} = \frac{5 - 0'2 - 0'7}{1K6} \approx 2'6mA \quad (7.2)$$

$$I_3 = \frac{V_{BE}(Q4)}{1K} = 0'7mA \quad (7.3)$$

$$I_4 = I_1 + I_2 - I_3 \approx 2'6mA \quad (7.4)$$

Cabe hacer notar que la corriente en la salida a nivel bajo será de entrada en la puerta y por tanto de signo positivo. Si esta corriente llega a ser β veces la corriente de base de Q4 (I_4) entonces el transistor saldrá del estado de saturación pasando a conducción, con lo que $V_{CE} > V_{CEsat}$ y el nivel bajo de salida superará el nivel de $0'2\ V$. Esto se analizará con más detalle cuando se analice la influencia de la carga en la salida en la sección 7.4.2.

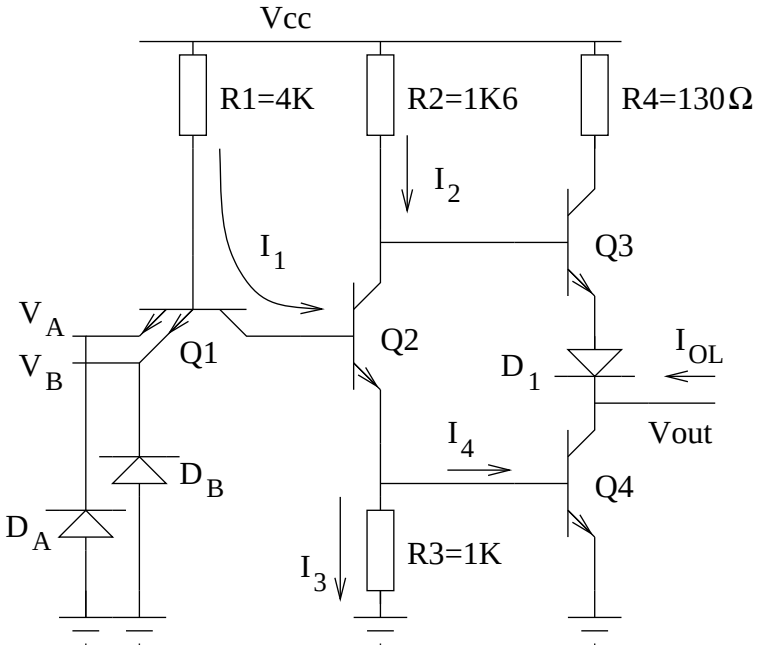


Figura 7.3: *Corrientes en la puerta NAND TTL estándar con entradas a nivel alto*

7.3.2 Alguna entrada a nivel bajo

En el caso de que haya alguna entrada a nivel bajo, la corriente que baja por la base de Q1 polariza el diodo base-emisor correspondiente (o ambos si ambas entradas están a nivel bajo), tal como se muestra en la figura 7.4.

Al no bajar corriente por la base de Q2 no se puede polarizar su diodo base-emisor, con lo que Q2 estará cortado. Esto a su vez corta a Q4, al no bajar corriente que polarice su diodo base-emisor. La corriente que baja por R2 se desvía por tanto por la base de Q3, polarizando este transistor que estará en la zona de conducción. La tensión de salida será por tanto un nivel alto dependiendo este valor de la corriente de salida I_{OH}, que a su vez dependerá de la impedancia conectada a la salida. Cabe hacer notar que en este caso el nivel típico estará por debajo de los 5 voltios, al caer tensión tanto en R2, como en el diodo base-emisor de Q3 y en el diodo de salida D_1. Esto se analizará con detalle en la sección 7.4.2.

7.4 Características funcionales de la lógica TTL

7.4.1 Niveles lógicos

Todas las características apuntadas en esta sección se supondrán para una alimentación de $5\ V \pm 5\%$ y una temperatura típica de 25^oC. Con estos parámetros se van a obtener los valores de los márgenes de tensión, tal como se ha hecho en la sección 6.2.1.

Los valores umbrales para los valores de entrada son una característica tecnológica de la puerta, y en este caso se tiene que $V_{ILumbral} = 1V$ y $V_{IHumbral} = 1'4V$. Si se desea un margen de ruido para la entrada a nivel alto y bajo de $V_{ML} = V_{MH} = 0'4V$, se

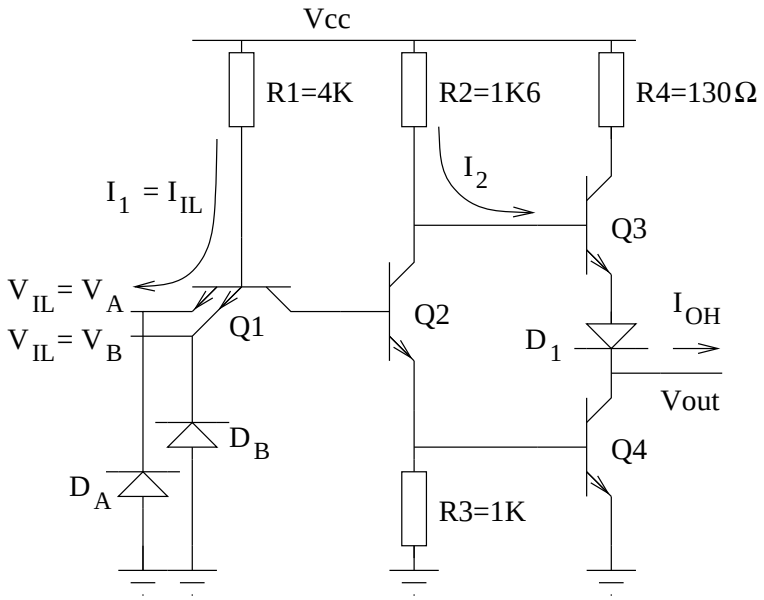


Figura 7.4: *Corrientes en la puerta NAND TTL estándar con alguna entrada a nivel bajo*

obtendrá que:

$$V_{ILmax} = V_{ILumbral} - V_{ML} = 1 - 0'4 = 0'6V \tag{7.5}$$
$$V_{IHmin} = V_{IHumbral} + V_{MH} = 1'4 + 0'4 = 1'8V \tag{7.6}$$

Análogamente si se desea tener un margen de ruido en la salida también $V_{ML} = 0'4$ V, se obtendrá que:

$$V_{OLmax} = V_{ILmax} - V_{ML} = 0'6 - 0'4 = 0'2V \tag{7.7}$$
$$V_{OHmin} = V_{IHmin} + V_{ML} = 1'8 + 0'4 = 2'2V \tag{7.8}$$

A partir de estos niveles ya se puede calcular el margen de transición con márgenes de ruido que será $V_{IHmin} - V_{ILmax} = 1'2V$. Análogamente también se podría calcular el margen de transición sin márgenes de ruido, que será $V_{IHumbral} - V_{ILumbral} = 0'4$ V.

Como resultado interesante se tiene que el margen del uno en la lógica TTL estándar es mayor que el margen del cero, con lo que los problemas de violación de niveles de tensión al suministrar más corriente de la permitida se producirán antes a nivel bajo que a nivel alto.

7.4.2 Corrientes eléctricas: Influencia de la carga en la salida

Para observar el comportamiento de las salidas TTL se van a interconectar varias entradas TTL a éstas. Sólo se va a tener en cuenta la parte de la puerta que está activa, desechando la parte de la etapa de salida que no interviene a nivel alto o bajo. Así por ejemplo en la salida a nivel bajo sólo interviene el transistor Q4, y en la salida a nivel alto sólo interviene el transistor Q3 y el diodo D1 de la salida *totem-pole*.

Salida a nivel alto

En la figura 7.5 se puede observar la etapa de salida que interviene en la puesta a uno conectada a n entradas TTL estándar. Cuando no se conecta ninguna impedancia a la salida, la corriente de salida I_{OH} será 0 y por tanto no caerá tensión y se tendrá que $V_{OUT} = V_{CC} = 5\ V$.

Este comportamiento cambia cuando hay una cierta carga. El efecto de la carga de la salida a nivel alto se puede observar a partir de la figura 7.5. La entrada de las puertas conectadas a la salida absorben una cierta cantidad de corriente de entrada I_{IH}. Esta corriente es de pérdidas, al atravesar el diodo base emisor inversamente, y cercana a $5\mu A$. Esta corriente de salida hace que se pierda tensión en la salida al atravesar las resistencias R2 y R4, el colector-emisor de Q3 y el diodo D1. Si en la salida hay n entradas TTL conectadas se cumple que $I_{OH} = \sum_{i=1}^{n} I_{IHi}$.

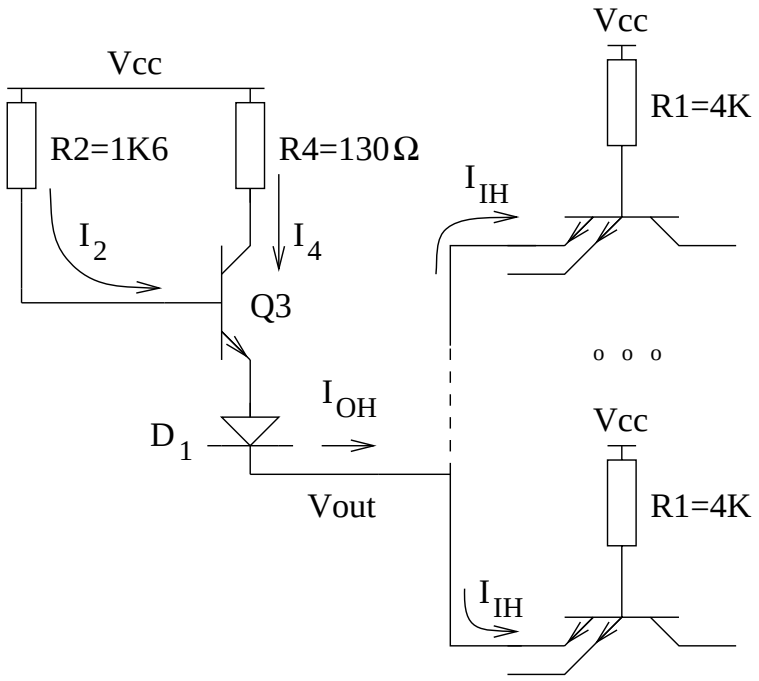


Figura 7.5: *Carga en la puerta TTL a nivel alto*

Cuando mayor sea la corriente de salida I_{OH} mayor tensión caerá en las resistencias y en el colector-emisor del transistor Q3 (Q3 está polarizado en conducción). Por tanto si se conecta a la salida una impedancia muy baja, esto provocará que la corriente de salida I_{OH} sea tan grande que el nivel de salida V_{OUT} sea menor que V_{OHmin}, con lo que el nivel de salida no será reconocido como un uno lógico. Esta corriente para un $V_{OHmin} = 2'2V$ (y por tanto con un margen de ruido de $0'4V$) es del orden de 10 mA.

Cuando la salida está conectada solamente a puertas TTL estándar para superar esta corriente máxima de salida se deberán de conectar muchas puertas, ya que la corriente de entrada a nivel alto es del orden de μA. Estas consideraciones son distintas con la salida a nivel bajo, tal como se muestra en la siguiente sección.

Salida a nivel bajo

De nuevo para considerar la salida a nivel bajo sólo se va a tener en cuenta la parte activa de la etapa. Por este motivo en la figura 7.6 sólo se muestra el transistor Q4, conectado a N entradas TTL.

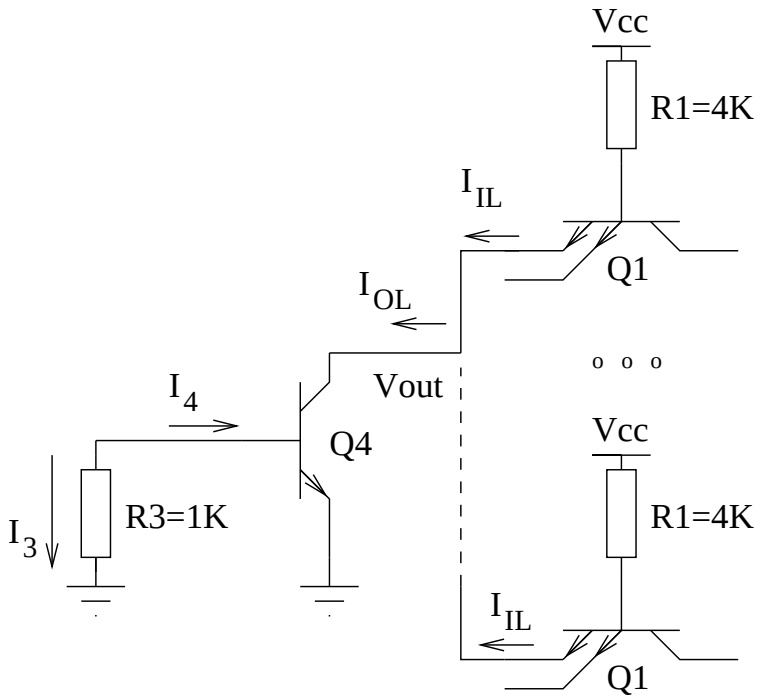


Figura 7.6: *Carga en la puerta TTL a nivel bajo*

De nuevo si no hay ninguna impedancia conectada a la salida y $I_{OL} = 0mA$, se cumplirá que $V_{OUT} = 0V$. En este caso el problema que aparece es que si I_{OL} es muy grande (supera a βI_4) el transistor Q4 sale de saturación, pasando a conducción. De esta manera V_{OUT}, que es la tensión V_{CE} de Q4, crecerá superando $V_{CEsat} = 0'2V = V_{OLmax}$ (estando V_{OLmax} definido para tener un margen de ruido de $0'4V$). Este valor de corriente es de $16mA$, con lo que $I_{OLmax} = 16mA$.

A partir de este resultado y teniendo en cuenta que la corriente I_{IL} de cada entrada TTL no es despreciable como en el caso anterior, se puede calcular el *fan-out* o cargabilidad máxima de una puerta. Como $I_{OL} = \sum_{i=1}^{n} I_{ILi}$, $I_{OLmax} = 16mA$ y $I_{IL} = 1'6mA$ para $V_{IL} = 0'2\ V$, se cumplirá que el fan-out$=\frac{16mA}{1'6mA} \approx 10$.

En el caso de la familia lógica TTL-estándar los problemas de cargabilidad máxima aparecen en la tensión a nivel bajo. ya que la corriente de entrada a nivel bajo es mayor que la corriente de entrada a nivel alto y el margen del cero es menor que el margen del uno.

Se puede extraer más corriente de una puerta TTL estándar que la especificada por los valores I_{OHmax} y I_{OLmax}. El problema es que entonces los niveles de tensión infringirán los valores V_{OHmin} y V_{OLmax}, con lo que el margen de ruido que se había tenido en cuenta para definirlos ya no existirá. De hecho se podría extraer por la salida totem-pole tanta corriente como se quiera hasta llegar a los valores umbrales $V_{IHumbral}$ y $V_{ILumbral}$. El problema aparecerá cuando el mínimo ruido electromagnético en el nodo

Parámetro	Valor típico	Valor máximo	Unidades
t_{PLH}	11	22	ns
t_{PHL}	7	15	ns

Tabla 7.1: *Parámetros dinámicos de la puerta SN7400 (puerta NAND de la familia TTL estándar de Texas Instruments)*

de interconexión haga que el nivel de tensión pase a la zona de transición y la lógica deje de funcionar.

Otro problema es que, si se extrae más corriente del circuito electrónico de la permitida, directamente se puede romper el transistor de salida. Efectivamente las uniones de los transistores BJT pueden soportar una corriente y disipar una potencia máxima, con lo que no basta con ser consciente de la disminución de los márgenes de ruido cuando se extrae más corriente de la permitida. Esto puede *quemar* transistores del circuito electrónico.

Es por tanto importante restringir el funcionamiento a los parámetros definidos por el fabricante, y en el caso de tener que superar los límites, analizar de forma cuidadosa los niveles de ruido permitidos y las corrientes que van a soportar los transistores.

7.4.3 Otros parámetros

Dentro de estas categoría vendrían definidos otros parámetros de absoluta importancia como es el consumo.

Inicialmente la lógica TTL estándar presentaba unos retrasos menores, y por tanto una velocidad de funcionamiento mayor, que las primeras familias CMOS con las que competía. Por contra presentaba un consumo estático mayor que sus equivalentes CMOS. Hoy en día la familia TTL estándar se ha visto ampliamente superada por sus sucesivas mejoras, aunque para aplicaciones de pocas prestaciones puede ser una buena elección por su bajo coste.

Los parámetros dinámicos para la puerta SN7400, (puerta NAND de la familia TTL estandar con márgenes de funcionamiento comercial), tal como se han definido en la sección 6.3 se muestran en la tabla 7.1.

El fabricante indicará las condiciones de medida (el tipo de carga que se ha conectado a la salida) en cada caso. Los valores típicos se supondrán para una temperatura de 25^oC y una alimentación de 5V. Los valores máximos se supondrá garantizados para la temperatura más alta dentro del rango comercial (70^0C) y la alimentación más desfavorable ($4'75V$).

Valores típicos de tiempos de establecimiento (tiempo de *set-up*) y de mantenimiento (tiempo de *hold*), en el biestable maestro-esclavo D de esta familia lógica son $t_{SU} = 20ns$ y $t_h = 5ns$.

Por otra parte hay que hacer notar como en esta lógica hay un consumo, independientemente del valor de la salida y de la frecuencia de funcionamiento. El consumo será distinto cuando la salida es un nivel alto y cuando la salida es un nivel bajo ya que se habilitan distintos transistores y las corrientes que atraviesan el circuito son distintas. En el caso de la puerta SN7400 el consumo de corriente cuando la salida es

alta es de $I_{CCH} = 1'1mA$ y cuando la salida es baja es de $I_{CCL} = 3'4mA$ (el consumo es mayor cuando la salida está a nivel bajo). Con estos datos se obtiene la potencia media consumida que será $P_D = \frac{1'1+3'4}{2} V_{CC} = 11.25mW$ por puerta lógica.

7.5 Familias TTL avanzadas

En la lógica TTL la mayoría de los transistores BJT trabajan en corte/saturación. Esto implica que la capacidad parásita base-emisor, (que es la más importante en estos transistores), debe cargarse y descargarse de forma total. La base debe de llenarse de *cargas* para pasar el transistor de corte a saturación, y estas cargas debe evacuarse para pasar de nuevo el transistor BJT a corte. Este funcionamiento en corte/saturación del transistor BJT impone una serie de retrasos que ralentizan el funcionamiento de la lógica TTL. Para reducir este problema se han realizado dos aproximaciones distintas. Por una parte se utiliza una mejora del transistor BJT (el transistor Schottky), evitando que el transistor se sature.

Hoy en día las familias TTL avanzadas se utilizan en aplicaciones que requieren un cierto margen de prestaciones. La elección de una familia u otra se reduce al compromiso entre la velocidad, el consumo y el coste.

Mayor velocidad por regla general implicará un mayor consumo y viceversa. Las características de un diseño concreto indicarán las familias más adecuadas en cada caso [CGT93], [Wak00], [Toc93].

7.5.1 El transistor Schottky

El principal retraso en la lógica TTL se debe a que la mayoría de sus transistores funcionan en corte/saturación, ralentizándose el cambio de un estado a otro debido a la carga/descarga de su capacidad de base. Para evitar este problema se utiliza el transistor **Schottky** cuya símbolo y estructura se muestran en la figura 7.7.

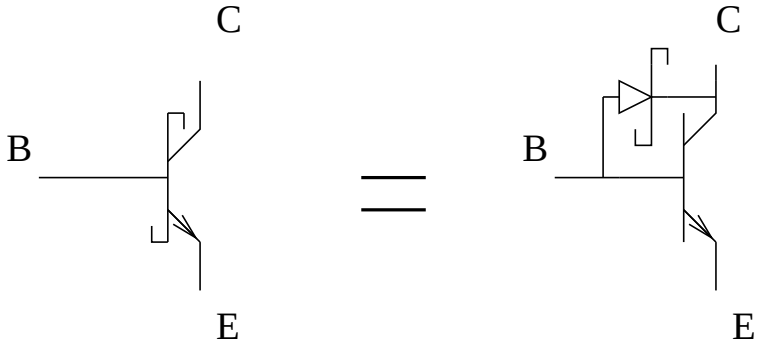


Figura 7.7: *Estructura del transistor Schottky*

El transistor Schottky está a partir del transistor BJT, añadiendo un diodo Schottky que conecta la base del transistor BJT con su colector. La caída de tensión del diodo Schottky es de $0'2\ V$ cuando está polarizado directamente.

El funcionamiento de este transistor se puede razonar partiendo del funcionamiento del transistor BJT y del diodo Schottky, que tiene el funcionamiento normal de un

diodo, salvo que su caída de tensión que es de $0'2\ V$. Para ello puede observarse el inversor mínimo Schottky de la figura 7.8.

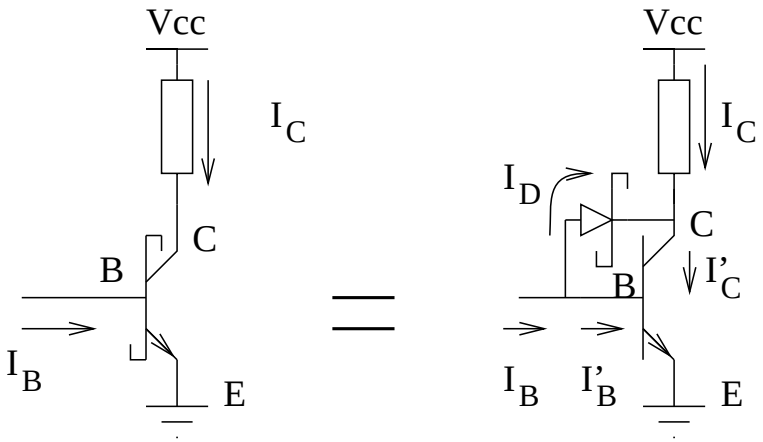


Figura 7.8: *Inversor mínimo Schottky*

Cuando V_{BE} sea 0 el transistor BJT estará cortado, y seguirá cortado hasta que $V_{BE} = 0'7\ V$. Entonces se polarizará directamente el diodo base-emisor y empezará a conducir una corriente de colector I_C. Si se aumenta la corriente de base aumentará la corriente de colector (el transistor estará en conducción). Esto hará que la tensión en el colector V_C disminuya.

Cuando $V_B = V_C + 0'2\ V$ entonces el diodo Schottky se pondrá a conducir una corriente I_D fijando la tensión de colector-emisor $V_{CE} = 0'5\ V$, ya que $V_{CE} = V_{BE} + V_{CB} = 0'7 - 0'2 = 0'5\ V$. Si se aumenta la corriente de base el exceso de corriente se desviará por I_D, ya que la tensión $V_{CE} = 0'5\ V$ fija la corriente de colector I'_C y por tanto, al estar en conducción, la corriente de base I'_B.

Se puede comprobar que por mucho que se aumente la corriente de base I_B el sistema se auto-regula y la tensión V_{CE} no baja de 0'5 voltios, con lo que el transistor BJT nunca se satura. Análogamente se puede decir que el transistor Schottky se *satura* con una $V_{CE} = 0'5\ V$, con lo que en la base se almacenan menos cargas, y por tanto su carga/descarga es más rápida.

La utilización del transistor Schottky acelerará por tanto el paso de corte a conducción y de conducción a corte de los transistores. Este transistor, además de mejoras en el circuito electrónico, originaron la evolución de las familia TTL estándar.

7.5.2 Puerta TTL-S

La primera evolución de la familia TTL-estándar, además de incorporar transistores Schottky, presentaba otras mejoras en el circuito electrónico que se muestran en la figura 7.9. Esta familia incorpora la letra S como distintivo identificador. De esta manera, la puertas SN74S00 corresponderá al circuito de *Texas Instruments* con 4 puertas NAND de la familia TTL-Schottky.

Una de las características que se pueden observar rápidamente es que las resistencias de la puerta NAND TTL-S son menores que su equivalente TTL-estándar. Esto se debe a que, de manera adicional al resto de cambios, se hace que las corrientes sean mayores

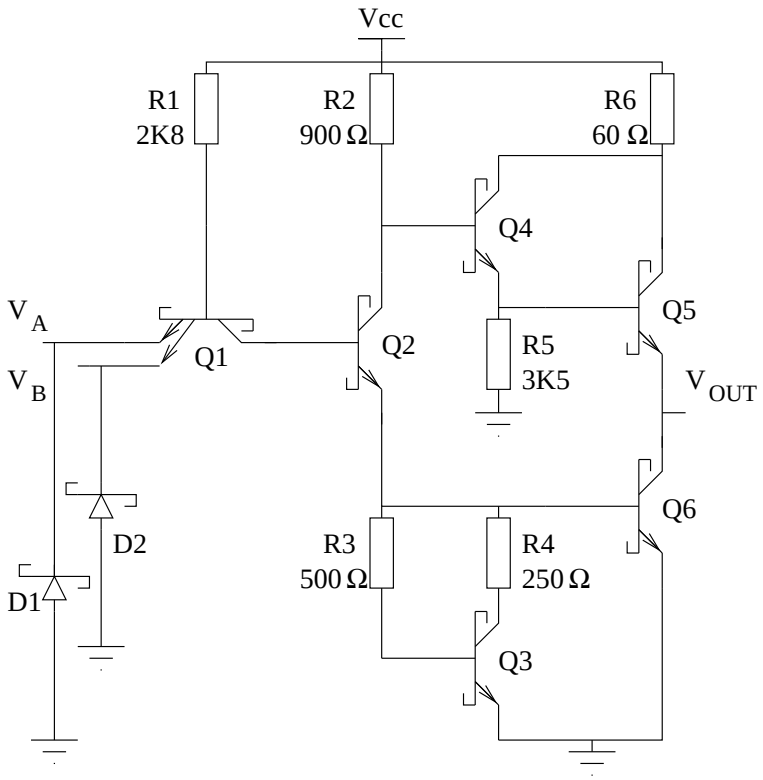


Figura 7.9: *Puerta NAND TTL Schottky*

para cargar/descargar más rápidamente las inevitables capacidades parásitas. Esto también puede verse como que se disminuye R para disminuir los productos RC a los que son proporcionales los retrasos de la puerta, tal como se ha justificado en la sección 6.3. El problema que va a presentar esta estrategia será que al ser mayores las corrientes el consumo estático será mayor, lo cual será el problema básico de esta familia.

La puerta TTL-S presenta de nuevo un transistor multiemisor de entrada Q1 que realiza la función AND. El transistor Q2 realiza la inversión de fase, y se mantiene a la salida la configuración *totem-pole* pero con ligeras modificaciones.

El transistor de puesta a uno se ha substituido por la **configuración Darlington** formada por Q4 y Q5, que funciona como un solo transistor pero con una $\beta = \beta_4 \cdot \beta_5$, con lo que se disminuye la impedancia de salida respecto a la familia TTL estándar.

El transistor de puesta a cero se ha substituido por los transistores Q3 y Q6, donde Q3 forma la **configuración *squaring network*** (red de encuadre). Este nombre proviene de que en realidad el transistor Q3 nunca conducirá estáticamente. Su misión es evacuar las cargas de la base de Q6 en el transitorio en el que Q6 pasa de saturación a corte. De esta manera se consigue acelerar el tiempo t_{PLH} mejorando la pendiente de la curva de carga. De ahí el sobrenombre de *red de encuadre.*

El funcionamiento de esta puerta es similar al funcionamiento de la puerta TTL-estándar. Cuando alguna entrada está a nivel bajo la corriente de base de Q1 baja por el emisor correspondiente. Esto hace que Q2 no tenga corriente de base, con lo que estará cortado. Al estar cortado Q2, Q6 tampoco podrá tener corriente de base, con lo que también estará cortado. La corriente que baja por R2, al estar Q2 cortado, polarizará el par *Darlington* Q4 Q5, con lo que en la salida habrá un nivel alto, cuya

tensión dependerá exactamente de la corriente que se extraiga del circuito.

De la misma manera que ocurría en la puerta TTL-estándar, si ambas entradas están a nivel alto, y no se pueden polarizar directamente los diodos base-emisor del transistor multiemisor Q1, se polarizará directamente el diodo base-colector de Q1. Esto hará que Q2 tenga corriente de base y a su vez Q6 tenga corriente de base. Es trivial comprobar como de nuevo Q3 estará cortado y como la relación de resistencias generarán unas corrientes que hacen que Q6 esté *saturado* en el sentido Schottky de la palabra, (es decir con $V_{CE} = 0'5\ V$). Es de nuevo fácil comprobar como el hecho de que Q2 y Q6 estén *saturados* hace que Q4 y Q5 estén cortados, con lo que a la salida se tendrá un nivel bajo.

El comportamiento es por tanto el esperado para una puerta NAND, es decir si ambas entradas son un nivel alto la salida es un nivel bajo, y si alguna entrada está a nivel bajo la salida estará a nivel alto.

Características funcionales de la lógica TTL-S

Como primer dato a tener en cuenta, de la características del transistor Schottky y de la puerta TTL-S se observa como el nivel más bajo posible será el de 0'5 voltios. Por tanto esto ha de ser tenido en cuenta a la hora de definir los márgenes que definan V_{OLmax}.

De la misma manera que se ha hecho con la lógica TTL-estándar se puede obtener, a partir de la función de transferencia, el valor de $V_{ILumbral} = 1'1\ V$ y el valor de $V_{IHumbral} = 1'3\ V$. Esto da una zona de transición sin márgenes de ruido de $V_{IHumbral} - V_{ILumbral} = 0'2\ V$, menor (y por tanto mejor) que en el caso de la familia TTL estándar.

A partir de los valores umbrales y tomando un margen de ruido a nivel bajo $V_{ML} = 0'3\ V$ y un margen de ruido a nivel alto $V_{MH} = 0'7\ V$ se obtiene:

$$V_{ILmax} = V_{ILumbral} - V_{ML} = 1'1 - 0'3 = 0'8V \tag{7.9}$$

$$V_{IHmin} = V_{IHumbral} + V_{MH} = 1'3 + 0'7 = 2V \tag{7.10}$$

La diferencia en los márgenes de ruido se debe a la diferencia de amplitud del 0 lógico y del uno lógico. De nuevo el uno lógico es más amplio que el cero lógico al igual que en la lógica TTL estándar.

A partir de los valores anteriores y repitiendo los márgenes de ruido para el cero y para el uno se tiene que:

$$V_{OLmax} = V_{ILmax} - V_{ML} = 0'8 - 0'3 = 0'5V \tag{7.11}$$

$$V_{OHmin} = V_{IHmin} + V_{ML} = 2 + 0'7 = 2'7V \tag{7.12}$$

Se observa como en este caso $V_{OLmax} = V_{OLmin} = 0'5\ V$, que es la tensión mínima que puede salir de la puerta TTL-S.

La interconexión con la familia TTL-estándar es posible tal como se puede observar en la tabla 7.2 al ser los niveles compatibles (De hecho esto pasará en todas las familias TTL avanzadas), lo único que hay que tener en cuenta son los márgenes adicionales a los márgenes de ruido que habrán cambiado.

Parámetro	TTL-estándar	TTL-S	Unidades
V_{ILmax}	0'6	0'8	Voltios
V_{IHmin}	1'8	2	Voltios
V_{OLmax}	0'2	0'5	Voltios
V_{OHmin}	2'2	2'7	Voltios

Tabla 7.2: *Comparación de niveles lógicos TTL-estándar y TTL-S*

En el caso de las corrientes el caso es similar al analizado en la sección 7.4.2. Hay que analizar cual es la corriente máxima que se puede extraer del circuito a nivel alto para que no se viole V_{OHmin}, y la corriente que se puede suministrar al circuito para que no se viole V_{OLmax}. Los valores que se obtienen son $I_{OHmax} = -38\ mA$ y $I_{OLmax} = 40\ mA$.

A partir de estos valores, y de la corriente de entrada que consumen las puertas TTL-S, se puede calcular la cargabilidad o *fan-out* de la familia TTL-S. Al igual que en la familia TTL-estándar, la corriente de entrada I_{IH} es una corriente de pérdidas, con lo que el problema de la cargabilidad aparece de nuevo en el nivel bajo.

Se tiene un valor de corriente de entrada a nivel bajo de $I_{IL} = 1'45\ mA$ con una entrada de 0'5 V, que será la aplicada por una puerta TTL-S. Esto da un fan-out= $\frac{40}{1'45} \approx 27$ puertas lógicas TTL-S con un margen de ruido de 0'3 voltios a nivel bajo.

De nuevo cabe hacer notar que se podrían conectar más entradas TTL-S a una salida TTL-S. El problema estribará en que se infringirán los niveles V_{OLmax} y V_{OHmin}, lo cual se podrá hacer siempre que no superen los respectivos niveles umbrales de entrada y sean detectados como ceros y unos lógicos. Si se hace esto cualquier ruido electromagnético acoplado al nodo de interconexión hará que deje funcionar correctamente el circuito, con lo que es necesario mantenerse dentro de los parámetros del fabricante.

Este problema es mínimo comparado con la rotura que se puede producir en la puerta lógica si se pide más corriente de la permitida. Las uniones de los transistores pueden soportar una corriente y disipar una potencia máxima. Por tanto, exceder los parámetros lógicos puede llevar a la rotura irreversible de la puerta, con lo que de nuevo se concluye que es necesario mantenerse dentro de los parámetros del fabricante.

La familia TTL-S puede considerarse como una evolución de la familia TTL-estándar con el objetivo de mejorar los tiempos de propagación. Este objetivo se cumple ampliamente ya que se consiguen tiempos de propagación del orden de 3ns, tal como se muestra en la tabla 7.3. De nuevo los valores típicos son los valores a 25^oC y con una alimentación de 5V. Los valores máximos se supondrá garantizados para la temperatura más alta dentro del rango comercial (70^0C) y la alimentación más desfavorable ($4'75V$). Tiempos típicos de de establecimiento (tiempo de *set-up*) y de mantenimiento (tiempo de *hold*), en el biestable maestro-esclavo D de esta familia lógica son $t_{SU} = 3ns$ y $t_h = 2ns$.

En lo referente al consumo, en esta familia lógica éste se dobla la corriente cuando la salida es alta de $I_{CCH} = 3\ mA$, y cuando la salida es baja de $I_{CCL} = 6\ mA$. Con estos datos se obtiene la potencia media consumida que será $P_D = \frac{3+6}{2} V_{CC} = 22'5\ mW$.

Se puede concluir que la estrategia seguida con la familia TTL-S (reducción del valor de las resistencias, utilización de transistores Schottky, modificación del circuito)

Parámetro	Valor típico	Valor máximo	Unidades
t_{PLH}	3	4'5	ns
t_{PHL}	3	5	ns

Tabla 7.3: *Parámetros dinámicos de la puerta SN74S00 (puerta NAND de la familia TTL-Schottky de Texas Instruments)*

consiguen mejorar la rapidez del circuito, pero aumentando el consumo.

7.5.3 Familia TTL-LS

Con la familia TTL-S se consigue mejorar la rapidez de las puertas en un factor 3 con respecto a la familia TTL-estándar, pero a costa de doblar el consumo.

A partir de la familia TTL-S se desarrolló la familia TTL-LS donde las siglas LS significan *Low-power Schottky*. Esta familia de nuevo presenta la utilización de transistores Schottky, añadiendo mejoras al circuito TTL-S y aumentando el valor medio de las resistencias con respecto a TTL-S y TTL-estándar. De esta manera se consigue una velocidad de conmutación ligeramente mejor que la familia TTL-estándar pero se reduce su consumo respecto a la misma familia en un factor 5.

Características funcionales de la lógica TTL-LS

De la función de transferencia de un inversor de la familia TTL-LS se pueden obtener los valores umbrales $V_{ILumbral} = 1'1\ V$ y $V_{IHumbral} = 1'5\ V$, lo que da una zona de transición de 0'4 voltios. Tomando de nuevo un margen de ruido del cero V_{ML} para la entrada de 0'3 voltios, al igual que en la familia TTL-S ya que de nuevo la salida a nivel bajo tendrá como nivel mínimo $V_{OLmin} = 0'5\ V$ se obtiene que:

$$V_{ILmax} = V_{ILumbral} - V_{ML} = 1'1 - 0'3 = 0'8V \tag{7.13}$$

De la misma manera, para el nivel alto se toma un margen de ruido en la entrada a nivel alto $V_{MH} = 0'7\ V$, con lo que se obtiene:

$$V_{IHmin} = V_{IHumbral} + V_{MH} = 1'5 + 0'7 = 2'2V \tag{7.14}$$

Análogamente si se desea tener en la salida los mismos márgenes de ruido a nivel bajo y alto se tiene que:

$$V_{OLmax} = V_{ILmax} - V_{ML} = 0'8 - 0'3 = 0'5V \tag{7.15}$$
$$V_{OHmin} = V_{IHmin} + V_{ML} = 2'2 + 0'7 = 2'9V \tag{7.16}$$

Estos niveles de entrada y salida garantizan la interconexión entre la familia TTL-LS, y las familias TTL-estándar y TTL-S.

Parámetro	Valor típico	Valor máximo	Unidades
t_{PLH}	9	15	ns
t_{PHL}	10	15	ns

Tabla 7.4: *Parámetros dinámicos de la puerta SN74LS00*

En cuanto a corrientes eléctricas, de nuevo se pueden obtener los valores de corriente en los pines de salida para que no se violen V_{OLmax} y V_{OHmin}. Estos parámetros para los márgenes de ruido apuntados son $I_{OLmax} = 15'5\ mA$, y $I_{OHmax} = -16'8\ mA$.

Los parámetros dinámicos para la puerta SN74LS00, con márgenes de funcionamiento comercial, tal como se han definido en la sección 6.3 se muestran en la tabla 7.4.

Como ejemplo de valores típicos de tiempos de establecimiento (tiempo de *set-up*) y de mantenimiento (tiempo de *hold*), en el biestable maestro-esclavo D de esta familia lógica son $t_{SU} = 20ns$ y $t_h = 5ns$, idénticos a los valores en TTL-estándar.

En el caso de la puerta SN74LS00 el consumo de corriente cuando la salida es alta es de $I_{CCH} = 0'25\ mA$ por puerta y cuando la salida está a nivel bajo es de $I_{CCL} = 0'65\ mA$, con lo que la potencia media disipada será de $P_D = \frac{0'25+0'65}{2} V_{CC} = 2'25\ mW$, que es la quinta parte de la TTL-estándar.

Como conclusión cabe decir que la familia TTL-LS consigue una velocidad similar a TTL-estándar (ligeramente mejor) pero consumiendo la quinta parte que ésta.

7.5.4 Familia TTL-ALS

A partir de la familia TTL-LS se desarrolló la familia TTL-ALS o TTL *Advanced Low-power Schottky*. Las mejoras en esta familia vienen básicamente del rediseño de la constitución de las entradas con transistores PNP, aunque se rediseña toda la puerta que aumenta bastante en complejidad.

Con la familia TTL-ALS se disminuye significativamente el tiempo de propagación de las familias TTL-estándar y TTL-LS, reduciendo simultáneamente el consumo de potencia con respecto a TTL-LS a la mitad (por tanto se reduce el consumo de potencia con respecto a TTL-estándar un décima parte).

Características funcionales de la lógica TTL-ALS

De la función de transferencia de un inversor de la familia TTL-ALS se pueden obtener los valores umbrales $V_{ILumbral} = 1.2\ V$ y $V_{IHumbral} = 1'5\ V$, lo que da una zona de transición de 0'3 voltios. De nuevo se observa como el margen del 0 es bastante más pequeño que el margen del uno, tal como ocurre con toda la lógica TTL. Si se toma un margen de ruido del cero V_{ML} para la entrada de 0'3 voltios, al igual que en la familia TTL-S y TTL-LS, se obtiene que:

$$V_{ILmax} = V_{ILumbral} - V_{ML} = 1'2 - 0'3 = 0'9V \tag{7.17}$$

Parámetro	Valor típico	Valor máximo	Unidades
t_{PLH}	3	11	ns
t_{PHL}	2	8	ns

Tabla 7.5: *Parámetros dinámicos de la puerta SN74ALS00*

De la misma manera, para la entrada a nivel alto con un margen de ruido $V_{MH} = 0'7$ V se obtiene que:

$$V_{IHmin} = V_{IHumbral} + V_{MH} = 1'5 + 0'7 = 2'2V \tag{7.18}$$

Manteniendo los mismos márgenes de ruido para la salida se obtendrá:

$$V_{OLmax} = V_{ILmax} - V_{ML} = 0'9 - 0'3 = 0'6V \tag{7.19}$$
$$V_{OHmin} = V_{IHmin} + V_{ML} = 2'2 + 0'7 = 2'9V \tag{7.20}$$

Para medir las corrientes eléctricas máximas de nuevo hay que tener en cuenta que el valor de tensión de la salida no descienda más del valor V_{OHmin} para el nivel alto, y que no supere V_{OLmax} para el nivel bajo. Con los valores definidos anteriormente, y conservando los márgenes de ruido que han servido para definirlos, se obtiene que $I_{OLmax} = 18'5\ mA$, y $I_{OHmax} = -24\ mA$.

Los parámetros dinámicos para la puerta SN74ALS00, con los márgenes de funcionamiento comercial son los que se adjuntan en la tabla 7.5.

Valores típicos de tiempos de establecimiento (tiempo de *set-up*) y de mantenimiento (tiempo de *hold*), en el biestable maestro-esclavo D de esta familia lógica son $t_{SU} = 18ns$ y $t_h = 2ns$.

El consumo en el caso de la puerta SN74ALS00 cuando la salida es alta es de $I_{CCH} = 0'13\ mA$ por puerta y cuando la salida está a nivel bajo es de $I_{CCL} = 0'45\ mA$, con lo que la potencia media disipada será de $P_D = \frac{0'13+0'45}{2} V_{CC} = 1'45\ mW$.

Como conclusión para esta familia se tiene que se disminuye el consumo de potencia respecto a TTL-LS a la mitad, aumentando simultáneamente la eficiencia de conducción, pero sin ser una familia más rápida que TTL-S.

7.5.5 Familia TTL-AS

Se desarrolló la familia TTL-S como mejora (más rapidez) de la familia TTL-estándar, pero aumentado mucho el consumo. Como mejora de la TTL-S se desarrollo la TTL-LS, y como mejora de la TTL-LS la TTL-ALS consiguiendo un buen compromiso entre potencia disipada y rapidez de la propagación.

A partir de la familia TTL-S se desarrolló, siguiendo otra línea de evolución de los circuitos distinta a TTL-LS, la familia TTL-AS (*Advanced Schottky*). Esta familia, al contrario que la familia TTL-LS que se centraba en bajar la potencia pero aumentando el retraso hasta niveles TTL-estándar, se basaba en mantener básicamente los tiempos

Parámetro	Valor típico	Valor máximo	Unidades
t_{PLH}	1	4'5	ns
t_{PHL}	1	4	ns

Tabla 7.6: *Parámetros dinámicos de la puerta SN74AS00*

de propagación, pero mejorando el consumo. De esta manera TTL-AS mejora los tiempos de propagación de TTL-S, reduciendo significativamente la potencia (casi a la mitad).

Características funcionales de la lógica TTL-AS

Como característica de la lógica TTL-AS, de la función de transferencia de un inversor de la familia TTL-ALS se pueden obtener los valores umbrales $V_{ILumbral} = 1.1\ V$ y $V_{IHumbral} = 1'5\ V$. Estos niveles definen una zona de transición de 0'4 voltios. Si se toma un margen de ruido del cero V_{ML} para la entrada de 0'3 voltios, y un margen de ruido del uno de $V_{MH} = 0'7\ V$, al igual que en la familia TTL-S, TTL-LS y TTL-ALS se obtiene que:

$$V_{ILmax} = V_{ILumbral} - V_{ML} = 1'1 - 0'3 = 0'8V \tag{7.21}$$
$$V_{IHmin} = V_{IHumbral} + V_{MH} = 1'5 + 0'7 = 2'2V \tag{7.22}$$

Análogamente si se desea tener en la salida los mismos márgenes de ruido en la salida a nivel bajo y alto se tiene que:

$$V_{OLmax} = V_{ILmax} - V_{ML} = 0'8 - 0'3 = 0'5V \tag{7.23}$$
$$V_{OHmin} = V_{IHmin} + V_{ML} = 2'2 + 0'7 = 2'9V \tag{7.24}$$

Estos niveles de entrada y salida garantizan la interconexión de todas las otras familias con la lógica TTL.

Se pueden obtener los valores de corriente en los pines de salida para que no se violen V_{OLmax} y V_{OHmin}. Estos parámetros para los márgenes de ruido apuntados son $I_{OLmax} = 65\ mA$, y $I_{OHmax} = -45\ mA$.

Los parámetros dinámicos para la puerta SN74LS00, con márgenes de funcionamiento comercial, tal como se han definido en la sección 6.3 se muestran en la tabla 7.6.

Valores típicos de tiempos de establecimiento (tiempo de *set-up*) y de mantenimiento (tiempo de *hold*), en el biestable maestro-esclavo D de esta familia lógica son $t_{SU} = 4'5ns$ y $t_h = 0ns$.

La potencia media disipada es sensiblemente menor a la disipada por TTL-S ya que $I_{CCH} = 0'5\ mA$ y $I_{CCL} = 2'8\ mA$, con lo que la potencia media disipada será de $P_D = \frac{0'5+2'8}{2} V_{CC} = 8'25\ mW$.

Como características finales de esta familia cabe decir que se consigue la mejor velocidad de las familias TTL con la mitad de consumo de TTL-S.

7.5.6 Familia TTL-FAST

A finales de los 80 se desarrolló la familia TTL *FAST* (*Fairchild Advanced Schottky TTL*) como mejora de la familia TTL-S, siendo esta familia es el ultimo paso en lógica TTL. La lógica TTL-F reduce la potencia disipada por TTL-S en una cuarta parte, y más de la mitad la disipada por TTL-AS. Simultáneamente casi mantiene los niveles de rapidez de TTL-S y TTL-AS.

Hoy en día se puede considerar la familia TTL-F como la mejor familia TTL porque, aunque no es la más rápida, ni es la que menos consume, es la que mantiene un mejor compromiso retraso/potencia disipada.

Características funcionales de la lógica TTL-FAST

De la función de transferencia de un inversor de la familia TTL-F se pueden obtener los valores umbrales $V_{ILumbral} = 1.2\ V$ y $V_{IHumbral} = 1'5\ V$, lo que da una zona de transición de 0'3 voltios. De nuevo se observa como el margen del 0 es bastante más pequeño que el margen del uno, cosa común a todas las familias TTL. Si de nuevo se toma un margen de ruido del cero V_{ML} para la entrada de 0'3 voltios, se obtiene que:

$$V_{ILmax} = V_{ILumbral} - V_{ML} = 1'2 - 0'3 = 0'9V \tag{7.25}$$

De la misma manera, para la entrada a nivel alto con un margen de ruido $V_{MH} = 0'7\ V$ se obtiene que:

$$V_{IHmin} = V_{IHumbral} + V_{MH} = 1'5 + 0'7 = 2'2V \tag{7.26}$$

Manteniendo los mismos márgenes de ruido para la salida se obtendrá:

$$V_{OLmax} = V_{ILmax} - V_{ML} = 0'9 - 0'3 = 0'6V \tag{7.27}$$
$$V_{OHmin} = V_{IHmin} + V_{ML} = 2'2 + 0'7 = 2'9V \tag{7.28}$$

Las corrientes eléctricas máximas con los márgenes de ruido definidos serán $I_{OLmax} = 73'5\ mA$, y $I_{OHmax} = -25'9\ mA$.

Los parámetros dinámicos para la puerta SN74F00 son los que se adjuntan en la tabla 7.7. Valores típicos de tiempos de establecimiento (tiempo de *set-up*) y de mantenimiento (tiempo de *hold*), en el biestable maestro-esclavo D de esta familia lógica son $t_{SU} = 3ns$ y $t_h = 1ns$.

El consumo de esta familia no es menor que TTL-AS, pero mayor que TTL-LS, siendo $I_{CCH} = 0'4\ mA$ y $I_{CCL} = 1'5\ mA$. Esto da una valor para la potencia media disipada por puerta de $P_D = \frac{0'4+1'5}{2} V_{CC} = 4'75\ mW$.

Con estos valores se concluye que la familia TTL-F no es ni la más rápida ni la de menor consumo, pero es la que mantiene un mejor compromiso de todas las familias TTL.

Parámetro	Valor típico	Valor máximo	Unidades
t_{PLH}	1'6	6	ns
t_{PHL}	1	5.3	ns

Tabla 7.7: *Parámetros dinámicos de la puerta SN74F00*

7.6 Utilización de la lógica TTL

En el diseño de sistemas digitales tradicionalmente lo más habitual ha sido utilizar unidades funcionales (como microprocesadores o memorias) junto con módulos estándar (multiplexores, contadores, ...etc). Muchas de las unidades funcionales digitales tienen niveles lógicos compatibles TTL, y dentro de las familias TTL se pueden encontrar una gran variedad de módulos estándar, lo que facilita el diseño de sistemas digitales complejos sin cambiar de familia lógica.

En un principio la familia TTL estándar era utilizada para el diseño de sistemas digitales sencillos, presentando como ventaja su rapidez frente a las primeras familias CMOS, y como desventaja su consumo. Para el diseño de sistemas basados en unidades funcionales se utilizaba como lógica de pegado o *glue logic*, aunque en la actualidad para esta función se utilizan mayoritariamente dispositivos programables, tal como se mostrará en el capítulo 9.

En la actualidad la lógica TTL se ha visto ampliamente superada en prestaciones (velocidad y consumo) por las familias TTL avanzadas. Incluso las familias CMOS y BiCMOS avanzadas superan en prestaciones la familia TTL estandar. La lógica TTL sin embargo se utiliza todavía en aplicaciones en las que las prestaciones que ofrece sean suficientes, debido a su bajo coste.

Dentro de la lógica TTL, además de la salida *totem-pole*, se tienen muchos dispositivos con salida en *colector abierto*. Una salida TTL estándar en colector abierto, tal como muestra la figura 7.10. A estas puertas para que funcionen correctamente habrá que añadirles necesariamente una resistencia de *pull-up* (conectada a alimentación).

Las salidas en colector abierto pueden tener diversas utilizaciones, que se pueden agrupar en 3 bloques:

1. **Actuar directamente sobre un LED**: Si se substituye la resistencia de colector Rc de la figura 7.10 por una resistencia en serie y un LED éste emitirá luz cuando el transistor Q4 esté en saturación.
2. **Realizar la función AND cableada**: Si se tienen varias salidas en colector abierto éstas se pueden cortocircuitar y conectar a V_{CC} mediante una sola resistencia de colector. En ese nodo se realizará la función AND cableada ya que cuando uno de los transistores Q4 de salida conduzca, (ponga un cero en la salida), en el nodo habrá un cero aunque el resto de salidas quieran poner un uno (es decir el resto de transistores estén cortados). En la figura 7.11 se muestra una función AND cableada construida con puertas NOT TTL en colector abierto.
3. **Conectar una salida a un bus con múltiples entradas**: La salida en colector abierto sirve para conexiones a buses, o equivalentemente, conexiones a un gran número de entradas. Estos nodos tendrán una gran capacidad parásita, y por tanto será necesario optimizar el valor de la resistencia de colector para que la corriente de carga sea lo mayor posible, y por tanto el retraso sea el menor posible. Por

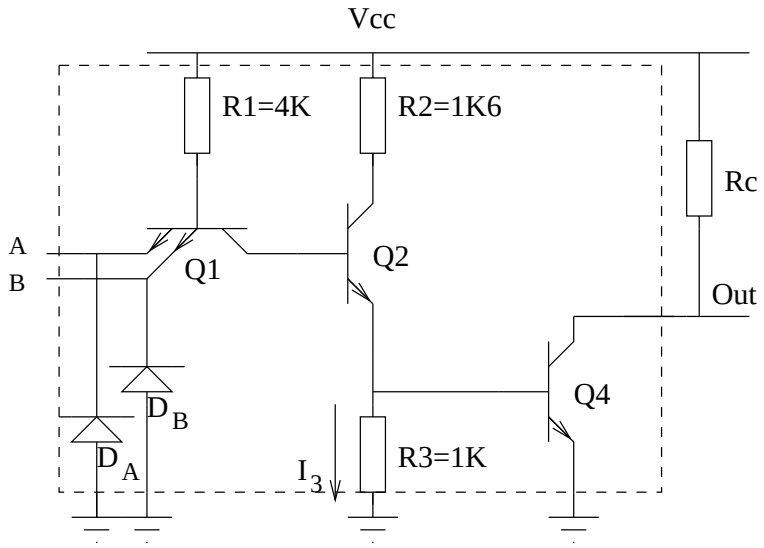


Figura 7.10: *Puerta NAND TTL en colector abierto*

el contrario el valor de Rc no puede ser muy pequeño, ya que entonces cuando el transistor de salida Q4 esté saturado si la corriente de colector es muy grande puede hacer pasar el transistor Q4 a conducción, con lo que que V_{OL} podrá ser mayor que V_{OLmax}, y por tanto perderse los márgenes de ruido e incluso dejar de funcionar. El valor de Rc será por tanto siempre un compromiso entre ambos valores. Valores típicos de Rc suelen tomarse cercanos a 5K, aunque se recomienda que se optimice su cálculo para cada caso.

7.7 Lógica ECL

El problema fundamental de la familia TTL-estándar es que sus transistores funcionan en corte saturación y esto obliga a descargar/cargar de forma completa las capacidades parásita de la base de los transistores BJT. Como solución a esto apareció el transistor Schottky que nunca pasaba a saturación (o equivalentemente se saturaba con $V_{CE} = 0'5$ V), además de mejoras en los circuitos que aumentaban la eficiencia de conducción.

En paralelo al desarrollo de las familias TTL surgió una lógica que también utiliza el principio de no saturar los transistores BJT. Esta lógica se llama ECL (*Emitters Coupled Logic*) o lógica acoplada por emisor. De esta lógica hay dos familias: la ECL 10K y la ECL 100K. Ambas son prácticamente idénticas, salvo que la ECL 100K es un poco más rápida.

La lógica ECL se caracteriza por que sus transistores BJT trabajan siempre en corte/conducción, lo que hace que se consigan velocidades de propagación del orden del nanosegundo. Como desventaja a su gran rapidez se tiene que son familias que consumen mucho estáticamente, llegándose a superar los 20 mW por puerta.

Otra particularidad de la lógica ECL es que utiliza tensiones negativas y la definición de los niveles lógicos, es radicalmente diferente a los niveles de las familias TTL y CMOS. Un cero lógico serán -5'2 voltios y un uno lógico se corresponde con una tensión de 0 voltios.

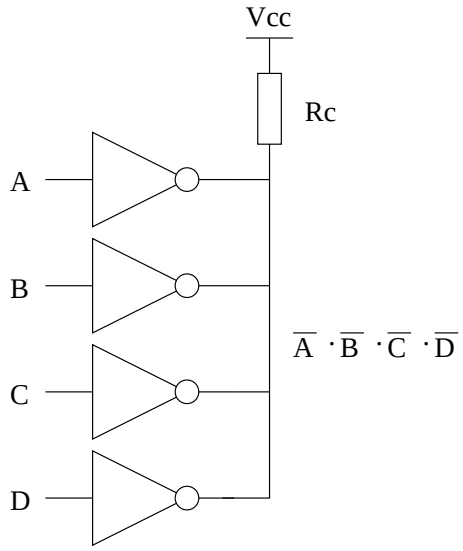


Figura 7.11: *Función AND cableada realizada con inversores TTL en colector abierto*

La lógica ECL se usa básicamente para dispositivos que tengan que funcionar a muy alta velocidad como sistemas de comunicaciones, o de cálculo a alta velocidad. Su utilización, está siendo relegada a pequeños dispositivos programables de alta velocidad debido a su alto consumo. Actualmente las lógicas BiCMOS y de bajo voltaje consiguen velocidades similares con un menor consumo.

7.7.1 Puerta lógica ECL básica: Funcionamiento

La puerta ECL básica se muestra en la figura 7.12. Se puede observar como el diseño de esta puerta no guarda ninguna relación con la filosofía TTL. En este caso la puerta básica es una puerta NOR al contrario de la lógica TTL donde la puerta básica siempre es una puerta NAND. Las entradas están en la base de transistores BJT, que siempre están polarizados en corte/conducción, y las resistencias de colector son muy bajas, lo que justifica el alto consumo de esta lógica.

Como particularidad adicional cabe destacar su incompatibilidad con la lógica TTL al corresponderse el nivel alto con 0 voltios y el nivel bajo con -5'2 voltios.

La tensión en V_{ref} se supone aproximadamente fija al despreciar la corriente de base de Q4 frente a la corriente I1 que bajará por el divisor de tensión formado por R3 y R6. Está tensión será de -1'15 voltios.

Si se tiene un nivel alto (0 voltios) en cualquiera de las entradas de la puerta se se conseguirá que produzca corriente base-emisor, lo que polarizará el diodo correspondiente y fijará una tensión V_E(Q1)=V_E(Q2)=V_E(Q3)=-0'7 V. Esta tensión en el emisor de Q3 cortará a Q3 ya que $V_B(Q3) = -1'15\ V$. Al estar Q3 cortado la corriente I2 derivará por la base de Q6, polarizando directamente su diodo base-emisor y conectando la la salida V_2 con la tensión mas alta del circuito (0 voltios).

Si en ambas entradas se tiene un cero lógico (-5'2 voltios) Q1 y Q2 estarán cortados. Esto hará que la corriente I3 derive por la base de Q5 y en la salida V_1 se tenga un 1 lógico. Simultáneamente Q3 estará conduciendo, lo que cortará a Q6, con lo que en el emisor de Q6 se tendrá un nivel bajo (-5'2 voltios). Este comportamiento se corresponde

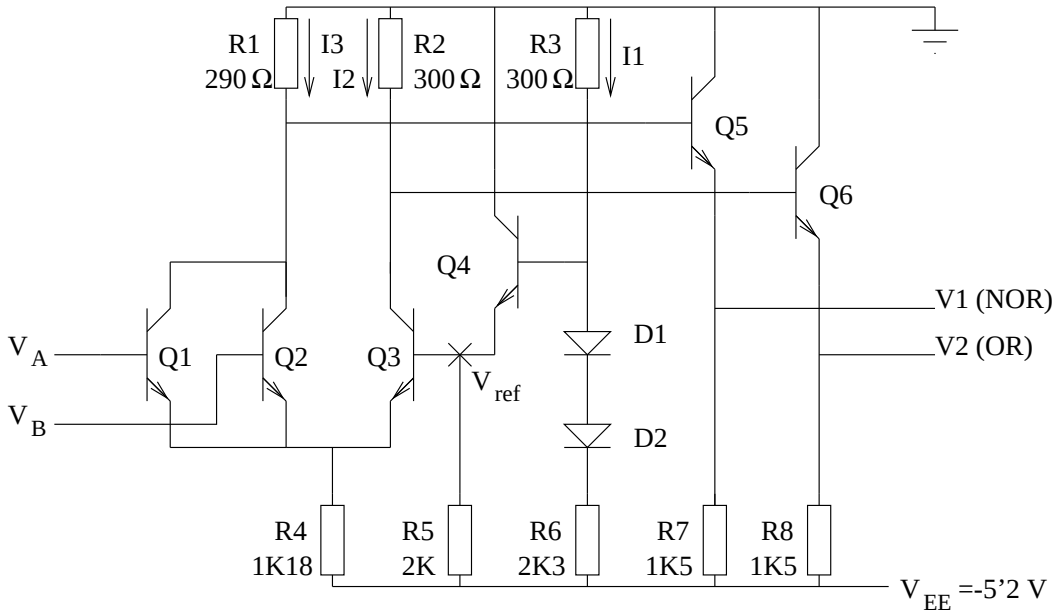


Figura 7.12: *Puerta ECL básica*

Parámetro	Puerta OR/NOR 10101 ECL 10K	Unidades
V_{OLmax}	-1650	mV
V_{OHmin}	-960	mV
t_{PLH}	2	ns
t_{PHL}	2	ns
I_{ILmax}	0'5	μA
I_{IH}	265	μA

Tabla 7.8: *Algunos parámetros significativos de la puerta lógica OR/NOR 10101 ECL 10K*

con una puerta lógica NOR para V_1 y una puerta OR para V_2.

7.7.2 Características funcionales

Como ejemplo simplemente se adjunta en la tabla 7.8 las características de la lógica ECL para dispositivos programables de la familia ECL 10K.

Cabe hacer notar su absoluta incompatibilidad con la lógica TTL, no sólo por las tensiones que definen los valores lógicos, sino también por el sentido de las corrientes que son siempre positivas (de entrada) para las entradas, y negativas (de salida) para las salidas.

7.8 Conclusiones: Análisis comparativo de las familias TTL y ECL

A partir de los transistores BJT se construye la puerta TTL estándar y toda la lógica TTL. Esta primera familia lógica tenía como característica fundamental su mayor rapidez frente a sus competidores CMOS, y como desventaja fundamental su mayor consumo estático. El uso inicial de la lógica TTL como lógica de pegado de módulos estandar ha sido substituido por la lógica programable, y el uso de sus módulo estándar ha quedado restringido a aplicaciones de bajas prestaciones y bajo coste.

La familia TTL estándar, a pesar de ser más rápida que sus contemporáneas CMOS no presentaba una velocidad suficiente para muchas aplicaciones. Basándose en mejoras del transistor BJT y en mejoras de los circuitos electrónicos se desarrollaron familias TTL más avanzadas. La figura 7.13 muestra la evolución y la situación comparativa de las diversas familias TTL y ECL con respecto a la disipación de potencia y retraso.

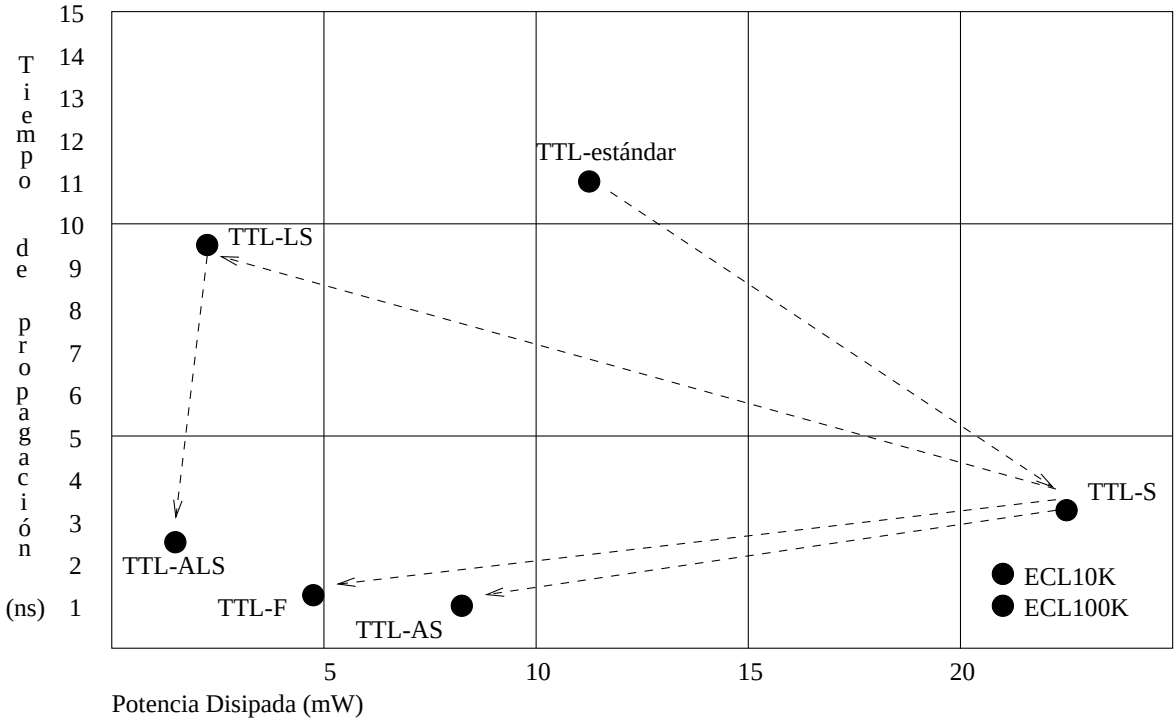


Figura 7.13: *Gráfica comparativa de las prestaciones de la lógica TTL y ECL*

Inicialmente se desarrolló la familia TTL-estándar como primera aproximación con unas ciertas prestaciones. Su capacidad de suministrar corriente y su bajo coste (fácil fabricación) hicieron que rápidamente se utilizase de forma masiva. Como única desventaja respecto a las primeras familias CMOS estaba su alto consumo estático. Aún hoy se sigue fabricando y utilizando, debido su bajo coste, en aplicaciones en las que sus prestaciones sean suficientes. Los principales fabricantes de lógica de todas maneras sugieren la migración a familias más modernas, con más prestaciones, cuyo coste se está reduciendo rápidamente.

A partir de la familia TTL-estándar, y con el ánimo de aumentar la rapidez de la lógica, se desarrolló la familia TTL-Schottky. La gran rapidez de esta familia se

consigue combinando 3 estrategias:

1. Utilizando transistores Schottky. Estos transistores son en realidad un transistor BJT que incorpora un diodo Schottky que hace que el transistor nunca se sature, aumentando de esta manera la velocidad en el cambio de estado.
2. Disminuyendo el valor de las resistencias y por tanto el valor de las constantes RC. El efecto colateral es que al aumentar el valor de las corrientes, aumenta el consumo estático.
3. Rediseñando el circuito para aumentar la eficiencia de conducción.

La gran rapidez de la lógica TTL-S se veía penalizada por su excesivo consumo. Por eso, a partir de TTL-S se desarrolló, rediseñando los circuitos electrónicos, la familia TTL-LS. Esta familia presenta la ventaja de disminuir a la mitad el consumo de TTL-estándar presentando una velocidad ligeramente mejor. A partir de TTL-LS se desarrolló TTL-ALS, de nuevo rediseñando la puerta TTL-LS. Esta familia mejora la velocidad y el consumo de TTL-estándar en un factor 5.

Posteriormente, y siguiendo una línea distinta se desarrollaron la familia TTL-AS, que es la familia más rápida TTL, y la familia TTL-F, que pasa por se el último desarrollo TTL y que se caracteriza por tener el mejor compromiso consumo/velocidad.

Las familias TTL se caracterizan por tener unos niveles de tensión que hacen posible la interconexión entre todas las familias, lo cual permite una gran versatilidad de combinación en un mismo diseño de dispositivos de diversas familias.

La pregunta *¿Qué familia es mejor?* tiene una fácil respuesta: **depende de cada caso**. Cada diseño tendrá unas características diferentes y deberá cumplir unas especificaciones mínimas de velocidad de funcionamiento y máximas de disipación de potencia. Cada familia lógica ofrece unas prestaciones diferentes, y en función de las especificaciones del diseño, se deberá de utilizar una u otra teniendo en cuenta los costes de la lógica. Habitualmente, y sin tener en cuenta estrategias de fabricantes y distribuidores, las familias con mayores prestaciones son las de coste más elevado.

La lógica TTL por su bajo coste, la gran cantidad de dispositivos que existen en las diversas familias, y las prestaciones aceptables de velocidad y suministro de corriente, sigue teniendo una gran utilización. Sobre todo las familias TTL-F y TTL-AS.

Una característica fundamental de la lógica TTL es su utilización para suministrar corriente en buses. Con este fin se utilizan las salidas en colector abierto, salidas en las que es necesario escoger la resistencia de colector a través de la cual se cargará la capacidad de ese nodo. Las salidas en colector abierto también pueden utilizarse para realizar funciones *AND cableadas*, debiéndose escoger el valor de Rc de manera adecuada para que no se sobrecargue el transistor de salida.

Capítulo 8

Lógica CMOS, BiCMOS y Familias de bajo Voltaje

8.1 Introducción

En paralelo a la lógica TTL basada en transistores BJT se ha desarrollado la lógica CMOS basada en transistores FET o de efecto campo (*Field Effect Transistor*). La principal ventaja de la lógica CMOS frente a la lógica TTL era inicialmente su despreciable consumo estático (CMOS sólo consume dinámicamente) y su alta inmunidad frente al ruido. La principal desventaja frente a TTL en las primeras familias CMOS era su mayor lentitud.

Adicionalmente, la lógica CMOS presenta otra interesante ventaja: es extremadamente simple e integrable. Partiendo de las ventajas CMOS se ha minimizado su prácticamente única desventaja (el retraso) mejorando los procesos tecnológicos CMOS. De esta manera, se han conseguido velocidades cercanas e incluso superiores a las mejores familias TTL.

Al mejorar los tiempos de retraso de la lógica CMOS, junto con los avances en el diseño de la arquitectura de CPUs, la frecuencia de los sistemas digitales ha ido aumentando rápidamente. De esta manera, lo que al principio era una ventaja (sólo consume en las transiciones), pasó a ser una desventaja y la potencia disipada por un circuito CMOS funcionando a una frecuencia muy alta puede llegar a ser inaceptable. Esta gran disipación de potencia provoca un calentamiento del circuito que hace que se desplace de su zona térmica de normal funcionamiento, modificándose los parámetros lógicos. Incluso este excesivo calentamiento puede llegar a quemar el encapsulado.

Estos problemas de alta disipación de potencia plantearon la necesidad de conseguir circuitos que combinasen la alta velocidad conseguida ya en circuitos TTL y CMOS con un bajo consumo. Con estas ideas surgió la lógica BiCMOS, que combina la fabricación de transistores BJT con transistores MOS, e intenta combinar las ventajas de la lógica TTL (capacidad de suministrar corriente y consumo dinámico razonable), con las ventajas de la lógica CMOS (alta inmunidad frente al ruido y bajo consumo estático).

Otra aproximación distinta para disminuir la potencia disipada, y aumentar la velocidad de los circuitos son las familias de bajo voltaje . Estas familias se caracterizan por tener una tensión de alimentación y un nivel alto menores a las habituales TTL

y CMOS (típicamente entre 3'3 y 1'2V, manteniendo el nivel bajo en 0 voltios. Estas familias son hoy en día las que consiguen un mejor compromiso velocidad/potencia disipada, teniendo como desventaja su baja inmunidad ante el ruido y sus niveles de tensión, incompatibles con las familias TTL y CMOS iniciales.

Se puede encontrar una buena aproximación a la lógica CMOS y las familias CMOS en [CGT93], [Wak00], [Mar97] y [Toc93]. Sin embargo como bibliografía de la lógica BiCMOS y de bajo voltaje se ha recurrido a manuales e información en el **web**.

8.2 Lógica basada en transistores de efecto campo

8.2.1 Transistores de efecto campo

La lógica CMOS se basa en los transistores de efecto campo o FET, de los cuales hay 3 tipos básicos, tal como se muestra en la figura 8.1. Para cada tipo hay a su vez transistores de canal N y de canal P, siendo su funcionamiento similar.

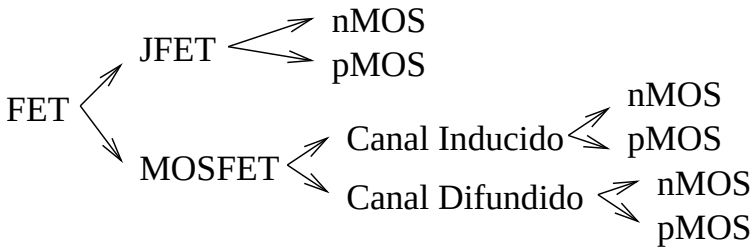


Figura 8.1: *Transistores de efecto campo*

La estructura de estos transistores se muestra, para canal N, en la figura 8.2. Los transistores JFET o transistores de efecto campo de unión (*Junction Field Effect Transistor*) no son más que un canal de un semiconductor de tipo N o P, con unas uniones de un semiconductor del tipo contrario.

Los transistores MOSFET, o transistores de efecto campo metal óxido semiconductor (*Metal Oxide Semiconductor Field Effect Transistor*), tienen como canal difusiones de silicio de tipo contrario al substrato, una puerta de polisilicio, y óxido de silicio. Los transistores MOSFET de canal inducido son los que se utilizan para el diseño de la lógica CMOS.

La forma de funcionamiento es similar para los 3 tipos. En el caso de los 3 transistores la corriente pasará del drenador D al surtidor S cuando se cumpla que la diferencia de tensión entre la puerta y el surtidor V_{GS} sea mayor que un cierto umbral V_T.

Al aplicar la tensión V_{GS} se provoca la aparición del canal por donde pasa la corriente I. En el caso del transistor JFET debe ser $V_T < 0$ y $V_{GS} < 0$ (para no polarizar la unión PN entre el surtidor y la puerta de forma directa). En el caso del transistor MOSFET de canal difundido de nuevo $V_T < 0$, sin restricciones para el signo de V_{GS} salvo que $V_{GS} > V_T$. En el caso del transistor MOSFET de canal inducido se cumplirá que $V_T > 0$, por lo que para que conduzca se deberá de cumplir que $V_{GS} > V_T > 0$.

Es interesante hacer notar la diferencia entre los transistores MOSFET de canal difundido y los de canal inducido. En el primer caso, el canal ya está previamente

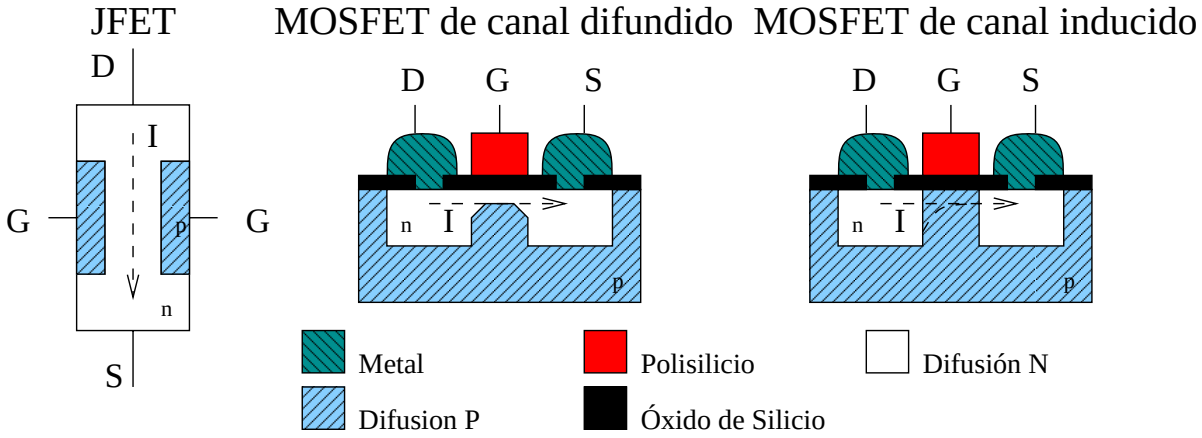


Figura 8.2: *Estructura de los transistores de efecto campo de canal N*

Transistor	V_{GS} **y** V_T	**Ecuaciones de corte, saturación y conducción**
JFET canal N	$V_{GS} < 0$ $V_T < 0$	$V_{GS} < V_T \rightarrow I_{DS} = 0$ $V_{GS} - V_T \leq V_{DS} \rightarrow I_{DS} = K(V_{GS} - V_T)^2$ $V_{GS} - V_T \geq V_{DS} \rightarrow I_{DS} = KV_{DS}[2(V_{GS} - V_T) - V_{DS}]$
JFET canal P	$V_{GS} > 0$ $V_T > 0$	$V_{GS} > V_T \rightarrow I_{DS} = 0$ $V_{GS} - V_T \geq V_{DS} \rightarrow I_{DS} = K(V_{GS} - V_T)^2$ $V_{GS} - V_T \leq V_{DS} \rightarrow I_{DS} = KV_{DS}[2(V_{GS} - V_T) - V_{DS}]$
MOSFET difundido N	$V_T < 0$	Igual que JFET N
MOSFET difundido P	$V_T > 0$	Igual que JFET P
MOSFET inducido N	$V_{GS} > 0$ $V_T > 0$	Igual que JFET N
MOSFET inducido P	$V_{GS} < 0$ $V_T < 0$	Igual que JFET P

Tabla 8.1: *Ecuaciones de funcionamiento de los transistores FET*

difundido, con lo que sin aplicar una diferencia de potencial ($V_{GS} = 0$) el transistor puede conducir corriente. En el segundo caso es necesario que se aplique una diferencia de potencial ($V_{GS} > V_T$ en el caso de tipo N), para que se cree el canal y pase corriente.

En función de la relación entre las tensiones V_{DS}, V_{GS} y V_T el transistor estará en la zona óhmica o en la zona de saturación, con lo que la corriente I_{DS} será función o bien solamente de V_{GS}, o bien de V_{GS} y V_{DS}. La tabla 8.1 muestra las condiciones y ecuaciones para que cada tipo de transistor esté respectivamente en corte, saturación o zona óhmica (conducción).

Cabe hacer notar que se puede considerar a los transistores de efecto campo como fuentes de corriente controladas por tensión (V_{GS}), al contrario de los transistores BJT que eran fuentes de corriente controlado por corriente (I_B).

Aplicando la tensión adecuada a la puerta el transistor conducirá de drenador a surtidor o viceversa si existe la diferencia de potencial adecuada entre el drenador y

el surtidor. A diferencia de los transistores BJT, en este caso no habrá corriente en la puerta. En el caso de los JFET no hay corriente de puerta debido a que V_{GS} será negativa, y eso polarizará inversamente el diodo PN que forman la puerta G y el surtidor S. En el caso de ambos tipos de MOSFET no hay corriente de puerta debido a que el óxido de silicio es un dieléctrico que no permite que pase la corriente. Aplicando la tensión adecuada en el polisilicio se crea el canal, pero no hay corriente de entrada por la puerta G salvo pérdidas del orden de μA.

Los transistores MOSFET de canal inducido serán los utilizables para el diseño de lógica por tener un comportamiento similar al de un interruptor, al igual que los BJT. Si se aplica una tensión en la puerta, entonces el transistor conducirá.

8.2.2 Lógica CMOS

Con estas ideas se ha desarrollado la lógica nMOS, que combina la utilización de transistores MOSFET de canal inducido de tipo N con resistencias, y la lógica pMOS que combina transistores MOSFET de canal inducido de tipo P con resistencias. Esta lógica no presenta ninguna ventaja frente a la lógica TTL, al consumir una corriente similar estáticamente pero ser mucho más lenta. Los transistores MOSFET tienen una gran capacidad en la puerta, tal como se puede adivinar de su estructura en la figura 8.2. Entre la puerta de polisilicio conductor y el canal hay una capa de óxido de silicio, que es un dieléctrico, con lo que se tiene una capacidad parásita apreciable.

Sin embargo la utilización combinada de transistores MOSFET complementarios de canal N y P presenta una serie de ventajas que justifican el desarrollo de la lógica CMOS (*Complementary MOS*). En la figura 8.3 se muestra la estructura de un inversor CMOS.

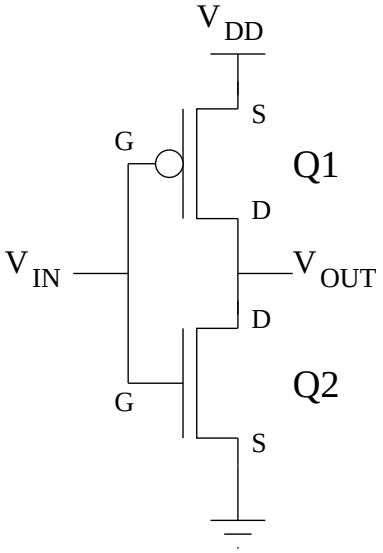


Figura 8.3: *Inversor CMOS*

Si se definen los valores *ideales* del nivel alto como 5 voltios y del nivel bajo como 0 voltios, se puede comprobar como el circuito de la figura 8.3 se comporta como un inversor CMOS. Al aplicarse un nivel alto en la entrada $V_{IN} = 5\ V$, se conseguirá que

$V_{GS}(Q2) = 5\ V > V_T(N)$, y que $V_{GS}(Q1) = 0\ V > V_T(P)$[1]. Un valor típico de V_T para el transistor Q2 será de 0'7 V y para Q1, al ser de tipo P de -0'7 V. De esta manera se cumple la condición de corte para Q1, y la condición de conducción para Q2. El punto de operación exacto de Q2 (zona óhmica o saturación) dependerá de la corriente I_{DS}, que a su vez dependerá de la impedancia conectada a la salida. Si esta corriente es pequeña Q2 estará en la zona óhmica y $V_{DS}(Q2) \approx 0\ V$. Por tanto para un nivel alto de entrada se tiene un nivel bajo se salida.

Si por el contrario se aplica un nivel bajo en la entrada $V_{IN} = 0\ V$,se cumplirá que $V_{GS}(Q2) = 0\ V < V_T(N)$, y que $V_{GS}(Q1) = -5\ V < V_T(P)$. De esta manera se cumple la condición de corte para Q2 y la condición de conducción para Q1, dependiendo de nuevo el punto de operación de Q1 de la corriente I_{SD} extraída. Por tanto, para un nivel bajo de entrada se tiene un nivel alto de salida, tal como cabría esperar de un inversor CMOS.

Una vez comprobado que el circuito de la figura 8.3 se comporta como un inversor cabe destacar las diferencias importantes que aparecen frente a la lógica TTL. En primer lugar no hay resistencias, lo que provoca que la disipación de potencia por la existencia de éstas sea nula. Adicionalmente se puede concluir que el inversor CMOS de la figura no consume corriente estáticamente. Para demostrar esta afirmación se han conectado 2 inversores, modelando la capacidad de carga C_L parásita de la salida.

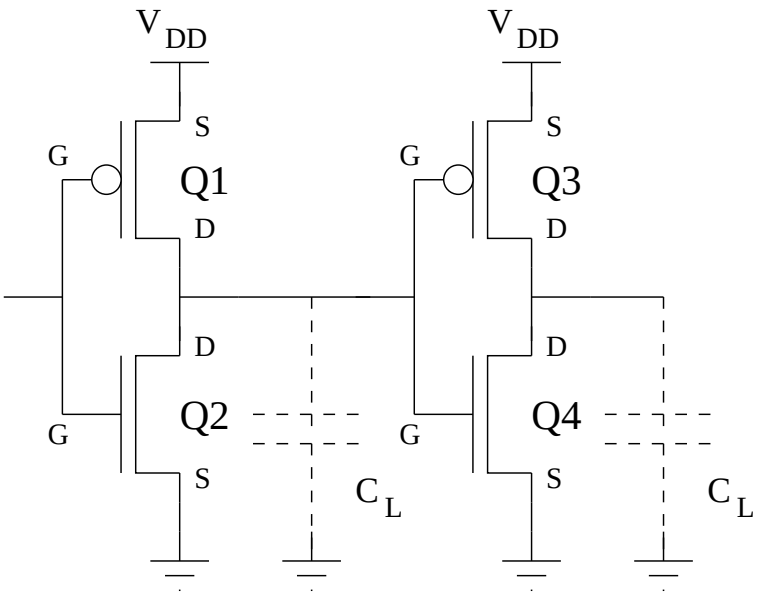


Figura 8.4: *Dos inversores CMOS conectados y con capacidades parásitas*

Estáticamente, al no entrar corriente por la puerta G en los transistores MOS no circulará corriente por ninguno de los transistores. Sólo se producirá paso de corriente en las transiciones, que es cuando se cargan (a través de los transistores P) y descargan (a través de los transistores N) las capacidades C_L.

Se tiene por tanto que estáticamente la lógica CMOS, al no tener resistencias y no absorber corriente por las puertas G, no consume corriente. El consumo se producirá cada vez que haya que cargar/descargar las capacidades parásitas, con lo que **el**

[1]La simetría en los transistores MOSFET hace que las terminales D y S vengan definidas en función de la terminal con mayor tensión, que será D para tipo N y S para tipo P

consumo en la lógica CMOS será directamente proporcional a la frecuencia de funcionamiento. El consumo estático es de pérdidas y puede considerarse despreciable.

Otra interesante propiedad se obtiene de la sencillez, simetría y de las características tecnológicas de los transistores MOS. Hay una amplio margen de alimentaciones (que pueden ser en muchas familias de 5 a 15 voltios), y el punto de transición $V_{ILumbral} = V_{IHumbral} = V_{DD}/2$. De esta manera se consigue una **alta inmunidad frente al ruido**, al ser los márgenes del uno y del cero iguales y con el máximo valor posible.

Este cúmulo de ventajas tiene como punto en contra la extremada lentitud de las primeras familias CMOS. A continuación se van a analizar las principales familias CMOS, mostrando sus ventajas y desventajas.

8.2.3 Familia lógica CMOS 4000

En la figura 8.5 se muestra la puerta NAND estándar de la familia CMOS 4000UB. Se deja al estudiante la sencilla comprobación de que dicho circuito se comporta como una puerta NAND. La familia CMOS 4000 fue la primera que se desarrolló con la filosofía CMOS, y tiene dos subfamilias: La 4000B, y la 4000UB. La familia 4000UB es una familia *Unbuffered* o sin *buffer* de salida[2]. La subfamilia 4000B consta sin embargo de un *buffer* de salida, lo que hace que suministre más corriente y sea un poco más lenta que la 4000UB.

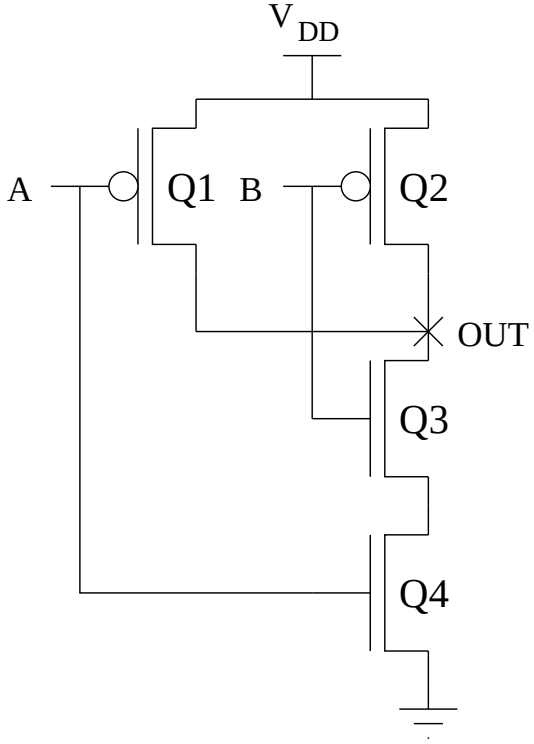


Figura 8.5: *Puerta básica NAND CMOS 4000UB*

[2]Un *buffer* es un dispositivo que suministra corriente sin transformar el valor lógico de entrada, habitualmente 2 inversores con los transistores diseñados especialmente para suministrar altas corrientes

El resto de familias CMOS más avanzadas utilizan los mismos circuito básicos. Las únicas mejoras estriban en los procesos tecnológicos que reducen las capacidades parásitas. Una puerta NAND de una familia CMOS avanzada es básicamente el mismo circuito de la figura 8.5, salvo diodos de protección ante descargas electrostáticas (a las que la lógica CMOS es en general muy sensible).

Característica funcionales

De un inversor CMOS de la familia 4000UB se pueden extraer las valores umbrales de entrada para el nivel alto y el nivel bajo. La función de transferencia que se obtiene es simétrica y casi ideal, de manera que la zona de transición no existe y se cumple que $V_{ILumbral} = V_{IHumbral} = V_{DD}/2$.

A partir de este resultado, en función de las posibles tensiones de alimentación y para tener unos márgenes de inmunidad al ruido se definen los valores V_{ILmax} y V_{IHmin}:

- **4000B**
  - Para una tensión V_{DD} entre 5 y 10 V: $V_{ILmax} = 30\%V_{DD}$ y $V_{IHmin} = 70\%V_{DD}$
  - Para una tensión V_{DD} de 15 V: $V_{ILmax} = 27\%V_{DD}$ y $V_{IHmin} = 73\%V_{DD}$
- **4000UB**
  - Para una tensión V_{DD} entre 5 y 10 V: $V_{ILmax} = 20\%V_{DD}$ y $V_{IHmin} = 80\%V_{DD}$
  - Para una tensión V_{DD} de 15 V: $V_{ILmax} = 17\%V_{DD}$ y $V_{IHmin} = 83\%V_{DD}$

Estos principios dan un valor con $V_{DD} = 5\ V$ de $V_{ILmax} = 1'5\ V$ y $V_{IHmin} = 3'5\ V$, con lo que los márgenes de ruido quedan definidos como $V_{ML} = V_{MH} = 1\ V$, (márgenes que se ven sobre-ampliados al ser $V_{ILumbral} = V_{IHumbral} = 2'5\ V$).

A partir de los valores anteriores y repitiendo los márgenes de ruido para el cero y para el uno se obtiene:

$$V_{OLmax} = V_{ILmax} - V_{ML} = 1'5 - 1 = 0'5V \tag{8.1}$$
$$V_{OHmin} = V_{IHmin} + V_{ML} = 3'5 + 1 = 4'5V \tag{8.2}$$

Se pueden medir las corrientes máximas de salida para que no se violen estos parámetros y por tanto se mantenga el margen de ruido. Las corrientes, en el caso de la puerta 4011B de MOTOROLA (puerta NAND de la familia CMOS 4000B) son $I_{OHmax} \approx -0'5\ mA$ y $I_{OLmax} \approx 0'5\ mA$. Estas corrientes son sensiblemente inferiores a la mayoría de familias TTL, con lo que se observa que la familia CMOS 4000 no suministra excesiva corriente, a pesar de estas cifras son de la familia 4000B que incluye un *buffer* de salida.

Las corrientes de entrada son de pérdidas y del orden de décimas de μA, con lo que no tiene sentido calcular el fan-out estático en cualquier familia CMOS. Simplemente hay que tener en cuenta que cuantas más entradas se conecten a una salida la capacidad parásita de ese nodo aumentará, con lo que la propagación de la señal será más lenta. Por tanto, a pesar de que el fan-out es casi ilimitado debido al casi nulo consumo estático de la lógica CMOS, es recomendable no hacer nodos con muchas entradas para evitar tener una gran capacidad parásita que implicaría un alto retraso.

Los parámetros dinámicos de la familia 4000B dados por el fabricante se muestran en la tabla 8.2, comprobándose la mayor lentitud de esta lógica respecto a TTL.

Parámetro	Valor típico	Valor máximo	Unidades
t_{PLH}	40	125	ns
t_{PHL}	40	125	ns

Tabla 8.2: *Parámetros dinámicos de la puerta 4011B (puerta NAND de la familia CMOS 4000B de Motorola)*

El consumo de la lógica CMOS, como se ha comentado con anterioridad, es tan sólo del orden de nW. Efectivamente, se tiene que $I_{CCH} = I_{CCL} = 0'5\ nA$, con lo que la potencia media disipada será $P_D = \frac{0'5+0'5}{2} V_{CC} = 2'5\ nW$.

Esta familia reporta una gran inmunidad frente al ruido (debido a una función de transferencia casi ideal), un consumo de potencia estático despreciable, una gran capacidad de integración, una amplia gama de valores de alimentación y un fan-out casi ilimitado. Como desventajas se tiene su gran lentitud y su poca capacidad para suministrar corriente. A partir de esta familia se han desarrollado familias lógicas CMOS más avanzadas, que mantienen las ventajas de la familia 4000 y reducen sus desventajas.

8.2.4 Familias lógicas CMOS avanzadas

Familias HE4000

La familia CMOS 4000 presenta el problema fundamental de su gran retraso de propagación debido a sus grandes capacidades parásitas. A partir de esta familia se desarrolló la familia CMOS HE4000 que incorpora la tecnología de puerta de silicio. Con esta tecnología se reducen considerablemente las capacidades parásitas de CMOS4000, manteniendo todas sus ventajas.

Dentro de esta familia hay dos subfamilias: HE4000B que incorpora un *buffer* de salida, y HE4000UB sin este *buffer*. En ambas familias la circuitería que implementa la función lógica es idéntica a la de la familia 4000. La evolución se produce en el proceso tecnológico de fabricación microelectrónica de los circuitos, no en cambios del circuito.

La tensión de alimentación puede estar en el rango de $V_{DD} \in [3, 15]\ V$. Para $V_{DD} = 5\ V$, se tiene que $V_{ILumbral} = V_{IHumbral} = 2'5\ V$, con lo que de nuevo se tienen márgenes de cero y uno simétricos. Los parámetros suministrados por el fabricante para la puerta HEF4011B son los que se muestran en la tabla 8.3.

Cabe hacer notar como se reduce más de la mitad el tiempo de propagación máximo de la familia CMOS 4000, pero sigue siendo más desfavorable que cualquier familia TTL. El resto de características CMOS se siguen manteniendo.

Familias HCMOS

Otro desarrollo a partir de la familia CMOS 4000 es la familia CMOS de alta velocidad HCMOS (*High-speed CMOS*). Esta familia de nuevo parte de una mejora tecnológica que reduce las capacidades parásitas en la fabricación de los circuitos integrados, pero manteniendo los mismos circuitos lógicos que de la familia 4000.

Parámetro	Valor	Unidades
V_{ILmax}	1'5	V
V_{IHmin}	3'5	V
V_{OLmax}	0'5	V
V_{OHmin}	4'5	V
I_{OLmax}	0'4	mA
I_{OHmax}	-0'4	mA
t_{PLHmax}	55	ns
t_{PHLmax}	55	ns

Tabla 8.3: *Parámetros lógicos más relevantes de la puerta HEF4011B (puerta NAND de dos entradas)*

Parámetro	74HC	74HCT	Unidades
V_{ILmax}	1'35	0'8	V
V_{IHmin}	3'15	2	V
V_{OLmax}	0'26	0'26	V
V_{OHmin}	3'98	3'98	V
I_{OLmax}	4	4	mA
I_{OHmax}	-4	-4	mA
t_{PLHmax}	9	12	ns
t_{PHLmax}	9	12	ns

Tabla 8.4: *Parámetros lógicos más relevantes de la puerta 74HC00 (puerta NAND de dos entradas)*

Dentro de la familia HCMOS hay tres subfamilias:

- **74HC**: Con márgenes de tensión idénticos a los de la familia CMOS 4000.
- **74HCT**: Con márgenes de tensión compatibles TTL, estando el umbral típico de conmutación en el 28 % de V_{DD}. La tensión de alimentación se reduce en este caso a $V_{DD} = 5\ V \pm 10\%$.
- **74HCU**: Tiene las mismas características que la subfamilia 74HC, salvo que sus salidas no tienen *buffer*.

En la tabla 8.4 se resumen estas características para las familias 74HC y 74HCT. Con estas familias se consiguen velocidades del orden de TTL-estándar y TTL-LS, con todas las ventajas adicionales de la lógica CMOS.

Familias ACL

Con las familias HCMOS se consiguen velocidades TTL-estándar, pero la lógica CMOS sigue estando lejos de las velocidades de las familias TTL más rápidas y ECL.

De nuevo, a partir de mejoras tecnológicas en el proceso de fabricación, se desarrolló

Parámetro	74AC	74ACT	Unidades
V_{ILmax}	1'65	0'8	V
V_{IHmin}	3'25	2	V
V_{OLmax}	0'36	0'36	V
V_{OHmin}	3'94	3'94	V
I_{OLmax}	24	24	mA
I_{OHmax}	-24	-24	mA
t_{PLHmax}	4	6'5	ns
t_{PHLmax}	4	6'5	ns

Tabla 8.5: *Parámetros lógicos más relevantes de la puerta 74AC00 y 74ACT00 de PHILIPS*

una nueva familia CMOS que mejoraba la velocidad de propagación, manteniendo la misma circuitería lógica. Esta familia es la familia lógica CMOS avanzada, o familia ACL (*Advanced CMOS Logic*).

La familia ACL consta de dos subfamilias:

- **74AC**: Con niveles de tensión compatibles CMOS. El umbral típico de conmutación está en el 50 % de V_DD. La tensión de alimentación puede variar desde 3 a 5'5 voltios.
- **74ACT**: Con márgenes de tensión compatibles TTL, estando el umbral típico de conmutación en el 28 % de V_{DD}, al igual que en la familia 74HCT.

En la tabla 8.5 se resumen estas características para las familias 74AC y 74ACT. Estas familias CMOS son las más rápidas, y si sólo se tienen en cuenta las familias CMOS y TTL, es la que tiene un mejor compromiso velocidad/consumo. Esta familia de manera adicional mantiene unos valores buenos de corriente de salida, lo que la hace sin duda ser la familia de mejores prestaciones globales entre TTL y CMOS. El problema del suministro de corriente en lógica CMOS viene definido por la relación W/L en los transistores de salida. Si W/L $\uparrow\uparrow$, entonces el tamaño de la lógica se hace muy grande, disminuyendo su escala de integración.

8.3 Lógica BiCMOS

Con las familias lógicas TTL más avanzadas se tienen las ventajas:

1. Suministro de corriente alto en las salidas.
2. Unos tiempos de propagación del orden de 3 *ns* (TTL-F).

Y como desventajas:

1. Un consumo estático y dinámico elevado.
2. Baja inmunidad frente al ruido.

Las familias más avanzadas CMOS tienen como ventajas:

1. Consumo estático prácticamente nulo.

2. Alta inmunidad frente al ruido.
3. Amplio margen de tensiones de alimentación.
4. Tiempos del orden de 3 ns (ACL).

Y como desventajas:

1. Un consumo dinámico prohibitivo.
2. Bajo suministro de corriente para relaciones W/L que permitan alta escala de integración.

Para intentar combinar las ventajas de ambas lógicas, y eliminar las desventajas, se ha desarrollado la tecnología BiCMOS. Esta tecnología, como su propio nombre indica, combina en un solo proceso tecnológico la fabricación de transistores bipolares BJT y transistores CMOS. El propósito de esta combinación es conseguir simultáneamente que los dispositivos suministren corriente con una alta escala de integración, y tengan un consumo estático y dinámico razonable.

8.3.1 Puerta mínima BiCMOS

En la figura 8.6 se puede observar la constitución de una puerta mínima BiCMOS. Se puede observar como se combinan los transistores CMOS para implementar la funcionalidad de la puerta, con la etapa de salida con transistores BJT que suministrarán corriente. La etapa de entrada también es CMOS, con lo que se tiene la alta impedancia de entrada de la lógica CMOS y un fan-out muy alto.

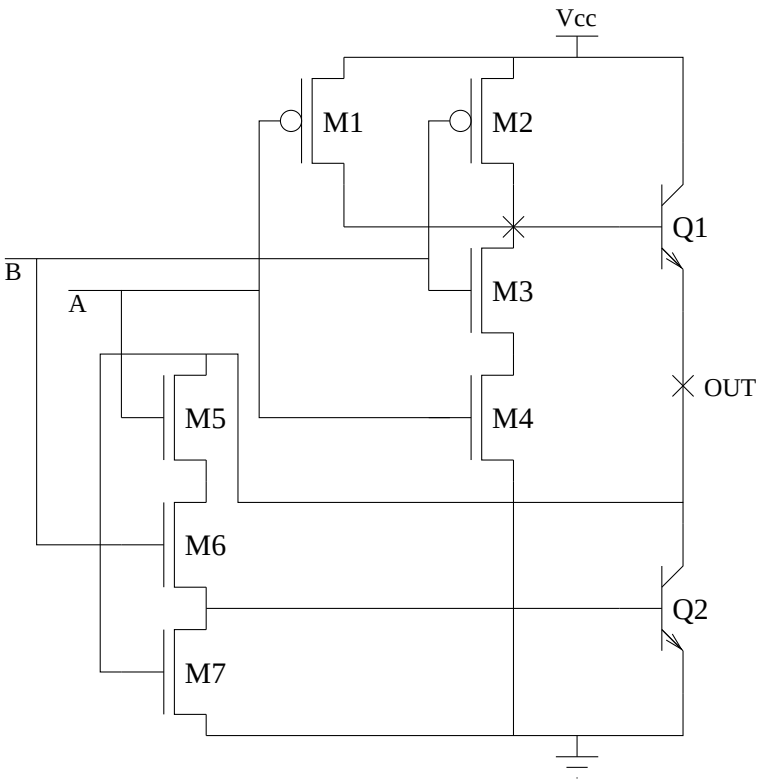


Figura 8.6: *Puerta NAND mínima BiCMOS*

Se supone que en la figura 8.6 siempre hay conexión en las conexiones de tipo T y nunca en los cruces, salvo que esté explícitamente indicado con un aspa X. El funcionamiento de esta puerta sería como sigue. Si se tiene un nivel alto en ambas entradas, los transistores de canal P M1 y M2 estarán cortados y los transistores de canal N M3, M4, M5 y M6 conducirán. Esta hará que el transistor BJT Q1 esté cortado al no tener corriente de base, lo que a su vez cortará a M7. Al estar cortado M7 y conducir M5 y M6, Q2 conducirá, con lo que se tendrá en la salida un nivel bajo.

Cualquier otra combinación de entrada conectará la base de Q1 con alimentación, con lo que Q1 conducirá. Esto hará que M7 conduzca, lo que cortará a Q2. Por tanto, si es tiene un nivel bajo en cualquiera de las dos entradas, en la salida se tendrá un nivel alto, tal como corresponde a una puerta NAND.

Se puede dividir una puerta lógica en 3 etapas: la de entrada que define la impedancia de entrada, la intermedia que implementa la función lógica, y la de salida que suministra corriente. Siempre en la lógica BiCMOS la etapa intermedia se implementará con transistores MOS, para conseguir una alta escala de integración y bajo consumo. La etapa de salida será siempre una etapa con transistores BJT que suministran corriente con un tamaño pequeño. Sin embargo la etapa de entrada, que inicialmente era siempre CMOS por su alta impedancia de entrada, se ha visto substituida en algunas familias por transistores BJT para aumentar la velocidad.

Inicialmente la lógica BiCMOS se ha introducido en *drivers* y circuitos de bus debido a sus prestaciones optimizadas para suministrar corriente, pero actualmente la tecnología BiCMOS se está utilizando en una gran cantidad de dispositivos lógicos estándar y circuitos *custom*[3]. Con la lógica BiCMOS se consigue una optimización del área de silicio necesaria para suministrar una cierta cantidad de corriente, optimizando el consumo de potencia estática y dinámica y la velocidad de propagación.

8.3.2 Familias BiCMOS

Las familias BiCMOS ofrecen sobre todo dispositivos de interfase con buses. Como ejemplo se tienen las familias BCT (*BiCMOS Technology*) y ABT (*Advanced BiCMOS Technology*), de *Texas Instrument*. Estas familias tienen una capacidad de suministrar corriente de hasta 64 mA y retrasos de propagación por debajo de los 5 ns, manteniendo un consumo menor que sus equivalentes TTL-F y TTL-ALS. De manera adicional los productos ABT están indicados para diseño de tarjetas con inserción en vivo.

8.4 Familias de bajo voltaje

La disipación de potencia para cualquier dispositivo que es atravesado por una corriente I, lo que provoca una caída de tensión V, es $P = V \cdot I$. En el caso de una resistencia, si se utiliza la ley de Ohm se obtiene que la potencia disipada es proporcional al cuadrado de la tensión $P = \frac{V^2}{R}$.

En el caso de la lógica CMOS (y para frecuencias altas el caso TTL) el consumo de potencia es función de la frecuencia de funcionamiento ν. De hecho, el fabricante suele incluir, en las hojas de características de la lógica CMOS, una ecuación de cálculo

[3]Hechos a medida.

del consumo en función de la frecuencia. La disipación de potencia dinámica es directamente proporcional a la frecuencia de conmutación del circuito, pero es cuadrática con la tensión. En el caso de la lógica ACL la potencia dinámica P_D se calcula de la siguiente manera:

$$P_D = C_{PD} V_{CC}^2 \nu_1 + C_L V_{CC}^2 \nu_0 \tag{8.3}$$

Donde C_{PD} es la capacidad de la puerta, valor dado por el fabricante, C_L es la capacidad de la carga, ν_0 es una frecuencia base de 0'1 MHz, V_{CC} es la tensión de alimentación y ν_1 es la frecuencia de funcionamiento.

Una posible aproximación para reducir fuertemente el consumo, manteniendo la frecuencia de funcionamiento, sería reduciendo la tensión de alimentación V_{CC}. Con esta filosofía se han desarrollado familias CMOS (y algunas BiCMOS) que reducen el consumo por puerta e incluso aumentan la velocidad de funcionamiento. Con esta filosofía se han definido las familias LV o de bajo voltaje *Low Voltage*, con tensiones de alimentación de 3'3 $V \pm 0'3V$, 2'5 $V \pm 0'2V$, 1'8 $V \pm 0'15V$, 1'2 $V \pm 0'1V$ y un nivel alto ideal de la misma tensión. La mayoría de estos dispositivos lógicos tienen unos margenes de alimentación de baja tensión, en lugar de una tensión fija. De esta manera los parámetros de tensión a nivel alto (V_{IHmin} y V_{OHmin}) se definen en función de la tensión de alimentación.

Las desventajas de esta aproximación son obvias. Se reducen los márgenes de tolerancia al ruido, al ser los márgenes del uno y del cero absolutos más reducidos. De manera adicional, el hecho de no seguir unos niveles compatibles TTL y CMOS hace que al optar por una familia LV se tenga que utilizar toda la lógica de familias LV o incluir *transceivers* (traductores de nivel), que ralentizarán el sistema.

La utilización de avanzadas técnicas de blindaje frente al ruido y el desarrollo completo de las familias LV, permite la progresiva introducción de estas familias, sólo frenada por su alto coste.

Como ejemplo de estas familias está la familia AVC (*Advanced Very-Low-Voltage CMOS*) de *Texas Instrument*, cuya tensión de alimentación es de 2'5 V y que consigue tiempos de propagación menores de 2 ns. Con estos retrasos se puede trabajar a frecuencias que serían prohibitivas si no existiera la reducción de disipación de potencia debida a la disminución de V_{CC}.

8.5 Conclusiones: Criterios globales de selección de lógica

La lógica CMOS presenta ventajas claras: bajo consumo estático, alta inmunidad al ruido, amplio margen de tensiones de alimentación y, con las familias avanzadas, una velocidad comparable a las mejores familias TTL. Sin embargo presenta el problema de que si es necesario suministrar corriente los dispositivos CMOS pasan a ocupar demasiado silicio, con lo que en este caso los dispositivos TTL serian más adecuados.

Sin embargo TTL presenta consumos altos para una alta velocidad y una baja inmunidad al ruido. Con la idea de combinar los beneficios de ambas familias lógicas, se desarrolló la lógica BiCMOS. Las familias de esta tecnología básicamente son dispositivos de interfase que suministran corriente con un consumo bajo y retrasos pequeños.

Debido a las velocidades de funcionamiento que se están alcanzando en los dispositivos CMOS, la potencia disipada es excesiva, con lo que varios fabricantes de lógica han optado por disminuirla. Para ello se han definido familias lógicas con tensión de alimentación más reducida. De esta manera se consigue reducir el consumo y el tiempo de propagación, pero disminuyendo la inmunidad ante el ruido.

De nuevo la pregunta *¿Qué familia es mejor?* tiene una fácil respuesta: **depende de cada caso**. El coste de las familias lógicas dependerá de sus prestaciones y de las políticas de sus fabricantes y distribuidores. Para seleccionar un cierto componente, además de las prestaciones requeridas y del precio, hay que tener en cuenta si es para un prototipo o para producción, si hay más de un proveedor o fabricante del dispositivo y la compatibilidad con otros dispositivos.

Siempre hay que ser cuidadoso y optimizar los costes del producto siempre que se cumplan las especificaciones del diseño (sin sobre-dimensionarlas).

Las familias con mejores prestaciones para módulos y puertas digitales son las ACL, seguidas por las TTL-AS y TTL-F. Las familias LV tienen unas prestaciones superiores, pero no son por ahora familias con toda la diversidad de componentes de TTL o CMOS.

Para *drivers* e interconexiones con buses la lógica BiCMOS y las TTL avanzadas, ofrecen las mejores prestaciones, siendo de nuevo el coste el elemento clave en la elección siempre que se cumplan las especificaciones.

De todas maneras la utilización de lógica discreta para implementar la lógica de pegado o *glue logic* en los sistemas digitales ha sido reducida por la utilización de lógica programable. Incluso hoy en día es posible implementar sistemas digitales relativamente complejos en circuitos programables, con el consiguiente ahorro en cantidad de componentes y el circuito impreso de interconexión. La lógica programable además aumenta la fiabilidad del circuito al reducir el numero de componentes y conexiones entre ellos. De esta manera, la utilización de lógica discreta en sistemas digitales se está viendo reducida a sistemas con altas prestaciones (que todavía no ha conseguido la lógica programable), o a sistemas donde sólo son necesarios unos pocos componentes (cuyo coste es menor que su equivalente implementado con lógica programable). En el capítulo 9 se describen y analizan estos dispositivos, así como se indican los criterios para su selección.

Parte III

Lógica Programable y Memorias

Capítulo 9

Lógica Programable

9.1 Introducción a los ASICs

Los ASICs o circuitos de aplicación específica (*Application Specific Integrated Circuit*, son sistemas digitales completos en un solo circuito integrado. A diferencia de los sistemas digitales habituales, que se construyen a partir de elementos lógicos discretos de propósito general, como son las puertas lógicas, los módulos MSI o los μprocesadores. Se recomienda como bibliografía de consulta de este capítulo [NNCI96], [GA95] y [Tav94].

Realizar el diseño de un sistema en un ASIC tendrá ventajas respecto a realizar el mismo diseño con componentes discretos:

1. **Facilidad de diseño**: Existen lenguajes de alto nivel para describir sistemas digitales. Posteriormente, con potentes herramientas de CAD, se puede simular y sintetizar el circuito. De esta manera se simplifica el proceso de diseño y se disminuye el tiempo de acceso al mercado.
2. **Prestaciones**: El hecho de que un sistema se haga en un solo circuito hace que sólo incluya la lógica necesaria, con lo que se optimiza el consumo. De manera adicional, al mantenerse la circuitería dentro de un solo circuito, se evitan los retrasos de propagación que aparecen en las pistas del circuito impreso, con lo que se aumenta la velocidad.
3. **Fiabilidad**: Se reduce la cantidad de componentes y pistas en el circuito impreso, además de evitar el ensamblaje del circuito, lo que reduce la probabilidad de fallos.
4. **Economía**: Hay veces en las que un sistema complejo utiliza muchos componentes distintos. Si el ASIC tiene un precio menor que la suma de los componentes lógicos discretos ya se tiene la ventaja económica. Si no es así hay que tener también en cuenta el coste del circuito impreso, el coste del ensamblaje y el coste de los montajes defectuosos.
5. **Seguridad**: Si se realiza un sistema digital en un solo circuito integrado es muy complicado copiarlo. Además se pueden incorporar a los ASICs métodos adicionales de protección contra la ingeniería inversa.

En la figura 9.1 se muestra una clasificación de los diversos tipos de circuitos de aplicación específica o ASICs.

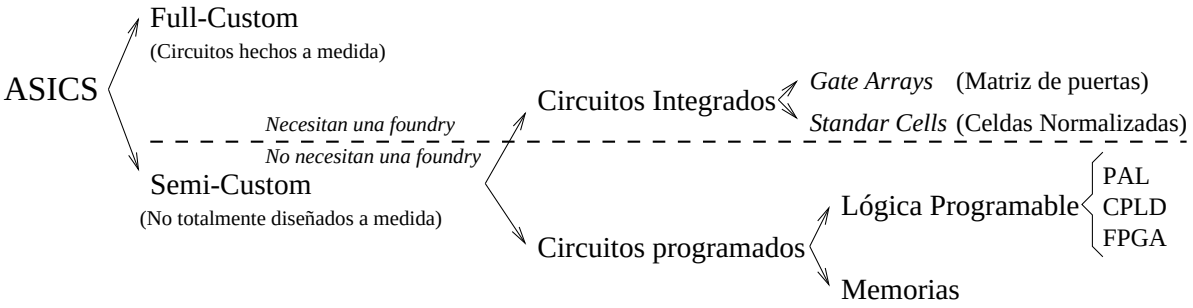


Figura 9.1: *Tipos de ASICs*

Los circuitos *full-custom* o hechos a medida, son los circuitos en los que se hace un diseño a nivel de máscara de silicio. Son los circuitos que tienen un mayor aprovechamiento del silicio ya que se diseña a nivel de transistor, y se incluye solamente lo estrictamente necesario. Tienen por tanto el más alto rendimiento y aprovechamiento de recursos. Estos circuitos tienen como desventaja su alto coste de diseño y de producción ya que es necesaria una *foundry* o fundición para fabricarlos. Esto hace que con este tipo de ASICs se implementen sólo sistemas de muy altas prestaciones (ya que no hay otra solución), o de fabricación masiva (ya que los altísimos costes de producción se amortizan al ser la producción muy alta y reducir el silicio).

Los circuitos *semi-custom* son circuitos parcialmente hechos a medida. Dentro de esta categoría estarían las *Gate Arrays* o matrices de puertas. En este caso los transistores ya están difundidos en la oblea de silicio. Sólo hay que fabricar las máscaras de metal que interconectan estos transistores y por tanto definen la lógica. Esta solución también necesita de una fundición pero es más económica que la *full-custom* al tener que fabricar sólo unas pocas máscaras (dependiendo del número de metales de la tecnología).

Los ASICs basados en celdas normalizadas o *standard cells* consisten en circuitos en los que el diseñador no edita a nivel de silicio el circuito, pero hay que fabricar todas las máscaras del circuito. El diseño se realiza a nivel de esquema lógico, en el que cada componente lógico tiene un equivalente directo con una celda de silicio sita en una biblioteca de celdas. Después, una vez realizado el diseño lógico, sólo queda realizar el emplazado de las celdas y su interconexión en la oblea de silicio. Este proceso, si se desea, puede realizar automáticamente con potentes herramientas de CAD, con lo que el diseñador sólo tiene que preocuparse del diseño lógico. Con estos ASICs se reduce el coste de ingeniería pero se mantiene el alto coste de producción de las máscaras, con lo que los costes siguen siendo altos.

Los dispositivos lógicos programables, o PLDs, son programables en el sentido de que en el circuito se tienen difundidos de forma fija los dispositivos lógicos, puertas lógicas y flip-flops. Lo que se *programa* son las interconexiones entre estos dispositivos lógicos. De esta manera se puede tener la velocidad del hardware sin haber hecho un costoso circuito a medida. Basta con programar las conexiones internas de manera adecuada para que el dispositivo realice la función deseada.

La lógica programable tiene como ventajas:

1. No es necesaria una fundición.
2. Es viable hacer producciones pequeñas y prototipos, al no tener grandes costes iniciales.

3. Hay una gran cantidad de herramientas de CAD que incorporan captura de esquemas, descripción con lenguajes de alto nivel y síntesis lógica en dispositivos programables.
4. Se ha llegado a dispositivos programables de más de 200 pines de entrada salida, 1.500 biestables y 250.000 puertas equivalentes, lo que los posibilita para implementar sistemas complejos. Análogamente se han desarrollado dispositivos que pueden funcionar a frecuencias muy altas, lo cual los hace una alternativa muy interesante a la lógica discreta.

9.2 Tecnologías de programación

En la figura 9.2 se muestra lo que en definitiva hay que programar para programar un dispositivo lógico programable. Lo que hay que hacer es establecer las conexiones que aparecen marcadas como X en los puntos de cruce. Esto es lo que define la función del circuito. En el ejemplo se tiene una puerta OR fija que tiene como entradas 3 funciones AND cableadas. Los productos de las AND cableadas se programan estableciendo las conexiones, tal como se muestra en la figura. Se van a revisar las diversas tecnologías que permiten realizar las conexiones, mostrando sus ventajas y desventajas [Ska96].

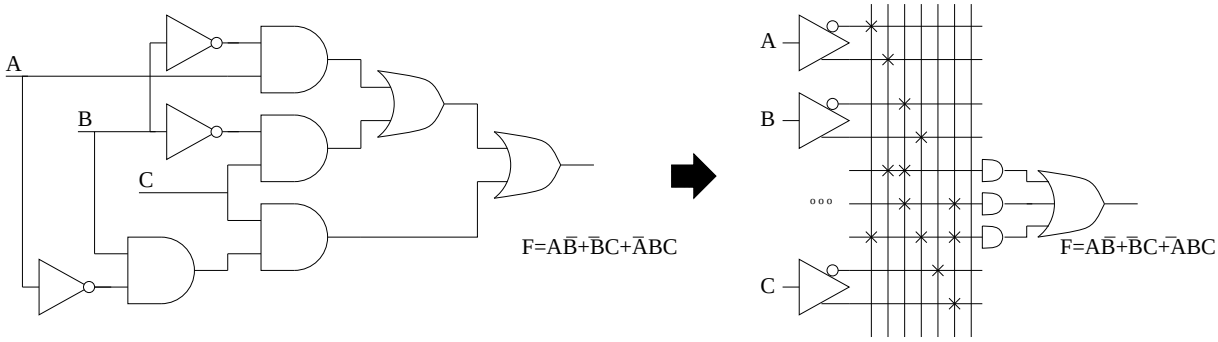


Figura 9.2: *Ejemplo de implementación de una función lógica en dispositivo programable*

9.2.1 Tecnología EPROM

La tecnología EPROM (*Erasable Programmable Read Only Memory*) se basa en la utilización del transistor FAMOS (*Floating-gate Avalanche-injection MOS*), del que se puede observar un corte transversal en la figura 9.3. La estructura es idéntica a la de los transistores MOS de canal inducido salvo que aparece una puerta flotante. Esta puerta de polisilicio está aislada eléctricamente de la puerta G normal del transistor, siendo permeable a las tensiones aplicadas V_{GS}.

Si se aplican corrientes grandes (tensiones altas) entre D y S y V_{GS} $\uparrow\uparrow$, habrán electrones que superen la barrera de potencial que es el dieléctrico (óxido de silicio) y se queden atrapados en la puerta flotante. De esta manera la tensión V_T aumenta y entonces, aunque en la pista B haya un nivel alto, no se conecta la pista A a tierra. De esta manera inutilizando, y manteniendo los transistores útiles se puede programar la lógica. Como ejemplo se tiene la figura 9.4, en la que se han tachado los transistores inutilizados (con la puerta flotante cargada). Los transistores que permanecen sin tachar

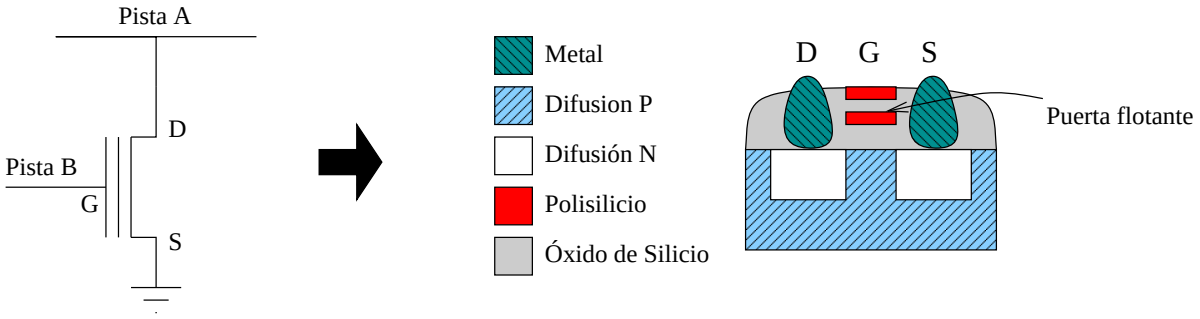


Figura 9.3: *Símbolo y corte transversal del transistor FAMOS, base de la tecnología EPROM*

son los que realizan la conexión y son los que se muestran sin una aspa X en el esquema lógico cableado.

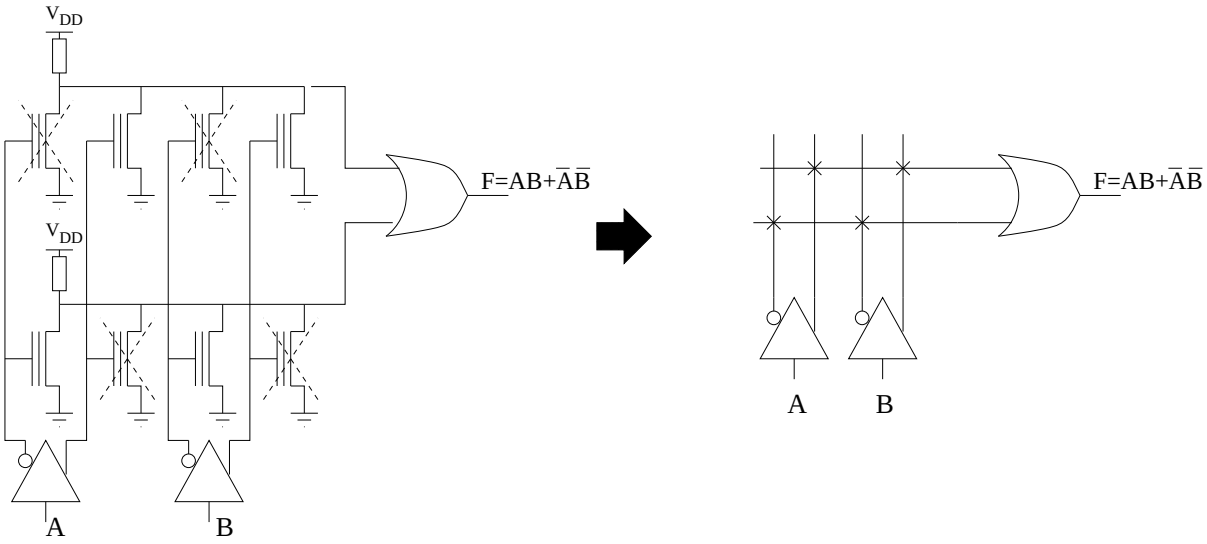


Figura 9.4: *Ejemplo de implementación de una función lógica con tecnología EPROM*

Un resultado interesante de la tecnología de programación EPROM es que la programación no desaparece aunque se elimine la alimentación del circuito. Efectivamente, las cargas permanecerán capturadas en la puerta flotante por que está aislada eléctricamente de las terminales del transistor. Esto es debido a la barrera de potencial que supone el óxido de silicio para las cargas atrapadas en la base.

Para *borrar* la programación EPROM hay que exponerlo a rayos UV, que son de una frecuencia suficientemente alta para suministrar la energía necesaria a las cargas para que superen la barrera de potencial.

Como ventaja de esta tecnología se tiene que la programación permanece cuando se elimina la alimentación del circuito, es decir, no es volátil. Como desventaja se tiene que hace falta un dispositivo especial para programar las celdas EPROM. Esto es debido a que las tensiones de programación son más altas que las TTL o CMOS normales de funcionamiento de los circuitos, habitualmente $\pm 12\ V$.

Además de que el circuito se debe programar fuera del sistema donde vayan a estar funcionando, para eliminar la programación es necesario extraerlo y borrarlo con una

lámpara especial de rayos UV varios minutos, lo cual puede llegar a ser incómodo y problemático en diseño de prototipos.

9.2.2 Tecnología EEPROM

Se ha visto como la tecnología de programación EPROM presenta el problema del complicado borrado del dispositivo programado. Por este motivo se desarrolló la tecnología EEPROM (*Electrically Erasable Programmable Read Only Memory*), que como su nombre indica es borrable eléctricamente.

Esta tecnología se basa en el mismo principio de transistor de puerta flotante ligeramente modificado. En este nuevo transistor, denominado FLOTOX (*FLOating gate Tunnel OXide transistor*), las cargas de la base pueden ser evacuadas al aplicar tensiones altas mediante el efecto túnel. Los transistores FLOTOX se usan conjuntamente con transistores de selección para establecer la interconexión porque en este caso V_T es negativa.

La programación es idéntica. De nuevo se aplican tensiones altas para producir corrientes altas que inutilicen el transistor FLOTOX. El borrado se realiza también eléctricamente en este caso.

A pesar de ser más cómodo el proceso de borrado se sigue teniendo el problema de que, tanto para programar como para borrar dispositivos de esta tecnología, hay que extraerlos del sistema y conectarlos a un periférico especial que genere las tensiones adecuadas. Como ventaja se sigue teniendo la no-volatilidad de la tecnología EPROM.

9.2.3 Tecnología FLASH

La desventaja de la tecnología EEPROM se ha visto superada con la tecnología de las celdas FLASH. De nuevo se utiliza el transistor FLOTOX pero con la característica de que la capa de óxido de silicio que lo aisla del canal es extremadamente delgada. De esta manera con tensiones TTL se pueden grabar y borrar los dispositivos basados en esta tecnología. No es necesario extraerlos del sistema para programarlos y borrarlos, es decir, los dispositivos FLASH son **programables en el sistema** o son ISP (*In-System Programmability*).

Se sigue teniendo la ventaja de que la programación se mantiene aunque se elimine la alimentación del circuito, es decir no es volátil.

Como desventaja de esta tecnología (frente a la tecnología antifuse que se expone posteriormente en la sección 9.2.5) se tiene que no ofrece una escala de integración óptima, al igual que la tecnología EPROM y EEPROM.

En teoría se puede realizar borrado y reprogramación a nivel de celda individual en los dispositivos de esta tecnología. En la práctica el acceso de programación y borrado, debido a la estructura interna de la pistas de programación, queda limitado a sectores o bloques que se pueden programar/borrar por separado.

9.2.4 Tecnología SRAM

La celda SRAM toma su nombre de que se utiliza en las memorias SRAM (*Static Random Access Memory*), y contiene 4 transistores MOS que almacenan un 1 o un 0 mientras se mantenga la alimentación del circuito.

La celda SRAM tiene la ventaja de que es programable con tensiones TTL, y tiene la desventaja de que es volátil, de manera que cada vez que se vuelva a conectar la alimentación el circuito debe de volverse a programar.

Para realizar la programación de los dispositivos programables basados en esta tecnología hay 2 métodos fundamentales: O bien hay un dispositivo inteligente externo que lo programa, o bien se lee el programa de una memoria EPROM local externa al dispositivo.

En el primer método se establece un protocolo en el que el dispositivo programable es esclavo de un maestro inteligente (*host*); habitualmente un microcontrolador o un procesador que leen y re-envían el programa. Para ello, el dispositivo programable se configura automáticamente como una máquina de estados que se comunica con el *host* y distribuye internamente los bits para cada celda SRAM. Se dice entonces que el dispositivo está en **modo de programación**. Cuando se programa la última celda SRAM el dispositivo ya está listo para ser utilizado como se ha previsto y entra en el denominado **modo usuario**.

En el segundo método de programación el programa reside en una EPROM (habitualmente serie) conectada al dispositivo programable. Cuando se suministra la alimentación al circuito el dispositivo programable se configura como una máquina de estados que accede a la memoria durante el modo de programación. Una vez se ha programado completamente el dispositivo programable deja las salidas de la EPROM en alta impedancia e inicia su funcionamiento previsto.

9.2.5 Tecnología de antifusibles

Con esta metodología las interconexiones no se realizan a través de transistores sino a través de antifusibles o conexiones que al *quemarse* conducen. Un esquema de la configuración de un antifusible se muestra en la figura 9.5.

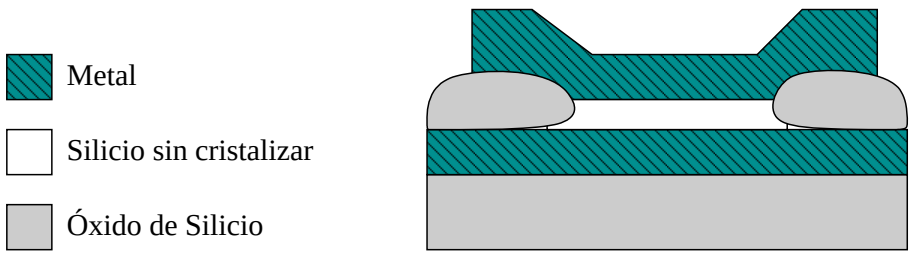


Figura 9.5: *Estructura de una conexión antifusible*

Entre las dos capas de metal que se pretenden conectar hay una capa de silicio amorfo sin cristalizar que es un dieléctrico. Al aplicarse pulsos de 12 V entre los dos metales se provoca que se perfore la capa de silicio aislante, conectándose ambos metales a través de una resistencia de unos 50 Ω.

Tecnología	Ventajas	Desventajas
Antifusible	No volatilidad, Alta integración	No reprogramabilidad
EEPROM	No volatilidad	No ISP
EPROM	No volatilidad	No ISP
SRAM	Reprogramabilidad, ISP	Volatilidad,
FLASH	Reprogramabilidad, No volatilidad, ISP	Baja integración

Tabla 9.1: *Tecnologías de programación ventajas y desventajas*

La desventaja de esta metodología es clara: Una vez fundido un antifusible no se puede volver a dejar en su estado inicial, con lo que los dispositivos basados en esta tecnología **no son reprogramables**. Además es necesario un dispositivo externo para programarlos fuera del sistema, al ser necesarias tensiones elevadas.

La ventaja de esta tecnología estriba en que que las celdas antifuse son más pequeñas que el resto de celdas de programación, con lo que se puede conseguir una mayor escala de integración que con el resto de tecnologías

La tabla 9.1 resume las ventajas y desventajas principales de cada tipo de tecnología.

9.3 Tipos de dispositivos programables

9.3.1 PALs

Los dispositivos programables más simples son las matrices lógicas programables, conocidas como PALs (Programmable And Logic). La arquitectura de una PAL genérica se muestra en la figura 9.6.

Las salidas no son más que una función OR de varias líneas AND cableadas. Las lineas horizontales cruzan completamente la PAL posibilitando una conexión en cada cruce con una línea vertical. Las salidas a su vez se pueden realimentar para hacer posible funciones más complejas que el número de minitérminos que caben en la función OR. En las salidas pueden haber biestables, útiles para realizar sistemas secuenciales sencillos.

Las PALs tienen una nomenclatura estándar en la que la primera cifra indica el número de entradas y la segunda el número de salidas. La letra intermedia indica si las salidas están invertidas (L), si no lo están (H), o si tienen un registro (R). De esta manera por ejemplo la PAL16R8 se corresponde con una PAL con un máximo de 16 entradas y 8 salidas con registro. Cabe hacer notar que esto no significa que el dispositivo lógico tenga 16+8=24 pines de entrada/salida, varios de estos pines pueden ser configurables como entradas o salidas.

Las PALs típicamente pueden tener un máximo número de 30 entradas, unas 15 salidas y unos 15 registros [Adv96].

Una característica importante de las PAL es el número de términos producto que tiene cada línea AND cableada, que en la figura 9.6 serian 4. Esto da la complejidad de la lógica que se puede implementar en dichos circuitos. Si se supera este número de minitérminos se deberá de realimentar externamente una salida a una entrada, con

la consiguiente pérdida de velocidad. Esto es evitable muchas veces en los sistemas secuenciales sin más que una recodificación adecuada de los estados.

A partir de esta simple arquitectura se definen las PALs universales. La arquitectura de la matriz de ANDs cableadas y interconexiones con realimentación apenas cambia, la única diferencia estriba que en las salidas en vez de tener flip-flops simples se tienen macroceldas. Las macroceldas a su vez incluyen un biestable (de tipo D generalmente) y lógica para multiplexar la salida y la entrada a él. El ejemplo más popular de PAL universal es la PAL $22V10$, en la que se pueden implementar máquinas de estados y sistemas secuenciales sencillos. La letra V indica *Versatile*, debido a que el ancho de la OR es variable para diversos pines.

Las PALs suelen ser de tecnología EPROM o EEPROM, consiguiéndose velocidades altas ($t_{PD} \approx 5\ ns$) debido a la simplicidad de los circuitos y a su simple modelo de temporización.

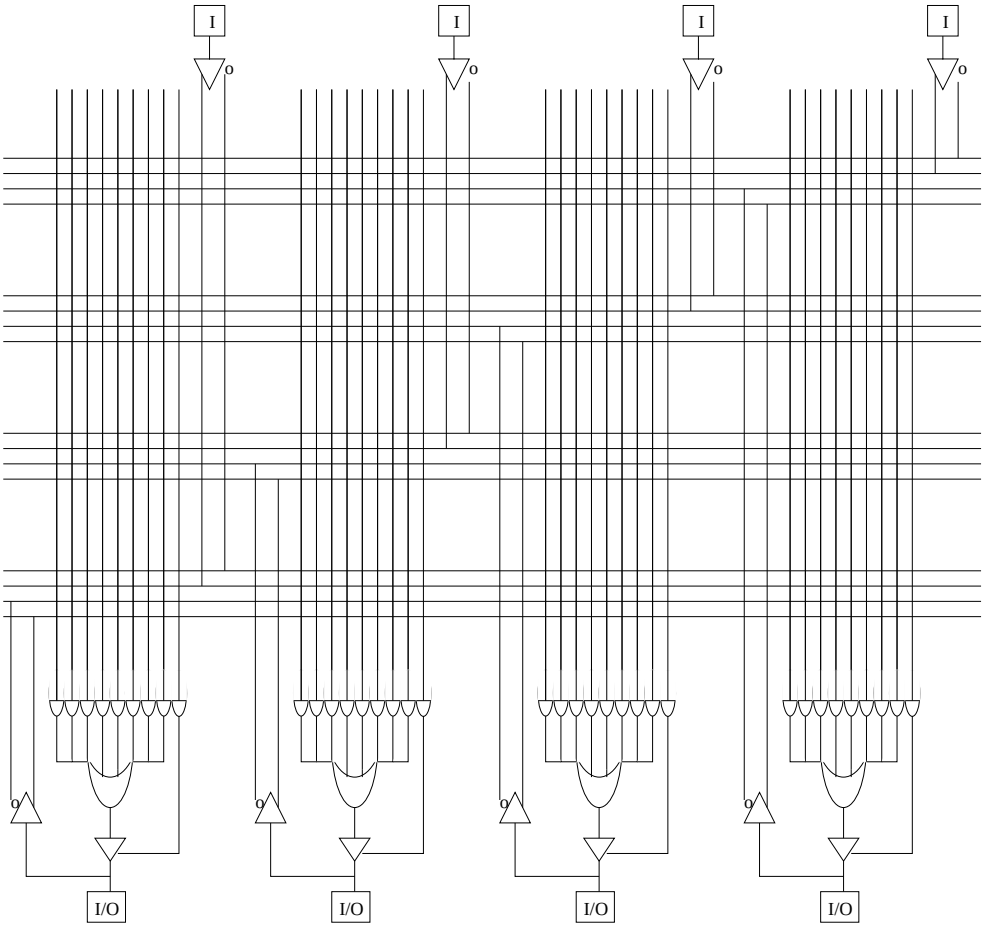


Figura 9.6: *Arquitectura de una PAL genérica*

9.3.2 CPLDs

A partir de las PALs, que son las PLDs más sencillas, se construyen las PLDs complejas o CPLDs. En la figura 9.7 se puede observar la arquitectura genérica de las CPLDs de

la familia Max7000 de Altera, siendo todas las arquitecturas de CPLDs actuales muy similares.

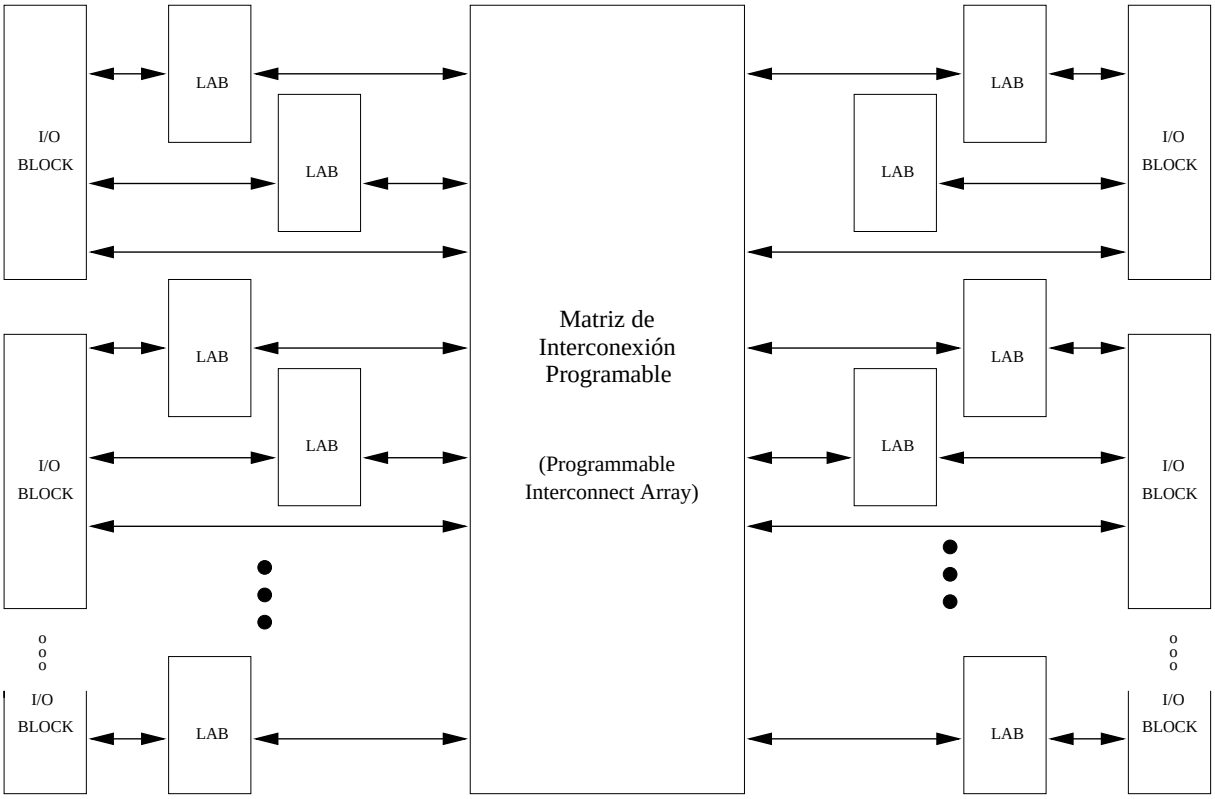


Figura 9.7: *Arquitectura de las CPLDS de la familia Max 7000 de ALTERA*

Las CPLDs tienen unos bloques internos, llamados bloques de matrices lógicas o LABs (*logic array blocks*), con una estructura similar a la arquitectura PAL mostrada en la figura 9.6. Además de estas pequeñas PALs, las CPLDs tienen una matriz de interconexión programable, que permite el conexionado de los LABs entre si y, por tanto, la realización de sistemas más complejos que con las PALs. Junto con los LABs, las CPLDs tienen unos bloques especiales de entrada/salida que incorporan salidas triestado y sirven también para suministrar corriente.

Las CPLDs son dispositivos realmente complejos que pueden llegar a incorporar más de 20.000 puertas lógicas equivalentes con más de 50 LABs [Alt95]. El mercado de CPLDs cambia muy rápidamente, ofreciendo los fabricantes cada vez CPLDs más complejas y con mayores prestaciones de velocidad, integración y menor consumo de potencia. Cualquier análisis del estado de la tecnología a la escritura de este capítulo sería superado unos pocos meses después.

La principal característica de las CPLDs son sus prestaciones. La arquitectura de las CPLDs hace que se puedan implementar funciones lógicas complejas con pocos niveles de realimentación, y por tanto, pocos retrasos. La arquitectura de las CPLDs también hace que se puedan predecir los retrasos de una manera sencilla y así el diseñador puede prever las prestaciones de su diseño (velocidad de funcionamiento) incluso antes de implementarlo.

9.3.3 FPGAs

Paralelamente al desarrollo de las CPLDs se han desarrollado las redes de puertas lógicas programables o FPGAs (*Field Programmable Gate Arrays*). En la figura 9.8 puede observarse la arquitectura genérica de una FPGA clásica.

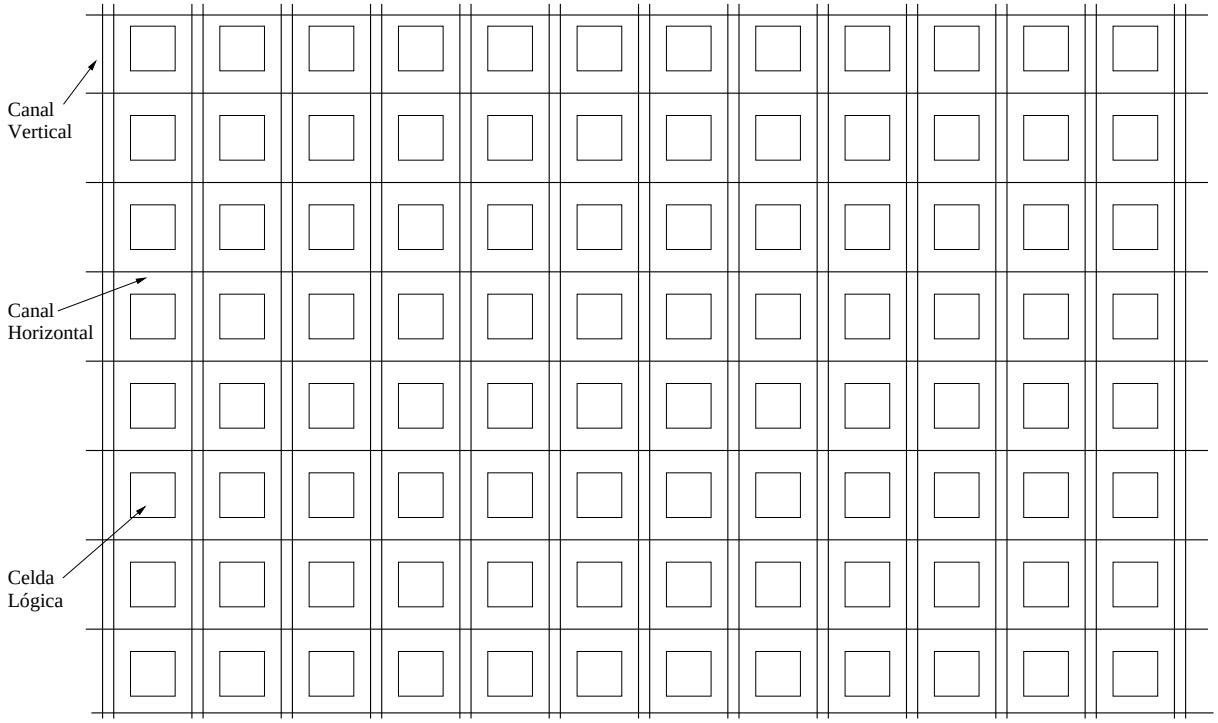


Figura 9.8: *Arquitectura de una FPGA genérica*

Al contrario de las CPLDs, aquí no se tienen varios bloques complejos como los LABs y una matriz de interconexión localizada. En las FPGAs se tienen muchas celdas lógicas pequeñas distribuidas regularmente por su superficie. Cada celda lógica incluye uno o dos flip-flops y unas pocas pequeñas LUTs (*lookup tables*) que pueden implementar funciones combinacionales sencillas. Las celdas lógicas suelen incorporar también algún multiplexor para interconectar las LUTs con los flip-flops.

El conexionado entre las diversas celdas lógicas no se centraliza en una matriz de interconexión sino que se realiza mediante canales de interconexión. Estos canales están distribuidos uniformemente entre las celdas lógicas y cruzan la superficie de la FPGA horizontal y verticalmente.

La ventaja de esta arquitectura regular es que, aprovechando la alta escala de integración de la tecnología VLSI, se pueden realizar dispositivos bastante más complejos que con las CPLDs más avanzadas. A la fecha de la escritura del presente capítulo las FPGAs más grandes podían tener una complejidad de hasta 250.000 puertas equivalentes con más de 10.000 flip-flops y celdas lógicas [Xil94].

La desventaja de la arquitectura FPGA parece clara a la vista de la figura 9.8; para realizar un sistema complejo se deben realizar muchas conexiones entre las celdas lógicas, generando caminos realmente largos y realimentaciones profundas, por tanto grandes retrasos impredecibles a priori.

Como conclusión a la revisión de la arquitectura genérica de las FPGAs y las CPLDs se tiene que a mayor complejidad (FPGAs) se obtiene una mayor pérdida de velocidad de funcionamiento.

9.3.4 Arquitecturas híbridas

El problema de la pérdida de prestaciones con las FPGAs al implementar sistemas complejos ha sido abordado por los diferentes fabricantes de dispositivos programables. La nueva filosofía de diseño consiste en realizar arquitecturas híbridas que intentan combinar los beneficios de ambas arquitecturas. Se intenta mantener la velocidad de las CPLDs mediante una nueva arquitectura de conexionado, agrupando las celdas lógicas elementales en estructuras más grandes. Por otra parte se intenta mantener la granularidad de las FPGAs difundiendo un gran número de estos bloques más grandes entre las líneas de interconexión. En la figura 9.9 se muestra, como ejemplo, la arquitectura híbrida de la familia FLEX 8000 de Altera [Alt94].

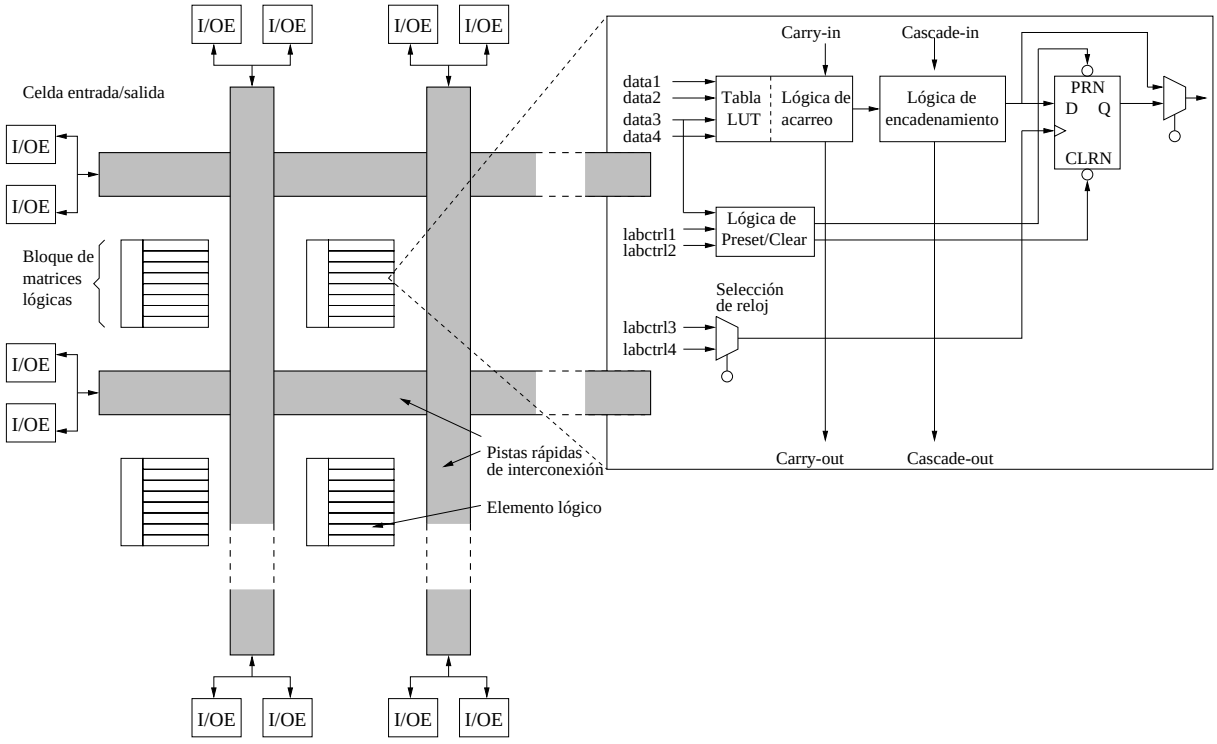


Figura 9.9: *Arquitectura de la familia FLEX 8000 de Altera*

Se puede observar que en esta arquitectura las celdas lógicas están agrupadas en bloques, que en el caso de la figura, son de 8 elementos lógicos. Esta agrupación por bloques facilita el agrupamiento de las señales y la aparición de buses durante la síntesis. En el caso de que se traten señales de 8 bits, un solo bloque cubriría una señal, evitando la aparición de retrasos distintos en los diferentes bits de la señal.

Además, la estructura de los elementos lógicos incluye líneas especiales de acarreo adelantado y de encadenamiento. De esta manera se mejora la velocidad de contadores y otros dispositivos lógicos con realimentaciones y acarreo. La interconexión de los

bloques de matrices lógicas se realiza mediante pistas rápidas de interconexión similares a las matrices de interconexión de las CPLDs.

Se puede concluir que las arquitecturas híbridas combinan la granularidad y flexibilidad, de las FPGAs, con la agrupación por bloques (y por tanto rapidez) de las CPLDs.

9.3.5 Mega-estructuras

A partir de las nuevas arquitecturas híbridas, y como consecuencia de la alta escala de integración que es está consiguiendo con la tecnología SRAM, se han desarrollado recientemente (principios de 1999) dispositivos programables más complejos.

Las mega-estructuras desarrolladas por Altera con su familia APEX20k y Xilinx con su familia VIRTEX XCV00 son un paso más en la complejidad de los dispositivos programables. Tal como se muestra en la figura 9.10, la alta escala de integración consigue agrupar los bloques lógicos de las arquitecturas híbridas en un nivel jerárquico superior llamado megabloques. La interconexión de los bloques lógicos en megabloques se realiza por pistas rápidas de conexión, de la misma manera que la interconexión entre megabloques.

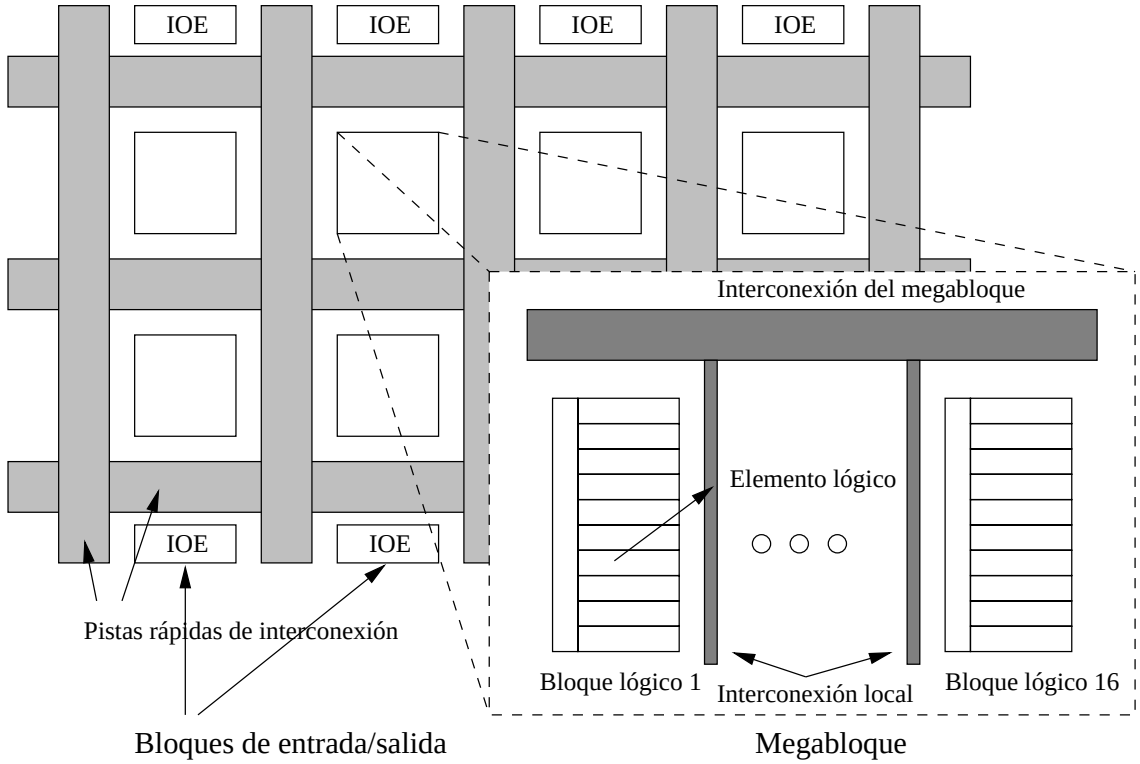


Figura 9.10: *Arquitectura de la familia APEX20K de Altera*

Esta agrupación de bloques lógicos (16 en la familia APEX) con pistas rápidas permite la aparición de buses con unas altas prestaciones. La alta escala de integración permite en estas familias llegar a 200 megabloques, más de 2 millones de puertas equivalentes, más de 40.000 elementos lógicos y medio millón de bits de RAM, con encapsulados de más de 700 pines de entrada/salida de usuario.

Estas cifras de integración, junto con las altas prestaciones conseguidas (del orden de 100 MHz), provocarían una disipación de potencia inaceptable, lo que ha obligado a desarrollar estas familias como lógica programable de baja tensión (desde 1'8V hasta 3'3V).

9.4 Conclusiones: Criterios de selección de lógica programable

Los ASICs pueden ser una opción con múltiples ventajas a la implementación de sistemas digitales con componentes lógicos discretos. Las múltiples ventajas que ofrecen hace que se vaya incorporando cada vez más rápidamente en los sistemas digitales.

La lógica programable, a su vez, ofrece ventajas que la hacen ser una opción muy a tener en cuenta en el diseño de sistemas digitales.

Los dispositivos programables van evolucionando más rápidamente si cabe que las familias lógicas. El gran volumen de mercado, y el hecho de que no se pueda decir que un dispositivo o tecnología son siempre mejores, hace que haya una gran competencia entre los diversos fabricantes.

Los dispositivos programables más sencillos, como son las PALs, son los más adecuados para implementar la inevitable *glue logic* o lógica de pegado que aparece en sistemas complejos. Las pocas puertas y/o biestables necesarios para conectar módulos estándar pueden ser implementables en PALs universales como la 22V10, o más sencillos como la 16V8.

De manera adicional, la arquitectura simple de estos dispositivos y su tecnología (habitualmente EPROM o EEPROM), hacen que se puedan conseguir altas velocidades de funcionamiento. De todas maneras para aplicaciones muy específicas de suministro de interconexión con suministro de corriente o de gran velocidad, pueden no ser adecuados.

Las CPLDs pueden ser adecuadas para implementar máquinas de estados complejas o sistemas digitales que necesiten más registros de los disponibles en una PAL. De nuevo su arquitectura y su modelo de temporización regular, hacen que su velocidad de funcionamiento pueda llegar a los 100 MHz e incluso superarlos. Estas altas prestaciones hacen adecuadas a las CPLDs en la implementación de sistemas complejos que funcionen a alta velocidad.

Las FPGAs por el contrario tienen una estructura de grano más fino lo que hace que, al hacer la interconexión de los elementos lógicos, la temporización sea imprevisible. Sin embargo esta misma estructura regular favorece la alta escala de integración de estos dispositivos, pudiéndose llegara a dispositivos con cientos de miles de puertas equivalentes. Estas características las hacen ser la única opción para implementar sistemas muy complejos con una gran número de registros, a pesar de las pérdidas de prestaciones. De manera adicional, la utilización de tecnologías SRAM (para garantizar la ISP), y de tecnologías antifuse (para aumentar la escala de integración), hacen que sean dispositivos más lentos que las CPLDs.

Como solución intermedia han aparecido las arquitecturas híbridas que combinan la granularidad de las FPGAs con estructuras de agrupamiento inteligentes. Con los dispositivos de arquitecturas híbridas se pueden llegara implementar sistemas complejos, antes reservados a las FPGAs, con las prestaciones de las CPLDs.

Por último la aparición de las mega-estructuras permite la integración completa de sistemas complejos como son procesadores y su lógica periférica en un solo dispositivo programable. La mejora de los compiladores de silicio, y la configuración del HDL en función de la aplicación están dirigiendo el diseño de sistemas empotrados hacia la lógica programable. En cualquier caso el compromiso prestaciones/precio se debe de tener siempre en cuenta para realizar la elección del dispositivo más adecuado.

Capítulo 10

Memorias

10.1 Introducción

El almacenamiento de la información es un problema importante en los sistemas digitales. Se ha descrito en el capítulo 5 el almacenamiento en registros y la construcción de éstos a partir de biestables. También se han descrito en el capítulo 9 las tecnologías existentes de almacenamiento de la información para circuitos integrados. Estas mismas tecnologías, que en el caso de la lógica programable almacenan conexiones entre pistas, son las que se utilizan para almacenar información digital.

En la asignatura **Estructura de Computadores II** se analizó como se organizan los diversos tipos de memorias en capas (en función de su coste y su tiempo de acceso) estableciendo una **jerarquía de memoria**. En este tema se van a abordar, desde el punto de vista tecnológico y de temporización para su acceso, los circuitos de memoria [Toc93].

Definición: Una memoria es un dispositivo que almacena información de forma binaria con la posibilidad de recuperarla.

10.2 Parámetros más importantes

La *bondad* de un tipo de memorias se mide por una serie de parámetros que caracterizan y permiten comparar los diversos tipos de memorias. Los más importantes y representativos son:

1. **Tiempo de escritura**: Es el tiempo que transcurre desde que se suministra al dispositivo la información a almacenar (y la dirección en su caso) y ésta queda grabada. Pueden variar desde los pocos nanosegundos de una SRAM a los milisegundos de una unidad de almacenamiento masivo.
2. **Tiempo de lectura**: Es el tiempo transcurrido desde que se solicita una lectura de información y ésta se produce. Estos tiempos suelen ser del orden de magnitud de los de escritura. Lo que se hace en la mayoría de casos es referirse al tiempo de lectura como el **tiempo de acceso** (t_{ACC}). Este tiempo es el referente básico de velocidad de una memoria.

3. **Capacidad**: Es la cantidad de bits que se pueden almacenar en una memoria simultáneamente. Así se tienen las correspondencias (en esta caso expresadas en bytes): 1 Kilobyte = 2^{10} bytes = 1.024 bytes, 1 Megabyte = 2^{10} Kilobytes = 2^{20} bytes y 1 Gigabyte = 2^{30} bytes. La capacidad de almacenamiento de los dispositivos de almacenamiento masivo más populares es del orden de decenas de gigabytes. El tamaño de las memorias principales de los ordenadores personales llega al centenar de megabytes.
4. **Volatilidad**: Una memoria se dice que es volátil si necesita un suministro constante de energía para mantener la información. Las memorias no volátiles son aquellas que mantienen la información a pesar de que se desconecte la alimentación al sistema.
5. **Densidad de información**: Es el número de bits por unidad de volumen físico o unidad de superficie. Expresa la integración del almacenamiento.

10.3 Clasificación

Se va a realizar una clasificación en función de algunos de los parámetros más importantes anterior descritos. De esta manera se podrá clasificar:

En función de la forma de acceso a la información:

- **Acceso aleatorio**: El tiempo necesario para recuperar o almacenar la información no depende de la posición de la celda ni del orden en que se haya introducido el dato. Es decir, el tiempo de escritura y tiempo de lectura es constante. Estas son las memorias de utilización más sencilla en sistemas digitales ya que este tiempo es fijo. Un ejemplo es la memoria RAM de un ordenador, que como su nombre indica es de acceso aleatorio (*Random Access Memory*).
- **Acceso secuencial**: El tiempo de acceso depende de la posición de la palabra con respecto a una referencia u origen. Suelen ser memorias de almacenamiento masivo de datos y no de uso intensivo por módulos procesadores (p.ej. un cinta DAT).También son memorias útiles para un tipo concreto de diseños de sistemas secuenciales. Un ejemplo son los registros de desplazamiento SISO (*Serial Input Serial Output*).
- **Acceso mixto**: La información está almacenada por bloques, a estos bloques se accede de forma aleatoria, pero dentro de cada bloque el acceso es secuencial. Ejemplo: Los discos magnéticos.

En función de la volatilidad:

- **Volátiles**: La información se almacena mediante dispositivos electrónicos que necesitan constantemente una alimentación para mantener los estados presentes (información). Habitualmente son biestables o capacidades las que almacenan esta información. Una falta de alimentación significa la perdida irreversible de los datos, aunque a veces pueden quedar cargas estáticas que brevemente mantengan la información. Un ejemplo de una memoria volátil es la memoria principal de un PC.
- **No volátiles**: Estas memorias no necesitan de alimentación para mantener los datos, aunque si pueden necesitarla para la lectura y/o almacenamiento de información. En estas memorias se almacena la información más necesaria para un sistema de manera que, aunque se pierda la alimentación del sistema, el sistema pueda recuperar la funcionalidad. Ejemplos son los discos magnéticos y la ROM de un

ordenador principal.

Por su función o jerarquía:

El precio de las memorias viene a ser proporcional a su velocidad. De esta manera hay que llegar a un compromiso entre la velocidad de acceso y el coste de los sistemas. Para ello se realizan diseños de memoria jerárquicos en los que las memorias mas rápidas (y caras) son las más *cercanas* a la CPU. Las memorias más lentas serán las destinadas al almacenamiento masivo debido a su alto coste y estarán más *lejos* del procesador o elementos de proceso. El diseño jerárquico organizará la información de manera que los datos más utilizados estarán en las memorias de acceso más rápido. Atendiendo a este criterio se puede realizar la clasificación:

- **Memoria Cache**: Es una memoria intermedia entre el procesador y la memoria principal. Su acceso es transparente al programador, se realiza por hardware y aprovecha los principios de localidad temporal y espacial. Son las memorias más rápidas (y más caras).
- **Memoria principal**: Es la memoria directamente accesible por la unidad de proceso. Como ejemplo se tiene la memoria RAM directamente accesible por la CPU.
- **Memoria tampón**: Es una memoria *buffer* que está entre los periféricos que son más lentos que la CPU. De esta manera los periféricos no bloquean al procesador. Un ejemplo son los buffers de impresora.
- **Memoria de masas**: Es la memoria más lenta y barata. Se utiliza para almacenar la información que no se está procesando en ese momento. Está en el nivel más externo de la jerarquía.

10.4 Circuitos de memoria

Dentro de los dispositivos de memoria se van a presentar los circuitos integrados que almacenan información. Los dispositivos de almacenamiento de memoria masivos como discos o cintas se analizarán en la sección de periféricos de la asignatura **Estructura de Computadores II**.

10.4.1 RAM estática asíncrona

Una RAM estática asíncrona o SRAM (*Static Random Access Memory*) es una memoria volátil y de acceso aleatorio muy rápido. Esta característica las hace ideales para su utilización en sistemas digitales y como memoria en computadores. Lamentablemente su alto coste hacen prohibitiva su utilización de forma masiva como memoria directamente direccionable en muchos computadores, quedando relegada su utilización a registros y memoria . En la sección 9.2.4 se describe la tecnología SRAM utilizada para almacenar los bits de programación en la lógica programable basada en esta tecnología.

El tiempo de acceso en las SRAM más rápidas es menor de 10 ns, aunque esta cifra baja rápidamente con las mejoras tecnológicas asociadas a los procesos de fabricación (las memorias SRAM actuales suelen ser de tecnología CMOS de alta velocidad). La capacidad en un sólo circuito integrado, a la escritura de este capítulo, es del orden de 4 Mbytes.

Estructura interna

La SRAM está formada por una matriz de celdas de memoria donde se almacenan los bytes. Un decodificador binario que selecciona la celda adecuada en función de la dirección suministrada, lógica de control y buffers para la comunicación con los buses de datos y direcciones. En la figura 10.1 se muestra un diagrama de bloques de una SRAM genérica.

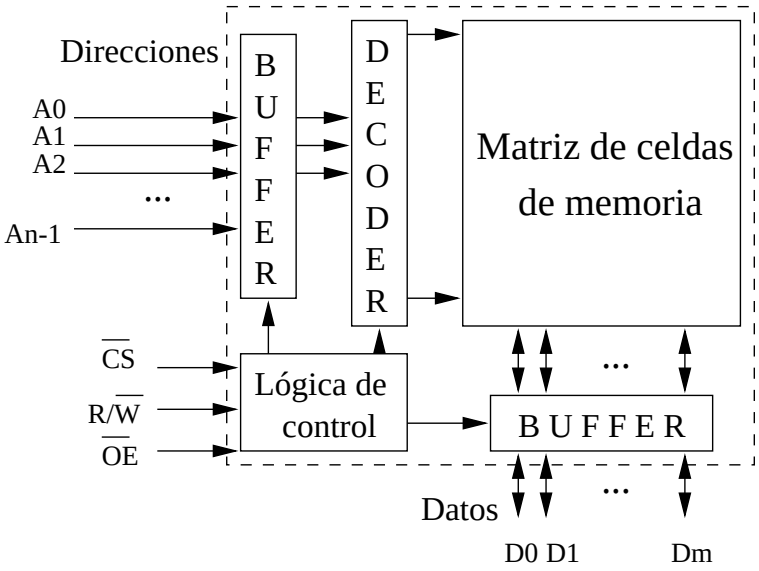


Figura 10.1: *Diagrama de una SRAM*

Modos de funcionamiento

Para leer un dato desde una SRAM debe realizarse el siguiente proceso:

1. Establecer la dirección de la posición de memoria a la que se quiere acceder.
2. Activar la línea de lectura $R/\overline{W}$ si no estaba ya activada.
3. Seleccionar el circuito y habilitar la salida mediante las señales $\overline{CS}$ y $\overline{OE}$ si no estaban ya activadas.

Durante la lectura la SRAM se comporta como un mero circuito combinacional en el que las salidas (bus de datos) dependen de las entradas (bus de direcciones). Estas acciones se deben realizar preferentemente respetando el orden indicado, pero se puede cambiar el orden. En la figura 10.2 se muestra un ciclo de lectura con los tiempos más característicos implicados.

- t_{ACC} **Tiempo de acceso desde las direcciones**: Es el tiempo necesario para que la SRAM suministre el dato correcto a partir de una dirección válida con las señales $\overline{CS}$ y $\overline{OE}$ ya habilitadas. En la figura 10.2 la última acción es la habilitación de $\overline{OE}$ y por tanto es la señal que gobierna la lectura.
- t_{CS} **Tiempo de acceso desde la selección del circuito**: Es el tiempo necesario para que la SRAM suministre el dato correcto a partir de la selección de dispositivo con $\overline{CS}$. Se supondrá que la dirección ya es válida y que la señal $\overline{OE}$ ya está habilitada.

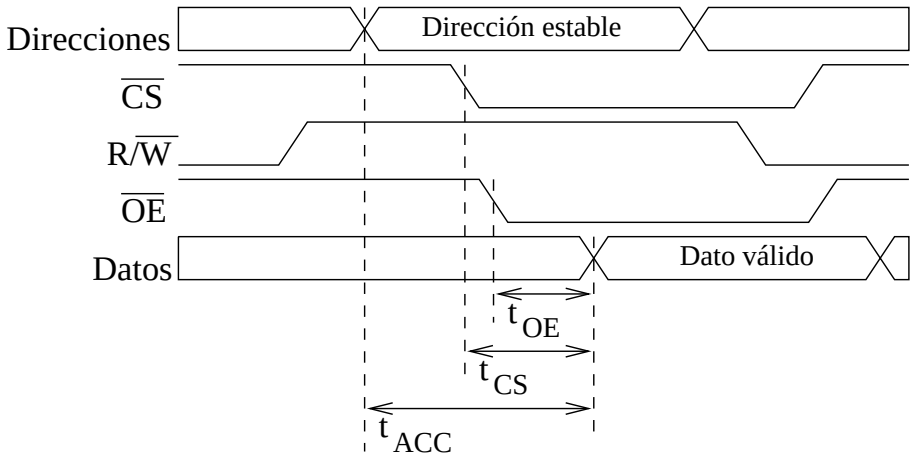


Figura 10.2: *Cronograma de lectura de una SRAM*

- t_{OE} **Tiempo de habilitación de las salidas**: Es el tiempo necesario para que la SRAM suministre el dato correcto a partir de la habilitación de las salidas. Las salidas por tanto pasarán de alta impedancia Z a suministrar el dato correcto. Se supondrá de nuevo que la dirección ya es válida y que la señal $\overline{CS}$ ya está habilitada.
- t_{RC} **Tiempo de ciclo de lectura**: Es el tiempo necesario para que se produzca la lectura y la memoria SRAM quede lista para cualquier otro ciclo. Este tiempo será al mayor de todos.

Una de las formas de escribir un dato en una SRAM es mediante una escritura controlada por $\overline{CS}$, tal como se muestra en la figura 10.3. Análogamente se podría realizar una escritura controlada por la señal $R/\overline{W}$. En la escritura controlada por $\overline{CS}$ debe de realizarse el siguiente proceso:

1. Establecer la dirección de la posición de memoria en la que se quiere grabar el dato.
2. Activar la línea de escritura $R/\overline{W}$.
3. Seleccionar el circuito mediante la señal $\overline{CS}$.
4. Dejar el dato válido en el bus.
5. Desactivar la señal $\overline{CS}$.

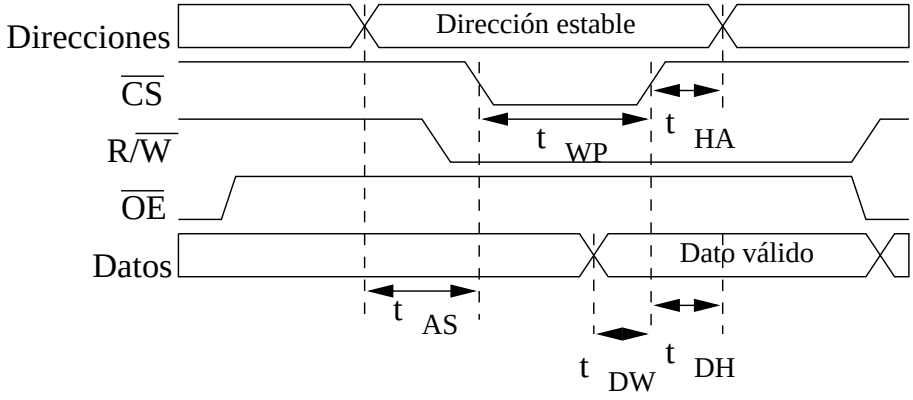


Figura 10.3: *Cronograma de escritura de una SRAM controlado por* $\overline{CS}$

Los tiempos implicados en el ciclo de escritura de la figura 10.3 son:

- t_{AS} **Tiempo de establecimiento de la dirección**: Es el tiempo que debe de permanecer la dirección válida antes de que $\overline{CS} = 0$.
- t_{WP} **Ancho de pulso mínimo**: Tiempo mínimo que se debe mantener $\overline{CS} = 0$.
- t_{DW} **Tiempo de establecimiento del dato**: Es el tiempo que debe permanecer estable el dato en el bus de datos previamente a que $\overline{CS} = 1$.
- t_{DH} **Tiempo del mantenimiento del dato**: Es el tiempo que debe permanecer estable el dato en el bus de datos posteriormente a que $\overline{CS} = 1$.
- t_{HA} **Tiempo de mantenimiento de la dirección**: Es el tiempo que debe permanecer estable la dirección de escritura posteriormente a que $\overline{CS} = 1$.
- t_{WC} **Tiempo de ciclo de escritura**: Es el tiempo necesario para que se produzca la escritura y la memoria SRAM quede lista para cualquier otro ciclo.

Cabe hacer notar como los tiempos y el funcionamiento son similares a la escritura en un biestable maestro-esclavo, donde la señal $\overline{CS}$ sería el reloj sensible a flanco de subida. Se puede repetir el mismo cronograma, pero en este caso la señal que *capturaría* el dato sería $R/\overline{W}$.

La memoria cuando no está seleccionada ($\overline{CS} = 1$) está en modo de espera o *stand-by*. En este modo la memoria conserva almacenados los datos, manteniendo un consumo muy bajo.

10.4.2 RAM estática síncrona

Esta RAM tiene una estructura interna similar a las SRAM asíncronas. La diferencia estriba en que existe una señal de reloj que sincroniza el proceso de lectura y escritura. Esto puede ser útil cuando se van a realizar accesos a direcciones consecutivas en memoria: Primero se suministra la dirección base y después, en cada flanco, la memoria proporciona el dato (lectura) o se le suministra el dato a escribir. Este es el tipo de acceso a memoria conocido como ráfaga o *burst*.

Las memorias cache externas de algunos microprocesadores son de este tipo para facilitar el acceso de datos en modo ráfaga y acelerar el proceso de acceso a bloques de memoria. De este tipo de acceso se ve un ejemplo en la figura 10.4. El inicio de la transacción en modo ráfaga se señala con la señal $\overline{ADSP}$. En el primer flanco se captura la dirección base de la transacción y en los sucesivos flancos se obtienen los datos de esa dirección y consecutivos (en el ejemplo sólo 2).

La temporización de este tipo de acceso será similar a los anteriores. De nuevo habrá que respetar tiempos mínimos de establecimiento y mantenimiento y anchos de pulsos mínimos, tanto para las direcciones como para los datos (sí se está escribiendo).

10.4.3 Memorias RAM dinámicas

Estructura interna

La RAM dinámica o DRAM (*Dynamic Random Access Memory*). Son memorias volátiles y de acceso aleatorio al igual que las SRAM. Los elementos de memoria son capacidades a las que se accede con un solo transistor, en vez de celdas con varios tran-

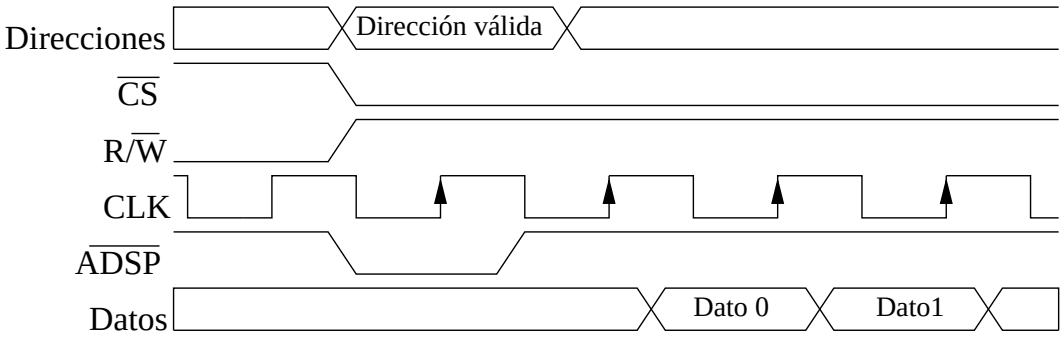


Figura 10.4: *Lectura de una SRAM síncrona en modo ráfaga*

sistores, con lo que se consigue una muy alta escala de integración (ver figura 10.5). El inconveniente principal es que las capacidades se descargan mediante la corriente de pérdidas de los transistores. Esto obliga a tener que realizar un *refresco* periódico de las capacidades si se quiere evitar que se pierda la información. En los ciclos normales de lectura y escritura habrá que intercalar ciclos de refresco y lógica adicional que se encargue de realizar estos ciclos. La periodicidad del refresco dependerá del caso concreto pero no suele exceder los 2 ms.

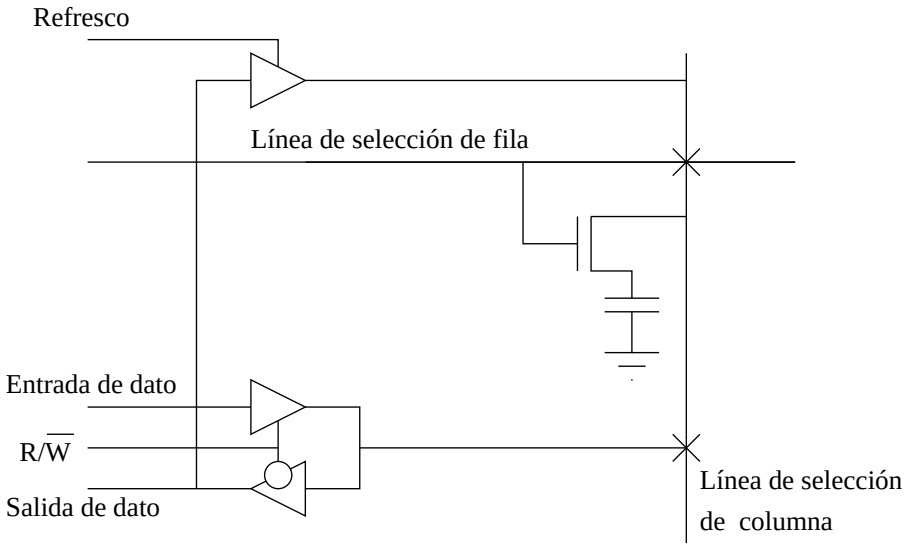


Figura 10.5: *Estructura de las celdas de memoria DRAM*

La estructura de las DRAM está organizada por filas y columnas, multiplexando las direcciones, para facilitar la tarea de refresco y reducir el número de líneas de direcciones (que sería grande debido a la gran capacidad de estas memorias). En la figura 10.6 se muestra esta estructura. La ventaja de tener memorias de gran tamaño se ve contrarrestada por el tiempo de acceso que puede llegar a ser un orden de magnitud mayor que en las SRAM.

Modos de funcionamiento

Para realizar una lectura de una DRAM se debe realizar el siguiente proceso:

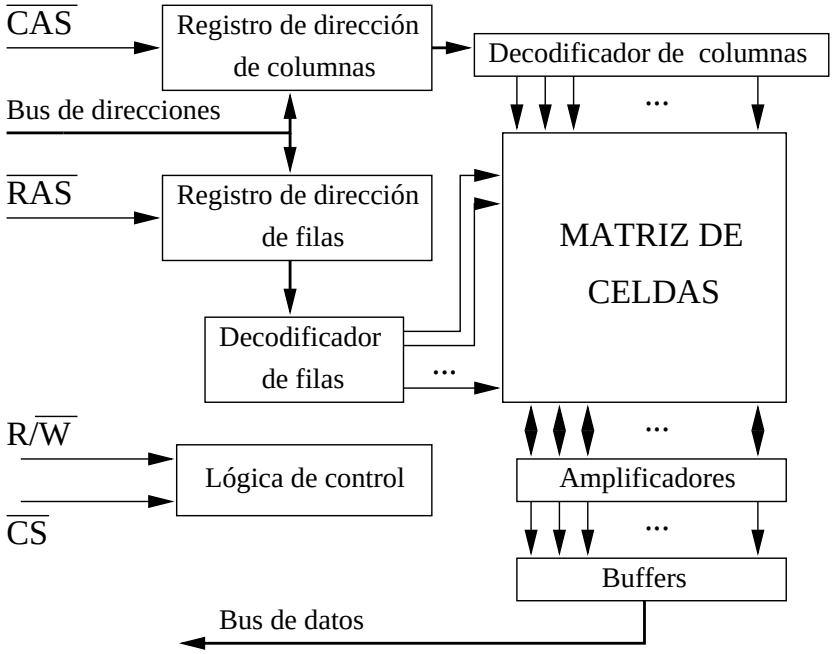


Figura 10.6: *Diagrama de bloques de una DRAM*

1. Dejar en el bus de direcciones la parte alta de la dirección (fila).
2. Validarla con $\overline{RAS}$.
3. Establecer en el bus de direcciones los bits bajos de la dirección (columna).
4. Validarla con $\overline{CAS}$.
5. Habilitar la señal de lectura $R/\overline{W}$.

Cuando se selecciona una fila se selecciona toda un línea de transistores y celdas. Con la selección de columna se particulariza la celda concreta poniendo esta señal a un nivel intermedio entre el nivel alto y bajo. La realimentación con los amplificadores de señal hace que la señal se incremente o decremente hasta el uno o el cero lógico en función del valor almacenado en el condensador. Posteriormente, se habilita la salida de la celda que está en la columna seleccionada. El proceso de lectura es destructivo, por lo que hay que refrescar la celda leída. Este proceso de refresco es automático y se produce en toda una fila simultáneamente debido a la estructura de realimentación con amplificadores de señal. Cada vez que se active la señal $\overline{RAS}$ se refresca toda la fila seleccionada.

Las acciones para leer una DRAM se deben realizar siguiendo una temporización que debe respetarse para asegurar el correcto funcionamiento de la DRAM. Estos tiempos se muestran en la figura 10.7, y se describen a continuación.

- t_{ASR} **Tiempo de establecimiento de fila**: Es el tiempo necesario que deben mantenerse los bits de mayor peso en el bus de direcciones previamente a la validación con la señal $\overline{RAS}$.
- t_{RAH} **Tiempo de mantenimiento de fila**: Es el tiempo necesario que deben mantenerse los bits de mayor peso en el bus de direcciones después de la activación de la señal $\overline{RAS}$.
- t_{RAS} **Tiempo de pulso de la señal** $\overline{RAS}$: Es el tiempo mínimo necesario que

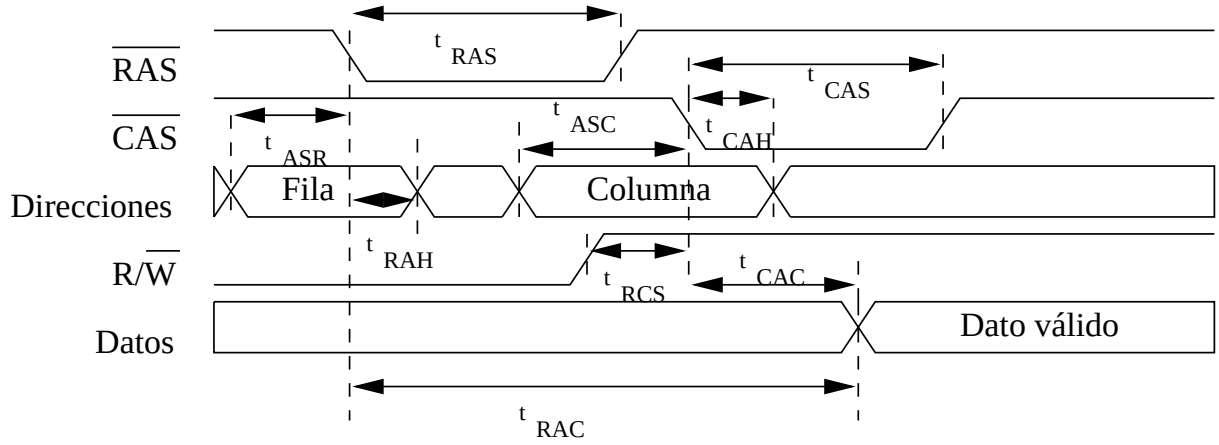


Figura 10.7: *Ciclo de lectura en una memoria DRAM*

debe estar activa la señal $\overline{RAS}$.

- t_{ASC} **Tiempo de establecimiento de columna**: Es el tiempo necesario que deben mantenerse los bits de menor peso en el bus de direcciones previamente a la validación con la señal $\overline{CAS}$.
- t_{CAH} **Tiempo de mantenimiento de columna**: Es el tiempo necesario que deben mantenerse los bits de menor peso en el bus de direcciones después de la activación de la señal $\overline{CAS}$.
- t_{CAS} **Tiempo de pulso de la señal** $\overline{CAS}$: Es el tiempo mínimo necesario que debe estar activa la señal $\overline{CAS}$.
- t_{CAC} **Tiempo de acceso desde la activación de** $\overline{CAS}$: Es el tiempo que transcurre desde que se activa la señal $\overline{CAS}$ hasta que aparecen los datos válidos.
- t_{RAC} **Tiempo de acceso desde la activación de** $\overline{RAS}$: Es el tiempo que transcurre desde que se activa la señal $\overline{RAS}$ hasta que aparecen los datos válidos.
- t_{RCS} **Tiempo de establecimiento de la señal de lectura**: Es el tiempo que se debe mantener la señal $R/\overline{W}$ activa previamente a la activación de la señal $\overline{CAS}$.
- t_{RC} **Tiempo de ciclo de lectura**: Duración total del proceso de lectura. Es decir, desde la activación de la señal $\overline{RAS}$ hasta que la memoria esté lista para el siguiente ciclo.

Para realizar una escritura en una DRAM se debe de realizar el siguiente proceso:

1. Establecer en el bus de direcciones los bits de mayor peso (fila).
2. Validar con $\overline{RAS}$.
3. Habilitar la señal de escritura $R/\overline{W}$.
4. Poner el dato válido en el bus.
5. Establecer en el bus de direcciones los bits de menor peso (columna).
6. Validar con $\overline{CAS}$.

Los tiempos son muy similares al ciclo de escritura, tal como se muestra en la figura 10.8. Los únicos tiempos que son diferentes son los siguientes:

- t_{DS} **Tiempo de establecimiento del dato**: Es el tiempo necesario que debe mantenerse el dato estable previamente a la bajada de $\overline{CAS}$.

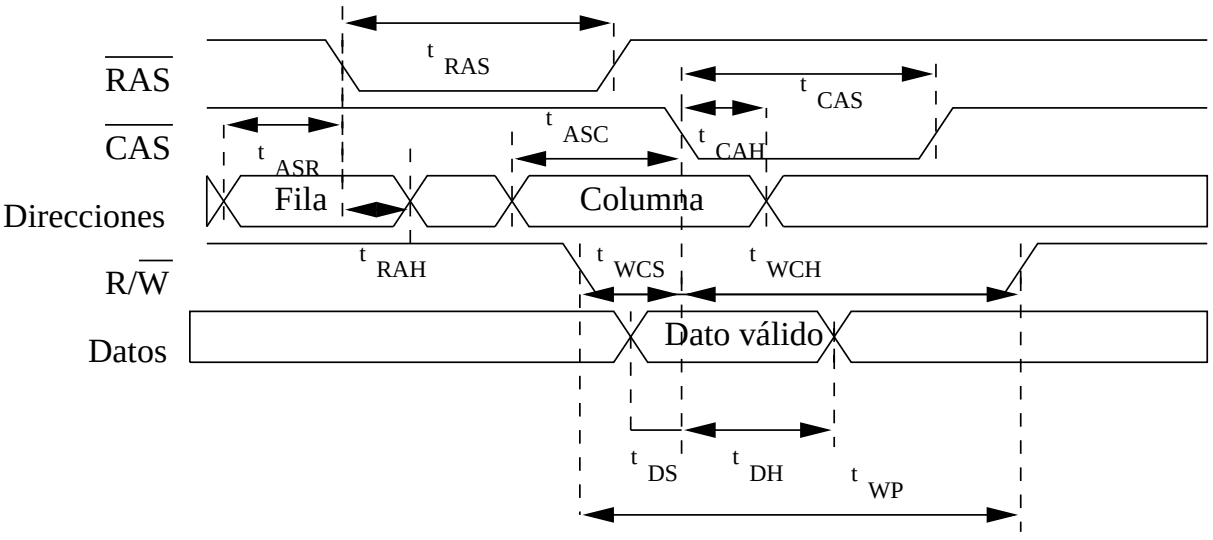


Figura 10.8: *Ciclo de escritura en una memoria DRAM*

- t_{DH} **Tiempo de mantenimiento del dato**: Es el tiempo que debe mantenerse el dato válido en el bus posteriormente a la activación de la señal $\overline{CAS}$.
- t_{WP} **Ancho de pulso de la señal de escritura**: Duración mínima que se debe mantener activa la señal de escritura.
- t_{WC} **Tiempo de ciclo de escritura**: Duración total del proceso de escritura. Es decir, desde la activación de la señal $\overline{RAS}$ hasta que la memoria esté lista para el siguiente ciclo
- t_{WCS} **Tiempo de establecimiento de la señal de escritura**: Tiempo que debe estar la señal $R/\overline{W}$ a cero previamente a la bajada de la señal $\overline{CAS}$.
- t_{WCH} **Tiempo de mantenimiento de la señal de escritura**: Tiempo que se debe mantener la señal $R/\overline{W}$ a cero posterior a la bajada de la señal $\overline{CAS}$.

Debido a la configuración interna de las DRAM siempre que se suministra el valor de una fila y se activa la señal $\overline{RAS}$ todas las celdas que tienen en común esa dirección de fila se refrescan. De esta manera, cuando se está leyendo o escribiendo en una dirección se está realizando un refresco de los bits de la DRAM que tienen en común la misma fila.

Este modelo de páginas de una DRAM es muy útil para realizar accesos rápidos a posiciones que estén en la misma página. Así sólo se suministra la dirección de la fila, ésta se capturará, y posteriormente se van suministrando las diversos bits de menor peso o columnas validando con $\overline{CAS}$.

Esta metodología se sigue para realizar el refresco en una DRAM. Se suministra sólo las filas y se activa la señal de $\overline{RAS}$. En cada fila se refresca una página completa de la DRAM. Los tiempos a respetar serán los relacionados con las filas, tanto de establecimiento como de mantenimiento y ancho de pulso. De esta manera se tendrán t_{ASR}, t_{RAH}, t_{RAS} y t_{RC}, que será el tiempo de ciclo de refresco (para una sola página). Se deja como ejercicio realizar el cronograma del ciclo de refresco.

Existen otros tipos de ciclos que no se van a mostrar al ser más dependientes de la DRAM concreta. Estos ciclos lecturas y escrituras suministrando sólo una parte de la

dirección y el acceso en modo página. Por regla general los computadores suelen tener como memoria principal memoria DRAM. El refresco y el acceso a memoria lo controla un circuito especial controlador de memoria que hace de interfase entre la memoria y el bus del sistema.

Las mejoras en las DRAMs han sido siempre históricamente relacionadas con la mejora del proceso litográfico de fabricación. Sin embargo, recientemente se produjo un cambio en la arquitectura de las DRAM que mejoraba la rapidez de los accesos en modo página mediante un cauce segmentado de dos etapas. Esta evolución es la que originó lo que se conoce como **EDO DRAM** o *Extended Data Output* DRAMs.

Análogamente a las SRAM síncronas se han desarrollado, y están teniendo una gran introducción en el mercado de ordenadores personales, las **SDRAM** o *Synchronous Dynamic Random Access Memory*. Estas memorias DRAM síncronas, al igual que las SRAM síncronas, permiten las transacciones en modo ráfaga y están optimizadas para combinar los modos de acceso a páginas con el modo ráfaga.

10.4.4 Memoria FLASH

Estructura interna

Las memorias Flash son memorias de lectura/escritura de acceso aleatorio no volátiles. Su comportamiento para lectura es exactamente igual al de una SRAM, pero para escritura es diferente, deben ser primero borradas y después escritas. Además la especial configuración interna de las Flash hace que deban ser borradas completamente o al menos el sector adecuado para escribir aunque sólo sea un byte.

La celda Flash se basa en el transistor FLOTOX (FLOating gate Tunnel OXide transistor) de oxido fino, tal como se ha descrito en la sección 9.2.3.

Una Flash tiene un conjunto de pines muy similar a una SRAM, con las señales $R/\overline{W}$, $\overline{OE}$, bus de direcciones y bus de datos. La diferencia estriba en la configuración interna. Una Flash tiene internamente un registro de instrucción y una máquina de estados que genera las señales de control internas necesarias para borrar/escribir en un bloque o toda la memoria.

Los tiempos de acceso para lectura son mayores que las SRAM más rápidas y suelen ser del orden de de 70 ns (a fecha de escritura de estos apuntes). El borrado de un sector (o de la memoria entera) en una Flash se puede realizar y comprobar en un tiempo de 1 segundo.

Estas memorias están desplazando a las EEPROM y EPROM como almacenamiento de programas de arranque reprogramables debido a que no necesitan alimentación de +12V para reprogramarlas, al contrario que las EEPROM. Sólo con la alimentación de +5V se puede realizar la reprogramación de estas memorias, con lo que se pueden soldar en un PCB como ROM del sistema y reprogramarlas sin extraerlas.

Modos de funcionamiento

Las memorias FLASH incorporan un registro especial que indica el modo de funcionamiento. Este registro puede estar en el espacio de direcciones de la memoria o puede accederse a él con señales especiales. En cualquier caso se debe especificar el tipo

de ciclo que se va a realizar escribiendo en este **registro de instrucción**. Una vez configurado esto empieza la lectura o programación de la memoria.

El ciclo de lectura es idéntico al de una SRAM descrito en la figura 10.2. Para escribir externamente el funcionamiento será muy similar al de una SRAM una vez esté borrado el sector al que pertenezca el byte a escribir. Internamente una vez se ha capturado la dirección y el dato a escribir un algoritmo interno se encarga de realizar la escritura en la dirección.

Para iniciar un ciclo de borrado debe escribirse en el registro de instrucción interno de la Flash que se va a iniciar un ciclo de borrado. Esto se realiza escribiendo en unas posiciones de la memoria ciertos valores concretos que corresponderán al modo de borrado (total o de un sector) y al número del sector.

10.4.5 Memorias ROM

Son las siglas de *Read Only Memory*. Son memorias no volátiles de acceso aleatorio y de sólo lectura. Una vez han sido escritas o programadas sólo se puede leer el contenido de las celdas. Se suelen utilizar para almacenar el código que permite arrancar a los sistemas una vez se suministra la alimentación o las rutinas del sistema. Las ROM se fabrican para aplicaciones masivas con máscaras de silicio. Si la aplicación es un prototipo o la producción va a ser baja no es viable la inversión en mascaras de silicio para la fabricación de las ROM. Afortunadamente hay tres tipos de ROM que pueden ser programadas en el laboratorio, algunas de ellas incluso pueden ser borradas.

PROM

Las PROM o *Programmable Read Only Memories* son memorias ROM programables eléctricamente mediante un programador especial, no volátiles y de sólo escritura.

EPROM

Las EPROM o *Erasable Programmable Read Only Memories* se programan también con un dispositivo de programación especial conectado a un ordenador. La diferencia con las PROM es que las EPROM sí que se pueden borrar El borrado se realiza mediante rayos UV. Para ello las EPROM tienen un pequeña ventana de cuarzo transparente a los UV mediante la cual se realiza la exposición de la matriz de celdas. Una vez están programadas deben de cubrirse con una etiqueta para evitar el borrado accidental de la memoria. Estas celdas se basan en el transistor FAMOS (Floating gate Avalanche-injection MOS), o transistor de puerta flotante, transistor cuyo funcionamiento ha sido descrito en la sección 9.2.1. El ciclo de lectura es idéntico al mostrado en la figura 10.3.

EEPROM

Las EEPROM o *Electrically Erasable Programmable Read Only Memories* son memorias programables y borrables mediante un dispositivo especial que se conectará a un ordenador. Este programador/borrador de EEPROMs genera señales de +12v para la programación y borrado de las EEPROM. La celda EEPROM se basa en el transistor FLOTOX (FLOating gate Tunnel OXide transistor), transistor cuyo funcionamiento ha

sido descrito en la sección 9.2.2. El ciclo de lectura es idéntico al mostrado en la figura 10.3.

10.4.6 Memorias NVRAM

Las NVRAM o *Non Volatile RAM* son memorias de acceso aleatorio que, como su propio nombre indica, son idénticas en funcionamiento a las SRAM salvo con que una vez se desconecta la alimentación no se pierden los datos. Esta no-volatilidad se puede deber a dos aproximaciones distintas: o hay una pequeña batería integrada en el chip que automáticamente suministra la alimentación a la RAM cuando la tensión externa baja de un cierto umbral, o bien se realiza un trasvase a celdas EEPROM al perder la alimentación.

Las NVRAM del primer tipo incluyen una circuitería que recarga la batería cuando la alimentación externa vuelve al sistema. Aplicaciones típicas de las NVRAM eran la BIOS en ordenadores con una cierta antigüedad. La información de una NVRAM sin alimentación externa puede durar varios años.

Los ciclos de lectura y escritura son idénticos a los de las SRAM, siendo totalmente transparente para el usuario el hecho de que disponga de una batería.

10.4.7 Memorias especializadas

Además de las memorias de propósito general descritas hasta aquí, existen dispositivos de memoria con una funcionalidad especial o con arquitecturas especializada. Estos circuitos son útiles para ciertas funciones específicas, ayudando a simplificar la lógica de control que sería necesaria si se utilizasen memorias normales [Int92].

Memorias FIFO

Estas memorias implementan la funcionalidad de una cola FIFO (*First In First Out*) directamente por hardware. El acceso a los elementos de memoria no es aleatorio, al contrario de los circuitos de memoria vistos hasta ahora.

Las FIFO de tipo SRAM incorporan una matriz de celdas de memoria SRAM, un puntero que indica la posición de la celda a escribir, un puntero que indica la posición de la celda a leer, *flags* o banderas que indicarán si la cola está llena o vacía, y lógica de control. El puerto de datos suele estar separado para lectura y escritura, de esta manera es posible leer y escribir simultáneamente (aunque esto dependerá de la memoria concreta).

Memorias multipuerto

Los módulos multipuerto tienen la particularidad de que tienen más de un puerto de datos y direcciones, es decir, el acceso de lectura y escritura a los datos (que se almacenan en una sola matriz) se puede realizar de forma independiente desde diferentes puertos.

Adicionalmente existe una lógica que controla las colisiones que se pueden producir. Es decir, se evita que se escriba simultáneamente en una misma dirección desde

diferentes puertos.

Las memorias más habituales son las de doble puerto. También existen de más puertos, pero el problema en estos casos es el encapsulado que debe de ubicar un número muy grande de pines.

Una utilidad de las memorias de doble puerto es su utilización en cauces de datos segmentados. Se puede realizar escritura y lectura en la misma memoria, aunque no en la misma dirección, simultáneamente. Esto evita contenciones de bus y acelera el proceso de transmisión de datos.

10.5 Conclusiones

Se puede realizar una clasificación de las memorias en función de diversos parámetros: Volatilidad, tipo de acceso, capacidad de integración y velocidad de acceso.

El acceso aleatorio es la mayoría de veces la prestación más interesante para un circuito de memoria que va a ser integrado en un circuito digital. La idealidad de las SRAM (alta velocidad y acceso aleatorio) viene contrarrestada por su alto precio y no muy alta escala de integración comparada con las DRAM. Por contra las RAM dinámicas tienen una alta escala de integración y una gran capacidad que se ve contrarrestada por ser más lentas que las SRAM, y por tener que intercalarse ciclos de refresco.

Como opciones interesantes a las SRAM y las DRAM están sus variantes síncronas, que permiten el acceso en modo ráfaga. Esto hace que, para diseños con procesadores que incluyan este modo de acceso, optimicen el flujo de información a través del bus.

Puede ser interesante la no-volatilidad de las memorias que almacenen datos de configuración del sistema. Para ello se pueden utilizar memorias basadas en los diversos tipo de tecnologías que permiten que se mantengan los datos después de un corte de alimentación. La tecnología EPROM y EEPROM permite el diseño de las ROM no-volátiles con los mismos nombres. La desventaja es que si se quiere cambiar su contenido debe extraerse el circuito y utilizar un programador especial. Las NVRAM de batería presentan el problema de la caducidad de la misma, y las basadas en celdas EEPROM su poca integración y alto precio. Si se busca no-volatilidad la memorias FLASH tienen la ventaja de que son reprogramables en el sistema y que el proceso de programación se va simplificando cada vez más.

Como siempre en un diseño digital habrá que garantizar las especificaciones demandadas, llegando a un compromiso que equilibre las prestaciones con el coste del sistema. En este tema se han descrito los diversos tipos de circuitos de memoria existentes en la actualidad, así como sus principales características. Una vez tomada la decisión del tipo de memoria a utilizar se impone la búsqueda del fabricante y del modelo, y ahí no sólo el precio es importante. Consideraciones como pedidos mínimos, fechas de extinción o si hay una segunda fuente compatible del circuito pueden ser decisivas.

Apéndice A

Bibliografía Comentada

La bibliografía de Tecnología y Diseño de Sistemas Digitales es muy extensa ya que es un tema tratado e impartido en diversas titulaciones. Los temas abordados en la asignatura (sistemas secuenciales, familias lógicas, VHDL, lógica programable y memorias) son muy amplios y hay una extensa bibliografía específica para cada materia.

Las referencias bibliográficas han sido clasificadas en:

- **Bibliografía básica:** Estos textos han sido seleccionados como básicos porque se consideran útiles para una gran parte de la asignatura. Se han incluido varios textos que tienen una gran coincidencia con el temario de la asignatura. Se ha reducido esta lista intencionadamente a pocos textos para centrar la atención del estudiante en ellos.
- **Bibliografía complementaria:** Se han incluido en esta sección los textos que se consideran especialmente útiles para algunos temas concretos, o los libros de problemas.
- **Manuales y catálogos:** Ha sido necesario recurrir a catálogos y manuales para completar los temas de tecnologías digitales y de lógica programable.
- **Otras referencias:** Además de las fuentes bibliográficas escritas se ha de hacer referencia a catálogos, manuales e información disponible por Internet. La red es ya el medio de difusión habitual de muchos fabricantes que no realizan copias impresas de sus catálogos de dispositivos lógicos.

A la pregunta de si existe un *libro de la asignatura* cabe hacer notar que no hay un texto que cubra exactamente el 100% de la materia, aunque algunos lo hagan en un porcentaje bastante alto. Se puede considerar la tercera edición del Wakerly [Wak00] (texto que se imparte en la Universidad de Stanford) como el más utilizado y que cubre un mayor porcentaje de la asignatura.

En **negrita** se muestran las referencias a los temas de la asignatura y en texto normal las referencias a los capítulos de los textos comentados.

A.1 Bibliografía básica

[**Wak01**] **Diseño digital. Principios y Prácticas**. *John F. Wakerly.* Edita Prentice-Hall, 2001.

Este texto es un clásico y en su nueva edición se cubre de manera casi completa la asignatura. Se presenta una muy buena aproximación desde el punto de vista teórico y práctico al diseño de sistemas digitales, combinando los aspectos lógicos con los tecnológicos. En el capítulo 2 se tratan los sistemas y códigos numéricos. En el capítulo 3 se describe la lógica TTL, familias TTL avanzadas, ECL y CMOS. Este extenso capítulo es útil para los **temas de tecnología digital 6, 7, 7, y 8.**

El capítulo 4 hace referencia a los principios de diseño de lógica combinacional y se introduce el VHDL de forma natural para describir circuitos digitales. Éste se completa con las prácticas de lógica combinacional propuestas en el capítulo 5. En los capítulos 7 y 8 se describen los principios de lógica secuencial, el análisis y síntesis de sistemas secuenciales con biestables y módulos MSI. Estos dos capítulos son útiles para los **temas 1, 2, 3 y 5.**

En el capítulo 10 se abordan los dispositivos lógicos programables y las memorias, tal como se hace en los **temas 9 y 10**. Finalmente se realiza una aproximación al CAD y a otros temas más avanzados en el capítulo 8 (diseño para la verificabilidad, fiabilidad y líneas de transmisión).

Es un texto muy recomendable para toda la asignatura ya que combina con total naturalidad los aspectos tecnológicos y los lógicos. La lógica Booleana se describe con gran profusión de ejemplos y problemas, sin descuidar aspectos más formales. Las consideraciones tecnológicas de las familias lógicas, la lógica programable y las memorias son válidas y con un enfoque muy práctico, incluyéndose referencias a las tecnologías más modernas.

[**PB03**] **VHDL: Lenguaje para síntesis y modelado de circuitos. Segunda Edición**. *Fernando Pardo y Jose A. Boluda* . Edita RA-MA, 2003.

El libro se compone de 12 capítulos, 3 apéndices y un CD-ROM, donde se recoge: La metodología y posibilidades en la descripción del diseño electrónico, introducción y sintaxis del lenguaje, estilos de descripción (estructural, flujo de datos y algorítmica), bibliotecas, paquetes y unidades, conceptos avanzados simulación y modelado, descripción y síntesis automática de circuitos con VHDL, ejemplos, ejercicios resueltos y herramientas de CAD. El texto es muy recomendable para los **temas 4** y **9**, donde se introduce el leguaje de descripción VHDL y la lógica programable.

[**GA95**] **Diseño lógico**. *Pedro-Joaquín Gil y José Albadalejo.* Servicio de publicaciones de la Universidad Politécnica de Valencia, 1995.

Este texto tiene un temario que en la parte referente a los sistemas secuenciales, las memorias y la lógica programable, puede servir como guía que aglutina todos estos contenidos.

En el capítulo 1 se hace una revisión de la lógica combinacional, introduciéndose

en el capítulo 2 el formalismo de los sistemas secuenciales. Cabe hacer notar que el capítulo 3 es una muy buena referencia para la síntesis de sistemas secuenciales asíncronos. En el capítulo 4 se construyen y describen los biestables y en el capítulo 5 se plantea el diseño de sistemas secuenciales síncronos con biestables. Los módulos MSI combinacionales se introducen en el capítulo 6 y el capítulo 7 se dedica a las memorias semiconductoras. El capítulo 8 se presenta la lógica programable y su programación con lenguajes de alto nivel como PALASM para PALS y AHDL para CPLDs. Sin embargo no hay referencias a las FPGAs ni a los dispositivos con arquitecturas híbridas.

Este texto es recomendable para cualquier contenido de la parte de los sistemas secuenciales. Especialmente para el formalismo de los sistemas secuenciales descrita en el **1** de la asignatura, y la síntesis de sistemas secuenciales del **3**. El tema de lógica programable es interesante por el enfoque de programación de las PALs y CPLDs por lo que puede ser útil para complementar el **9**. La arquitectura, tecnologías y modos de utilización de las memorias semiconductoras también tienen un tratamiento útil para el **10**.

A.2 Bibliografía complementaria

A continuación se comentan brevemente otras referencias bibliográficas ordenadas alfabéticamente que no se han incluido en la bibliografía básica, pero que pueden ser interesantes para ampliar temas concretos. También se han incluido varios libros de problemas ya que el enfoque de la asignatura es práctico y se pretende que el estudiante amplíe el boletín de problemas.

[**ABOU02**] **Electrónica Digital. Aplicaciones y problemas con VHDL**. *José Ignacio Artigas, Luís Ángel Barragán, Carlos Orrite y Isidro Urriza.* Edita Prentice Hall, 2002.
Excelente colección de problemas de tecnología y VHDL. Incluye otros temas más alla de la asignatura, pero puede ser útil por sus problemas interconexión de circuitos integrados y los ejemplos de sistemas descritos en VHDL.

[**BBM+97**] **Problemas de circuitos y sistemas digitales**. *Carmen Baena, Manuel Jesús Bellido, Alberto Jesús Molina, María del Pilar Parra y Manuel Valencia.* Edita McGraw-Hill, 1997.
Completa colección de problemas de lógica digital. Cada tema incluye una pequeña introducción teórica, problemas resueltos y problemas propuestos. En este libro se pueden encontrar problemas de análisis y síntesis de sistemas secuenciales, además de interesantes problemas de máquinas ASM.

[**CGT93**] **Tecnologías digitales. De la teoría a la práctica**. *P. Casanova, N. García y J. A. Torres.* Edita Paraninfo, 1993.

En este libro se analizan, con un nivel de detalle que puede ser excesivo, todas las familias lógicas presentadas en la asignatura salvo las lógicas BiCMOS y de bajo voltaje. Carece de referencias a la lógica Booleana y se centra exclusivamente en el

análisis a bajo nivel de los circuitos electrónicos que constituyen las familias lógicas. El texto compara los datos del fabricante con datos experimentales obtenidos por los autores. Al final presenta unos capítulos muy interesantes de comparación e interconexión de familias lógicas.

[**Flo97**] **Fundamentos de sistemas digitales**. *T. L. Floyd.* Edita Prentice Hall, 1997.
Libro básicamente orientado a la lógica combinacional, por lo que es recomendable para la asignatura de primer curso **Fundamentos de los Computadores**. Este libro también describe los biestables básicos, pero no aborda el formalismo de los sistemas secuenciales, ni para análisis ni para síntesis. Un aspecto interesante a tener en cuenta de este libro es la descripción comportamental de sistemas secuenciales con ABEL para PLDs, sin entrar en la estructura o el diseño canónico.

[**Gaj97**] **Principios de diseño digital**. *Daniel D. Gajski.* Edita Prentice Hall, 1997.
El capítulo 6 de este texto describe los biestables, y la metodología de análisis y síntesis de una forma resumida muy eficiente, mientras el capítulo 7 aborda los registros y contadores. La más interesante de este libro es sin duda el capítulo 8 en el que se describe el diseño de máquinas de estados algorítmicas con una gran profundidad. De hecho se exponen técnicas de segmentación en el diseño de algoritmos secuenciales por grupos. Interesante para ampliar los conceptos del **tema 5**.

[**Hay96**] **Introducción al diseño lógico digital**. *John P. Hayes.* Edita Addison-Wesley Iberoamericana, 1996.
Libro que contiene toda la parte inicial de sistemas secuenciales e indicaciones breves a aspectos tecnológicos. Cada capítulo tiene un interesante resumen final donde se especifican los conceptos básicos que se espera que el estudiante haya adquirido. Este libro es recomendable para realizar ejercicios de síntesis de sistemas secuenciales, como complemento al **tema 3** y para profundizar en el diseño de máquinas algorítmicas o diseño a nivel de registros explicado en el **tema 5**.

[**Kat94**] **Contemporary logic design**. *R. H. Katz.* Edita Benjamin Cummings, 1994.
Texto utilizado en la Universidad de Berkeley como referencia en la enseñanza de diseño lógico. Trata con gran profundidad y rigor el diseño de máquinas de estados finitos, pero carece de referencias a los aspectos tecnológicos. Incluye temas relacionados con la estructura de computadores, y simulación de circuitos digitales.

[**LP96**] **Diseño lógico**. *Antonio Lloris y Alberto Prieto.* Edita McGraw Hill, 1996.
Libro muy completo que trata tanto los aspectos lógicos como los tecnológicos en el diseño digital. Incluye una introducción a los microprocesadores y microcontroladores.

[**Man98**] **Sistemas electrónicos digitales**. *Enrique Mandado.* Edita Marcombo, 1998.
Este libro es un clásico en esta materia, escrito además en castellano y con muchas ediciones. Su contenido es muy extenso, cubriendo tanto los aspectos lógicos como los tecnológicos en el diseño digital. Sin embargo no trata el tema de lógica programable.

[**Mar96**] **Circuitos TTL digitales**. *R. M. Marston.* Editorial Paraninfo, 1996.
Este libro se centra en la descripción de las familias TTL estándar desde el punto

de vista electrónico y desde el punto de vista funcional. Incorpora una amplia descripción de los dispositivos más útiles y populares, además de incorporar muchas tablas de características.

[**Mar97**] **Circuitos CMOS**. *R. M. Marston*. Editorial Paraninfo, 1997.
En este otro libro el autor se centra en la descripción de las familias CMOS desde el punto de vista funcional. Incorpora una amplia descripción de los dispositivos CMOS más útiles y populares, además de proponer diseños prácticos con éstos.

[**MK98**] **Fundamentos de diseño lógico y computadoras**. *M. Morris Mano y Charles R. Kime*. Edita Prentice-Hall Hispanoamericana, 1998.
Este texto tiene un capítulo de sistemas secuenciales con poca profundidad, pero incorpora un interesante capítulo de contadores y registros, un capítulo de diseño a nivel RTL y una comparación entre la familias lógicas MAX7000, ACT3 y XC4000. Puede ser útil para revisar algunos de estos conceptos.

[**Oje94**] **Problemas de electrónica digital**. *Francisco Ojeda Cherta*. Editorial Paraninfo, 1994.
En este libro es interesante el último capítulo en el que se plantea el diseño de contadores y registros de desplazamiento a partir de biestables. Los problemas están resueltos y no presentan una gran dificultad.

[**Pri96**] **Semiconductor memories. A handbook of design, manufacture and application**. *Betty Prince*. Edita John Wiley & Sons, 1996.
Este libro es una excelente referencia para el estudio de las memorias semiconductoras desde el punto de vista de la evolución y la arquitectura interna.

[**Sca97**] **VHDL and AHDL Digital System implementation**. *Kevin Skahill*. Edita Prentice Hall, 1997.
Muchos ejemplos muy orientados a la implementación práctica con la herramienta de ALTERA Max+PlusII. Hace un especial énfasis en VHDL para síntesis. El AHDL es un lenguaje particular de ALTERA para síntesis.

[**Ska96**] **VHDL for Programmable Logic**. *Kevin Skahill*. Edita Addison-Wesley, 1996.
Este libro tiene un capítulo inicial en el que se describen las diversas tecnologías de programación y las arquitecturas de los dispositivos programables, realizando un análisis de los compromisos entre ambas características.

[**Tav94**] **Circuitos lógicos programables**. *Christian Tavernier*. Editorial Paraninfo, 1994.
Obra interesante totalmente dedicada a la lógica programable. Lamentablemente se analizan dispositivos que en la actualidad son totalmente obsoletos y la clasificación de éstos se centra en las tecnologías más que en las arquitecturas. De todas maneras es una obra muy válida para revisar los conceptos genéricos sobre las PALs, CPLDs y FPGAs. Se incluyen descripciones en lenguaje PALASM.

[**Toc93**] **Sistemas digitales. Principios y aplicaciones**. *Ronald J. Tocci*. Edita Prentice-Hall Hispanoamericana, 1993.
Este texto es recomendable para ampliar el diseño lógico con las referencias a contadores asíncronos. También tiene un enfoque con la profundidad adecuada para

los temas de parámetros y tecnología de las familias lógicas. Por último es una aproximación válida para ela parte final, pero los contenidos en cuanto a lógica programable son algo pobres y obsoletos.

[VO95] **Problemas de sistemas electrónicos digitales**. *Joaquín Velasco y José Otero*. Editorial Paraninfo, 1995.
Interesante libro con problemas de análisis y síntesis de sistemas secuenciales realizados con módulos estándar. Todos los problemas están resueltos y explicados con detalle. Incluye problemas realizados con módulos estándar MSI comerciales.

A.3 Otras referencias bibliográficas

En esta sección se incluyen los manuales, catálogos y referencias en la red que han servido para completar la información de la bibliografía básica y adicional. Muchos de estos catálogos son la referencia de la última versión impresa distribuida por el fabricante. Las nuevas versiones y actualizaciones se pueden encontrar en la página de *web* del fabricante.

[Adv96] **PAL Devices Data Book and Design Guide**. Edita *Advanced Micro Devices Inc.*, 1996.
Excelente manual que describe la arquitectura y características de las PALs fabricadas por AMD (muchas de ellas consideradas como estándar). Incluye un apéndice con un resumen básico de los principales conceptos de la lógica Booleana.

[Alt94] **FLEX 8000 Handbook**. Edita *Altera Corporation*, 1994.
Referencia básica de las familias FLEX de Altera y de la nueva filosofía y arquitectura de estos dispositivos programables. Útil como referencia a las arquitecturas híbridas.

[Alt95] **Altera Data Book**. Edita *Altera Corporation*, 1995.
Descripción de todos los dispositivos programables de Altera. Incluye una interesante introducción al mercado de PLDs y criterios para seleccionar dispositivos (evidentemente sólo de Altera).

[Int92] **Specialized Memories and Modules**. Edita *Integrated Device Technologies Inc.*, 1992
Se describe con detalle la arquitectura genérica de las memorias multipuerto y FIFO.

[Tex98] **Texas Instruments Data Books**. Edita *Texas Instruments*, 1995-98.
Completos manuales de todas las familias lógicas de TI. Todas las hojas de especificaciones están en el WEB, con notas de aplicación. El rango va desde las familias clásicas TTL hasta las más avanzadas BiCMOS y de bajo voltaje.

[Xil94] **The Programmable Logic Data Book**. Edita *Xilinx Inc.*, 1994.
Descripción de todos los dispositivos programables de Xilinx. Incluye un capítulo inicial en el que se describe de manera exhaustiva las ventajas de las FPGAs. Inte-

resante referencia para profundizar en la arquitectura FPGA clásica.

- **Actel:** Fabricante de FPGAs y software de programación de FPGAs: *http://www.actel.com.*

- **Altera:** Fabricante de lógica programable (CPLDs y dispositivos CPLDs avanzadas) con una gran cantidad de información sobre lógica programable. Dispone de manera adicional de software de distribución gratuita de síntesis de PLDs (MAX + PLUS II versión educacional): *http://www.altera.com.*

- **Amazon:** Ejemplo de librería digital donde se pueden encontrar y adquirir cientos de referencias sobre electrónica digital: *http://www.amazon.com.*

- **AMD:** Fabricante de, entre otras cosas, memorias no volátiles y PALS. En su servidor de web se puede encontrar abundante información sobre PALS y el lenguajes PALASM: *http://www.amd.com.*

- **Chip Directory:** *WEB site* que se puede considerar como referencia de los fabricantes de circuitos digitales. Se pueden encontrar enlaces ordenados por categorías y alfabéticamente: *http://www.chipdir.com.*

- **Cypress:** Fabricante de circuitos integrados, entre otros de lógica programable y memorias SRAM y FIFOs. Distribuye el software de programación de PLDs WARP y WARP2: *http://www.cypress.com.*

- **Fairchild:** Fabricante de circuitos integrados analógicos y digitales, entre ellos lógica discreta y memorias. *http://www.fairchildsemi.com.*

- **Integrated Devices Technology:** Fabricante de, entre otros circuitos integrados, módulos de memoria y memorias especializadas. *http://www.idt.com.*

- **Philips Semiconductors:** División de Philips que fabrica lógica de propósito general, PLDs y procesadores especializados. *http://www-us.semiconductors.philips.com.*

- **Texas Instruments:** Uno de los mayores fabricantes de lógica discreta y dispositivos programables: *http://www.ti.com.*

- **Xilinx:** Fabricante de FPGAs y de dispositivos lógicos programables de muy alta escala de integración: *http://www.xilinx.com.*

Bibliografía

[ABOU02] J. I. Artigas, L. A. Barragán, C. Orrite, y I. Urriza. *Electrñonica Digital.* Editorial Prentice Hall, 2002.

[Adv96] Advanced Micro Devices Inc. *PAL Devices Data Book and Design Guide*, 1996.

[Alt94] Altera Corporation. *FLEX 8000 Handbook*, 1994.

[Alt95] Altera Corporation. *Altera Data Book*, 1995.

[BBM+97] Carmen Baena, Manuel Jesús Bellido, Alberto Jesús Molina, María del Pilar Parra, y Manuel Valencia. *Problemas de circuitos y sistemas digitales.* McGraw-Hill, 1997.

[CGT93] P. Casanova, N. Garcia, y J. A. Torres. *Tecnologías digitales. De la teoría a la práctica.* Editorial Paraninfo, 1993.

[Flo97] T. L. Floyd. *Fundamentos de sistemas digitales.* Prentice Hall, 1997.

[GA95] Pedro-Joaquín Gil y José Albadalejo. *Diseño Lógico.* Colleción: Libro-Apunte. Servicio de publicaciones de la Universidad Politécnica de Valencia, 1995.

[Gaj97] Daniel D. Gajski. *Principios de diseño digital.* Prentice Hall, 1997.

[Hay96] John P. Hayes. *Introducción al diseño lógico digital.* Addison-Wesley Iberoamericana, 1996.

[Int92] Integrated Device Technologies Inc. *Specialized Memories and Modules*, 1992.

[Kat94] R. H. Katz. *Contemporary logic design.* Benjamin cummings, 1994.

[LP96] Antonio Lloris y Alberto Prieto. *Diseño lógico.* McGraw Hill, 1996.

[Man98] Enrique Mandado. *Sistemas electrónicos digitales.* Marcombo, 1998.

[Mar96] R. M. Marston. *Circuitos TTL digitales.* Editorial Paraninfo, 1996.

[Mar97] R. M. Marston. *Circuitos CMOS.* Editorial Paraninfo, 1997.

[MK98] M. Morris Mano y Charles R. Kime. *Fundamentos de diseño lógico y computadoras.* Prentice-Hall Hispanoamericana, 1998.

[NNCI96] V. P. Nelson, H. T. Nagle, B. D. Carroll, y J. D. Irwin. *Análisis y diseño de circuitos lógicos digitales.* Prentice-Hall Hispanoamericana, 1996.

[Oje94] Francisco Ojeda. *Problemas de electrónica digital.* Editorial Paraninfo, 1994.

[PB99] Fernando Pardo y Jose A. Boluda. *VHDL: Lenguaje para síntesis y modelado de circuitos.* Editorial RA-MA, Marzo 1999.

[PB03] Fernando Pardo y Jose A. Boluda. *VHDL: Lenguaje para síntesis y modelado de circuitos. Segunda Edición.* RAMA, Madrid,, 2003.

[Pri96] Betty Prince. *Semiconductor memories. A handbook of design, manufacture and application.* John Wiley & Sons, 1996.

[Sca97] Frank A. Scarpino. *VHDL and AHDL Digital System implementation.* Prentice Hall, 1997.

[Ska96] Kevin Skahill. *VHDL for Programmable Logic.* Addison Wesley, 1996.

[Tav94] Christian Tavernier. *Circuitos lógicos programables.* Editorial Paraninfo, 1994.

[Tex98] Texas Instruments. *Texas Instruments Data Books*, 1995-98.

[Toc93] Ronald J. Tocci. *Sistemas digitales. Principios y aplicaciones.* Prentice-Hall Hispanoamericana, 1993.

[VO95] Joaquín Velasco y José Otero. *Problemas de sistemas electrónicos digitales.* Editorial Paraninfo, 1995.

[Wak00] John F. Wakerly. *Digital design. Principles & Practices.* Prentice-Hall, 2000.

[Wak01] John F. Wakerly. *Diseño digital. Principios y prácticas.* Prentice-Hall, 2001.

[Xil94] Xilinx Inc. *The Programmable Logic Data Book*, 1994.