El hardware: arquitectura y componentes

José López Vicario
Antoni Morell Pérez

PID_00177261

Universitat Oberta de Catalunya
www.uoc.edu

Índice

Introducción

En este módulo estudiaremos los diferentes elementos de hardware que conforman los sistemas empotrados, partiendo de la hipótesis de que los estudiantes tenéis conocimientos básicos tanto de los circuitos lógicos combinacionales como de los secuenciales.

Si clasificamos los diferentes componentes desde un punto de vista tecnológico, veremos que actualmente existe en el mercado toda una serie de dispositivos que podemos ordenar según el grado de complejidad. En lo alto de la lista, encontraríamos los circuitos integrados a muy gran escala (VLSI, *very large-scale integrated*), los cuales pueden llegar a integrar millones de transistores en un solo chip. Los circuitos VLSI aportan un grado significativo de funcionalidad y es donde encontramos los microprocesadores, microcontroladores, matrices de puertas programables in situ (FPGA, *field programmable gate arrays*), dispositivos lógicos programables (PLD, *programmable logic devices*), circuitos integrados de aplicación específica (ASIC, *application specific integrated circuits*) e, incluso, las memorias. En un nivel por debajo de estos encontramos los circuitos integrados a escala media (MSI, *medium-scale integrated*), que contienen cerca de un millar de transistores y ejecutan tareas más sencillas que los LSI, si bien su funcionalidad continúa siendo completa. Finalmente, encontramos los circuitos integrados a pequeña escala (SSI, *small-scale integrated*) y los componentes simples. Entre todos ellos, se establece comunicación mediante las señales eléctricas, que pueden ser interrumpidas por ruido o por señales no deseadas. A continuación, se desarrolla una visión del hardware presente en los sistemas empotrados siguiendo esta clasificación.

En primer lugar, se describe la arquitectura básica de cualquier sistema empotrado con sus bloques principales, es decir, procesador (CPU, unidad central de procesamiento), memoria, entradas y salidas. Inicialmente, se centra la atención en el núcleo del sistema y se lleva a cabo una descripción de alto nivel. Se continúa con el estudio de los diferentes tipos de memoria existentes y cuál es su uso. En este punto, el discurso tiene en cuenta algunos aspectos de nivel más bajo. Finalmente, hablaremos de otros dispositivos menos complejos pero de gran trascendencia en los sistemas empotrados, como son los registros, los relojes, los contadores y los divisores. En este punto, consideraremos los diferentes efectos que sufren las señales eléctricas para tenerlos en cuenta durante la fase de diseño.

Este módulo finaliza con el estudio de otras partes del hardware, algunas de las cuales están siempre presentes e incluso pueden llegar a ser críticas, como el subsistema de alimentación o la detección de errores en el hardware. En cambio, otras partes, como la comunicación con el exterior por diferentes puertos o por comunicaciones de radio, tendrán más o menos importancia en función

de la aplicación específica. Por ejemplo, en una máquina expendedora solo se utilizarán para tareas de mantenimiento (por ejemplo, actualizar precios) y, en cambio, en una red de sensores inalámbricos las comunicaciones de radio son parte indispensable del sistema. En este último caso vemos la importancia del subsistema de alimentación, puesto que queremos sensores con gran autonomía. En algunos casos, incluso, el sensor se abandona una vez agotada la batería.

Objetivos

El estudio de este módulo didáctico os ayudará a lograr los objetivos siguientes:

1. Conocer los componentes de hardware principales que forman un sistema empotrado.

2. Conocer los diferentes subsistemas de la arquitectura de un sistema empotrado.

3. Conocer los diferentes mecanismos de comunicación en un sistema empotrado.

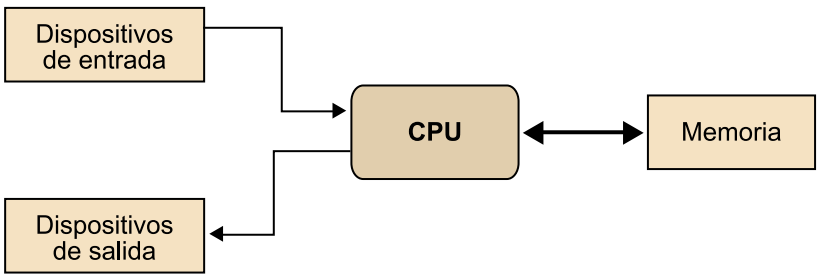
1. Arquitectura del sistema

La arquitectura básica de un sistema empotrado no difiere demasiado de la arquitectura de cualquier computador, en el que figuran cuatro elementos básicos:

- el núcleo o procesador (CPU),
- la memoria principal,
- la entrada del sistema y
- la salida del sistema,

tal como se aprecia en la figura siguiente:

Arquitectura básica de un sistema empotrado



En el momento de ejecución, la CPU recoge de la memoria las instrucciones que tiene que llevar a cabo (el programa) y, junto con otros datos, que pueden ser obtenidos tanto de la misma memoria del sistema como de sus entradas, ejecuta las tareas indicadas y devuelve el resultado a la memoria o a la interfaz de salida.

La CPU es, pues, la parte principal del sistema y todo el diseño y funcionamiento del sistema empotrado gira en torno a esta parte.

2. Componentes de los sistemas empotrados

En este apartado veremos los componentes principales que integran un sistema empotrado.

2.1. Núcleo del sistema

Hay cuatro opciones principales a la hora de elegir la CPU, cada una de las cuales impone pequeñas variaciones en la arquitectura del sistema, y que son:

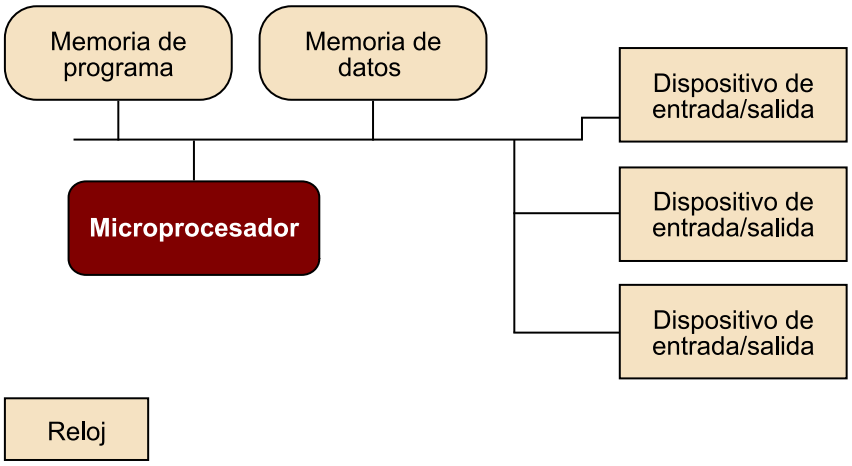
- microprocesador,
- microcomputador,
- microcontrolador y
- procesador de señales digitales (DSP, *digital signal processor*).

2.1.1. El microprocesador

El microprocesador es un dispositivo que integra en un solo chip las partes indispensables de la CPU, es decir, los registros internos, la unidad aritmeticológica (ALU, *arithmetical and logical unit*) y la unidad de control. Esta última se encarga de gestionar la ejecución del programa, es decir, tomar las instrucciones de la memoria, interpretarlas, recoger los datos necesarios para llevarlas a cabo, ejecutarlos, entregar los resultados y pasar a la instrucción siguiente. La ALU, tal como su nombre indica, se ocupa exclusivamente de hacer los cálculos aritméticos y resolver las relaciones lógicas. Finalmente, los registros internos son memorias de alta velocidad y tamaño reducido en las que se guardan, de una manera temporal, los datos que se utilizan habitualmente durante la ejecución, ya sea porque es necesario o con el objetivo de favorecer la velocidad del sistema.

Cuando un sistema empotrado está basado en un microprocesador, su arquitectura típica es la de la figura siguiente, en la que se aprecian varios elementos externos: la **memoria**, que habitualmente se divide en memoria de datos y de programa (microsoftware, *firmware*), los **dispositivos de entrada/salida** y el **reloj** del sistema. La memoria de datos suele ser memoria RAM, mientras que el microsoftware suele estar constituido por memoria ROM y es donde se guarda el conjunto de instrucciones que tiene que ejecutar el sistema. En general, la arquitectura del sistema será de tipo Von Neumann o Harvard, en función de si se permite el acceso simultáneo a las dos memorias o no. Por lo tanto, en una arquitectura Harvard serán necesarios dos buses de información separados.

Arquitectura de un sistema empotrado basado en un microprocesador



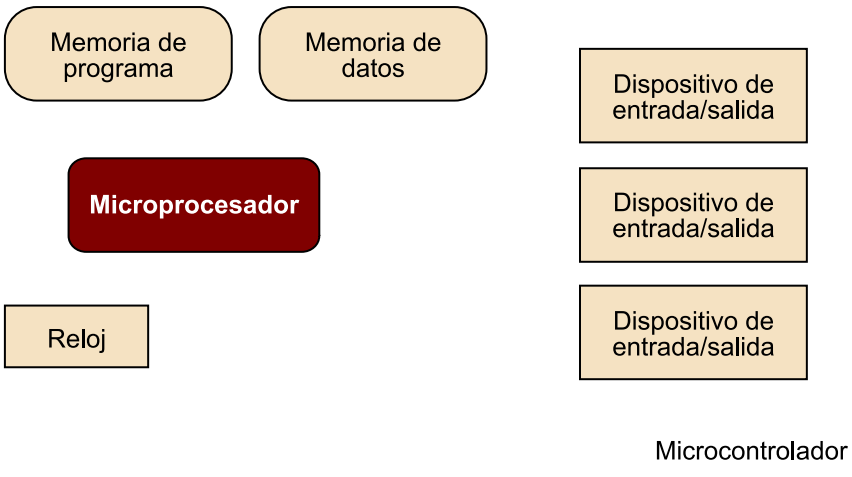
2.1.2. El microcomputador

El microcomputador es una versión más *completa* de microprocesador en el sentido de que integra elementos indispensables en cualquier sistema empotrado y de que, en el caso del microprocesador, son externos a este. Así pues, tenemos en un solo chip el microprocesador, una cierta cantidad de memoria y algunos circuitos de entrada-salida a los que conectar los periféricos.

2.1.3. El microcontrolador

Finalmente, el último nivel de integración está representado por los microcontroladores, dispositivos que pueden llegar a incluir en un único chip todas las funcionalidades necesarias. Entre otras, pueden incluir reloj propio, dispositivos de entrada, convertidores de analógico a digital y viceversa, puertos de comunicación, etc. En el ámbito de los sistemas empotrados, los microcontroladores son especialmente útiles cuando queremos obtener sistemas de bajo coste, especialmente si se han de producir a gran escala. En este caso, la arquitectura del sistema es la que aparece en la figura siguiente:

Arquitectura de un sistema empotrado basado en un microprocesador y con microcontrolador



2.1.4. El procesador de señales digitales (DSP)

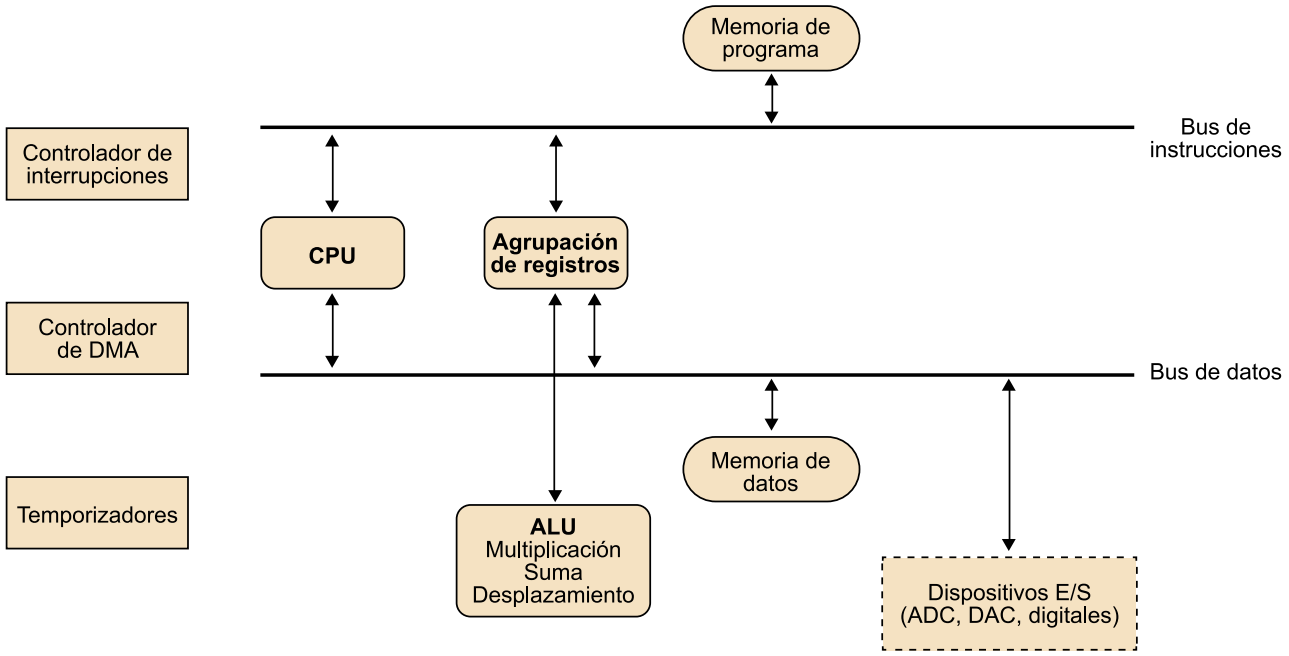
Además de los dispositivos de propósito general que hemos comentado hasta el momento, hay que destacar el DSP como un microprocesador de carácter específico diseñado para ofrecer un alto rendimiento en las aplicaciones que utilizan procesamiento de la señal, como pueden ser el tratamiento de voz, música, vídeo, imágenes, etc.

La diferencia básica entre un DSP y el resto de los procesadores es la capacidad para ejecutar operaciones aritméticas, como la suma, la multiplicación, el desplazamiento de bits o la multiplicación-acumulación en un solo salto de reloj. Este hecho permite, pues, hacer cálculos matriciales o convoluciones con pocos saltos. Además, suelen utilizar aritmética de saturación, es decir, se limitan los resultados de las operaciones entre un valor máximo y mínimo en vez de permitir el paso del valor máximo al mínimo, como sucede en los cuentakilómetros analógicos de los automóviles.

Debido a su especialización, pueden ser usados en solitario o junto con un procesador de carácter general que encarga al DSP las tareas específicas de procesamiento de la señal. Dado que los DSP trabajan muchas veces con señales del mundo real y, por lo tanto, analógicas, es común equiparlos con convertidores analógico-digitales (ADC) y digital-analógicos (DAC), así como con dispositivos digitales de entrada/salida de alta velocidad.

En la figura siguiente se puede apreciar el diagrama de bloques de un DSP, donde se aprecia una arquitectura de tipo Harvard con acceso independiente a datos y a instrucciones, que utiliza aplicaciones de alta velocidad junto con buses de datos de gran ancho de banda (pueden transportar una gran cantidad de bits por segundo).

Arquitectura de un sistema empotrado basado en DSP



2.2. Buses de comunicaciones (direcciones, datos y control)

Tal como se ha podido entrever en los diagramas anteriores, cualquier sistema empotrado, sea cual sea su arquitectura, necesita establecer comunicaciones entre las diferentes partes que lo forman.

Denominamos *buses* a estos canales de comunicación, y habitualmente esta información se transporta en forma de señales eléctricas. En su forma más simple, podemos entender el bus como un conjunto de "hilos" por donde viaja la información y normalmente un solo bus interconecta múltiples dispositivos.

Ejemplo

Un valor de +5 V suele representar un "1" lógico, mientras que un valor de 0 V representa el "0" lógico.

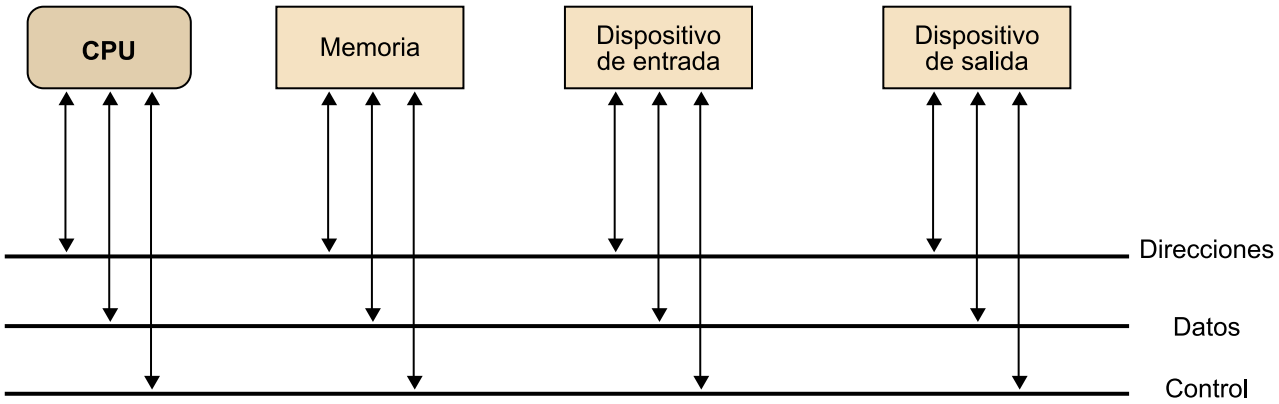
La información que viaja por ellos puede ser de tres tipos: **direcciones**, **datos** y **control**. En primer lugar, la dirección nos indica dónde hay que guardar la información o de dónde se tiene que extraer, ya sea de la memoria del sistema o de los dispositivos entrada-salida. En segundo lugar, los datos contienen la información que estamos procesando. Finalmente, en el bus de control encontramos señales que son necesarias para el funcionamiento correcto de los dispositivos que forman el sistema. Por ejemplo, cuando accedemos a la memoria del sistema para recoger unos datos determinados, aparte de saber la dirección, hay que saber en qué momento están preparados para ser leídas. En el mundo digital, es muy importante remarcar que todo aquello que no tiene dirección no existe, puesto que no hay manera de acceder a ello. Pensemos en cuando borramos un fichero del disco duro de nuestro PC: la información (datos) no se borra físicamente, pero sí que dejamos de recordar la ubicación (dirección) y, por lo tanto, no podemos ver la información aunque permanezca allí.

Ejemplo

En una máquina expendedora de bebidas nos puede interesar saber el número de unidades que quedan de un artículo determinado, que hemos de actualizar en el momento de hacer la reposición y que irá decreciendo cada vez que una bebida es dispensada. Este dato, que estará ubicada en una dirección determinada (posición de memoria), es el que viajará por el bus de datos.

La figura siguiente muestra los diferentes elementos de un sistema empotrado interconectados mediante los buses de direcciones, datos y control. En este caso, el bus de direcciones transporta 12 señales, el bus de datos 16 y el bus de control 4:

Los buses de direcciones, datos y control



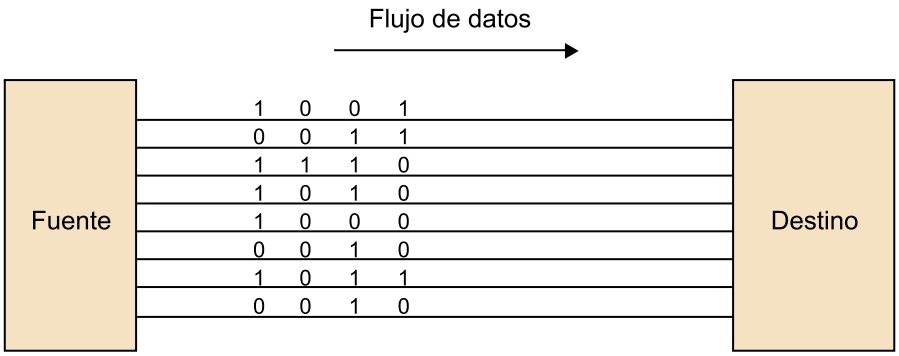
Definimos el ancho del bus como el número de bits que puede transportar simultáneamente y nos da una idea aproximada de la velocidad del bus. Es decir, si debemos transferir 64 bits sobre un bus de ancho de 16 bits, necesitaremos cuatro instantes temporales para completar la transmisión, mientras que si el bus tiene un ancho de 64 bits, con un solo instante tendremos suficiente. Así pues, queda claro que la velocidad del bus en bits por segundo vendrá determinada por el producto de su ancho y la frecuencia máxima de funcionamiento, medida en ciclos por segundo.

Ejemplo

Un bus que opera a 800 MHz y tiene un ancho de 32 bits es capaz de transportar datos a una velocidad de 25,6 GBps.

En la figura siguiente podemos ver el movimiento de datos entre fuente y destino. En este caso, son necesarios ocho instantes temporales para transmitir 32 bits. En el caso particular de que el ancho del bus sea 1, diremos que se trata de un **bus serie**, mientras que en el resto de los casos nos referiremos a él como **bus paralelo**.

Movimiento de una palabra de 32 bits en un bus de 8 bits

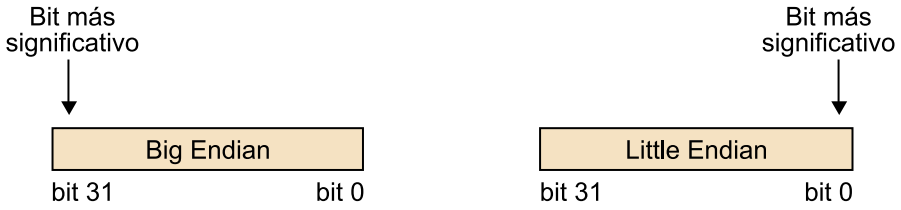


Dado que el sistema trabajará con palabras de *N* bits para describir tanto los datos como las direcciones, hay que establecer un orden en los bits que recibimos o enviamos por medio de los buses. En caso de que la palabra ocupe completamente el bus, hay que establecer un orden espacial entre las líneas

que forman el bus. Si, por el contrario, una palabra requiere varias transmisiones para ser enviada (es seguro el caso de los buses serie), hay que establecer un orden espaciotemporal. Cualquier ordenación que se lleve a cabo es una simple cuestión de convenio y, en general, se usa la notación Big Endian, en la que el bit 0 (de más a la derecha) corresponde al de menos peso (LSB) y el bit con un índice más grande corresponde al de más peso (MSB), o la notación Little Endian, que funciona justo al revés.

La figura siguiente nos muestra ambas notaciones en una palabra de 32 bits:

Notaciones Big Endian y Little Endian



2.3. Direcciones

Ejemplo

Si el tamaño de la dirección es de 32 bits, todas estarán comprendidas entre 00000000 y FFFFFFFF en hexadecimal.

Anteriormente hemos visto que los datos guardados en memoria necesitan una dirección asociada para poder ser identificadas y acceder a ellos. Así mismo, también necesitaremos identificar los dispositivos de entrada/salida para poder interactuar con ellos. Si pensamos en la memoria del sistema, la podemos ver como una agrupación de datos en la que cada posición tiene un índice diferente (dirección) y una capacidad determinada (tamaño máximo en número de bits). En los sistemas empotrados, como en cualquier ordenador, las direcciones se codifican en binario empezando por el 0 y hasta el número más grande que se puede representar. Por lo tanto, podemos decir que no hay direcciones negativas. Esto limita la memoria máxima que el sistema es capaz de soportar, puesto que 32 bits proporcionan 4.294.967.296 identificadores diferentes y, por lo tanto, no seremos capaces de encaminar todas las posiciones de memoria si esta tiene un tamaño más grande. Es decir, si los datos tienen un ancho de 8 bits, no podremos tener memorias más grandes de 4 GB con 32 bits de dirección. Finalmente, hay que remarcar que el direccionamiento de memoria es una cuestión que muchas veces no es transparente al programador del sistema, puesto que si se emplean lenguajes de programación tipo C++, es habitual el uso del operador & para obtener la posición de memoria donde se ubica una variable. Por otro lado, esta flexibilidad requiere actuar con precaución, puesto que el acceso descontrolado sobre la memoria puede suponer errores graves en el funcionamiento del sistema.

2.4. Instrucciones

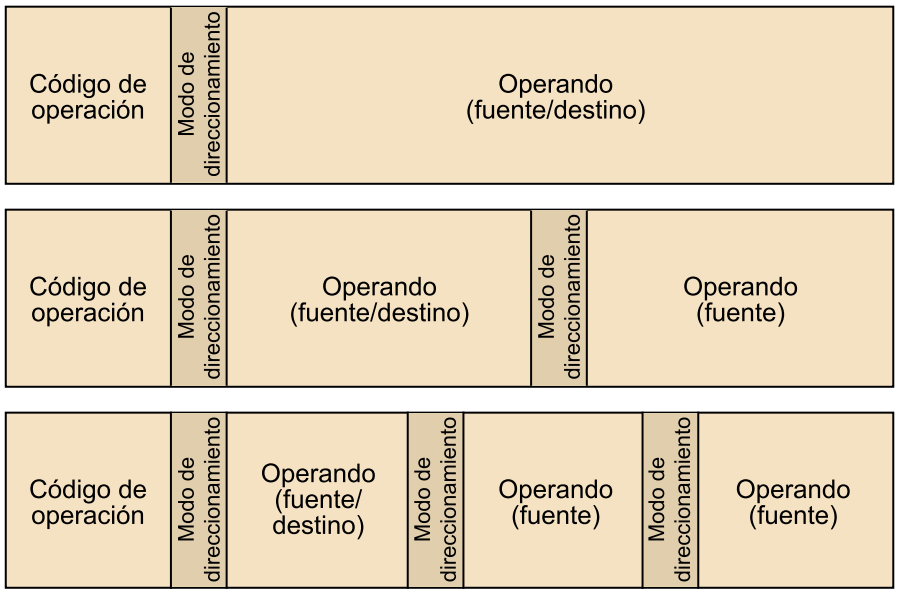
El conjunto de instrucciones de un sistema empotrado conforma el nexo de unión entre su hardware y el software. Así pues, cada instrucción representa una acción que el hardware de nuestro sistema es capaz de llevar a cabo. Nos referiremos a estas acciones como *operaciones* y actúan siempre sobre el operando. El número de operandos que una operación contiene denota su grado, habitualmente comprendido entre 1 y 3.

Ejemplo

Como ejemplo, pensemos en la instrucción $z = x + y$, escrita en C/C++. Vemos que contiene tres operandos: los operandos fuente x e y y el operando destino z. Si miramos la instrucción con detalle, podemos observar que en realidad habrá que llevar a cabo dos operaciones básicas, que son la de suma y la de asignación. Por lo tanto, el procesador hará primero la suma de x e y, la guardará temporalmente y escribirá el resultado en z. En contraposición, la instrucción $x = x + y$ tiene solo dos operandos, pero a la hora de ejecutarla habrá que hacer dos operaciones básicas del mismo modo que en el caso anterior.

A la hora de representar las instrucciones en la memoria de programa y de manera que el procesador sea capaz de interpretarlas, se agrupan los bits de datos en campos. Siempre tiene que haber un campo destinado a describir cuál es la operación que se ha de llevar a cabo y el resto de los campos (habitualmente entre 1 y 3) permite localizar los operandos. En la figura siguiente, podemos ver algunos ejemplos de formato de instrucciones en una máquina que utiliza palabras de 32 bits y en notación Big Endian (en Little Endian los campos irían al revés, de tal manera que el código de operación se ubica siempre en los bits de más peso).

Diferentes formatos de instrucción



2.4.1. Tipos de instrucción

De cara al procesador, cada instrucción queda unívocamente identificada por una combinación determinada de 0 y 1 y, por lo tanto, el número de bits dedicado a esta parte estará determinado por el total de instrucciones que el micro sea capaz de ejecutar. En la fase de programación, sin embargo, no trabajamos directamente con la información binaria, sino que utilizamos lenguajes de diferentes niveles de abstracción. El más básico (de nivel más bajo) es el lenguaje de ensamblador, pero hay una gran cantidad de micros que se programan con lenguajes de alto nivel tipo C/C++. En cualquiera de los casos, necesitaremos un software específico que transforme nuestro código a 0 y 1, es decir, a len-

guaje de máquina. Por otro lado, el conjunto de instrucciones del que dispone un procesador adquiere una gran importancia, puesto que define la interfaz entre el programador y la máquina.

Como nuestro objetivo en estos momentos es poder clasificar el conjunto de instrucciones de un procesador, utilizaremos el lenguaje de ensamblador para mantenernos tan cerca como sea posible del lenguaje de máquina. De hecho, la traducción de ensamblador a código máquina es directa, puesto que simplemente sustituimos los códigos de operación binarios por mnemónicos, que facilitan que el ser humano los interprete. A grandes rasgos, clasificamos el conjunto de instrucciones en tres grandes bloques:

- Instrucciones de transferencia de datos, que nos permiten mover los datos de un lugar a otro.

- Instrucciones de control de flujo, que servirán para controlar el orden de ejecución de las instrucciones que forman un programa.

- Instrucciones relacionadas con la aritmética y la lógica, que nos permitirán hacer las operaciones lógicas y aritméticas que sean necesarias.

Instrucciones de transferencia de datos

Las instrucciones de transferencia de datos contienen tres partes claramente diferenciadas: la **información que hay que mover**, su **localización** o su **origen** y el **lugar donde se quiere guardar** o **destino**. El origen y el destino pueden ser:

- La memoria del sistema.
- Un registro.
- Un puerto de entrada o salida del sistema.

Más adelante hablaremos de los registros, pero de momento diremos que son pequeñas memorias volátiles en las que se almacena tan solo una palabra y sirven para guardar información de carácter temporal, como, por ejemplo, los operandos de una instrucción. Una de sus características más destacadas es que son de acceso rápido, tanto para la lectura como para la escritura.

En la tabla siguiente, podemos encontrar algunas de las instrucciones de transferencia de datos habituales, si bien no todas tienen que aparecer en un mismo conjunto de instrucciones. Por ejemplo, las instrucciones LD y ST no las encontraremos nunca junto con la instrucción MOVE.

Ejemplos de instrucciones de transferencia de datos

Instrucción	Significado
LD destino, origen	*Load*. El operando origen es transferido al de destino, que puede ser un registro o la memoria principal.

Instrucción	Significado
ST origen, destino	*Store*. El operando origen es transferido al de destino. En este caso, el origen es siempre un registro y el destino, la memoria del sistema.
MOV destino, origen	*Move*. Copia el operando origen al destino. Ambos pueden ser memoria o registros.
XCH destino, origen	*Exchange.* Intercambia los operandos origen y destino.
PUSH/POP operando	Apilar el operando en la pila o extraer un dato de la pila y transferirlo al operando.
IN/OUT destino, origen	Transferir datos a un puerto de entrada o salida.

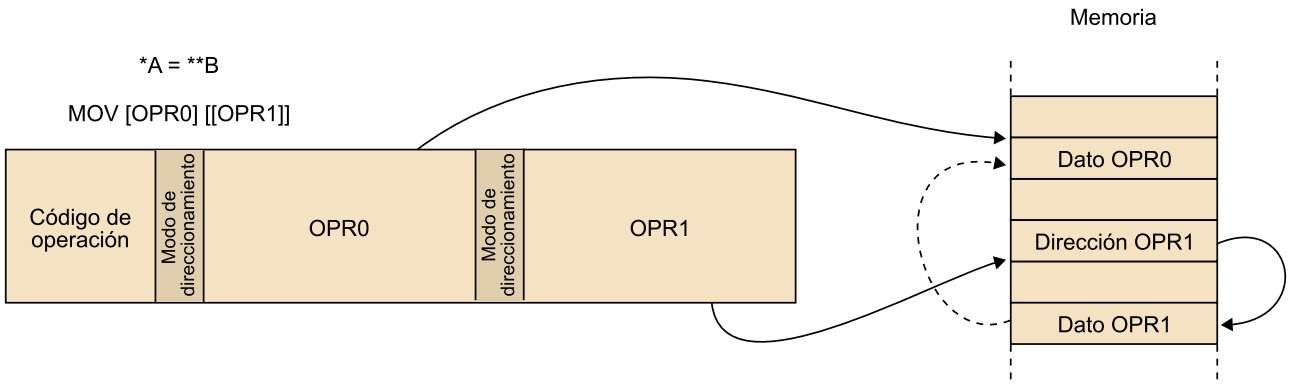
Tal como hemos comentado anteriormente, justo después del código de operación aparecen los campos correspondientes a los operandos. Sin embargo, hay diferentes maneras de obtener estos operandos y, en consecuencia, hay que indicarlo al inicio de cada campo, tal como se muestra en la primera figura del apartado 2.4. Los modos de direccionamiento más comunes son:

- Inmediato.
- Directo e indirecto.
- Directo de registro e indirecto de registro.
- Indexado.
- Relativo al contador de programa.

En el modo de **direccionamiento inmediato**, es la misma instrucción la que contiene el valor del operando y, por lo tanto, es la opción que requiere un número inferior de accesos a memoria. Este modo de direccionamiento puede aparecer en instrucciones de un operando o de dos y funciona bien cuando el valor que hay que transferir es pequeño (en caso contrario, puede no caber en la instrucción), como sucede en la inicialización de variables, o en los índices del lazos de código. En el caso de un solo operando, el destino tiene que estar implícito, como, por ejemplo, el acumulador de la ALU.

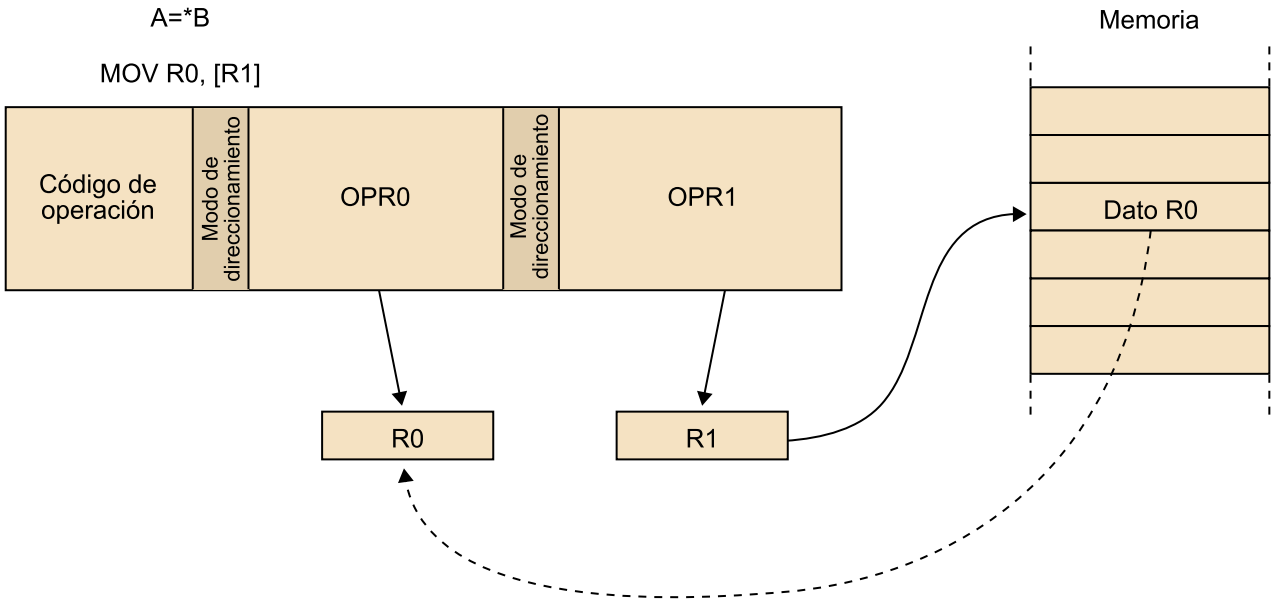
En los modos de **direccionamiento directo e indirecto**, los operandos contienen la dirección en memoria de los datos que queremos transferir. La diferencia entre el modo directo e indirecto radica en el número de niveles de indirección. En el direccionamiento directo hay un solo nivel de indirección y la dirección que contiene el operando es donde se encuentra el dato de interés. En cambio, en el modo indirecto, el operando apunta a una posición de memoria que contiene la dirección donde está el dato. En la figura siguiente, podemos ver un ejemplo de transferencia de datos utilizando direccionamiento directo e indirecto; en él se utiliza el operador de referencia * de C++, que podemos interpretar como "el contenido de", es decir, *A indica el contenido de la posición de memoria que se encuentra en A. En ensamblador, expresaremos [*A*] de una manera equivalente.

Transferencia de datos usando direccionamiento directo e indirecto



En el modo de **direccionamiento directo de registro e indirecto de registro**, el operando especifica uno de los registros del sistema. En el caso de direccionamiento directo de registro, el registro seleccionado directamente contiene el dato de interés, mientras que en el direccionamiento indirecto de registro, este contiene una dirección de memoria a la que habrá que acceder para recuperar o escribir datos. Evidentemente, los modos de direccionamiento directo e indirecto (usando registros o no) requieren más accesos a memoria que el modo inmediato. La ventaja de emplear registros es que tenemos un abanico más amplio de direcciones de memoria donde podemos apuntar. Fijémonos en que el registro siempre dispondrá del tamaño de palabra completa del sistema, mientras que un operando en una instrucción tiene que compartir este mismo tamaño con el código de instrucción y el resto de los operandos. La figura que incluimos a continuación muestra un ejemplo de transferencia de datos usando los modos directo e indirecto por medio de registro.

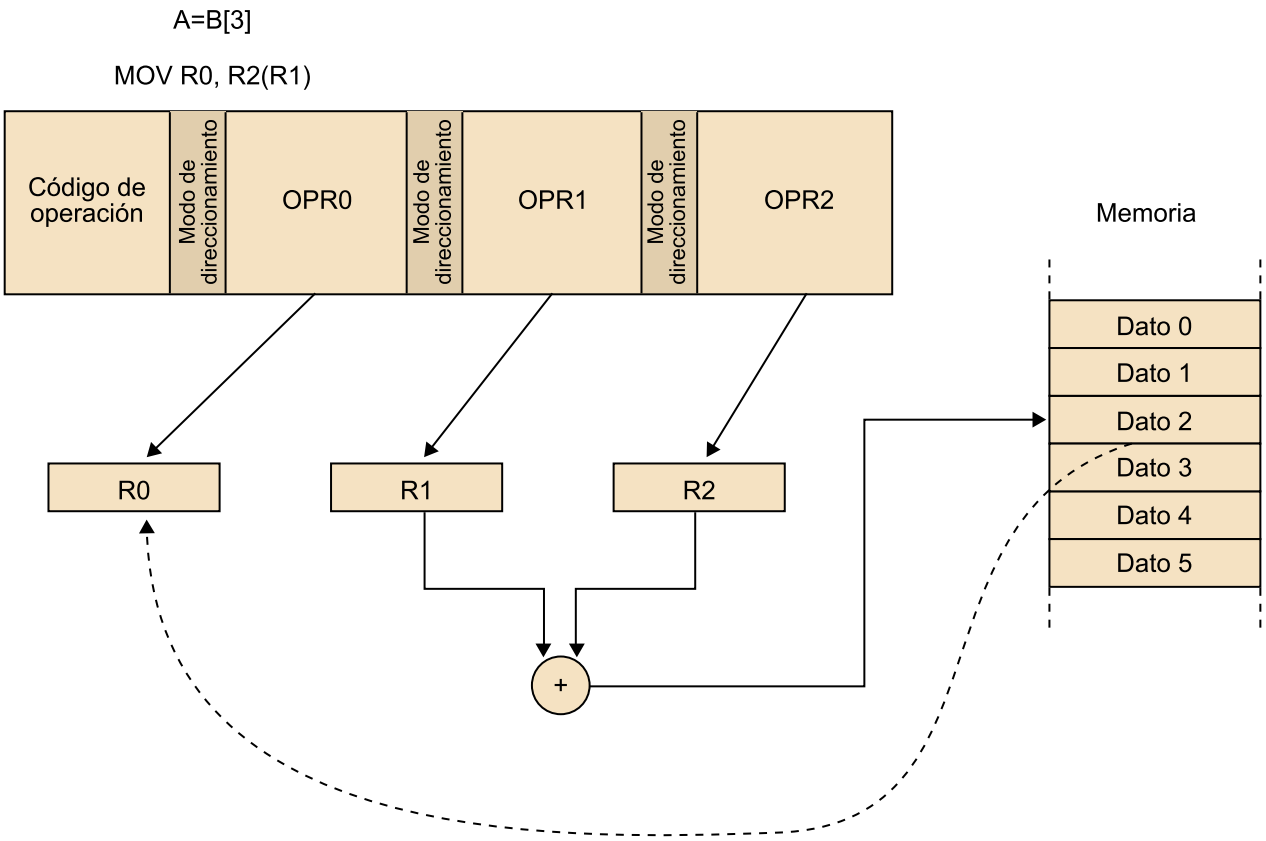
Transferencia de datos usando los modos directo e indirecto de registro



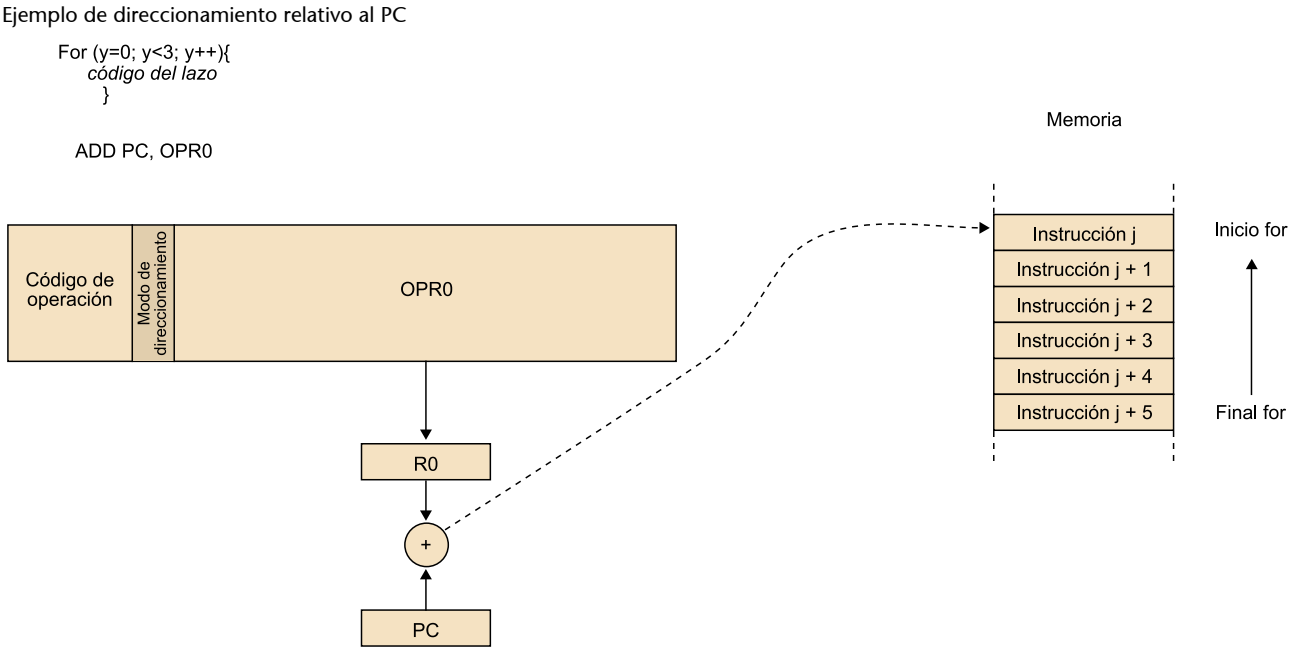
En el modo indexado o de desplazamiento, trabajamos con un conjunto de direcciones base más offset, es decir, la dirección de interés se obtiene sumando un cierto desplazamiento a la dirección base. Se trata, por lo tanto, de un método adecuado para acceder a estructuras de datos de tipo agrupación y la

mayor desventaja que presenta es el tiempo necesario para calcular la dirección deseada y hacer el acceso a memoria. En general, el acceso indexado es más costoso en este sentido que el resto de los modos presentados hasta el momento. Es importante remarcar que, después de la ejecución de la instrucción, ni la dirección base ni la offset cambian. Podemos ver un ejemplo de acceso indexado en la figura siguiente, en la que el operando 2 apunta al registro que contiene el offset, el operando 1, al registro que contiene la dirección base y finalmente el operando 0 apunta al registro donde está la dirección de destino. En la figura siguiente podemos ver tanto la instrucción escrita en ensamblador como el equivalente en C/C++.

Ejemplo de acceso indexado



Finalmente, el modo de direccionamiento relativo al contador de programa presenta una mecánica de funcionamiento parecida al método indexado, si bien hay algunas diferencias significativas entre los dos. En primer lugar, la dirección base es implícita y corresponde siempre al valor actual del registro PC (contador de programa). Lo que sí que se indica es el offset que hay que añadir, que en este caso puede ser tanto positivo como negativo. Finalmente, una vez calculada la nueva dirección, esta sobrescribe la anterior (ubicada en el PC). En la figura siguiente, vemos un ejemplo en el que se utiliza este tipo de direccionamiento para reiniciar un lazo tipo *for* (hay que comprobar que la condición de lazo se cumpla antes de hacer el salto). Del mismo modo que en el caso anterior, el método requiere más tiempo para completar la instrucción que los tres primeros métodos presentados.



Instrucciones de control de flujo

Las instrucciones de control de flujo son las que nos permiten establecer el orden de ejecución de un programa determinado. En este sentido, podemos identificar hasta cuatro maneras de operar diferentes, que son:

- Ejecución secuencial.
- Ejecución ramificada.
- Ejecución en lazo.
- Llamada a una función.

La mayor parte del código de cualquier programa se ejecuta en modo secuencial, es decir, una vez se ha completado una instrucción se pasa a ejecutar la instrucción que se encuentra justo detrás de la primera en memoria. Un ejemplo de ejecución secuencial lo podemos ver en el pequeño fragmento de código de la figura siguiente, escrito tanto en C/C++ como en ensamblador.

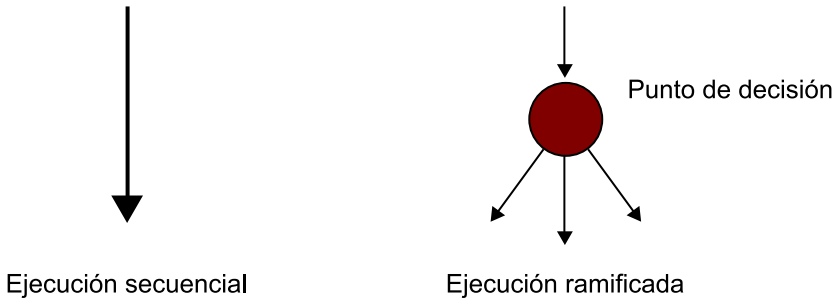
Fragmento de código de ejecución secuencial

```
x = 5;       MOV R1, #5H   // pone 5 en hexadecimal en R1
y = 10;      MOV R2, #AH    // pone 10 en hexadecimal em R2
z = x + y;   ADD R3, R1, R2   // calcula R1 + R2 y lo guarda en R3
```

A diferencia de esta última, la ejecución ramificada es la que se inicia en un punto determinado del código que denominamos **punto de decisión**. A partir de este, se prevén varias posibilidades para seguir la ejecución. Esta es la manera de operar que tienen instrucciones de alto nivel del tipo *if / else, switch* o *case*. Gráficamente, podemos ver en la figura siguiente la representación tanto de la ejecución secuencial como de la ramificada. Habitualmente, hay algún tipo de condición asociada a la instrucción y se utiliza el *flag register* (agrupa

varios bits, en este caso *flags*) para almacenar temporalmente el resultado de la condición. Las condiciones más habituales son de tipo lógico entre variables (si son iguales, una más grande que la otra, etc.) o de tipo aritmético (si el resultado de una operación es cero, o negativo, o provoca un desbordamiento [*overflow*]).

Ejecuciones secuencial y ramificada



En función del resultado obtenido, habrá que ejecutar una parte del código u otra. Algunas de las instrucciones encargadas de hacer estos saltos dentro del programa se pueden ver en la tabla siguiente. Fijaos en que también es posible hacer un salto a un punto de código de manera incondicional. En lenguaje de ensamblador, estos puntos de código se marcan de una manera sencilla utilizando etiquetas.

Ejemplos de instrucciones de salto

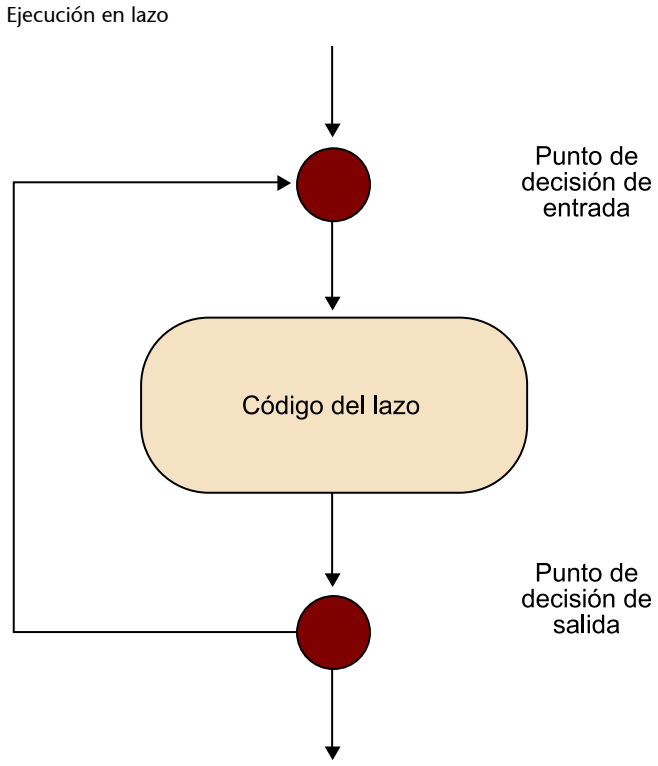
Código de operación	Descripción
BR etiqueta	Salto incondicional al punto marcado por etiqueta.
BE/BNE etiqueta	Salto al punto marcado por etiqueta cuando el indicador (*flag*) de igualdad está activado/desactivado.
BZ/BNZ etiqueta	Salto al punto marcado por etiqueta cuando el indicador de cero está activado/desactivado.
BGT etiqueta	Salto al punto marcado por etiqueta cuando el indicador más grande está activado/desactivado.
BV etiqueta	Salto al punto marcado por etiqueta cuando el indicador de desbordamiento está activado/desactivado.
BC/BNC etiqueta	Salto al punto marcado por etiqueta cuando el indicador *carry* está activado/desactivado.
BN etiqueta	Salto al punto marcado por etiqueta cuando el indicador negativo está activado/desactivado.

En el cuadro siguiente, vemos un ejemplo de un fragmento de código en C/C++ y su versión correspondiente en ensamblador en el que se utilizan las instrucciones de control de flujo. Aquí asumimos que las variables x, y, z, a y b se encuentran ya en los registros R1-R5, respectivamente.

Fragmento de código de ejecución ramificada

```
if (x==y)  CMP R2, R1          // compara x e y (actualiza el flag de igualdad)
  z=a+b;   BE $1               // si son iguales (flag=1) salta a $1
else       SUB R3, R4, R5      // resta a y b poniendo el resultado en z
  z=a-b;   BR $2               // salto incondicional a la etiqueta $2
           $1: ADD R3, R1, R2  // calcula R1+R2 y lo guarda en R3
           $2: ...             // continúa la ejecución del código
```

En la *ejecución en lazo*, del mismo modo que en el caso anterior, podemos identificar un punto de decisión que nos indicará si hay que continuar ejecutando un trozo de código determinado (el mismo código del lazo) o no. La decisión se puede tomar antes o después de empezar esta parte del código. En el primer caso diremos que se trata de un lazo con condición de entrada, mientras que el segundo será un lazo con condición de salida, tal como vemos en la figura siguiente. La ejecución en lazo es la que utilizan instrucciones de alto nivel, como por ejemplo *do/while*, *while* o *for*.



Finalmente, vemos en el recuadro siguiente un fragmento de código escrito tanto en C/C++ como en ensamblador en el que se implementa una ejecución en lazo. Suponemos que inicialmente el registro R1 ha sido cargado con varCond y el registro R2 con index.

Fragmento de código de ejecución en lazo

```
                        $1: CMP R1, #AH   // comparamos varCond con 10
while (varCond <10){    BGE $2            // si R1 es más grande o igual que 10, saltamos a $2
     index = index+2;   ADD R2, #2H       // añadimos 2 a index
     varCond++;         ADD R1, #1H       // añadimos 1 a varCond
```

```
}                       BR $1           // volvemos al inicio del lazo
                        $2: ...         // continúa la ejecución del código
```

Finalmente, la **llamada a una función** es quizá la instrucción de flujo más compleja que encontramos, en parte porque involucra a la pila, una estructura de datos especial. Esta estructura ocupa una región de memoria determinada, exclusivamente dedicada a ella, en la que los datos entran y salen siguiendo una política LIFO (último en entrar, primero en salir), a diferencia del acceso a memoria, donde es aleatorio. Además, siempre tenemos un puntero asociado a la pila o *stack pointer* (SP) que indica la posición del último dato almacenado. Las instrucciones para apilar y desapilar son *PUSH origen* y *POP destino*, respectivamente, y ninguna de ellas admite el modo de direccionamiento inmediato. La pila tiene varias aplicaciones, la mayoría de ellas relacionadas con la ejecución de subrutinas, que son:

- Salvar/restaurar registros.
- Llamadas sucesivas a subrutinas.
- Interfaz para paso de parámetros a subrutinas.

En la primera aplicación, utilizamos la pila para guardar la información existente en los registros del procesador. Esto se hace sobre todo cuando entramos en una subrutina para que no tenga ningún efecto colateral una vez volvemos a la función que ha efectuado la llamada. Antes de ejecutar la función apilaremos los registros que queramos guardar y al salir los desapilaremos, pero siempre en orden inverso para devolver la información al registro que procede.

Cuando utilizamos la pila para hacer la llamada a funciones, mediante las instrucciones CALL y RET. En el momento que llamamos a la función, ejecutamos `CALL nombreFuncion`, lo cual provoca el salto al punto del código etiquetado como `nombreFuncion` y, además, apila la dirección de regreso, es decir, la dirección en la que se encuentra la instrucción inmediatamente posterior al `CALL`. Así, al acabar la función, ejecutaremos `RET` para recuperar (desapilar) la dirección de regreso y continuar con la ejecución de la función principal.

Finalmente, la pila también nos sirve para pasar parámetros a una función, como en el caso de no tener suficientes registros disponibles. Consideramos, por ejemplo, la función C, que tiene como cabecera `void acumula (int *a, int b)`. Si mantenemos el convenio *C* de paso de parámetros, los apilaremos de derecha a izquierda, es decir, primero pondremos *b* y después la dirección de *a*. Aparte, dentro del cuerpo de la función, nos puede interesar movernos por la pila para poder acceder a estas variables. En ocasiones, como en los sistemas basados en x86, nos encontraremos con que no podemos usar el registro SP indirectamente. Entonces, se utiliza un registro alternativo: el *base pointer* (BP) en el caso de x86, en el que copiamos el valor del SP. El regreso de parámetros de la función, en cambio, se realiza habitualmente por medio de registros.

En la figura siguiente, podemos ver el código de la subrutina `void acumula (int *a, int b)`, para un sistema basado en i8086, en el que los registros AX y BX se utilizan para operar con los datos (AX sería el registro de regreso en caso de que la función no fuera de tipo void). También vemos el aspecto de la pila en el momento de entrar en el cuerpo de la función. Los pasos que seguimos son:

- Enlace dinámico y variables locales: guardamos el valor del BP anterior para restaurarlo antes de salir. Podría suceder que fuera necesario para la función que ha llamado *acumula*. Si quisiéramos poner variables locales en la pila, podríamos dejar el espacio necesario a continuación.

- Guardar registros: salvamos el valor de los registros AX y BX para restaurarlos al final de la función, puesto que podrían ser necesarios para la función principal.

- Cuerpo de la función: accedemos a los parámetros de la función a partir de BP para ponerlos en registros y hacemos la acumulación. Fijémonos en que el resultado ya queda en memoria.

- Restauración de registros y eliminación de variables locales (si hay): es importante seguir el orden inverso debido al mecanismo LIFO de la pila.

- Restauración del enlace dinámico: recuperamos el valor anterior de BP.

Si nos fijamos en el estado de la pila, veremos que, llegados a este punto, el dato que queda arriba de todo es la dirección de regreso y, por lo tanto, al ejecutar RET continuaremos con la ejecución de la función principal sin causar ningún tipo de interferencia.

Ejemplo de subrutina en un sistema x86

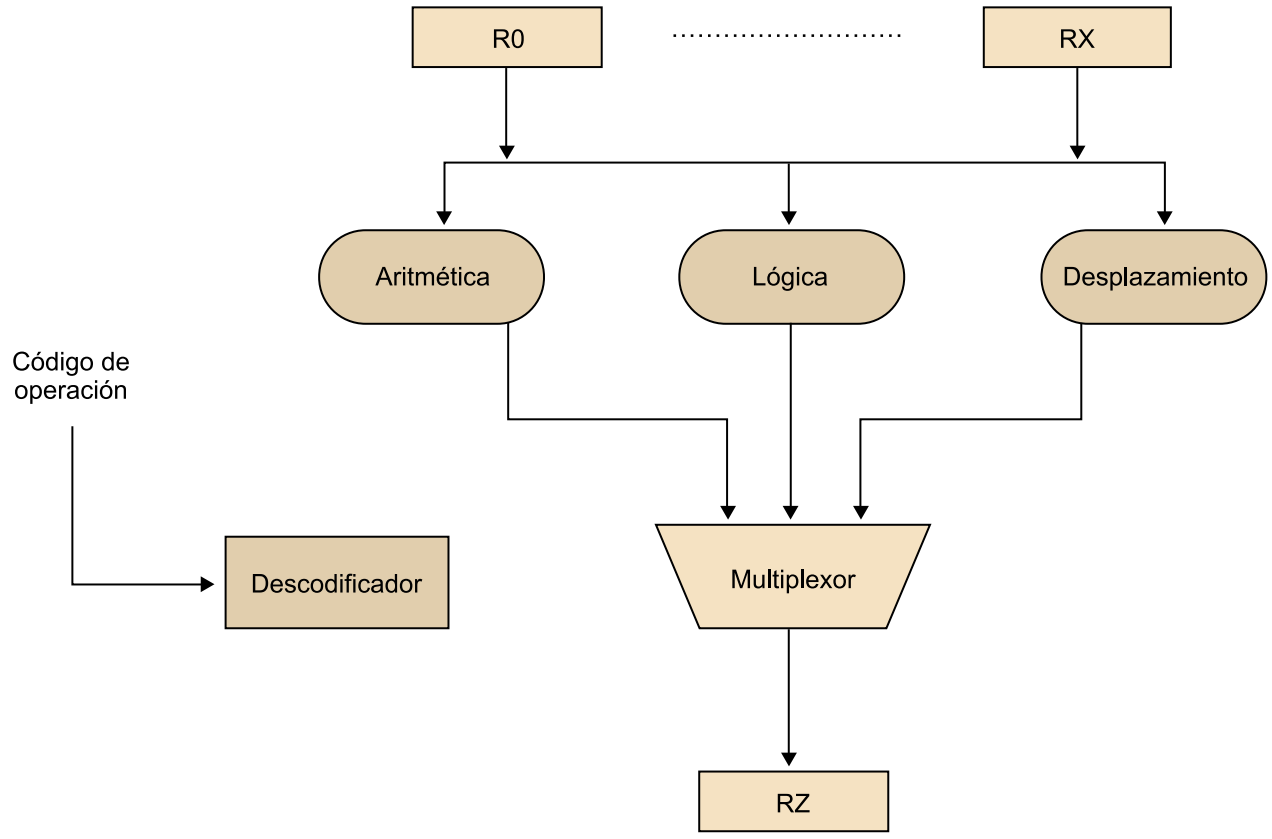
```
acumula:    PUSH BP
            MOV BP, SP
            PUSH BX
            PUSH AX

            MOV BX, [BP+4]
            MOV AX, [BP+6]
            ADD [BX], AX

            POP AX
            POP BX
            POP BP
            RET
```

Instrucciones de lógica y aritmética

En cualquier sistema empotrado, las instrucciones de lógica y aritmética son siempre ejecutadas en la unidad aritmeticológica (ALU) del procesador, la cual aglutina varios elementos de hardware para poder llevar a cabo las diferentes tareas que se le encomiendan.

Un posible diagrama funcional de ALU se puede encontrar en la figura siguiente. En ella apreciamos varios registros que almacenan temporalmente los operandos, un bloque dedicado a las operaciones aritméticas, otro dedicado a las operaciones de lógica, un bloque específico para el desplazamiento de bits, un multiplexor y un registro de salida. En función del código de operación que contiene la instrucción, el controlador de la ALU activa uno de los bloques anteriores para computar lo que se le pide. Finalmente, el resultado se escribe en un registro de salida para ponerlo a disposición del sistema.

Diagrama de bloques de una ALU



Normalmente el procesador del sistema es capaz de llevar a cabo cuatro funciones aritméticas: suma, resta, división y multiplicación, si bien algunos procesadores implementan solo suma y resta, y dejan las otras dos operaciones para ser tratadas en el nivel de software. Asimismo, encontraremos sistemas capaces de trabajar con coma flotante y otros que solo serán capaces de hacer operaciones con enteros.

Entre las instrucciones lógicas, encontraremos típicamente las funciones AND, OR y XOR bit a bit. Finalmente, las instrucciones de desplazamiento nos permitirán desplazar bits o palabras a derecha e izquierda. Podemos encontrar diferentes versiones de desplazamiento en función de cómo se llene el nuevo bit generado a consecuencia del desplazamiento y cómo se utilice el bit que se extrae.

2.5. Memoria

El subsistema de memoria es una parte esencial de cualquier sistema empotrado, puesto que es donde guardamos tanto los datos como las instrucciones que forman parte de nuestras aplicaciones. El diseño correcto de este subsistema en cuanto a dimensionado, gestión de los propios recursos y adecuación de los tiempos de ejecución afecta directamente al rendimiento global del sistema. Es más, asignaciones de memoria inadecuadas pueden incluso dejar nuestra aplicación en un punto muerto, "colgada". En general, nos podemos encontrar en dos situaciones claramente diferenciadas: si nuestra aplicación es pequeña, la memoria interna del procesador puede resultar suficiente y, por lo tanto, no tendremos problemas de compatibilidad. En cambio, en aplicaciones más grandes, tendremos que dotar al sistema de un módulo externo de memoria. En este caso, la compatibilidad, sobre todo en cuanto a tiempo de ejecución, es fundamental, tal como veremos más adelante.

En cuanto al tipo de memoria, de entrada podemos hacer dos grandes grupos: memoria de acceso aleatorio o RAM y memoria solo de lectura o ROM. A continuación, se presenta una clasificación más detallada de los diferentes tipos de memoria, si bien no es exhaustiva. Tenemos:

- **RAM**. Su nombre indica que podemos acceder a cualquier posición de memoria en cualquier momento, a diferencia de lo que sucede en otros tipos de memoria (normalmente antiguos), como pueden ser las cintas magnéticas, donde el acceso a una posición determinada pasa forzosamente por el acceso a posiciones anteriores. Además, los tiempos de lectura y escritura son del mismo orden de magnitud. De entre los diferentes tipos de memoria RAM podemos destacar:
  - **Memoria RAM dinámica (DRAM)**. Se trata de un diseño sencillo de la celda de memoria elemental (1 bit), basado en el almacenamiento de carga sobre un condensador. Como la carga almacenada es siempre volátil, las memorias de tipo DRAM requieren la operación de refresco, es decir, cada cierto tiempo hay que hacer una lectura de la memoria y volver a guardar el valor resultante. Finalmente, hay que remarcar que la lectura/escritura se realiza de una manera asíncrona al reloj del sistema.

  - **Memoria RAM estática (SRAM)**. Se trata de un diseño más complejo que el anterior y utiliza la filosofía de los biestables (podéis ver la figura 24 como ejemplo), con lo cual se evita la necesidad del refresco.

Continúan siendo memorias asíncronas como las anteriores, pero son más rápidas.

– **Memoria RAM pseudoestática (PSRAM)**. Las memorias PSRAM están formadas por un núcleo de memoria DRAM y una circuitería periférica que se encarga tanto del refresco como del control de direcciones. Técnicamente es una memoria DRAM pero se comporta como si fuera SRAM.

– **DRAM sincrónica (SDRAM)**. A diferencia de la memoria DRAM, la SDRAM sincroniza las direcciones, los datos y las señales de control con el reloj del sistema, lo cual permite incrementar las tasas de transmisión respecto a la versión asíncrona.

- **ROM**. La memoria ROM es una memoria de acceso aleatorio como la RAM pero está pensada, como su nombre indica, solo para lectura. Se puede escribir en ella, obviamente, pero el proceso de escritura es siempre mucho más lento que el de lectura y normalmente nos referimos a él como programación de la ROM. Los tipos más destacados de memoria ROM son:
  – **Memoria ROM programable (PROM)**. Se trata de una memoria que se programa una sola vez utilizando un aparato específico.

  – **Memoria EPROM (*erasable programmable read only memory*)**. Es una memoria de tipo PROM, pero con la particularidad de que puede ser borrada. Para hacerlo, hay que situar la ventana del dispositivo bajo luz ultravioleta durante un intervalo de tiempo determinado.

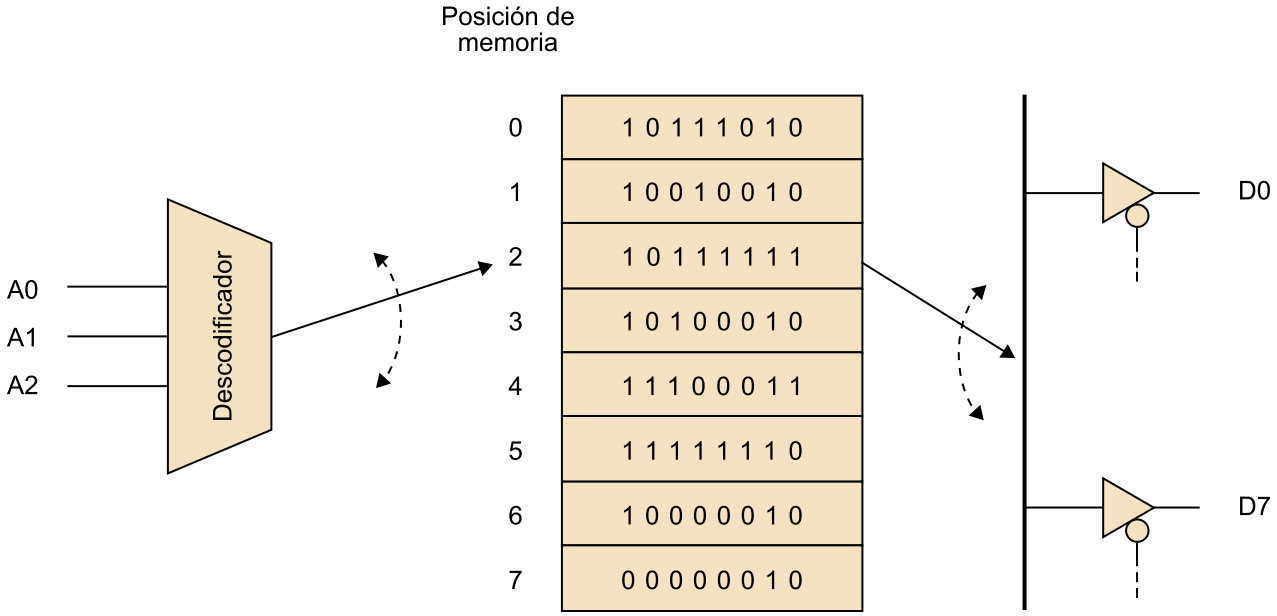
  – **Memoria EEPROM (*electrically erasable programmable read only memory*)**. En este caso, se puede borrar eléctricamente con el mismo aparato utilizado para la programación.

  – **Memoria flash**. Se trata de un tipo de memoria EEPROM que ha adquirido gran importancia en los últimos años (memorias USB, tarjetas SD, discos duros de estado sólido, etc.). En este caso, no es necesario sacar la memoria del circuito donde se encuentra habitualmente para reprogramarla.

En cuanto a la interacción entre memoria y sistema, cabe señalar que la interfaz de cualquier memoria está formada por tres tipos de señales, que, coincidiendo con los tipos de buses, son: direcciones, datos y señales de control. Independientemente del tipo de memoria y del mecanismo de almacenamiento que emplee, la interfaz más básica con la memoria la podemos ver en la primera figura del apartado 3.5.2., en la que de momento se han obviado las señales de control. Fijémonos en que un conjunto de bits del bus de dirección identifica una palabra de la memoria, que se lee o se escribe mediante los *buffers* (seguidores de tensión) de entrada/salida. Sin embargo, y por cuestiones de fabricación, es posible que las memorias no se adecuen siempre a las nece-

sidades del sistema en términos de tamaño, es decir, que no haya suficientes posiciones o direcciones o que la longitud de la palabra en memoria no sea suficiente. Más adelante veremos cómo se pueden solucionar las dos situaciones, entre otras.

Modelo e interfaz básicos de una memoria



Las señales que contienen las direcciones son siempre entradas del subsistema de memoria, y también lo son normalmente las señales de control. En cambio, los datos pueden ser de entrada o salida según escribimos o leemos en/de la memoria. La función de las señales de control es, en general, validar las operaciones que se tienen que llevar a cabo. Dado que las señales digitales experimentan un transitorio a la hora de cambiar de nivel, es de vital importancia saber en qué momento los valores son válidos.

Ejemplo

En el momento en el que queremos leer el contenido de una dirección de memoria determinada, no es suficiente indicar la dirección, sino que también debemos indicar en qué momento las señales que representan la dirección en cuestión son válidas para evitar interpretaciones erróneas.

En la práctica, esto se traduce en esperar el tiempo mínimo necesario entre la carga de la dirección y la activación de la señal de lectura. A continuación, mencionamos algunas de las señales de control más habituales:

- **Chip select (CS)**. Habilita el dispositivo de memoria. Generalmente es válido con valor bajo y puede servir para evitar el consumo de energía cuando el chip de memoria no se utiliza.

- **Output enable (OE)**. Control del *buffer* de salida del dispositivo. Suele ser válido con valor bajo y es entonces cuando permite la lectura de los datos. En caso contrario, el *buffer* mantiene un estado de alta impedancia.

- **Read (R)**. Indica que se quiere hacer una operación de lectura.

- **Write (W)**. Indica que se quiere hacer una operación de escritura.

- **Column address strobe (CAS)**. Indica que las señales presentes en el bus de datos corresponden a la dirección de la columna para el dispositivo. Tal como veremos más adelante, las memorias se organizan internamente de manera matricial y, por lo tanto, una posición queda completamente determinada si indicamos la fila y la columna donde se encuentra.

- **Row address strobe (RAS)**. Indica que las señales presentes en el bus de datos corresponden a la dirección de la fila para el dispositivo.

Finalmente, la tabla siguiente muestra la presencia de estas señales más habituales según el tipo de memoria con el que trabajamos.

Señales de control habituales

Tipo de memoria	CS	OE	R	W	CAS	RAS
ROM	x	x				
SRAM	x	x	x	x		
DRAM	x	x	x	x	x	x

En cuestión de temporización, a pesar de que se detallará más adelante teniendo en cuenta el tipo de memoria con el que trabajamos, sí que podemos decir que, a grandes rasgos, se sigue siempre la secuencia siguiente. En primer lugar, el bus de direcciones y el de datos (en caso de escritura) deben estar preparados o cargados, es decir, con la tensión estabilizada.

Nota

Recordemos que los cambios de tensión en los buses y circuitos lógicos no son inmediatos, sino que experimentan un transitorio.

Después de este primer paso, ya es posible dar la orden de lectura o escritura, y asegurarnos así de no tener comportamientos inesperados. Asociada a esta temporización, encontramos una serie de parámetros que caracterizan a las memorias. Algunos de los más destacados son:

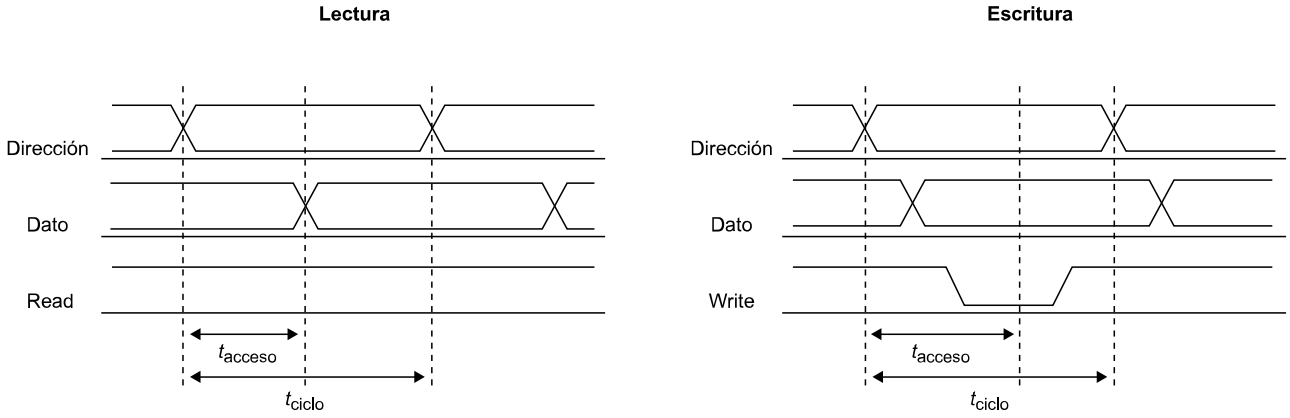
- **Tiempo de acceso**: es el tiempo necesario para obtener una palabra de la memoria o para escribirla. Se mide desde el momento en el que aplicamos la dirección hasta que la operación finaliza, es decir, hasta que los datos están en el *buffer* de lectura en el caso de lectura o hasta que los datos se han almacenado en memoria en el caso de escritura (podéis verlo en la figura siguiente). Asumimos que la escritura se ha completado en un tiempo determinado, después de que se inicie la transición alto-bajo en la señal "Write" para activar la escritura.

- **Tiempo de ciclo (lectura o escritura)**: es el tiempo mínimo que ha de pasar entre dos inicios de lectura o de escritura consecutivos y, por lo tanto, nos dice cuál es la velocidad a la que podemos acceder a la memoria.

- **Ancho de banda**: se utiliza para medir la capacidad de una memoria determinada para transmitir o recibir un flujo de datos y se mide en palabras

enviadas/recibidas por unidad de tiempo. La unidad empleada habitualmente es el MBps.

Tiempo de acceso y de ciclo en la lectura o escritura de/en memoria



Finalmente, hay otros parámetros temporales asociados a ciertas magnitudes de memoria que van más allá de la palabra. Estas otras magnitudes son el bloque y la página. Un bloque es simplemente una agrupación lógica de un conjunto de palabras y su utilidad se hace evidente en el momento en el que un sistema determinado quiere transferir ciertas cantidades de datos, puesto que se suele hacer a nivel de bloque y no de palabra. Asimismo, también se define la página como una agrupación de bloques. A partir de estas definiciones podemos introducir:

- **Latencia**: es el tiempo necesario para obtener la dirección del primer elemento de una secuencia determinada y acceder a él.

- **Tiempo de acceso de bloque**: es el tiempo necesario para acceder a un bloque completo. Por lo tanto, incluye el tiempo de acceso al primer elemento (latencia) y la transferencia del resto los de elementos.

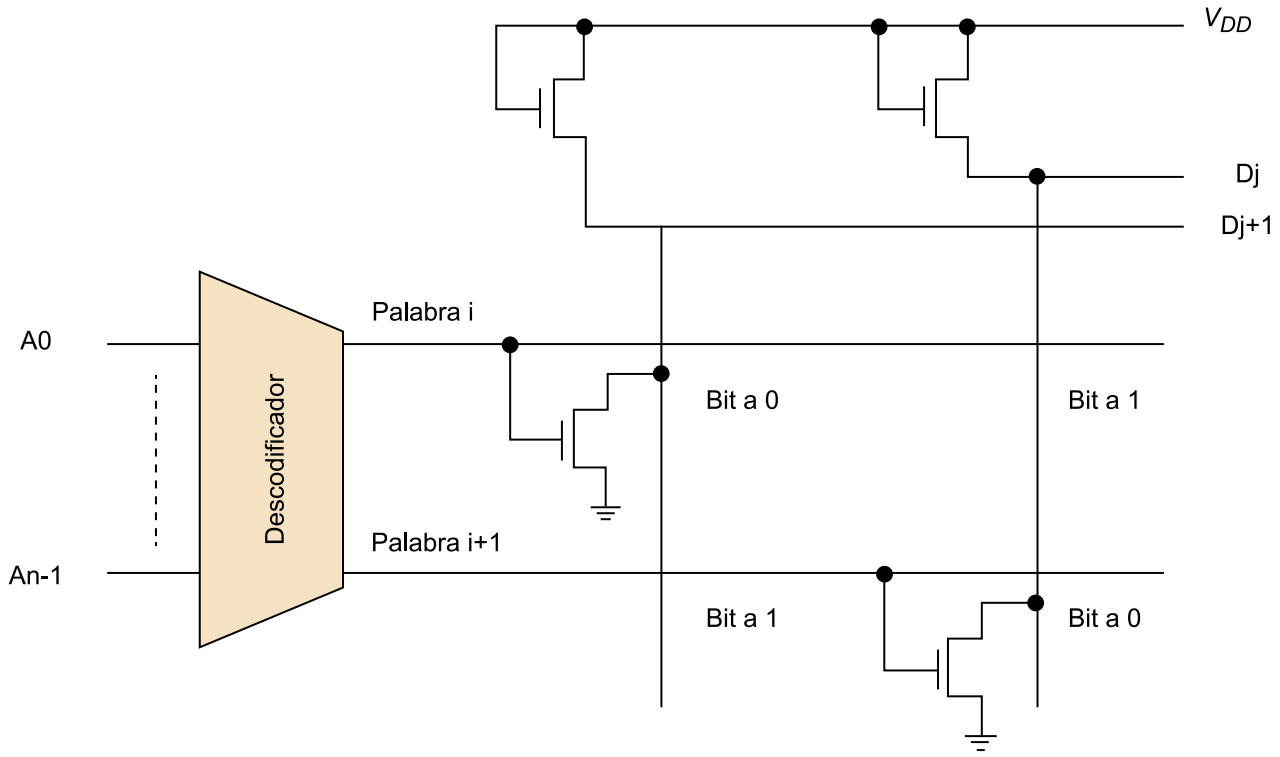
A continuación, se profundiza en los dos tipos básicos de memoria mencionados anteriormente: ROM y RAM.

2.5.1. Memoria ROM

En este apartado consideraremos la memoria ROM un dispositivo de solo lectura, es decir, obviaremos la parte de programación o escritura. En cuanto a la implementación, se puede conseguir almacenar un bit con la presencia o no de un transistor. En la figura siguiente vemos un posible esquema de memoria ROM utilizando únicamente transistores NMOS. Fijémonos en que cada fila interna de la memoria es activada o seleccionada por una única combinación de los bits de entrada, que forman la dirección. A priori, todas las columnas están cargadas a un valor lógico 1 (transistores NMOS con la puerta conectada a la tensión de referencia) y, por lo tanto, si no hay ninguno otro elemento, todos los bits de la palabra seleccionada tomarían un valor lógico 1. Sin em-

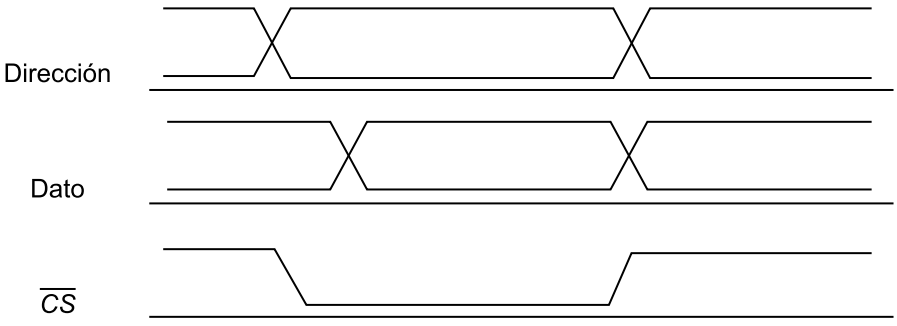
bargo, la colocación de un transistor NMOS conectado a tierra provoca que la tensión final de la celda o bit en cuestión se sitúe en un valor lógico 0, puesto que se drena la corriente a masa.

Memoria ROM implementada con transistores NMOS



En cuanto a la interacción con el dispositivo, deberemos indicar en primer lugar la dirección de la que se quiere hacer la lectura y también de la que queremos hacer la operación. En el ejemplo de la figura siguiente, fijémonos en que se habilita la señal $\overline{CS}$ (activo con valor bajo). A partir de este momento, hay que esperar el tiempo de acceso determinado por el fabricante para tener los datos disponibles en el *buffer* de salida. Hay que destacar que en este caso no se ha considerado la señal de control OE, si bien su funcionalidad es necesaria, puesto que para evitar conflictos en el bus datos conviene que la salida de la memoria se mantenga en estado de alta impedancia cuando no se utiliza.

Temporización de las señales en una memoria ROM



2.5.2. Memoria RAM

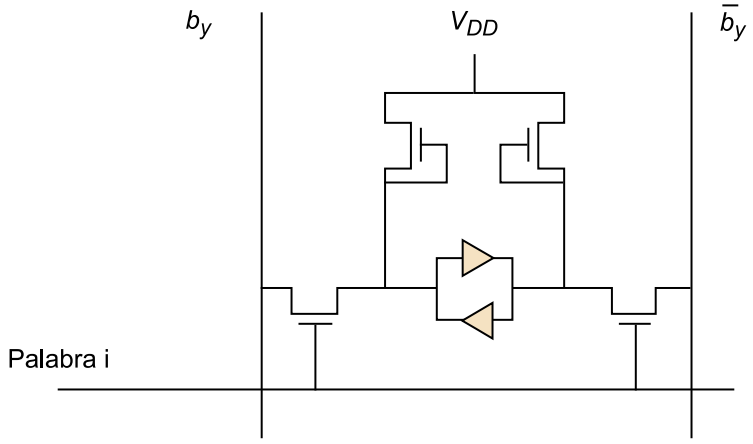
En este punto nos centraremos en los dos tipos básicos de memoria RAM, es decir, memoria RAM estática y memoria RAM dinámica. En primer lugar, describiremos la celda básica de memoria en cada uno de los casos y, a continuación, trataremos diferentes aspectos relacionados con la integración de la memoria en un sistema empotrado.

Memoria RAM estática (SRAM)

La celda básica de memoria SRAM se puede ver en la figura siguiente y está formada por seis transistores. Recordemos que cada inversor está formado por dos transistores y, además, son necesarios los dos transistores de carga conectados a la tensión de referencia. Además, se requieren dos transistores más de acceso para poder ejecutar las operaciones de lectura y escritura. La parte esencial de la memoria y donde se almacena el bit de información es el *latch*[1] que forman los dos inversores.

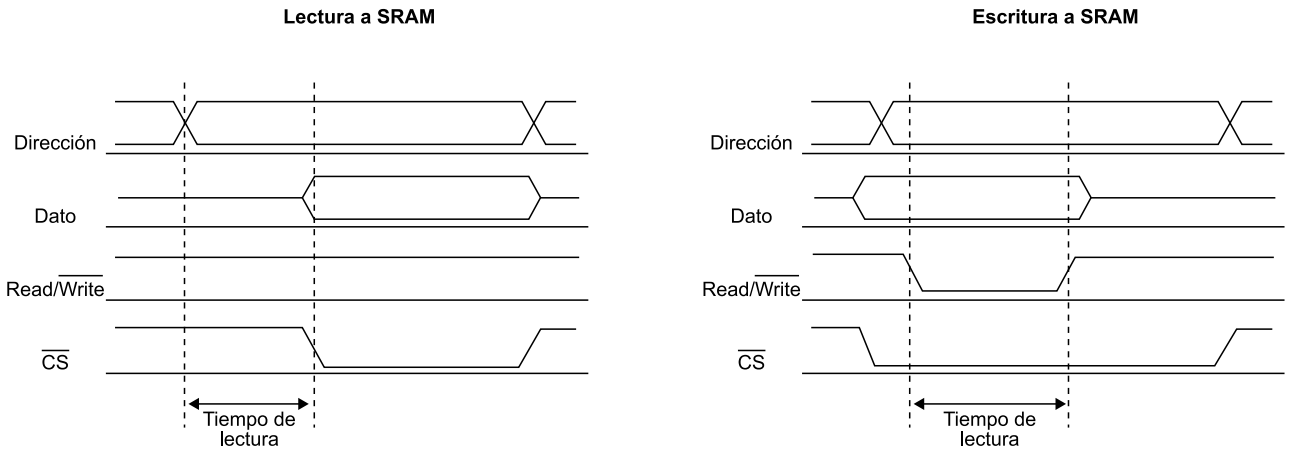
(1)*Latch* es sinónimo de biestable o *flip flop*.

Celda básica de una memoria SRAM



Para leer de la memoria, hay que tener precargadas las líneas b_i y $\bar{b}_i$ a una tensión intermedia entre masa y la de referencia. Entonces, cuando se activan los transistores de acceso seleccionando la dirección de memoria correspondiente, la línea b_i y la $\bar{b}_i$ toman los valores almacenados. Finalmente, el valor del bit tiene que ser sensato, amplificado y transmitido al exterior. A la hora de escribir, simplemente cargamos las líneas b_i y $\bar{b}_i$ mediante los amplificadores de escritura. Una vez se activen los transistores de acceso, el bit de información quedará almacenado en el *latch* de la figura. En la figura siguiente vemos la temporización típica en los procesos de lectura y escritura a una memoria SRAM. Fijémonos en que se distinguen los tiempos de acceso a memoria para cada una de las operaciones, puesto que no han de ser iguales.

Temporización de las señales en una memoria SRAM



En el momento de integrar una memoria determinada en un sistema empotrado, es posible que el procesador tenga un tamaño del bus de datos o de direcciones que coincida con el número de líneas disponibles en la memoria. En este caso, la integración es inmediata. Pero esto no es siempre así. Nos podemos encontrar con que esta condición no se cumpla por alguna de las dos partes, es decir, puede ser que el número de líneas del procesador no sea suficiente o que la memoria se quede corta en cuanto al tamaño de palabra o en cuanto al número de direcciones posibles. A continuación, se plantea un ejemplo de diseño lo más general posible y se propone una solución de integración.

Supongamos que las especificaciones del sistema en cuestión requieren una memoria SRAM capaz de almacenar 4 K palabras de 16 bits. Sin embargo, los módulos que tenemos disponibles son de 1 K palabras y solo 8 bits. Así pues, serán necesarios 8 módulos para poder cumplir las especificaciones. Además, el procesador tiene un bus de datos de 8 bits de ancho y un bus de direcciones también de 8 bits. Sabiendo que para poder distinguir 4 K direcciones diferentes necesitamos 12 bits, serán necesarias dos transferencias tanto para las direcciones como para los datos.

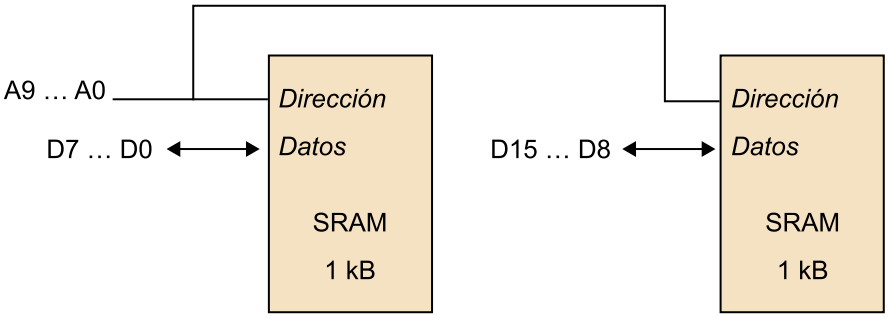
En primer lugar, utilizaremos bloques de dos módulos de memoria para obtener 1 K palabras de 16 bits de ancho, tal como se muestra en la figura siguiente. Como se puede apreciar, las líneas de dirección son comunes a los dos módulos, mientras que los bits de datos se dividen entre ellos, es decir, un módulo se encarga de los bits D7...D0 y el otro de los bits D15...D8. Una vez obtenidos los bloques de 1 K × 16 bits, resulta más o menos sencillo ampliar el diseño, en este caso cuadruplicar, a un sistema SRAM de 4 K × 16 bits. Si disponemos de 4 bloques de memoria de 1 K × 16 bits, solo hay que seleccionar en cuál de los 4 bloques tenemos que escribir o leer los datos en cada acceso. Fijémonos en que 4 K palabras se representan con 12 bits. De estos, 10 bits (A9...A0) serán necesarios para especificar la palabra deseada dentro del bloque de memoria. Los otros dos bits (A11, A10) servirán para determinar de manera unívoca qué bloque de memoria está activo en cada momento. Fijémonos en que los bits de dirección A9...A0 deben ser comunes a los cuatro bloques de memoria y,

Ejemplo

La dirección 001011001111 corresponde al primer bloque de memoria (00) y, dentro de este, tenemos que mirar la dirección 1011001111.

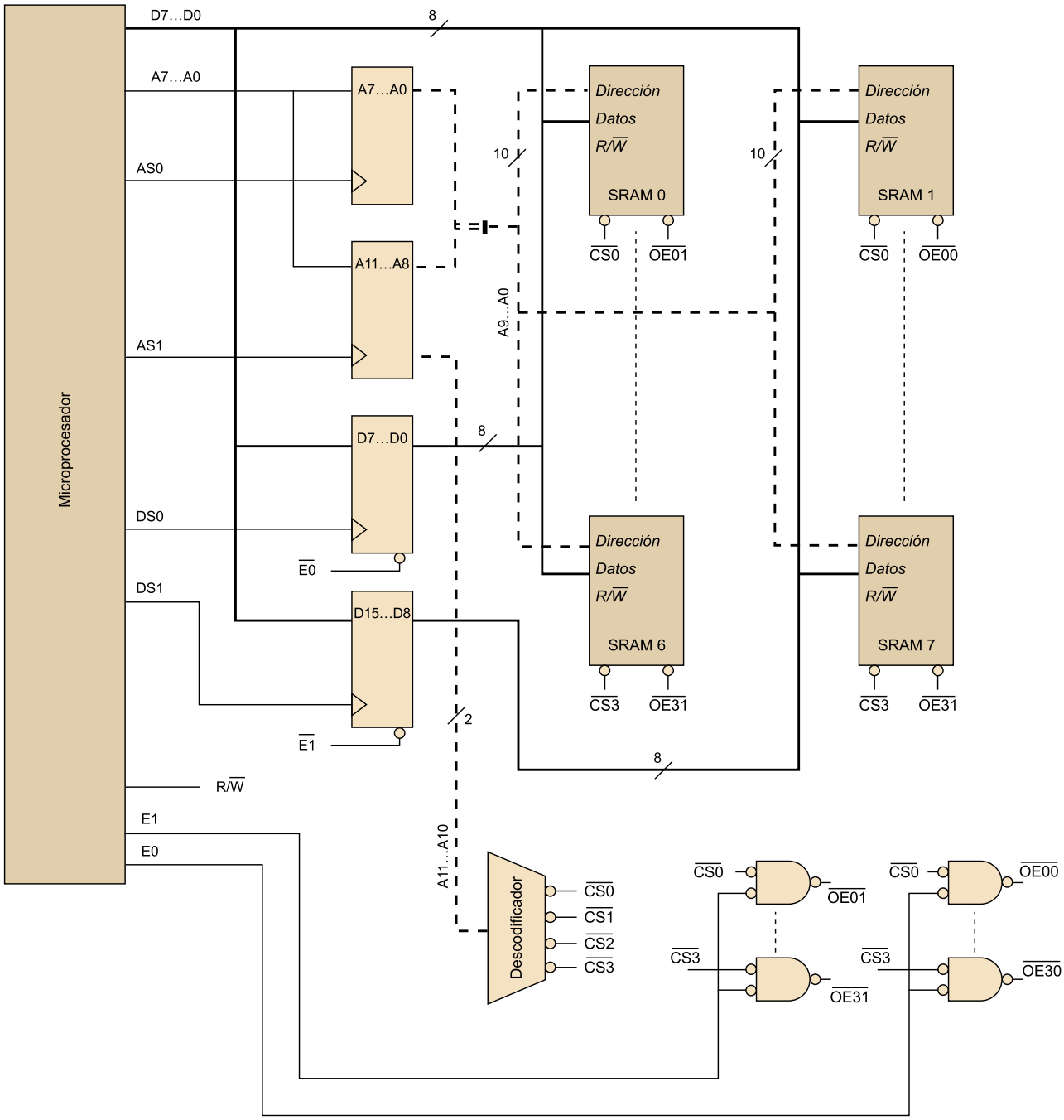
por lo tanto, es imprescindible desactivar los tres últimos bloques para no acceder a una posición indeseada. Un sencillo bloque de lógica combinacional nos servirá para llevar a cabo esta misión.

Memoria SRAM de 16 bits a partir de dos bloques de 8 bits



Finalmente, hemos de solucionar el hecho de que el procesador disponga de solo ocho líneas tanto para los datos como para las direcciones. Recordemos que las palabras (16 bits) y las direcciones (12 bits) del sistema en cuestión requieren dos transferencias. Sin embargo, necesitamos los 16 bits de datos y los 12 bits de dirección disponibles a la hora de acceder a la memoria, al menos para escribir. Con esta finalidad utilizaremos 4 registros (2 para los datos y 2 para las direcciones). A pesar de que hablaremos de los registros más adelante, de momento quedémonos con la idea de que son elementos de memoria de tamaño pequeño (unos cuantos bits). Cuando hay una señal de activación, almacenan los bits que se encuentran a su entrada en aquel momento y, a continuación, se encuentran en disposición de ser leídos a la salida hasta que se aparezca la señal siguiente, momento en el que los bits guardados serán reemplazados por los nuevos bits de entrada. El diseño completo del sistema de memoria y su interacción con el microprocesador se puede ver en la figura siguiente. A continuación detallamos el funcionamiento de las operaciones de escritura y lectura.

Sistema de memoria SRAM 4 K × 16 bits

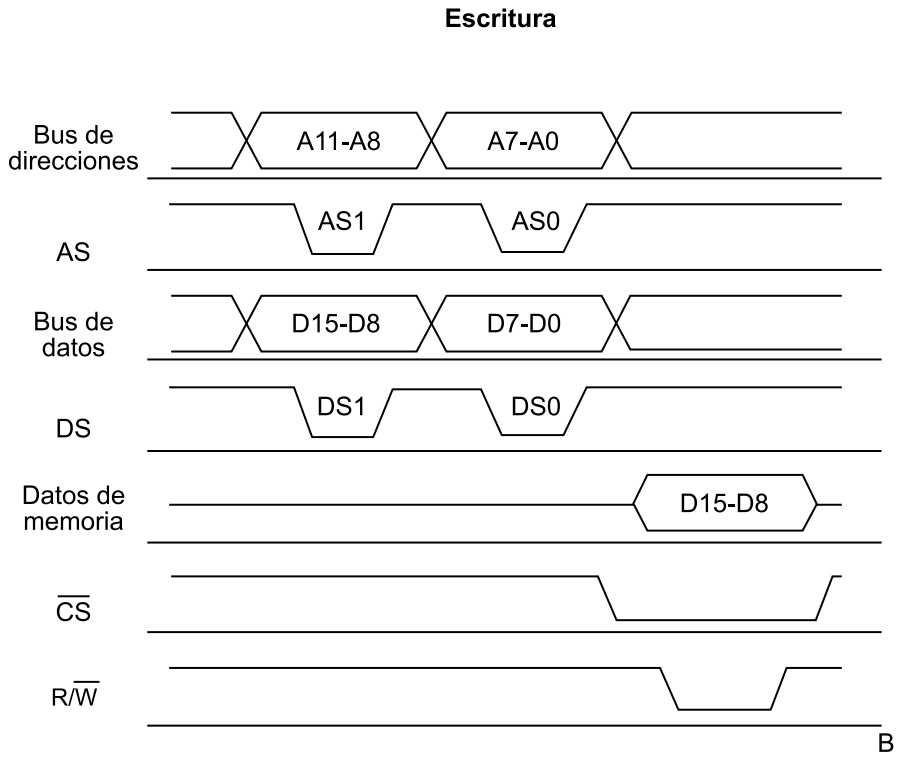


Escritura a memoria

El proceso de escritura a memoria se puede resumir en tres pasos que siguen el diagrama temporal de la figura siguiente. En primer lugar, se cargan los registros tanto de direcciones como de datos. Para hacerlo, utilizamos las señales *address strobe* (AS) y *data strobe* (DS) para indicar cuándo las señales están disponibles en las líneas de salida del procesador. Gracias a los bits A11 y A10, leídos en la primera transferencia, se calcula el bloque de memoria que hay que activar por medio de la correspondiente señal de *chip select* (CS). Finalmente, solo hay que dar la orden de escritura para ejecutar definitivamente la operación. Aparte, hay que mantener el *buffer* de salida de datos de las memorias en alta impedancia durante todo el proceso de escritura para evitar un conflicto

en el bus de datos. El lector puede comprobar en el diagrama de bloques de la figura siguiente cómo se ha tenido en cuenta esta cuestión mediante las señales E1 y E0 del microprocesador, que, tal como veremos a continuación, se utilizan para la lectura de datos en memoria.

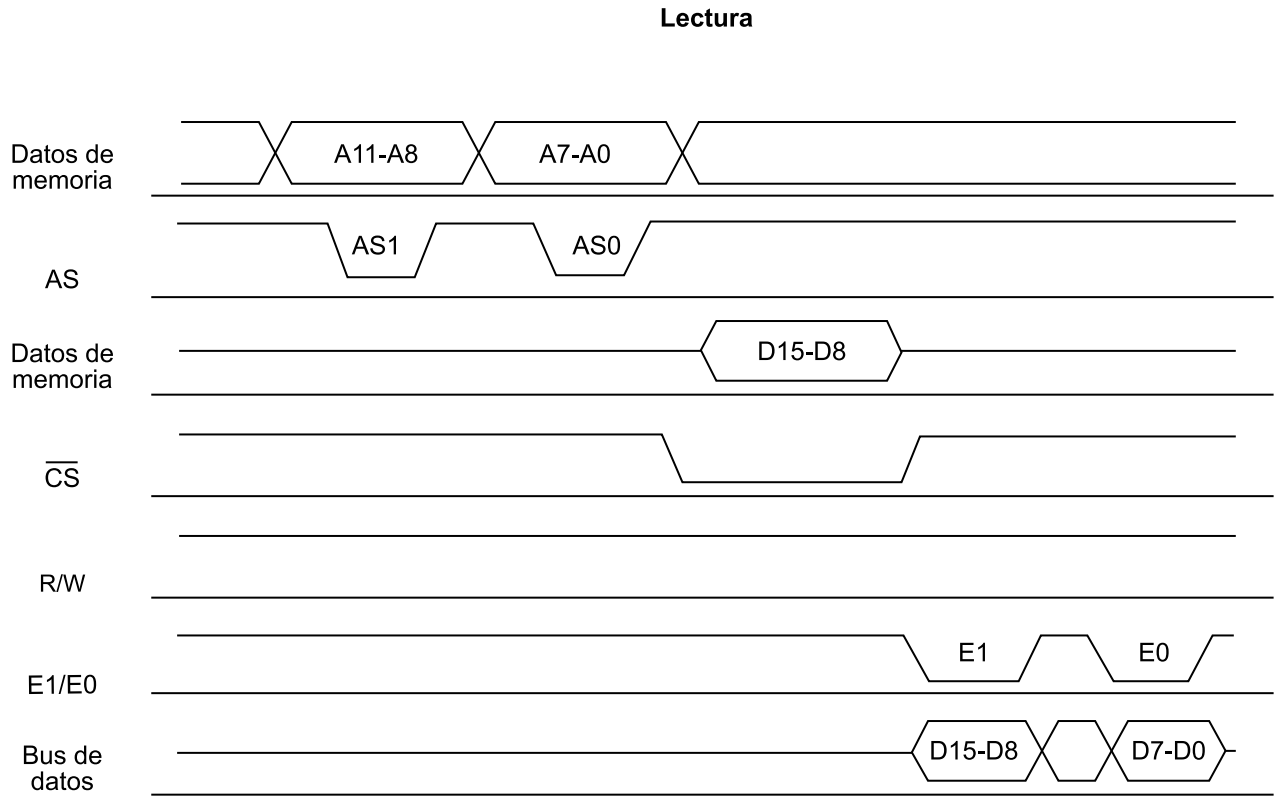
Proceso de escritura del sistema basado en SRAM



Lectura de memoria

El proceso de lectura se encuentra descrito en el diagrama de la figura siguiente. En este caso, solo utilizamos los registros de direcciones como memoria temporal, puesto que los bits de datos viajarán directamente de la memoria al procesador en dos transferencias de 8 bits cada una. Igual que en el caso de escritura, hay que evitar un conflicto en el bus de datos y, por ello, en esta ocasión nos debemos preocupar de dejar las salidas de los registros de datos en alta impedancia en el momento de la lectura de los bits. Podemos dividir el proceso de lectura en tres etapas. En primer lugar, se carga la dirección deseada a los registros correspondientes del mismo modo que en el caso anterior. Esto provoca, en segundo lugar, la habilitación del bloque de memoria donde se ubica el dato, si bien no está todavía disponible, puesto que los *buffers* de salida permanecen desactivados. Finalmente, las señales de habilitación que emite el procesador (E1 y E0) activan sucesivamente los *buffers* de salida de los dos módulos de memoria donde se guarda el dato para acabarlo transfiriendo al microprocesador.

Proceso de escritura del sistema basado en SRAM



Ya para acabar, hay que remarcar que este es solo un ejemplo de diseño de un sistema de memoria SRAM. Se ha procurado hacerlo lo más completo posible, pero, obviamente, habrá que estudiar cada caso por separado y tener en cuenta las características tanto del procesador como de la memoria con la que trabajamos.

Ejemplo

Nos podemos encontrar con que el procesador tenga suficientes líneas tanto para direcciones como para datos, si bien esta situación no es el habitual en sistemas empotrados. En el otro extremo, también nos podemos encontrar con un bus para datos y direcciones compartido, que requerirá un diseño parecido al expuesto, con un registro para direcciones y otro para datos.

En resumen, pues, hay que examinar cada caso con detenimiento y adaptarse a las características del hardware.

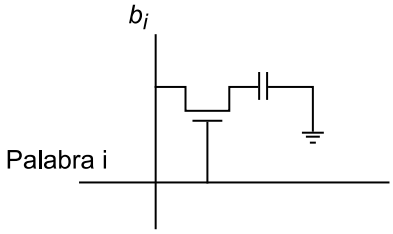
Memoria RAM dinámica (DRAM)

La celda básica de memoria DRAM es mucho más simple que la de SRAM y se puede ver en la figura siguiente. Como podemos ver, se trata de un solo transistor MOS con los *buffers* de entrada/salida correspondientes y el mecanismo de almacenamiento de información se deriva de la capacidad parásita del propio transistor. De este hecho se pueden extraer dos características importantes de las memorias DRAM:

- Permiten integrar cantidades de memoria más grandes que las SRAM porque son menos complejas.

- Como el mecanismo de almacenamiento es volátil, puesto que la carga acumulada drena por medio de la resistencia del transistor, hay que renovar continuamente la información guardada. Para hacerlo, se efectúa una lectura de los bits para reescribirlos inmediatamente después. Esta renovación de la información se conoce con el nombre de *refresco* o *ciclo de refresco* y es el fabricante de la memoria quien debe determinar el tiempo máximo permitido entre refrescos o intervalos de refresco.

Celda básica de una memoria DRAM

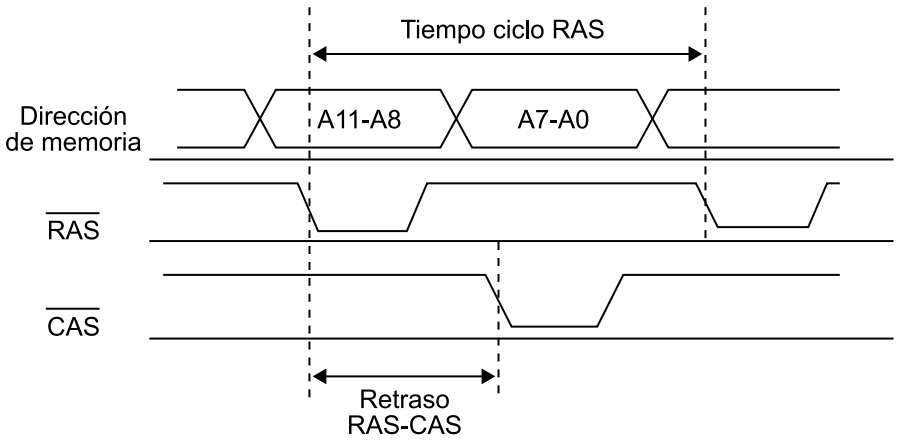


La lectura del bit guardado se hace precargando b_i a una tensión intermedia entre masa y la tensión de referencia. Después se activa la puerta del transistor y se comparte así la carga almacenada. Por lo tanto, si el valor guardado era un 0, la tensión b_i baja y si era un 1, la tensión sube. Finalmente, hay que detectar y amplificar esta variación para traerla al exterior de la memoria. Sin embargo, es obvio que el proceso de lectura deteriora la información almacenada y es la circuitería de la misma memoria la que se encarga de restaurarla o reescribirla. La operación de escritura es todavía más simple: solo se carga la línea b_i al valor deseado y se activa la puerta del transistor, con lo cual se carga o descarga la capacidad parásita para guardar el valor lógico que queremos.

En cuanto al diseño del sistema de memoria, todo lo que se ha visto para la memoria SRAM se puede trasladar al caso que nos ocupa, si bien ahora hemos de tener en cuenta algunas consideraciones adicionales. La primera es consecuencia del empaquetado del chip, puesto que muchas veces no es posible incluir un pivote para cada línea o bit de dirección. La solución en estos casos es multiplexar los bits de dirección aprovechando la estructura matricial de la memoria, es decir, primero se suministran los bits de dirección que identifican la fila donde se encuentra nuestra palabra y después se suministran los bits que identifican la columna. Para gestionarlo, utilizamos las señales RAS y CAS, que indican, respectivamente, cuándo los bits de la fila o de la columna están disponibles. Asociado a estas señales, definimos los parámetros temporales siguientes:

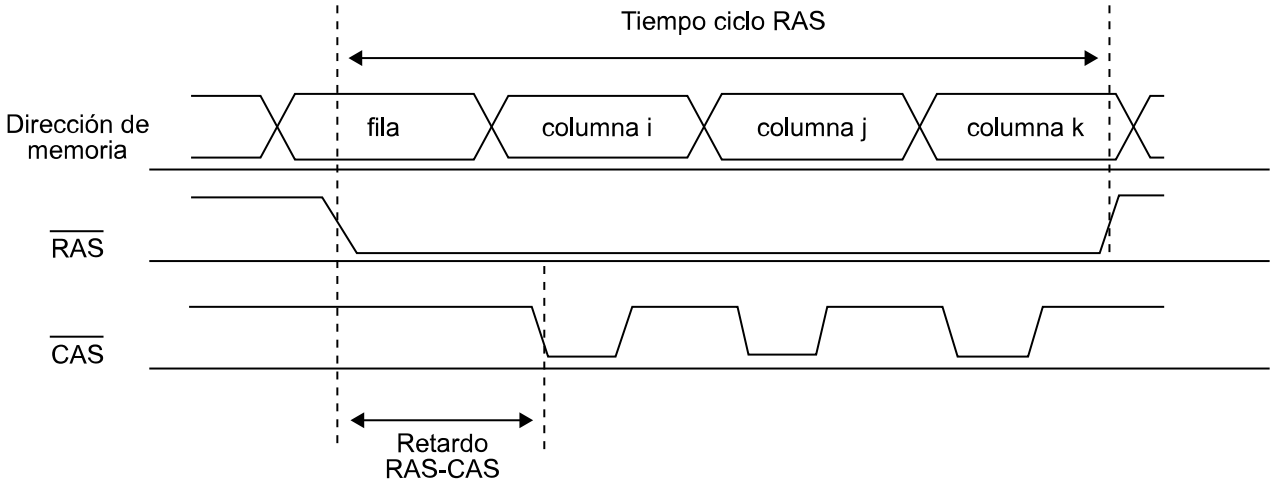
- Tiempo de ciclo del RAS: indica el período de esta señal.
- Retardo RAS-CAS: indica la separación que hay entre las señales RAS y CAS.

Temporización de las señales de dirección en una memoria DRAM



La figura muestra la señalización multiplexada de direcciones junto con los parámetros que acabamos de definir. Finalmente, hay que comentar que los diferentes tipos de memoria DRAM se distinguen básicamente por cómo se organizan internamente y por cómo se accede a los datos. Por poner un ejemplo, mencionamos las memorias DRAM que soportan EDO (*extended data output*) y que permiten acceder a varias palabras a la vez. En particular, deben ser palabras que tengan la parte de la dirección correspondiente a la fila en común y cuya señalización empleada sea la de la figura siguiente.

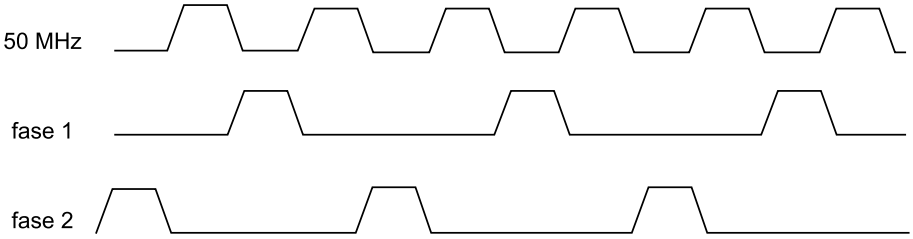
Temporización de las señales en una memoria DRAM con EDO



La segunda consideración es consecuencia de la necesidad de refresco de las memorias. Definiremos el *período de refresco* como el período temporal máximo con el que las celdas de memoria deben ser refrescadas para evitar pérdidas de información. Existen varias maneras de refrescar la memoria y hay que hacerlo de la manera más eficiente posible, puesto que el refresco en sí es un trabajo extra que no nos aporta ningún beneficio desde el punto de vista funcional. Esto implica toda una circuitería adicional que se encargue de recorrer las posiciones de memoria con los tiempos adecuados y de evitar conflictos con el funcionamiento normal de la memoria. A continuación, presentamos un posible diseño para una memoria de 4 M palabras de 16 bits cada una.

La memoria está organizada en 4 K filas por 1 K columna y es posible refrescar al mismo tiempo todas las palabras que constituyen una única fila, con lo cual conseguimos que el refresco sea más eficiente. En el supuesto que nos ocupa, asumimos que el período de refresco es de 64 ms y que disponemos de un reloj de 50 MHz con dos fases diferentes, tal como muestra la figura siguiente.

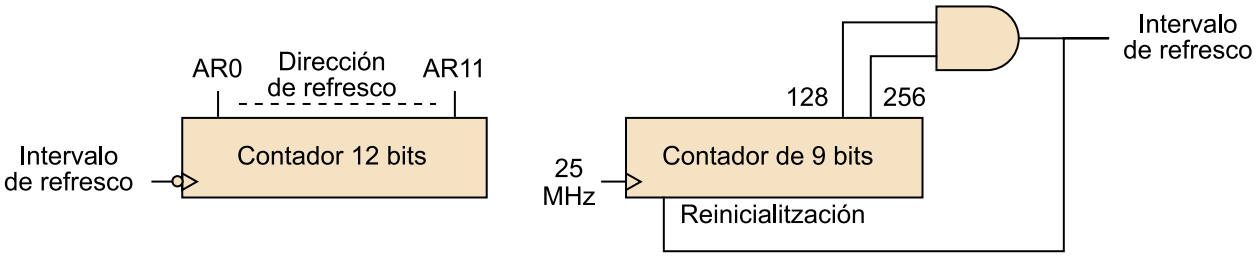
Reloj de dos fases empleado



En primer lugar, nos ocupamos del tiempo de refresco. Si tenemos en cuenta las 4 K filas de la memoria, vemos que hay que refrescar una fila cada 16 µs. Esto se puede conseguir con un contador de 9 bits alimentado por una de las fases del reloj (25 MHz). Fijémonos en que 400 golpes de reloj hacen los 16 µs que necesitamos. Si queremos incluir un cierto margen de seguridad y a la vez facilitar la implementación, podemos considerar 384 golpes de reloj, equivalentes a 15,36 µs. Poniendo una puerta AND entre los dos bits de más peso del contador conseguimos una señal que se mantiene a 0 hasta que no han transcurrido los 384 períodos de reloj deseados. Entonces, toma el valor 1 y se mantiene en él hasta que el contador no da la vuelta, es decir, cuando pasa de 511 a 0. Fijémonos en que, si no hiciéramos nada más, tendríamos una señal que se activaría cada 512 golpes de reloj, lo cual no nos interesa. Por lo tanto, una solución será optar por un contador con entrada de inicialización síncrona, que será alimentada por la señal de salida de la puerta AND, el mismo que marcará el intervalo de refresco deseado.

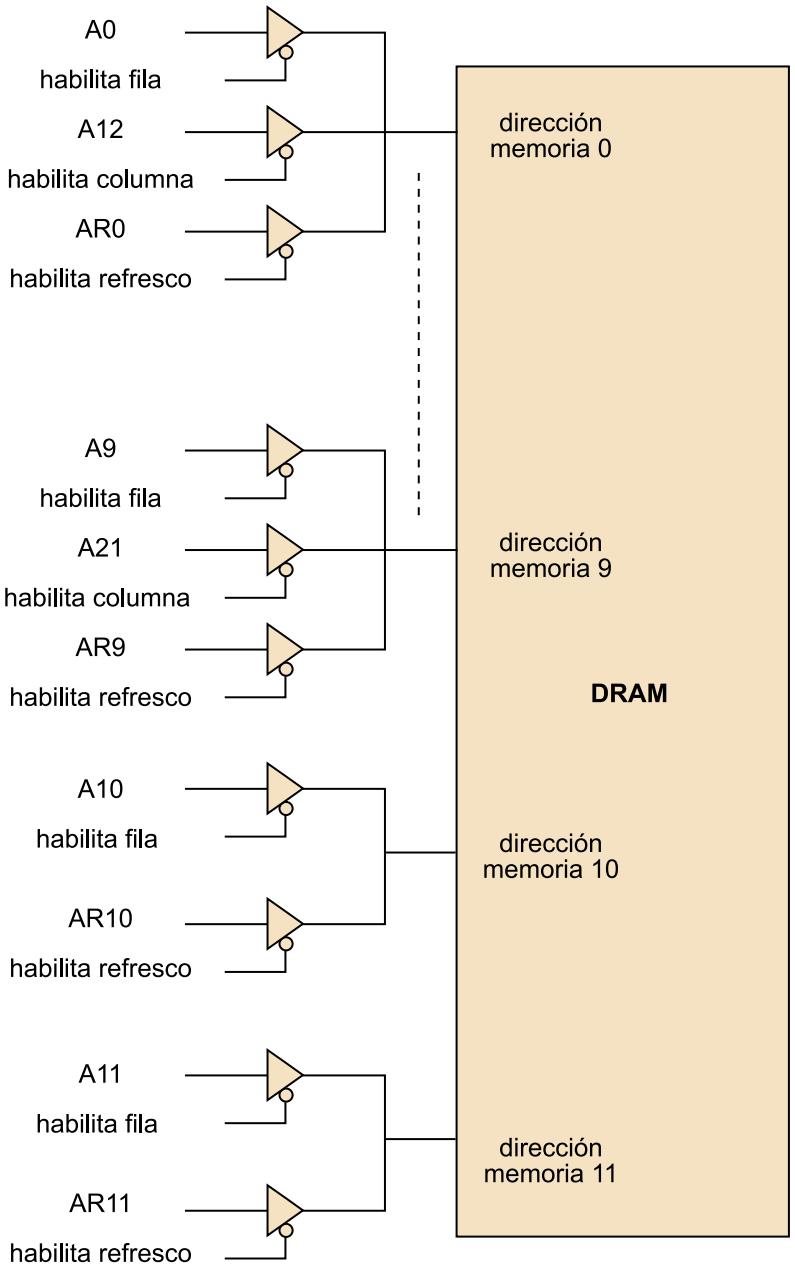
Una vez establecido el tiempo de refresco, hemos de calcular la posición o posiciones de memoria que hay que refrescar. En el supuesto que nos ocupa, donde refrescamos una fila de memoria cada vez, un contador de 12 bits será suficiente para recorrer todas las posiciones, es decir, las 4 K filas. Además, activaremos el contador en cada flanco de bajada de la señal intervalo de refresco, puesto que de este modo la dirección que hay que refrescar estará disponible con toda certeza a la hora de ejecutar la operación. En la figura 34 podemos ver tanto el generador de direcciones de refresco como el generador del intervalo de refresco.

Generador de direcciones de refresco (izquierda) y del intervalo de refresco (derecha)



Como tenemos 12 bits para indicar la dirección en la memoria DRAM y estos bits pueden tener hasta tres funciones diferentes (fila de memoria para lectura/escritura, columna de memoria para lectura/escritura o fila para refresco), utilizaremos *buffers* de salida para diferenciar cada una de estas funciones y evitar conflictos, tal como se muestra en la figura siguiente. De este modo, podremos dejar los *buffers* en estado de alta impedancia cuando no se utilicen. Esto implica el diseño de circuitería adicional para generar las señales de control de los *buffers*.

Control de las direcciones (normales y refresco)



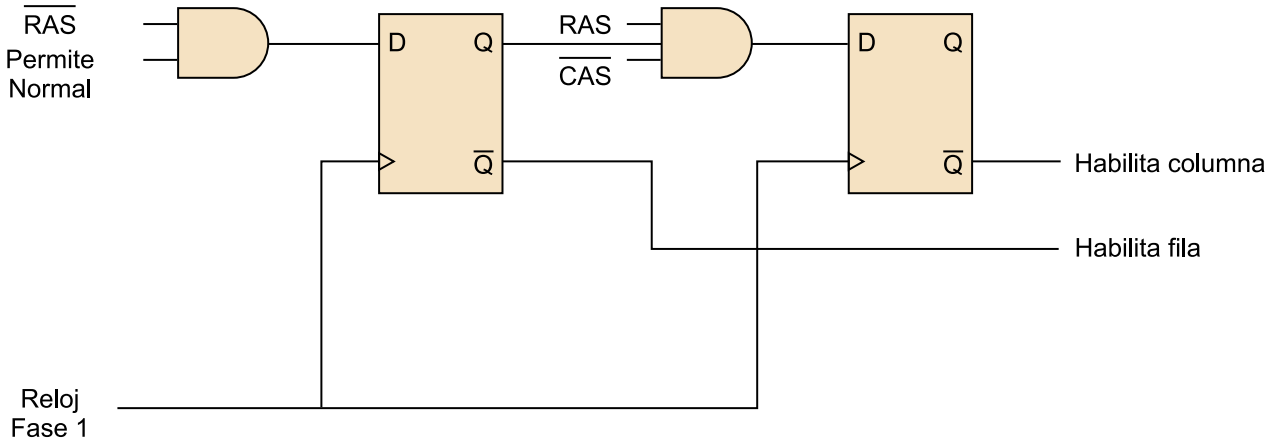
Ejemplo

En la figura siguiente vemos una manera de generar las señales de habilitación de fila y columna cuando la señal Permite Normal (que permite las operaciones normales de lectura o escritura) es válida. En la misma figura, podemos ver un diagrama temporal de las señales que intervienen. En este ejemplo, las señales RAS y CAS provenientes del procesador están sincronizados con la fase 2 del reloj, mientras que las señales de habilitación que generamos son gobernados por la fase 1. Este mecanismo evita interpretar erróneamente las señales tomando algún valor del transitorio.

Llegados a este punto, hemos sido capaces de adecuar la memoria DRAM para que pueda soportar procesos de lectura/escritura y también de refresco. Sin embargo, todavía falta lo más importante: la gestión temporal de estos procesos. Fijémonos en que el refresco de la memoria es una acción más controlada que las operaciones de lectura/escritura habituales, puesto que podemos predecir qué posiciones de memoria se verán afectadas en cada momento. En cambio, el acceso normal puede ser más arbitrario, y dejar abierta la posibilidad de querer acceder a memoria y refrescar al mismo tiempo. Para evitar este tipo de conflictos, deberemos incluir un módulo de arbitraje. Una posible definición funcional de este es la siguiente:

- Cuando empieza una operación de lectura/escritura, se deja acabar.
- Si una operación de refresco ha empezado, se deja acabar recordando la operación normal.
- Si las dos operaciones coinciden, se da prioridad a la operación normal.

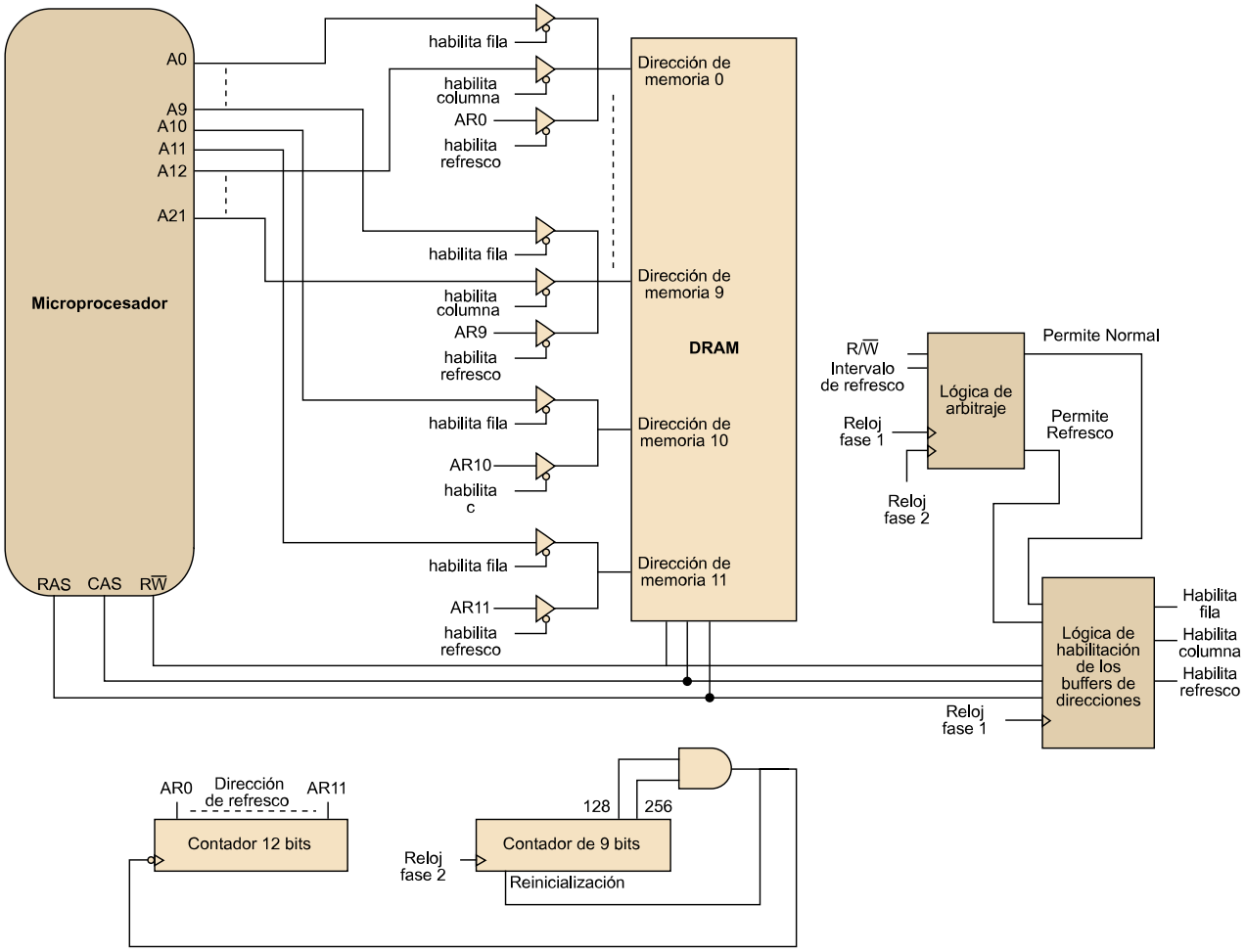
Ejemplo de generación de las señales de habilitación de los *buffers* de dirección



Finalmente, la figura siguiente muestra el diseño completo del sistema de memoria basado en DRAM, en el que usamos las señales siguientes:

- **Lectura-escritura**: indica que se ha iniciado un proceso normal de lectura/escritura.
- **Permite Normal**: indica que el módulo de arbitraje permite las operaciones de normales de lectura/escritura.
- **Permite Refresco**: indica que el módulo de arbitraje permite las operaciones de refresco.

Sistema de memoria DRAM completo



2.5.3. Sistema de memoria

En un sistema empotrado se suelen combinar memorias de diferentes tipos que finalmente dan lugar a su sistema de memoria. Habrá una parte de memoria ROM en la que guardaremos las aplicaciones y una parte de memoria RAM que utilizaremos a la hora de ejecutarlas. Asimismo, dividiremos la parte que corresponde a memoria volátil en tres tipos diferentes, que a la vez están formados por memorias de características diferentes. A continuación, las presentamos según su tamaño, de mayor a menor:

- **Memoria secundaria**: es opcional en sistemas empotrados y está formada por grandes memorias con velocidades de acceso moderadas. El coste también es el menor de las tres.

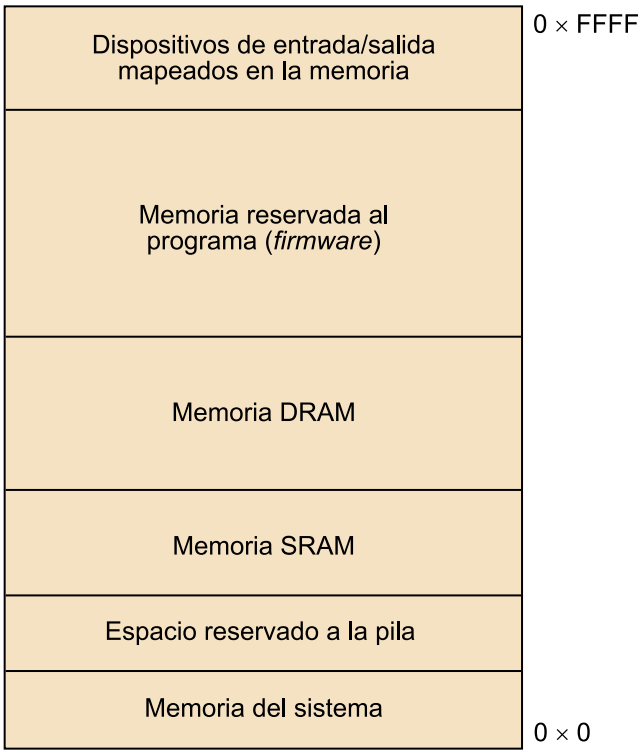
- **Memoria principal o primaria**: de tamaño más pequeño, está constituida por memorias más rápidas y también más caras. Típicamente son memorias DRAM o SRAM de baja velocidad.

- **Memoria caché**: es la parte más pequeña y rápida del sistema de memoria. También es la más costosa. Está formada por memorias SRAM de alta velocidad.

La lógica que hay detrás de esta configuración del sistema de memoria responde al comportamiento de la mayoría de las aplicaciones. Así, hay una pequeña cantidad de datos a los que se accede de una manera mucho más continuada que al resto en un momento determinado. Por lo tanto, este diseño se acomoda a esta situación y favorece la velocidad de ejecución del software si se ubican los datos de uso más frecuente en la memoria caché y el resto en la memoria principal. También se trata de una solución efectiva en términos de coste, puesto que se mantiene la capacidad del sistema de memoria y se penaliza poco en velocidad de ejecución respecto a un sistema equivalente que integre solo memorias de alta velocidad. En ocasiones, también se consideran los mismos registros del microprocesador en una categoría adicional que podríamos denominar **memoria caché interna**.

Asociado al sistema de memoria que acabamos de comentar y más en el nivel lógico, hemos de considerar el **mapa de memoria** de cualquier sistema empotrado. Se incluyen en él todas las direcciones físicas existentes en el sistema y, aparte de la zona dedicada a memoria volátil, comentada anteriormente y que contiene tanto datos del usuario como del sistema, también consideramos la memoria de tipo ROM en la que se guardan las instrucciones de programa o los dispositivos de entrada/salida mapeados a memoria, que van desde una pantalla o un visor a un disco duro con acceso directo a memoria (DMA). En la figura siguiente podemos ver un ejemplo de mapa de memoria para un sistema de 16 bits:

Mapa de memoria en un sistema empotrado



2.5.4. *Caching*

Anteriormente, hemos visto los diferentes tipos de memoria que integran un sistema empotrado y también cómo el uso de diferentes tipos de memoria RAM responde al patrón de ejecución típico de una aplicación. Desde ya hace tiempo, se sabe que la mayoría de las aplicaciones se ejecutan o bien de manera secuencial o bien en pequeños lazos de código (zona de referencia secuencial), lo que podemos interpretar como una pequeña ventana de actividad que se va desplazando a lo largo del código de nuestro programa. Además, la velocidad de desplazamiento de esta ventana es mucho más lenta que la velocidad de acceso de las memorias más rápidas.

Hecha esta observación, la idea de tomar bloques de datos de la memoria principal y trasladarlos a una memoria pequeña y rápida (la caché) tiene muy sentido, puesto que durante un rato el procesador podrá interactuar con la memoria caché, mucho más rápida, y se ahorrará el acceso a la memoria principal, más lenta. Esta es la idea del *caching*.

Lo primero que necesitamos para poder hacer *caching* es establecer una zona de referencia donde se centre la actividad del programa durante un intervalo de tiempo determinado. Si no es así, el mecanismo resultante no será válido porque no podremos evitar el acceso continuado a la memoria principal y se retardará así la ejecución del programa. Tenemos dos maneras principales de definir la zona de referencia:

- **Zona de referencia espacial**: suponemos que en un futuro inmediato se accederá a datos que se encuentran físicamente cercanos a los utilizados en el tiempo presente.

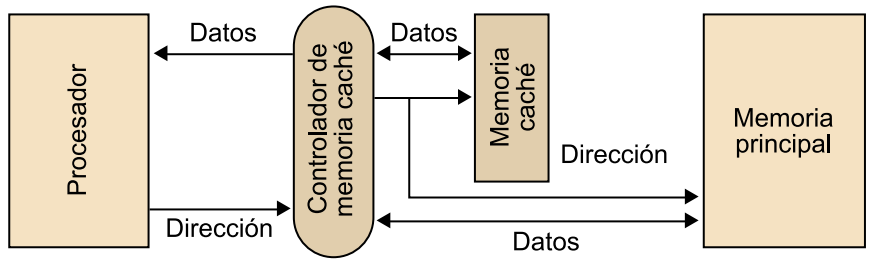
- **Zona de referencia temporal**: suponemos que los datos utilizados en el tiempo actual pueden ser reutilizados en instantes futuros cercanos.

Ved también

Un caso particular y conocido de zona de referencia espacial es el que hemos comentado al comienzo de este mismo subapartado, conocida también como **zona de referencia secuencial**, en la que la ejecución se produce de manera secuencial y en pequeños lazos de código.

Una vez sabemos que se dan las condiciones para que el *caching* sea efectivo, debremos implementar el sistema de *caching* siguiendo la arquitectura que se muestra en la figura siguiente:

Arquitectura del subsistema de *caching*



Como vemos, necesitaremos un controlador encargado de ejecutar el algoritmo de *caching* y de hacer de intermediario entre el procesador y el sistema de memoria. El controlador será el que decida cuándo y cómo se hacen los traspasos de datos entre la memoria caché y la principal. Se trata, por lo tanto, de un elemento crucial de cara al rendimiento del sistema.

En tiempo de ejecución, cuando el procesador necesita algún dato o instrucción, mira primero la memoria caché. Si la información se encuentra allí, decimos que ha habido un *cache hit*, se toma la información y se continúa la ejecución. En caso contrario, decimos que ha habido un *cache miss* y tenemos que ir a la memoria principal a buscar la información. Entonces, se toma el bloque de datos donde está la información y se traslada a la caché. Si hay espacio, se guarda directamente y, si no, se debe crear borrando un bloque existente. Antes de hacerlo, hay que comprobar si ha habido modificaciones en los datos de aquel bloque y guardarlos en la memoria principal en caso afirmativo.

A partir de aquí, hay que decidir cómo se llevan a cabo todas estas operaciones, es decir, cómo sabemos si un bloque está en caché o no, cómo seleccionamos el bloque que se debe borrar o cómo sabemos si la información ha sido modificada. A continuación, veremos tres algoritmos de *caching* diferentes que dan respuesta a estas preguntas. En los tres casos, tendremos en cuenta las especificaciones siguientes del sistema de memoria:

- Se trabaja con palabras de 32 bits de ancho.

- La caché permite almacenar 64 K de palabras.

- Cada bloque de datos contiene 512 palabras y, por lo tanto, la caché permite almacenar 128 bloques.

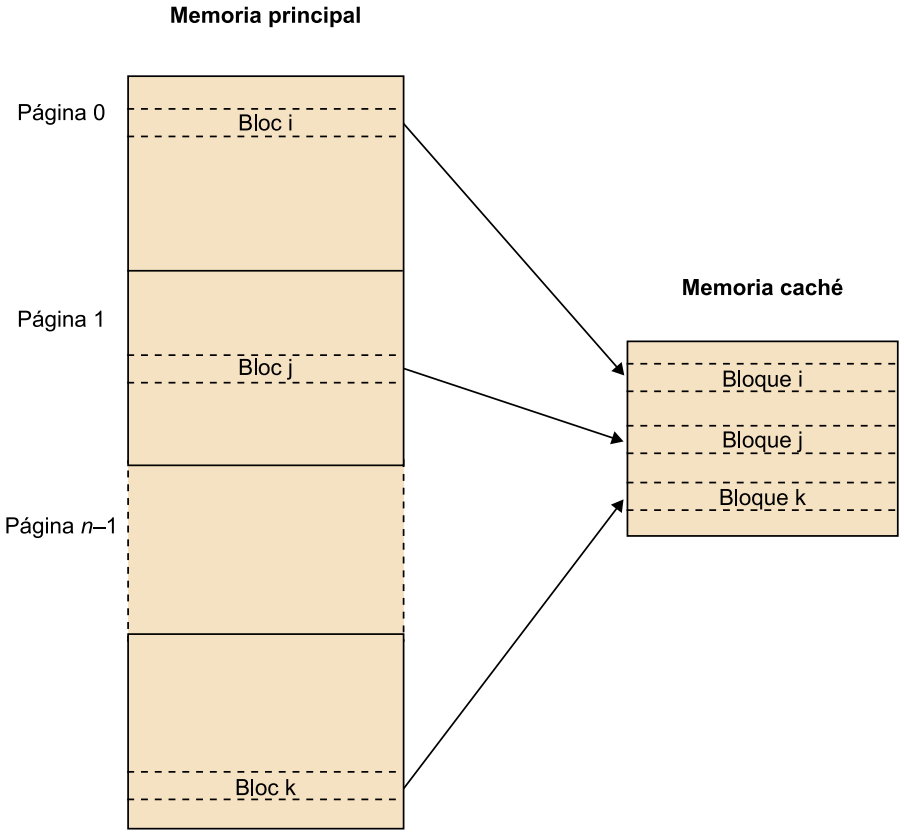
- El bus de direcciones tiene un ancho de 32 bits, suficiente para direccionar las 128 M de palabras que contiene la memoria principal.

- La memoria principal está dividida en 2 K de páginas y, teniendo en cuenta que el tamaño del bloque está fijado en 512 palabras, cada página contiene 128 bloques. Por lo tanto, una página contiene 64 K de palabras.

Implementación con mapeo directo

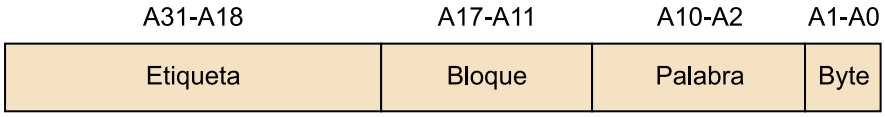
Esta implementación fija el tamaño de la memoria caché igual al de la página en la memoria principal para poder ubicar directamente un bloque en caché dentro de la página en memoria correspondiente. Es decir, independientemente de la página de memoria de donde provenga, el *k*-ésimo bloque de la *i*-ésima página en memoria principal se encontrará siempre en el lugar correspondiente al *k*-ésimo bloque en la memoria caché (podéis verlo en la figura siguiente). Este hecho facilita mucho el movimiento de datos, puesto que solo nos hemos de preocupar por saber a qué página de memoria corresponde cada bloque de datos que hay dentro de la caché. El resto está implícito gracias al mapeo directo.

Sistema de *caching* con mapeo directo



Si tenemos en cuenta que la memoria principal contiene 128 M palabras, 27 bits son suficientes para poder distinguirlas. Si, además, quisiéramos distinguir entre los 4 bytes que forman una palabra, necesitaríamos 2 bits adicionales. Supongamos este último caso (29 bits en total). De estos, los 2 últimos identifican el byte dentro de la palabra, los 9 siguientes la palabra dentro del bloque de datos (512 palabras por bloque) y los 7 siguientes identifican el bloque dentro de la página (128 bloques por página), tal como se puede ver en la figura siguiente:

Interpretación de la dirección desde el punto de vista del sistema de *caching*



Fijémonos en que cuando el procesador quiere acceder a un dato determinado, el valor que toman estos bits es el mismo tanto si vamos a buscar el dato directamente a memoria como a la caché. En caso de ir a buscarlo a la caché, deberemos comprobar que estamos accediendo a la réplica de la página deseada, es decir, tendremos que ver si los 11 bits siguientes coinciden o no.

Una observación

Fijaos en que los 3 bits de más peso no se utilizan y, por lo tanto, quedan a 0.

Esta información adicional se guarda en una estructura especial para este propósito: la **tabla de etiquetas**. Contiene una entrada para cada uno de los bloques que integran la caché (128 en nuestro caso) y se guarda allí la misma etiqueta, que identifica la página de donde proviene el bloque, el **bit de validez**, que indica si los datos del bloque son válidos, y el ***dirty bit***, que indica si se han modificado los datos del bloque. A pesar de que no se utiliza en este algoritmo, también se puede guardar en la tabla un campo temporal, que indica cuándo se ha traído el bloque a la caché o cuándo se ha usado por última vez.

Con todos estos ingredientes, el mecanismo de *caching* es el siguiente. Inicialmente, tenemos todos los bits de validez y los *dirty bit* a FALSE. A partir de aquí, cuando el procesador necesita datos se mira si ya se encuentran en caché y se comprueban la etiqueta y el bit de validez correspondientes. En caso afirmativo, se emplean los datos. En caso negativo, se van a buscar a la memoria principal, siempre copiando todo el bloque de datos donde se encuentra la palabra de interés. Previamente a la copia, hay que verificar si la posición en caché es válida (bit de validez). Si no lo es, se continúa con la operación, pero si es válida, hay que comprobar que no haya habido modificaciones en los datos de la caché (podéis ver el *dirty bit*). En caso afirmativo, copiamos previamente el bloque de caché en la memoria principal para mantener la información actualizada. Este método de actualización a nivel de bloque se denomina *escritura retardada* y se fundamenta en el hecho de que si un dato se ha escrito una vez, puede volver a ser escrito al cabo de poco tiempo y, por lo tanto, múltiples escrituras a memoria retardarían el sistema. Sin embargo, se

puede optar también por la **escritura directa**, es decir, actualizar la memoria principal cada vez que hay un cambio en la caché para mantenerlas *coherentes* en todo momento.

Implementación con mapeo asociativo

Si optamos por un algoritmo de *caching* con mapeo directo, facilitamos la implementación del sistema debido a su simplicidad. Sin embargo, esta técnica puede resultar ineficiente en términos de velocidad de ejecución.

Ejemplo

Imaginemos un lazo de código que utilice dos datos que se encuentran en el bloque 4 de memoria pero en páginas diferentes. Es evidente que durante la ejecución del programa deberemos ir modificando continuamente el bloque 4 de la caché, cuando sería más eficiente tener los dos bloques siempre en caché. El mapeo asociativo soluciona este problema y permite colocar un nuevo bloque de datos en cualquier posición de la memoria caché.

El funcionamiento es parecido al anterior, si bien ahora hay que decidir cuál es el bloque de la caché que hay que sustituir cuando se va a buscar un nuevo dato a la memoria principal. Además, el sistema de busca e identificación se hace más complejo, puesto que trabajaremos con etiquetas más grandes que incluirán tanto los bits que identifican la página, como el bloque (18 bits en nuestro caso). Habitualmente, se hace una busca en paralelo para acelerar el proceso, pero esto añade complejidad y coste al sistema. Fijémonos también en que el mapeo entre la dirección que proporciona el procesador y la dirección en caché no tienen que coincidir como antes y, por lo tanto, deberemos hacer la conversión.

Finalmente, a la hora de decidir cuál es el bloque que se debe sustituir, podemos optar por varias soluciones, que pueden hacer uso de las marcas temporales existentes en la tabla de etiquetas. Dos de los algoritmos más usados son:

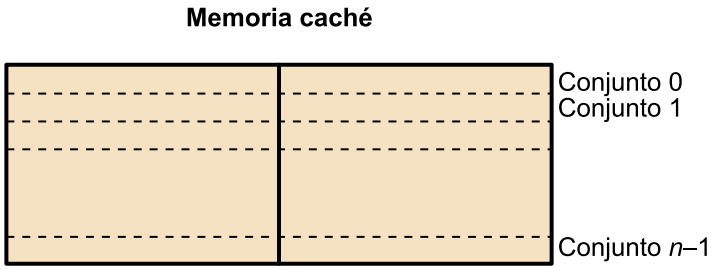
- ***Least recently used*** **(LRU)**: si pensamos que el bloque más antiguo es el que tiene menos probabilidades de volverse a usar, reemplazaremos el bloque más antiguo en caché.

- ***Most recently used*** **(MRU)**: si, por el contrario, creemos que el último bloque usado es el que tiene menos probabilidades de volver a ser necesario, optaremos por esta última solución.

Implementación con mapeo asociativo dentro de un conjunto de bloques

La última solución que vemos está a medio camino entre el mapeo directo y el asociativo y pretende aprovecharse de las ventajas de cada una de las técnicas, es decir, de la flexibilidad del mapeo asociativo y de la simplicidad del mapeo directo. Para conseguirlo, introducimos una nueva organización de la memoria caché. Si antes cada entrada de la tabla de etiquetas identificaba un bloque de la memoria principal, ahora cada fila de la tabla sirve para señalar un conjunto de bloques. En el supuesto que nos ocupa, y optando por conjuntos de dos bloques, tendremos una caché de 64 conjuntos de 2 bloques, tal como muestra la figura siguiente.

Organización de la caché usando mapeo asociativo dentro de un conjunto de bloques

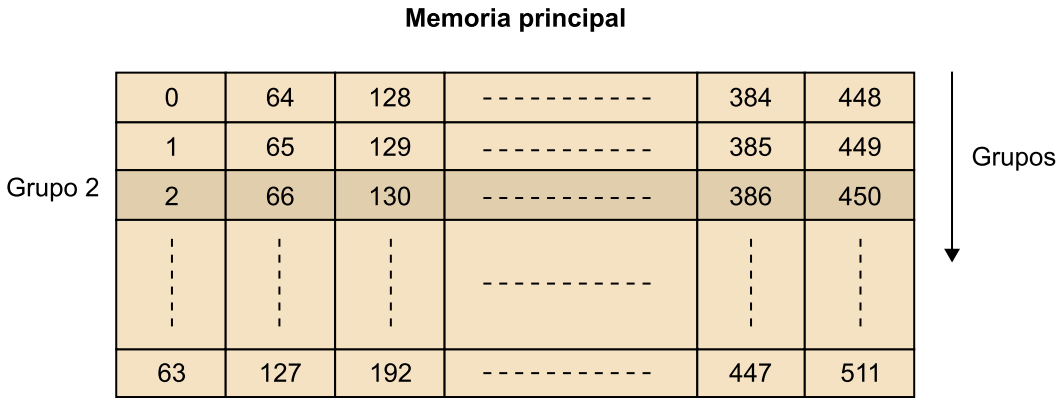


El paso siguiente es ordenar la memoria principal de acuerdo con esta organización de la caché. Lo que hacemos es dividir la memoria en tantos grupos como conjuntos hay en la caché (64 en nuestro caso). Para determinar qué bloques de datos pertenecen a cada grupo, calculamos el módulo del índice del bloque sobre el número de conjuntos.

Ejemplo

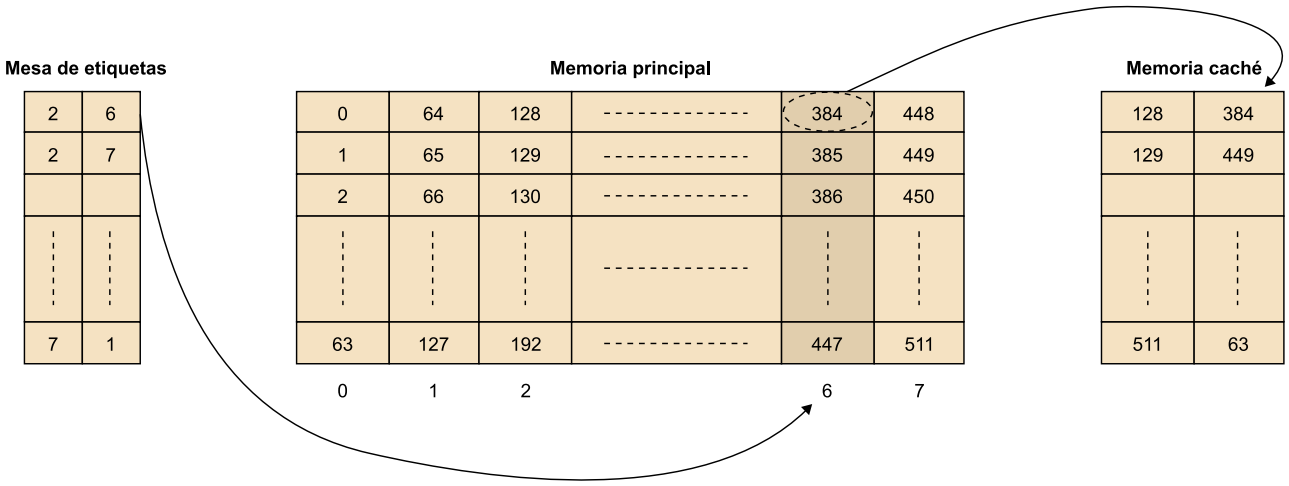
Por ejemplo, el bloque 453 pertenece al grupo 5, puesto que 453 mod 64 = 5. Y, dentro de cada grupo, los bloques que pertenecen a él se ordenan de más pequeño a más grande. Esta organización de la memoria principal en nuestro caso, donde tenemos hasta 256 K bloques, se puede ver en la figura 43.

Organización de la memoria principal usando mapeo asociativo dentro de un conjunto de bloques



Por lo tanto, podremos guardar en la memoria caché hasta 2 bloques de cada grupo (de un total de 4 K). A la hora de buscar un bloque en caché, sabemos que hay un mapeo directo entre grupo de memoria y grupo de caché, mientras que la posición dentro del grupo queda reflejada en la tabla de etiquetas y se debe buscar de manera asociativa, tal como muestra la figura 44. La ventaja es que ahora el espacio de busca es más pequeño (12 bits en el ejemplo) y, por lo tanto, menos complejo.

Funcionamiento del mapeo asociativo con conjuntos de bloques



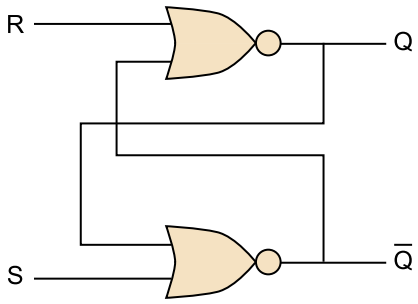
2.6. Registros

Anteriormente hemos hablado y utilizado los registros. Hemos visto que pueden ser internos al procesador, donde tienen un papel importante para que este pueda llevar a cabo sus tareas, y también externos. Nosotros los hemos utilizado para adaptar el bus de direcciones del procesador a un determinado bloque de memoria DRAM multiplexando cada dirección del bus en dos partes: fila y columna. De momento, sabemos que son memorias pequeñas (de unos cuantos bits de ancho) y rápidas. Estos registros tienen la función de almacenamiento de datos, pero también existen los registros de desplazamiento y los *linear feedback shift register* (LFSR).

2.6.1. Registros de almacenamiento

Si vemos con algo más de detenimiento esta parte del hardware, veremos que la pieza básica de cualquier registro es un biestable (capaz de mantener un estado de dos posibles), ya sea de tipos *latch* o de tipos *flip-flop*, y que nos permite guardar un solo bit de información. La agrupación de más de un biestable es lo que forma los registros y habitualmente tienen un tamaño que es potencia de 2, es decir, solemos encontrar registros de 4, 8, 16, 32 o 64 bits de ancho. Hemos visto en las SRAM un *latch* muy sencillo formado por dos inversores. El problema con esta configuración es que la lectura y escritura del bit no es fácil: hay que incluir los transistores de paso y trabajar con las tensiones adecuadas. El *latch* más simple que nos permite interactuar de una manera más fácil es el *latch* Reset Set (RS), que corresponde al circuito lógico de la figura siguiente. El funcionamiento es el que aquí indicamos: si las entradas R y S son 0, la salida Q no cambia y por lo tanto sirve para leer el bit que almacena el *latch*. En el momento en el que una de las entradas toma valor lógico 1, la salida es 0 si se ha activado la entrada R y 1 si se ha activado la entrada S. Esta es la manera de escribir un bit en el *latch*. En caso de que las dos entradas sean 1, el valor de salida es indeterminado.

Latch RS



Por lo tanto, un *latch* es en principio un dispositivo asíncrono, es decir, los cambios en la salida se producen como consecuencia de los cambios en la entrada.

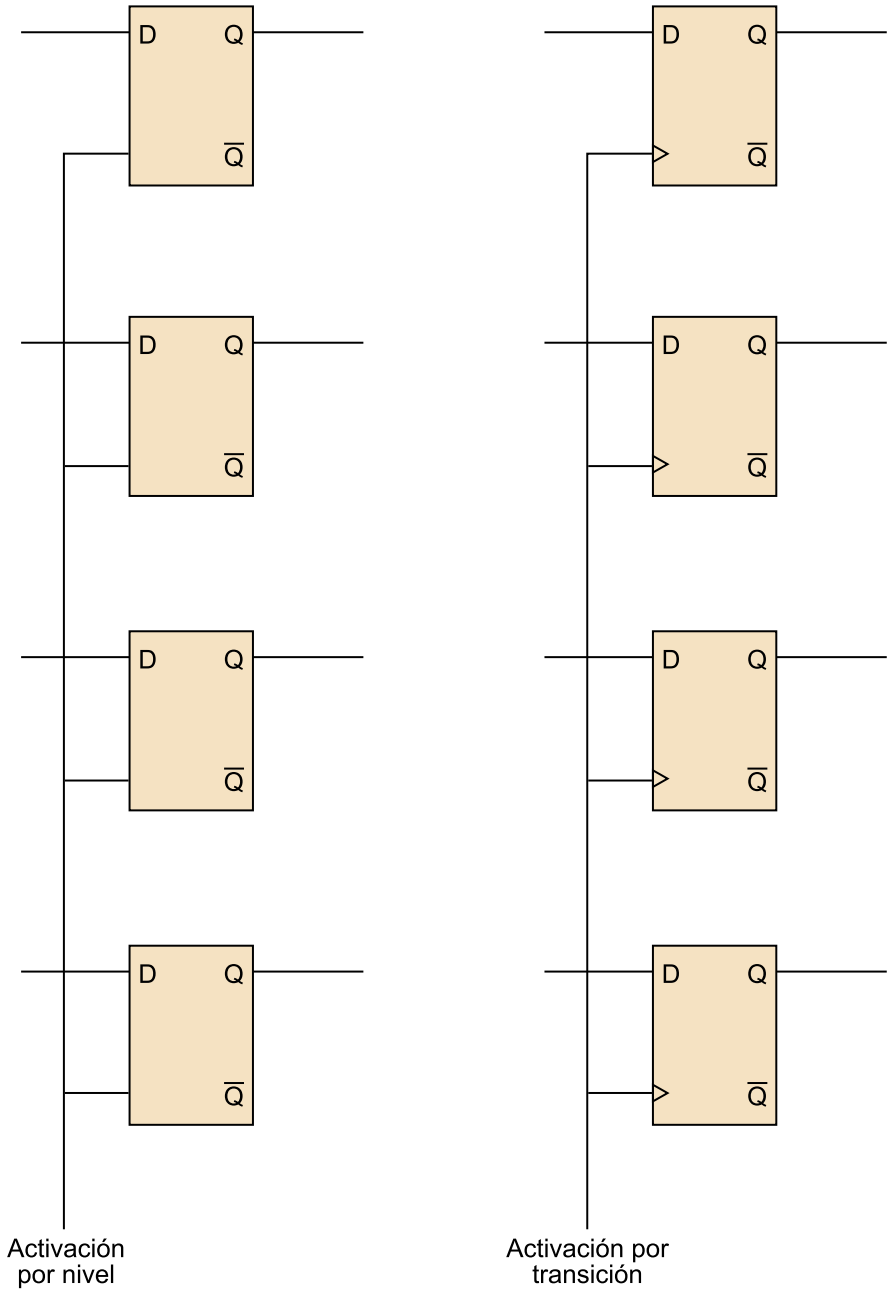
Sin embargo, habitualmente requeriremos cierto tipo de sincronismo, aunque no sea con el reloj del sistema. Por ejemplo, en el diseño anterior de sistema de memoria basado en DRAM hemos utilizado las señales AS y DS para indicar cuándo hay que capturar en los correspondientes registros de dirección y de datos los bits en la entrada. Modificando el *latch* básico de la figura precedente, añadiendo algunas puertas lógicas podemos conseguir el comportamiento deseado, es decir, un dispositivo que almacena el bit en la entrada según una señal de habilitación.

En este caso, decimos que la sincronización es por nivel porque la operación de escritura se lleva a cabo cuando la señal de habilitación toma un valor determinado (alto o bajo dependiente de la implementación). Este *latch* es conocido como *D latch*, donde la *D* proviene de *delay*, puesto que se trata de un dispositivo que retrasa la señal de entrada.

La segunda opción es utilizar *flip-flops*, que son los elementos básicos de los sistemas digitales secuenciales, más complejos que los anteriores *latches*. De hecho, un *flip-flop* de tipo D está formado esencialmente por dos *latches* de tipo RS conectados en cascada, aunque su funcionalidad sea casi idéntica al *D latch*. La diferencia es sutil pero extremamente importante: mientras que el *latch* se activa por nivel, el *flip-flop* se activa por flanco (de subida o bajada según la implementación). Además, es un dispositivo mucho más robusto, puesto que cuando la señal de habilitación (puede ser el reloj) se encuentra a nivel alto, se guarda el bit de entrada en el primer *latch* pero **no se transmite** al segundo (salida del *flip-flop*), lo que no sucede hasta que la señal de activación o reloj pasa de 1 a 0 (si se sincroniza por flanco de bajada). Por lo tanto, la entrada y la salida se encuentran aisladas, es decir, las variaciones en la señal de entrada no pasan a la salida. En la figura siguiente podemos ver un registro de 4 bits implementado con *latches* (izquierda) o con *flip-flops* (derecha). Fijémonos en que son prácticamente iguales y en que trabajan con la misma señal de activación o reloj. La única diferencia entre los dos radica en el triángulo que

hay en los *flip-flop*, que precisamente sirve para indicar la sincronización por flanco. Opcionalmente, puede haber otras señales comunes, como la señal de Reset si nos interesa poder inicializar el registro.

Registro de 4 bits usando *latches* (izquierda) o *flip-flops* (derecha)

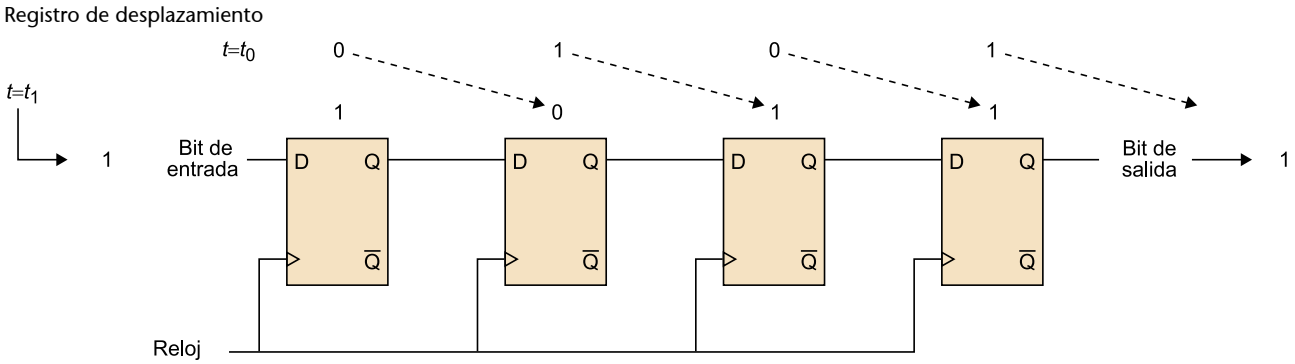


2.6.2. Registros de desplazamiento

En determinadas ocasiones hay que transformar datos que tenemos en paralelo a serie o al revés para adaptarlos, por ejemplo, a un puerto o bus determinados. En este caso, deberemos utilizar los registros de desplazamiento, que, tal como su nombre indica, permiten desplazar los bits almacenados. Un des-

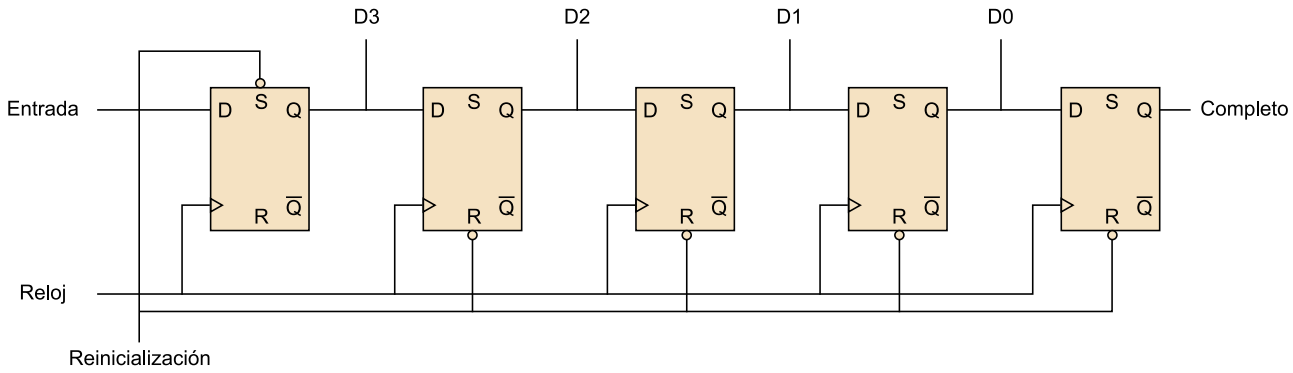
plazamiento de bits puede servir, también, para hacer multiplicaciones o divisiones por potencias de 2 o para generar retardos muy bien controlados que pueden ser útiles en ciertas aplicaciones.

Consideramos primero el registro de desplazamiento con entrada serie y salida serie. Estos tipos de registros se construyen fácilmente conectando *N flip-flops* en cascada con un reloj común. Por ejemplo, la figura siguiente muestra un registro de desplazamiento de 4 bits. El funcionamiento es sencillo: suponemos que en el instante $t = t_0$ los *flip-flops* almacenan los bits 1010 y que el bit de entrada es 1. Por lo tanto, la salida a t_0 es 0 y las entradas de los biestables son 1101, respectivamente. Pasado un flanco de reloj, en el instante t_1, estos serán los valores almacenados por los *flip-flops* y, por lo tanto, lo que veremos a la salida será 1. Es decir, el bit 1 ha entrado al registro desplazando la cadena 101 a la derecha, y ha sacado el bit 1 y eliminado el bit 0 anterior, tal como muestra la figura siguiente. Este comportamiento se repetirá a cada flanco de reloj, lo cual provocará el desplazamiento de un bit cada vez. Aparte, hemos de tener en cuenta que si queremos inicializar el registro a todo 0, deberemos colocar una señal de Reset común a todos los biestables. Es importante remarcar que este tipo de registro no funcionaría con *latches* debido a la transparencia que hay entre entrada y salida y, por lo tanto, en un mismo período, el bit de entrada se podría propagar a más de un biestable.



Este mismo registro de desplazamiento nos sirve para implementar un registro de entrada en serie y salida en paralelo. Simplemente, hemos de leer las salidas de los biestables al mismo tiempo. Opcionalmente, si es necesario multiplexar estos bits en un bus común, situaremos *buffers* de salida de tres estados en cada una de las salidas de los *flip-flops*. De este modo, podremos poner el registro de desplazamiento en estado de alta impedancia cuando no se lean los bits guardados. Sin embargo, deberemos contar los golpes de reloj transcurridos para saber cuándo queda lleno el registro. Para evitarlo, podemos modificar este diseño añadiendo un biestable más, que sirve para marcar si el registro está lleno. Esta variante se puede ver en la figura siguiente:

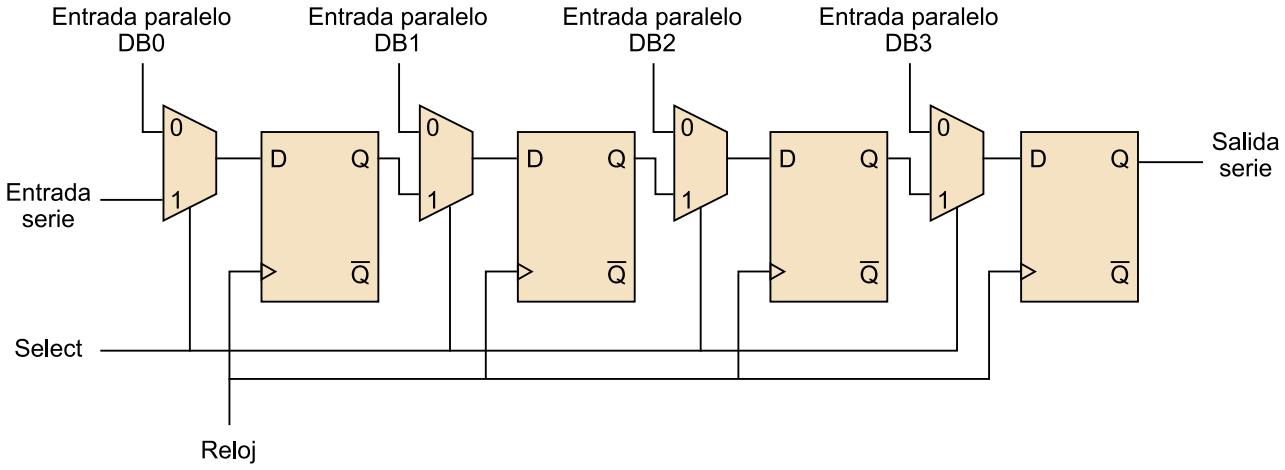
Registro de desplazamiento como convertidor serie a paralelo con bit de marca



Fijémonos en que antes de entrar la palabra al registro, hay que activar la línea Reset. Esto fijará a 1 la salida del primer biestable y a 0 el resto. A partir de aquí, se empieza a introducir la palabra. La señal de completo se mantendrá a 0 hasta que el registro esté pleno, momento en el que podemos hacer la lectura.

Finalmente, vemos el registro de desplazamiento de entrada en paralelo y salida en serie, que podemos observar en la figura siguiente. En este caso, hay que situar un multiplexor de dos entradas y una salida a cada entrada de biestable, todos controlados por una señal de selección. En caso de que la señal tome valor lógico 1, el registro funciona de manera habitual, desplazando un bit a cada golpe de reloj. En cambio, cuando la señal toma valor 0, cargamos los bits de entrada en paralelo a los biestables correspondientes, operación que se hace efectiva en el flanco de reloj siguiente.

Registro de desplazamiento como convertidor paralelo a serie



2.6.3. Registros LFSR

Los registros *linear feedback shift register* (LFSR) son registros de desplazamiento donde la entrada al registro es una combinación lineal de los bits almacenados en cada uno de los biestables. Por lo tanto, no hay una entrada externa al registro de desplazamiento. Además, la combinación lineal no es cualquiera, puesto que los valores que multiplican cada una de las variables de estado son o bien 0 o bien 1, es decir, la entrada es una suma *or exclusiva* de algunos de los bits que guardan los biestables. Por lo tanto, si el registro de desplazamiento se

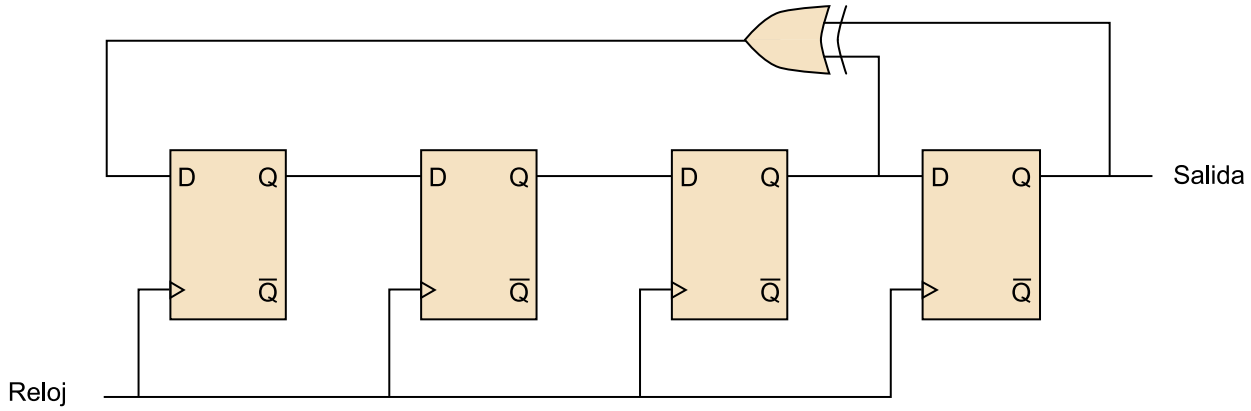
inicializa en el estado de todo 0, la realimentación será siempre 0 y el estado se mantendrá de manera indefinida. Obviamente, esta no es la misión del registro.

En la práctica, estos dispositivos se utilizan para obtener secuencias pseudoaleatorias. Para describir qué variables intervienen en la realimentación se utiliza un polinomio C(x) que marca las conexiones. Este polinomio tiene grado N, es decir, el número de biestables, en el que el término de grado más alto está siempre presente, puesto que marca la entrada al registro (la realimentación). Aparte, al menos uno de los otros coeficientes también debe ser 1 para tomar al menos uno de los bits del registro.

Ejemplo

Por ejemplo, la figura siguiente muestra un registro de cuatro elementos con el polinomio de conexiones $C(x) = 1 + x + x^4$. Evidentemente, la secuencia resultante no es totalmente aleatoria, puesto que llegará un momento en el que repetiremos el estado del registro (el número de estados es finito) y a partir de allí la secuencia se repetirá. Sabemos que la secuencia de longitud máxima que podemos obtener antes de que se empiece a repetir es de $2^N - 1$. Estas secuencias se obtienen siempre con polinomios de conexiones primitivos, es decir, que no se pueden factorizar. Finalmente, hay que comentar que las secuencias resultantes de los LFSR se suelen denominar secuencias *pseudo noise* (PN), dado que tienen la forma de ruido aleatorio. Entre sus aplicaciones, aparte de la que acabamos de mencionar, está la de generar vectores aleatorios para comprobar el funcionamiento correcto de los sistemas, encriptación, codificación (por ejemplo, en sistemas de espectro ensanchado) o sincronización.

Registro LFSR



2.7. Contadores y divisores

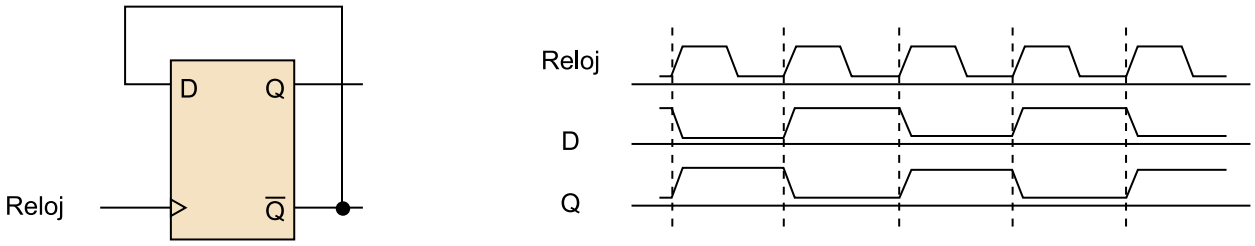
Las tareas de contar y dividir forman parte de muchos sistemas empotrados y han sido esenciales en muchas de sus aplicaciones. Nos interesa contar para acumular eventos, contar bits o determinar si algo ha sucedido un número de veces determinado. También podemos modificar un contador de manera fácil para que cuente de acuerdo con un acontecimiento que se repite periódicamente cada cierto tiempo (un reloj), de manera que se obtiene un temporizador. Los temporizadores nos servirán para medir el tiempo entre dos eventos determinados, crear una cuenta atrás, etc. Finalmente, emplearemos los divisores para generar, a partir de una señal de referencia, otra de frecuencia infe-

rior. A veces, será el mismo procesador quien podrá llevar a cabo estas tareas y en otras deberemos integrar estas funcionalidades dentro de nuestro sistema por medio de circuitos integrados especializados.

2.7.1. Divisores y contadores asíncronos

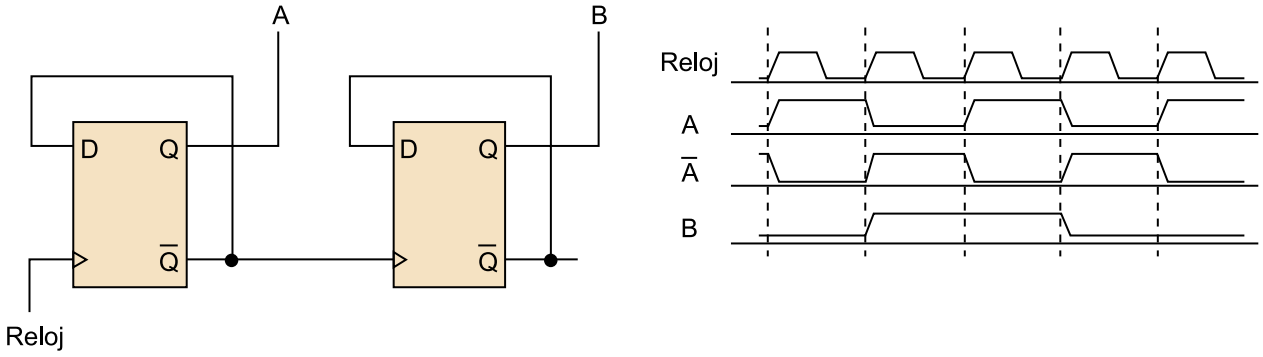
El divisor más sencillo que podemos construir necesita solo un *flip-flop* de tipo D y se obtiene conectando la salida negada a la entrada tal como se puede ver en la figura siguiente (izquierda). De este modo, a cada golpe de reloj actualizamos el valor almacenado por su negado y, como podemos ver en la figura siguiente (derecha), la señal resultante a la salida tiene la mitad de frecuencia que el reloj de la entrada.

Divisor simple por 2



Este diseño se puede ampliar conectando más biestables en cascada, tal como se muestra en la figura siguiente (izquierda) por el caso de dos *flip-flops*. Fijémonos en que la realimentación de un biestable se utiliza como señal de reloj para el siguiente. Así, el funcionamiento del segundo biestable es exactamente igual al del primero, pero los cambios se producen a una frecuencia que es la mitad que la del reloj del primero. Por lo tanto, la señal de salida del segundo *flip-flop* habrá reducido la frecuencia original cuatro veces, tal como se puede ver en la figura que se muestra a continuación (derecha):

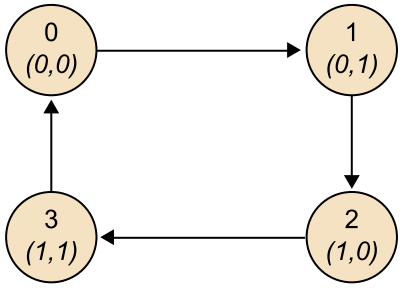
Divisor asíncrono por 4



Fijémonos ahora en la secuencia de valores {(B,A)} que se va obteniendo de manera periódica, que es {(B,A)} = {(0,0), (0,1), (1,0), (1,1)}. Vemos que se trata claramente de un contador de 2 bits. Si lo analizamos como una máquina de estados donde el estado es (B,A), obtenemos el diagrama de estados de la figura siguiente. Decimos que se trata de un contador asíncrono, puesto que los biestables no comparten una misma señal de reloj, sino que la salida de

uno es reloj del otro. Esto provoca que haya un retardo de propagación que se incrementa biestable por biestable y que puede ser significativo dependiendo del número de biestables y del retardo entre la entrada de reloj y la salida en cada uno de ellos. Se trata, por lo tanto, de un efecto tipo dominó que se debe tener en cuenta si queremos descodificar los bits del contador, puesto que sufriremos el riesgo de variaciones a la salida debido a la falta de sincronización entre las salidas de los biestables. En general, es mejor evitar esta práctica.

Diagrama de estados del contador asíncrono de 2 bits



2.7.2. Divisores y contadores síncronos

Si queremos evitar problemas de transitorios no deseados, no nos queda otro remedio que emplear contadores síncronos, donde todos los biestables trabajan con una señal de reloj común. Los diseñaremos del mismo modo que cualquier circuito de lógica secuencial.

Ejemplo

Veamos a continuación el contador de 2 bits. En primer lugar, es evidente que necesitaremos dos *flip-flops* para poder almacenar los dos bits, que denominaremos A y B. Fijémonos en que el diagrama de estados que queremos generar es el de la figura anterior y, por lo tanto, podemos construir la tabla de verdad de las entradas a los biestables, es decir, D_A y D_B, en función de los valores de las variables de estado A y B. Representamos esta información en forma de mapa de Karnaugh en la figura siguiente y usando el método del mismo nombre obtenemos las funciones lógicas que indicamos a continuación:

$$D_A = \overline{A}$$
$$D_B = \overline{A}B + A\overline{B} = A \oplus B$$

Mapas de Karnaugh de las entradas a los biestables D_A y D_B

$$D_A = \bar{A}$$
$$D_B = \bar{A}B + A\bar{B} = A \oplus B$$

D_A

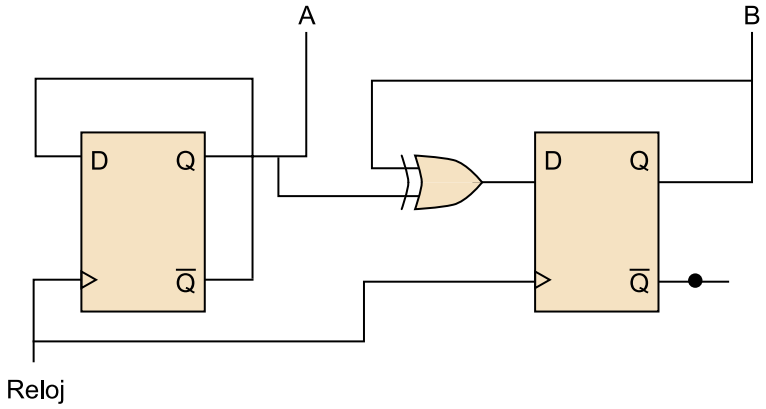
A \ B	0	1
0	1	1
1	0	0

D_B

A \ B	0	1
0	0	1
1	1	0

Esto nos da finalmente el contador síncrono de 2 bits de la figura 55 y, siguiendo el mismo procedimiento, podremos obtener cualquier contador que necesitamos.

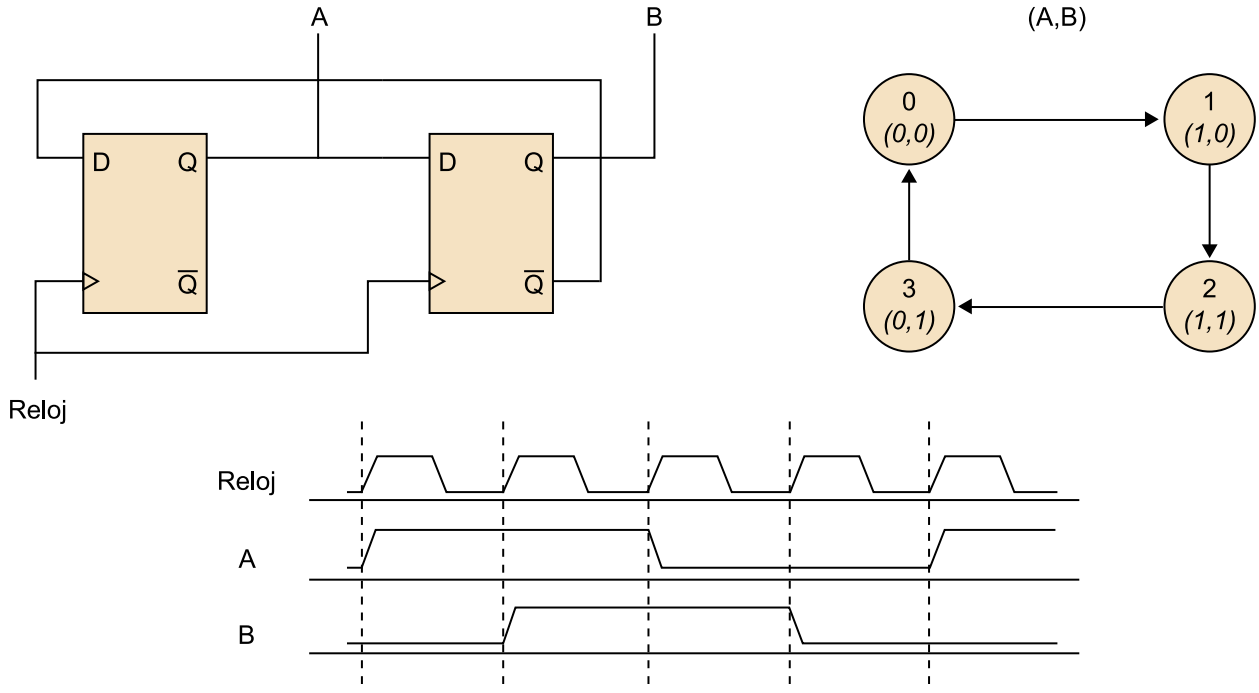
Contador síncrono de 2 bits



2.7.3. El contador de Johnson

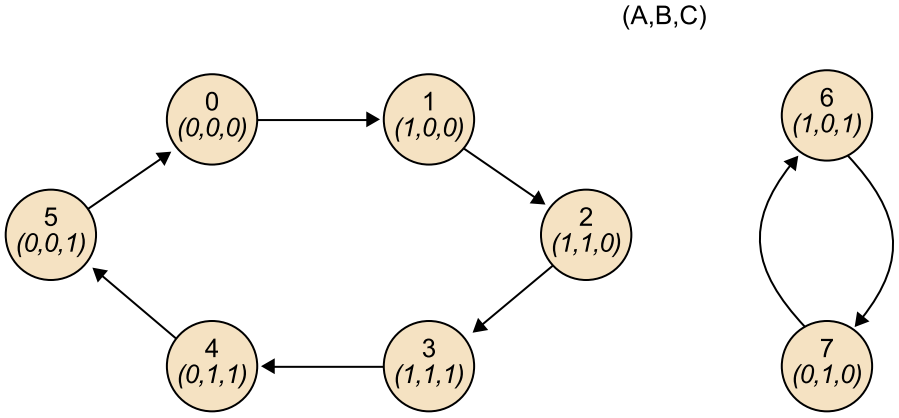
El contador de Johnson es un caso particular de contador que resulta especialmente útil a la hora de diseñar bases temporales para los sistemas empotrados. El diseño es muy simple: se trata solo de un registro de desplazamiento donde se realimenta la entrada con la salida, tal como podemos ver en la figura siguiente para el caso de 2 bits. En la misma figura, encontramos tanto su diagrama de estados como la evolución temporal de las variables de estado A y B. Fijémonos en que el contador de Johnson cambia una de las salidas a cada golpe de reloj y, por lo tanto, para entenderlo propiamente como contador, hemos de interpretar los estados según la codificación de Gray. Fijémonos también en que las salidas A y B se encuentran las dos en mitad de frecuencia que el reloj pero con fases diferentes.

Contador de Johnson de 2 bits (circuito, diagrama de estados y respuesta temporal)



En los contadores de más de 2 bits se debe tener en cuenta una particularidad, por lo que nos fijamos ahora en el contador de 3 bits. En caso de que el contador se inicialice en el estado (A,B,C) = (0,0,0), el comportamiento es parecido al anterior pero recorriendo seis estados diferentes en lugar de ocho. Los dos estados que quedan se van alternando indefinidamente, de manera que si el contador entra en uno de estos dos estados por accidente (ruido u otras causas), el contador deja de funcionar como nosotros queremos. Por ello es importante inicializarlo bien cuando el sistema arranca e incluso evitar que pueda caer en un estado no deseado controlando las entradas a los biestables. En caso de que el funcionamiento sea correcto, podemos ver en el diagrama de estados de la figura siguiente que cada una de las variables proporciona una señal de frecuencia igual a un sexto de la del reloj y con tres fases diferentes.

Diagrama de estados del contador de Johnson de 3 bits



En general, un contador de Johnson de N bits tiene un período de $2N$ en el funcionamiento correcto, mientras que el resto de los estados $2^N - 2N$ forman un subgrafo de estados ilegales que hay que evitar. La secuencia resultante es siempre de tipo Gray y, por lo tanto, nos ayudará si queremos descodificar el estado, puesto que no solo se trata de un contador síncrono, sino que, además, solo una de las entradas cambia a cada golpe de reloj.

2.8. Reloj del sistema

El reloj es un elemento básico de cualquier sistema empotrado y de cualquier circuito digital, puesto que es quien marca el ritmo de funcionamiento. Por ejemplo, entre dos golpes de reloj siempre debe haber el tiempo suficiente para llevar a cabo las operaciones establecidas. Los parámetros que hemos de tener en cuenta a la hora de elegir el reloj son básicamente dos:

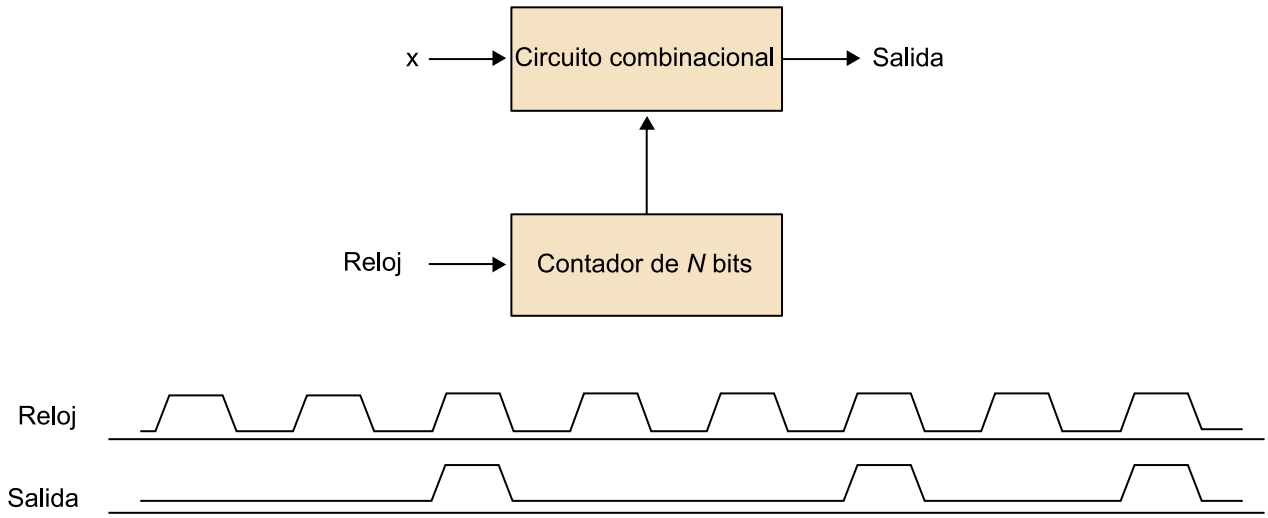
- frecuencia,
- estabilidad y precisión.

A la hora de elegir la frecuencia del reloj, lo más habitual es que sea igual o superior a la necesaria. En el segundo caso, podremos utilizar circuitos más o menos sencillos que permitirán reducir la frecuencia hasta la deseada. Hemos

visto anteriormente que los contadores se pueden utilizar como divisores de frecuencia. Hemos comentado que, en el caso de utilizar contadores asíncronos, no conviene descodificar las salidas del contador porque el retardo entre los diferentes biestables puede causar transitorios no deseados (*gltiches*) que, en último término, pueden provocar un funcionamiento inadecuado del sistema. En la práctica, se utilizan divisores asíncronos de hasta 12-14 bits, lo cual permite reducciones en frecuencia de hasta 16 K. Sin embargo, hemos de tener en cuenta el retardo que la señal de baja frecuencia experimenta respecto al reloj principal.

En definitiva, sabemos cómo dividir la frecuencia por un número que sea potencia de 2 o múltiplo de 2 utilizando los contadores (síncronos o asíncronos) o los contadores de Johnson, respectivamente. Ahora bien, también es posible dividir la frecuencia por un número fraccionario, lo cual se consigue con un contador de *N* bits y un circuito combinacional en el que la entrada también es un número de *N* bits y la salida es la señal de frecuencia reducida, tal como mostramos en la figura siguiente, suponiendo $N = 3$ y un factor de reducción de 3/8.

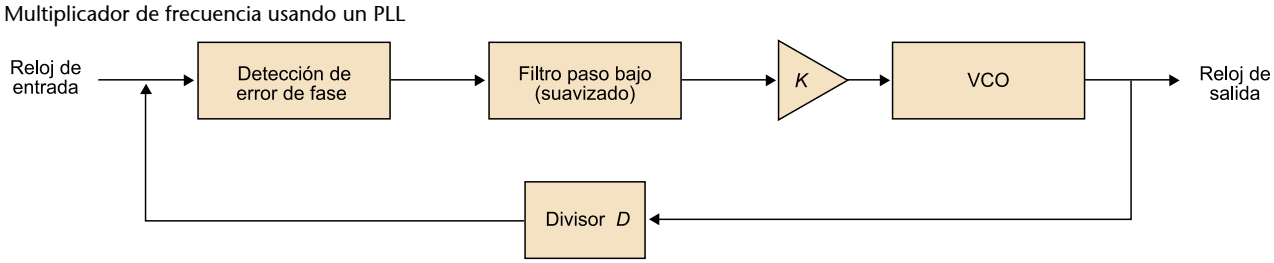
División de la frecuencia de reloj por un número fraccionario



En general, si utilizamos un contador de *N* bits, podremos aplicar factores de reducción del tipo $x/2^N$, donde x es el número, expresado en binario, que situaremos a la entrada del circuito. Para acabar, habrá que diseñar este circuito combinacional para que su respuesta sea 1 solo en x valores que tome el contador, los cuales se deben elegir uniformemente distribuidos entre 0 y $2^N - 1$, tal como vemos en el ejemplo de la figura.

Finalmente, si lo que necesitamos es aumentar la frecuencia del oscilador, deberemos implementar un circuito tipo lazo de seguimiento de fase (PLL, *phased locked loop*) cuyo esquema se puede ver en la figura siguiente, en la que la frecuencia de salida se multiplica por el valor del divisor de frecuencia situado en el lazo de retroalimentación. Para entender el funcionamiento del PLL, consideramos primero que la retroalimentación es directa, esto es, dividimos

por la unidad. En estas circunstancias, la señal de salida se compara con la de entrada, es decir, el reloj del sistema. La diferencia de fase resultante es filtrada paso bajo, esto es, se suaviza esta señal de error, para atacar (aplicando la ganancia adecuada *K*) un VCO, es decir, un oscilador controlado por tensión. El circuito, pues, sigue la señal de entrada y se mantiene sincronizado en fase. Si en el lazo de regreso añadimos un divisor de frecuencia de valor *D*, forzamos que la señal a la salida del VCO tenga una frecuencia *D* veces superior a la del reloj, puesto que es la única manera para que la pueda copiar una vez atraviese el divisor.



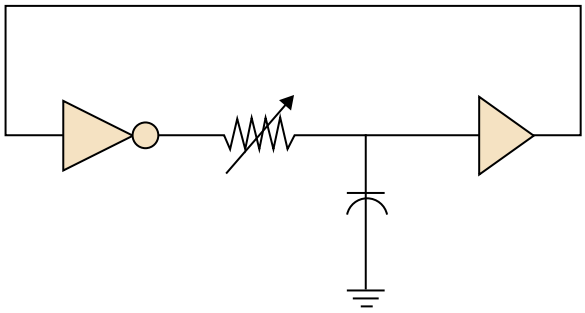
En términos de estabilidad y precisión, la elección del reloj del sistema dependerá de los requisitos de cada aplicación. La estabilidad determina cómo cambia la frecuencia del reloj en el tiempo, mientras que la precisión nos indica cómo se ajusta a su frecuencia nominal. En aplicaciones sencillas puede ser suficiente un reloj basado en resonadores RC (resistencia-condensador). En cambio, en aplicaciones más exigentes, como puede ser el sistema de comunicaciones de un sensor inalámbrico, necesitaremos una base temporal precisa y que no fluctúe en el tiempo. La mayoría de los sistemas utiliza cristales como resonadores, como es el caso del popular reloj de cuarzo. En el caso de necesitar más precisión y estabilidad se suelen estabilizar los cristales en temperatura con pequeños calentadores. Hay otros tipos de relojes, como son los basados en microsistemas (MEMS, *microelectromechanical system*) o los conocidos relojes atómicos. La principal ventaja de utilizar un resonador basado en MEMS es que podemos integrar el reloj en la pastilla de silicio de nuestro integrado y, de cara al futuro, se presenta como un competidor serio a los cristales de cuarzo. Los MEMS tendrían características parecidas a las de cristales. En cambio, si queremos un reloj más preciso y estable, la alternativa son los relojes atómicos. Son los que se usan, por ejemplo, en los satélites GPS. Su aplicación en sistemas empotrados todavía deberá esperar debido a su coste, tamaño y consumo, pero ya ha habido intentos de sintetizar relojes atómicos en un chip.

2.8.1. Algunas consideraciones prácticas

En la mayoría de los casos, optaremos por resonadores basados en cristales, puesto que de entrada obtendremos una base temporal más estable y precisa. Además, nos aseguraremos de que, al utilizar esta base temporal en una copia de nuestro sistema, el funcionamiento continúe siendo el deseado (repetibilidad del diseño). En caso de usar relojes basados en resonadores RC, deberemos utilizar diseños contrastados y comprobar que cumplan los requisitos de

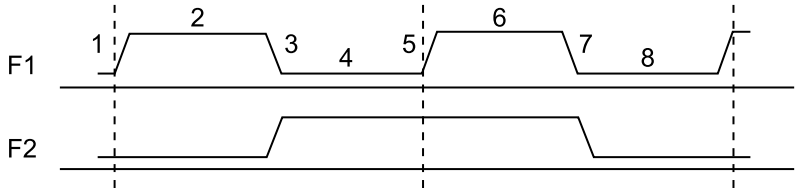
nuestro sistema. Evitaremos, por ejemplo, un diseño como el de la figura 60, que puede llegar a tener un comportamiento metaestable (diferentes puntos de resonancia). Este comportamiento se produce porque el bloque RC afecta significativamente a los tiempos de subida y bajada del *buffer*, que no han sido pensados para trabajar en estas condiciones.

Diseño RC peligroso



En algunos casos nos interesa tener una precisión temporal mejor que la que nos proporciona el reloj del sistema. Fijémonos en que en un reloj normal tenemos cuatro puntos de decisión como mucho, que son: flanco de subida, flanco de bajada, nivel alto y nivel bajo. Si utilizamos un reloj de dos fases, los puntos de decisión se amplían a ocho, puesto que la segunda fase del reloj nos permite distinguir los cuatro puntos de decisión anteriores en dos ciclos consecutivos de la primera fase del reloj, tal como vemos en la figura siguiente. Fijémonos en que los puntos 2 y 6 no se pueden distinguir con una sola fase (F1), pero sí que es posible si tenemos en cuenta la segunda fase (F2). En el caso de utilizar relojes de múltiples fases, procuraremos siempre derivar las distintas fases a partir de una misma base, puesto que de este modo se mantendrán siempre síncronas.

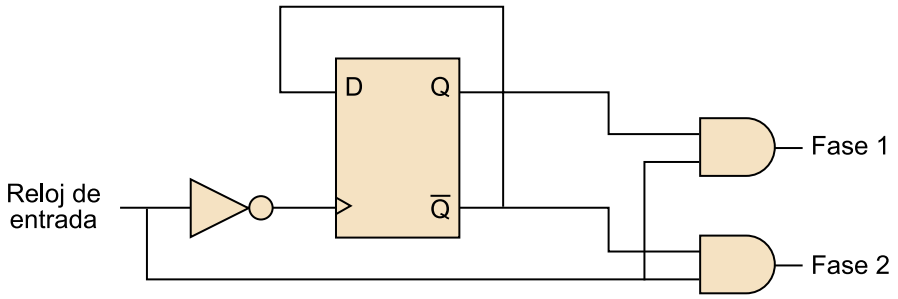
Puntos de decisión en un reloj de dos fases



Hemos visto un ejemplo de uso del reloj de dos fases en el ejemplo anterior de diseño DRAM. En general, podemos usar una fase para marcar los momentos de cambio en el sistema y la otra para guardar los resultados. Esto nos permite dar suficiente tiempo a los diferentes circuitos lógicos para estabilizar sus valores, añadiendo robustez al sistema. Por ejemplo, esta es una buena opción si trabajamos con registros de tipos *latch*, mientras que no es necesario empleando registros basados en *flip-flops*. Una manera sencilla de obtener un reloj de dos fases la podemos ver en la figura. Fijémonos en que, potencialmente, puede haber problemas de transitorios si las señales a la entrada de las puertas

AND llegaran en tiempos parecidos. Sin embargo, el retardo de una de las señales siempre es más grande que el otro, puesto que ha de cruzar el biestable, con lo cual no tendremos nunca picos de señal en las salidas.

Obtención de un reloj de dos fases

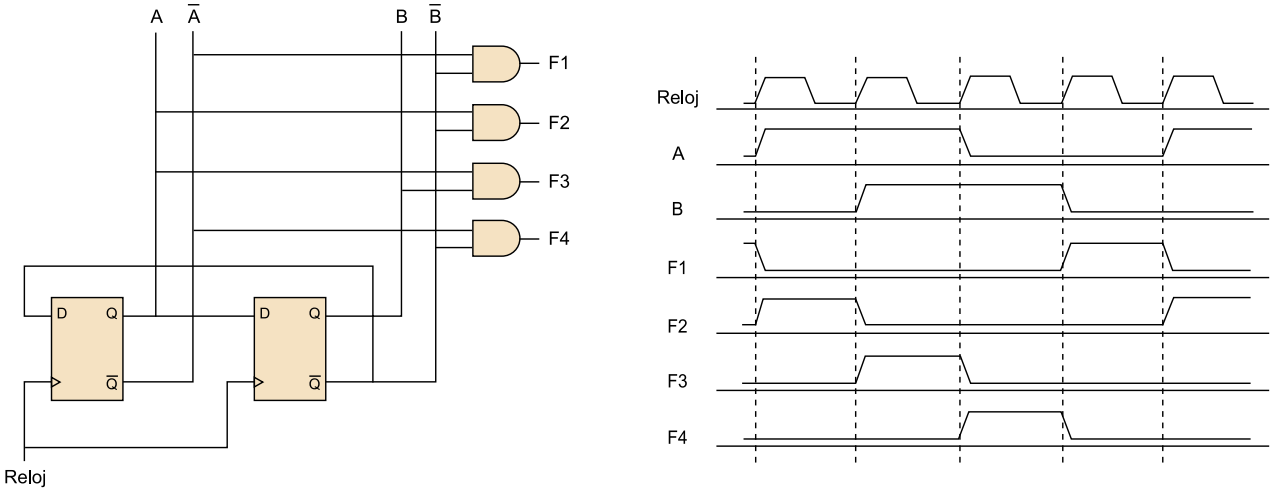


Si queremos incrementar el número de puntos de decisión en un mismo ciclo de reloj, una buena opción es optar por el contador de Johnson y descodificar las salidas para obtener cada una de las fases. Recordemos que gracias a la codificación de Gray solo cambia una variable de estado en cada ciclo del reloj base y, por lo tanto, no tendremos nunca transitorios en las salidas.

Ejemplo

En el ejemplo de la figura siguiente podemos ver el contador de Johnson de 2 bits para obtener un reloj de cuatro fases, y también el diagrama temporal correspondiente.

Obtención de un reloj de dos fases mediante un contador de Johnson de 2 bits



Finalmente, y como hemos comentado, en el caso de utilizar el contador de Johnson para obtener un reloj, nos hemos de asegurar de que este no entre en alguno de los estados ilegales, lo cual sería fatal para el sistema.

3. Comunicación con el exterior

Dependiendo del tipo de aplicación, el sistema empotrado llevará a cabo sus tareas de una manera aislada o, por el contrario, necesitará comunicarse con el exterior, ya sea para intercambiar datos con otros sistemas o para recibir o dar órdenes. En este último caso, hay que proveer al sistema de un mecanismo de comunicación adecuado según las restricciones del equipamiento y las necesidades de la aplicación. En esta sección, se presentan diferentes opciones disponibles para llevar a cabo esto y se discute el grado de aplicabilidad de cada una de ellas. Concretamente, la sección se encuentra dividida en dos partes según el tipo de estrategia utilizada para llevar a cabo la comunicación. En la primera parte, se presentan los mecanismos basados en el uso de cableado, donde la comunicación con el exterior se realiza conectando el sistema empotrado con otros dispositivos exteriores con un cable. Esta conexión, concretamente, se efectúa mediante un punto de acceso que tiene el mismo sistema empotrado, que es conocido con el nombre de *puerto* y varía dependiendo de la tecnología utilizada. En la segunda parte, se discuten estrategias que evitan el uso de cableado para facilitar las tareas de comunicaciones, que, en este caso, son conocidas como *estrategias de comunicación inalámbrica*.

3.1. Puertos

En informática, los **puertos** de un ordenador sirven para crear una interfaz de comunicación entre el propio sistema y otros ordenadores, o entre el sistema y otros dispositivos que actúen como periféricos. La comunicación en sí se lleva a cabo mediante el uso de cable (de diferente naturaleza según el tipo de puerto) y el puerto, concretamente, es el punto de acceso del ordenador donde se conecta este cable. En el caso de los sistemas empotrados, los puertos también se utilizan para ofrecer una interfaz de comunicación del sistema con el exterior, ya sea con otros sistemas empotrados, con periféricos o con otros tipos de hardware.

A continuación, se describen los tipos de puertos más utilizados en los sistemas empotrados, se destacan sus características más importantes y se comentan los aspectos relacionados con su aplicación a este tipo de sistemas.

3.1.1. Puerto serie

Junto con el puerto paralelo, el puerto serie era uno de los puertos por excelencia en los primeros años que aparecieron los PC. Concretamente, este tipo de puerto se utiliza para establecer una comunicación punto por punto entre dos dispositivos. En cuanto al nombre de *puerto serie*, está determinado por el hecho de que solo se puede enviar un bit de datos a cada instante de tiempo. A

pesar de que hay unos cuantos estándares para llevar a cabo comunicaciones mediante el puerto serie, normalmente, cuando se habla de este tipo de puerto (también conocido como *puerto COM*), se considera el estándar RS-232.

En la figura siguiente se presenta, junto con el cableado relacionado, el aspecto del puerto serie basado en RS-232 con el tipo de conector más utilizado, que es de 9 pines y se conoce con el nombre de *DB-9* (existe una versión de 25 pines pero es menos extensa y más cara). Este estándar concreto permite comunicaciones a 20 kBps hasta una distancia de 25 metros (de cable). Sin embargo, hay que señalar que la velocidad que se puede establecer depende de la longitud del cable y, en realidad, se pueden conseguir velocidades más altas (en torno a los 115.200 Bps).

Puerto serie basado en RS-232 con conector de 9 pines DB-9

En el caso de los sistemas empotrados, este puerto se utiliza normalmente para llevar a cabo tareas de programación o depuración del sistema empotrado. Es una opción atractiva en cuanto a coste y simplicidad y, por este motivo, se encuentra muy extendido en una amplia gama de dispositivos. Sin embargo, tiene como limitación la baja velocidad y el tamaño del puerto en sí; por este motivo, se usan actualmente también otros alternativas para hacer estas tareas, como es el caso del puerto USB (que se presenta más adelante).

3.1.2. Puerto paralelo

Como se ha comentado en el caso anterior, el puerto paralelo es otro de los puertos clásicos en la informática convencional. Normalmente, se utilizaba para conectar el PC con impresoras. Sin embargo, este uso se está dejando actualmente al puerto USB.

Este tipo de puerto también establece una conexión punto por punto pero, a diferencia del puerto serie, en este caso se permite la transmisión de hasta 8 bits en paralelo (por ello se denomina así el puerto). También hay que comentar que sigue el estándar de comunicación IEEE 1284, el cual permite velocidades de comunicación que oscilan entre los 4 y los 16 MBps a unas distancias de entre 6 y 10 metros (según la calidad del cable). En la figura 65 se presenta el puerto paralelo.

Puerto paralelo

En cuanto a los sistemas empotrados, este tipo de puerto no se suele utilizar mucho. Es decir, raramente este tipo de puerto forma parte del hardware de un sistema empotrado. A veces, se incluye porque el sistema se tiene que conectar con una impresora u otro tipo de periférico, pero cada vez es más extraño encontrar este tipo de aplicación con la gran implantación del puerto USB.

Sin embargo, hay que comentar que una aplicación interesante del puerto paralelo en un sistema empotrado es su utilización para la fase de desarrollo, concretamente para llevar a cabo tareas de depuración del funcionamiento del sistema. Para hacer esto, se deben introducir órdenes de llamada al puerto paralelo en diferentes partes del código y, como el puerto paralelo permite que el último valor del pin se mantenga hasta que este no cambie, uno puede observar el desarrollo del código en el sistema cómodamente. Esta tarea viene facilitada por el hecho de que muchos sistemas operativos (como el caso de Linux) ofrece controladores (*drivers*) para poder llevar a cabo estas tareas sin dificultad.

Ejemplo

Es muy fácil insertar LED en los pines del puerto paralelo y usarlos para indicar en qué línea del código está.

3.1.3. USB (*universal serial bus*)

Este puerto, desarrollado por el consorcio de compañías USB Implementers Forum (USB-IF), nació con el objetivo de sustituir a los puertos paralelo y serie. El tipo de puerto utilizado es de un tamaño más reducido (tal como se puede observar en la figura siguiente) y sus principales características son el bajo coste de la solución, las velocidades de datos más altas que se pueden conseguir y la facilidad de interconexión entre dispositivos, que se hace siguiendo la filosofía de la integración automática, con una configuración simplificada y sin la necesidad de reiniciar el sistema (concepto conocido como ***plug-and-play***).

Puerto USB

El tipo de conexión USB se basa en el uso de una topología en estrella donde un dispositivo hace de ordenador central (*host*) y el resto se conectan a él. Además, se ofrece la posibilidad de que un ordenador central se conecte a otro ordenador central, lo cual da como resultado la creación de una topología en árbol con diferentes ramas con un máximo de cinco niveles y de 127 dispositivos conectados al ordenador central principal (o *root host*). A pesar de que este número máximo de dispositivos puede parecer limitante en redes de gran escala, esto no suele representar un problema en las típicas configuraciones encontradas en los sistemas empotrados.

En cuanto a la velocidad, hay diferentes límites, que se pueden conseguir con longitudes de cables no superiores a 5 metros, y estos dependen de la versión del puerto:

- USB 1.0: 1,5 MBps.
- USB 1.1: 12 MBps.
- USB 2.0: 480 MBps, pero normalmente se trabaja a 125 MBps (esta es la versión de USB más extendida actualmente).
- USB 3.0: 4,8 GBps (con una longitud de cable recomendable de 3 metros en este caso). Hay que mencionar que esta tecnología no se ha acabado de

instaurar comercialmente a causa, en parte, del incremento de coste que implica.

Otra de las ventajas de este tipo de puerto es que los dispositivos conectados pueden ser alimentados por el mismo puerto, siempre que el nivel de alimentación requerida por el dispositivo no sea demasiado elevado.

Fruto de las ventajas comentadas, el puerto USB es hoy en día una de las soluciones más comunes para poder ofrecer interconexión de numerosos dispositivos y se ha convertido, en muchos casos, en puerto estándar, como sucede en el caso de las impresoras y cámaras digitales.

Ejemplo

Entre los dispositivos que se pueden conectar mediante el USB, podemos mencionar impresoras, cámaras digitales, teléfonos móviles, cámaras de vídeo, dispositivos de almacenamiento, grabadoras, teclados, ratones, etc.

A pesar de que este tipo de puerto está principalmente orientado a la informática de consumo, su utilización en sistemas empotrados ha proliferado en los últimos años. Tal como se ha comentado anteriormente, el puerto USB se está utilizando como sustitutivo del puerto serie para llevar a cabo tareas de programación y depuración, ya sea porque el conector es más simple, por la posibilidad de detección y configuración automática, por el bajo coste, por la posibilidad de utilizar el mismo puerto para alimentar el sistema y porque tiene más velocidad.

Otra utilidad del puerto USB, dada la mayor velocidad que puede ofrecer al sistema, es la conexión del sistema empotrado con otro dispositivo para transferir datos.

Ejemplo

Un ejemplo de esto podría ser el caso de que el sistema empotrado estuviera dedicado a la monitorización de un acontecimiento físico con una complejidad de hardware limitada, para ofrecer una solución de bajo coste y con bajo consumo energético. Este ejemplo se encuentra comúnmente en redes de sensores, que son redes designadas para el control y la monitorización de eventos físicos y están formadas por un alto número de nodos de baja complejidad con la finalidad específica de tomar medidas. En este caso, el sistema empotrado (nodo de la red) se puede conectar mediante USB con otro dispositivo más complejo encargado de recoger los datos medidos para analizarlos y procesarlos.

Por otro lado, dadas las posibilidades de interconexión de este sistema, se puede establecer una red de recogida de datos de los diferentes dispositivos empotrados mediante el uso de USB. Siguiendo con el ejemplo anterior, se podría formar una red tipo estrella en la que los diferentes sensores se conectarían con el dispositivo recolector de datos.

Finalmente, dada la gran popularidad del puerto USB en los dispositivos de consumo, este puerto se puede utilizar para conectar dispositivos adicionales al sistema empotrado, como pueden ser cámaras digitales, de vídeo u otros tipos de periféricos según la aplicación del dispositivo en sí.

Ejemplo

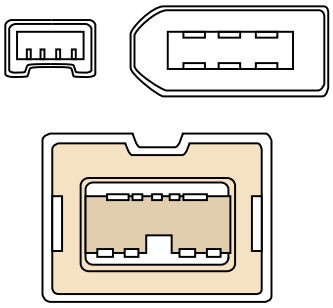
Como muestra de esto, podríamos señalar las tareas de monitorización o vigilancia.

3.1.4. Firewire

La tecnología de este tipo de puerto fue originalmente creada por la compañía Apple en la década de 1980 (concretamente, fue iniciada en 1986) para ser posteriormente trasladada al IEEE, lo que dio como resultado el estándar IEEE1394 (finalizado en el año 1995). Básicamente, el objetivo de esta tecnología era parecido al de la USB: ofrecer altas velocidades a bajo coste. Por sus prestaciones, Firewire se utiliza normalmente en dispositivos relacionados con aplicaciones de audio y vídeo. Hay varias versiones del estándar, y cada una de ellas ofrece diferentes velocidades límite, como el Firewire 400 (versión original con 400 MBps), el Firewire 800 (versión con 800 MBps) y las versiones Firewire s1600 y s3200 (con 1,6 GBps y 3,2 GBps, respectivamente).

Las conexiones entre dispositivos se efectúa siguiendo una topología en árbol pero, en este caso, no es necesario que uno de los dispositivos actúe como máster de la red. Es decir, se pueden hacer transmisiones directas entre dispositivos sin la necesidad de que alguno de ellos u otro de la red actúe como máster y controle el protocolo de comunicaciones. Concretamente, en este caso, se utiliza un sistema de comunicación puramente de igual a igual (*peer-to-peer*). Hay que mencionar que el máximo número de dispositivos que se pueden conectar es de 63, con unas longitudes de cable por conexión de unos 4,5 metros.

Actualmente, las versiones más utilizadas son Firewire 400 y 800. En la figura siguiente se muestran los conectores utilizados en estas configuraciones. En el caso del Firewire 400, se pueden utilizar conectores con 6 o con 4 pines. En el caso de 6 pines, 4 son utilizados para enviar datos y los 2 restantes para proveer de alimentación al dispositivo conectado (hasta 45 W), de manera que se evita su alimentación externa en los casos en los que no se necesiten niveles de alimentación más altos. En el caso de los 4 pines, los pines de alimentación son eliminados. En el caso de Firewire 800, se utilizan conectores de 9 pines (que se adopta también en versiones superiores del estándar), de los que 6 son para datos y 3 para alimentación. A pesar de que las diferentes versiones de Firewire son compatibles entre ellas, para poder conectar dispositivos Firewire 400 con dispositivos de otras versiones hay que usar adaptadores, que se pueden encontrar fácilmente y son proveídos por varias compañías. A diferencia de USB, la velocidad de una red Firewire no disminuye cuando un dispositivo de velocidad inferior es introducido en la red. En todo caso, disminuyen solo las velocidades de transferencia hacia el dispositivo, o desde este dispositivo, de velocidad inferior.



Conectores utilizados en Firewire
Izquierda: conector de 4 pines para Firewire 400. Centro: conector de 6 pines para Firewire 400. Derecha: conector de 9 pines para Firewire 800.

A pesar de que las últimas versiones de USB ofrecen velocidades nominales semejantes a Firewire (por ejemplo, 480 MBps del USB 2.0 frente a 400 MBps del Firewire 400), el último sistema es, en la práctica, más eficiente en cuanto a reducción del tiempo de transferencia para enviar un mismo volumen de datos. Esto se debe al hecho de que la conexión de Firewire es tipo de igual a igual y, por lo tanto, reduce la complejidad del sistema. El sistema USB, al ba-

sarse en una configuración máster-esclavo, sufre unos retardos extra relacionados con tareas llevadas a cabo por el protocolo de comunicación, además de requerir un consumo más grande de CPU. Por otro lado, el hecho de que una red Firewire no requiera un nodo que actúe como máster y que se puedan hacer conexiones directas entre dos o varios equipos ofrece un potencial de gran valor para establecer redes de alta velocidad y de bajo coste.

En el caso de los sistemas empotrados, sin embargo, el uso de este tipo de puerto no es muy frecuente. Cuando es utilizado, se suele hacer para llevar a cabo tareas de depuración del sistema aprovechando el alto volumen de datos que se pueden transferir. Aun así, hay que señalar que su utilización podrá extenderse en el futuro si aumenta la demanda de aplicaciones que requieren altas velocidades.

3.1.5. Ethernet

La creación de Ethernet data de la década de 1970 y su estandarización viene determinada por la norma IEEE 802.3. Ethernet ofrece diferentes variantes que trabajan a distintas velocidades; las versiones más populares son las de 10 MBps, 100 MBps y 1 GBps. Estas variantes utilizan también diferentes versiones de cableado físico y ofrecen, por lo tanto, distintos comportamientos que se ven reflejados en la longitud máxima que pueden tener los cables.

Ejemplo

En el caso de 10 MBps, algunas variantes son las referidas como 10Base2 (cable coaxial con longitud máxima de 185 m) y 10BaseT (par trenzado con 100 m); esta última es la variante más utilizada.

El acceso al medio es compartido y se emplea el protocolo de acceso múltiple con escucha de portadora y detección de colisión (CSMA-CD, *carrier sense multiple access with collision detection*). Los dispositivos que utilizan este protocolo escuchan el medio antes de transmitir sus paquetes para ver si hay alguna otra transmisión activa y, por lo tanto, para evitar colisiones. A la vez que va transmitiendo los datos, el dispositivo escucha también el medio para ver si hay colisión con otro dispositivo, es decir, ver si otro dispositivo también ha empezado a transmitir. Si se detecta una colisión, se para la transmisión y no se vuelve a intentar transmitir hasta que pasa un tiempo aleatorio.

Inicialmente, Ethernet se basaba en *colgar* todos los dispositivos del mismo cable (cable coaxial) y, por este motivo, se usaba este tipo de protocolo de acceso compartido. Este sistema ofrecía muy buenos resultados pero era bastante ineficiente cuando se aplicaba en redes relativamente grandes. Para reducir el número de colisiones y hacer la red más robusta, se cambió a una topología en estrella en la que todos los dispositivos se conectaban a un nodo central. En cuanto a este nodo central, hay dispositivos de diferente naturaleza para llevar a cabo esta tarea y ofrecen la posibilidad de dividir la red en diferentes segmentos (o subredes), y por lo tanto se dividen el tráfico global y facilitan la escalabilidad. Según las funcionalidades que incorporan, los dispositivos más comunes son:

- **Concentrador (*hub*)**: actúa como unión física de los diferentes dispositivos. Es decir, es un simple repetidor; si recibe un paquete de un dispositivo, lo retransmite a todos los dispositivos que tiene conectados.

- **Conmutador (*switch*)**: es un dispositivo más sofisticado que el concentrador, puesto que analiza el paquete recibido para ver la dirección de destino. De este modo, reenvia solo el paquete al dispositivo de destino o al segmento donde se encuentra el dispositivo de destino o el segmento donde se encuentra.

- **Encaminador (*router*)**: es un tipo de dispositivo que se suele utilizar en redes grandes donde diferentes subredes (o segmentos) están interconectadas. Básicamente, tiene la misma funcionalidad que el conmutador, pero también tiene la habilidad de encaminar el paquete por la mejor salida en términos de camino más corto y menos congestionado.

Ethernet es una de las opciones más sencillas y con un coste más bajo de las existentes para montar una red de dispositivos empotrados con una velocidad relativamente alta. Sin duda, el puerto Ethernet es el más utilizado para conectar dispositivos en redes de computadores y por esta razón este tipo de puerto también se encuentra implementado en muchas soluciones empotradas. Básicamente, ofrece la posibilidad de conectar el dispositivo de manera sencilla a otros computadores, impresoras, bases de datos e Internet.

Ejemplo

Una opción interesante es utilizar Ethernet para conectar el sistema empotrado a Internet para ofrecer una solución remota de monitorización y control.

Cabe comentar que, hace tiempo, incluir Ethernet en un sistema empotrado era bastante complicado porque un dispositivo basado en este estándar tenía una alta complejidad, tanto en cuanto al circuito como a los componentes requeridos. Sin embargo, esta tarea se ha facilitado bastante, dado el alto nivel de integración logrado últimamente. En concreto, hay implementaciones de chips de controladores Ethernet específicamente diseñadas para su implementación en sistemas empotrados y tienen como característica principal ofrecer interfaces Ethernet de bajo coste y simples. Las dos más populares son las dadas por los chips Realtek 8019 y el Cirrus CS8900A. No obstante, la mayoría de estas implementaciones trabajan normalmente a 10 MBps.

3.1.6. CAN

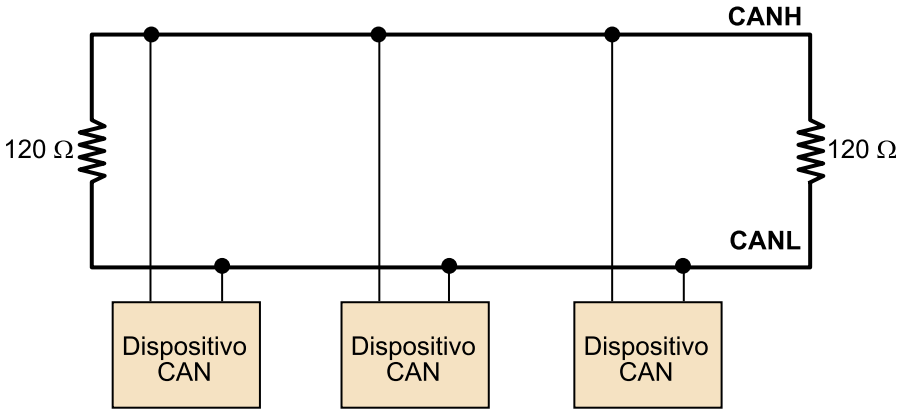
Esta sección la concluimos presentando una tecnología utilizada en entornos industriales. Estos tipos de entornos están caracterizados por su hostilidad en términos de ruido electromagnético severo. Por ello, son diseñados teniendo en cuenta otros criterios. En esta sección concreta, nos centramos en la tecnología *controller area network* (CAN), que se suele utilizar en entornos severamente afectados por interferencia electromagnética.

En las décadas de 1970 y 1980, la automoción sufrió un cambio importante en el sentido de que los automóviles disponían cada vez de más dispositivos electrónicos, como es el caso del ABS, la transmisión electrónica, el sistema de

climatización, el cierre centralizado, etc. Naturalmente, todas estas partes de los vehículos se debían interconectar para poder llevar a cabo una interacción entre ellos, ya sea para llevar a cabo órdenes de control o para intercambiar datos. Originalmente, todos estos dispositivos se conectaban usando cableado punto por punto, pero esto limitaba tanto las prestaciones del sistema como su escalabilidad y, además, aumentaban el coste del vehículo y su peso. Para solucionar esto, la empresa Bosch desarrolló CAN a finales de la década de 1980. Aunque estaba orientado principalmente para la industria de la automoción, fue adoptado después por otras aplicaciones industriales. En particular, CAN está confirmada por la International Standards Organization (ISO) en forma de la estándar ISO 11898 e ISO 11518-2.

La solución CAN se basa en reemplazar la complejidad del múltiple cableado por el uso de un único bus en el que todos los dispositivos se conectan y en el que se pueden conseguir velocidades de hasta 1 MBps con cables de longitud inferiores a 40 m (el mínimo de velocidad es de 40 kBps, que se puede lograr con cables de 1 km). Concretamente, este bus consiste simplemente en una línea balanceada formada por un par de cables (podéis ver la figura siguiente). En la figura, se puede observar también que las líneas están terminades con resistencias de 120 Ω y el motivo de esto es evitar reflexiones de las transmisiones efectuadas sobre la línea.

Red CAN



En una red CAN, los dispositivos se pueden añadir o desconectar en cualquier momento y se permite que diferentes dispositivos actúen como máster, donde cada uno de estos másteres se puede encargar del control local de la red. A pesar de ello, el tipo de acceso al bus se lleva a cabo usando un mecanismo de contención basado en CSMA/CD + AMP (*carrier sense multiple access with collision detection and arbitration message priority*). Básicamente, este esquema es similar al caso de CSMA/CD utilizado en Ethernet en el sentido de que un nodo mira la línea antes de transmitir para ver si está libre. Si es el caso, va transmitiendo los diferentes bits de información y, a la vez, va comparando si los bits existentes en la línea son los mismos que los que va transmitiendo. En caso de que no sea así, es porque hay colisión. Entonces se para la transmisión y se inicia un período de arbitraje que no se da en el protocolo CSMA/CD, y por ello se denomina CSMA/CD+AMP. Este arbitraje se basa en dar paso al

dispositivo con prioridad más alta, en el que la prioridad de los diferentes dispositivos ha sido asignada durante el diseño del sistema. En cuanto al número de dispositivos que se pueden conectar a una línea, el máximo es 30.

Teniendo en cuenta que en un automóvil se encuentra, desde el punto de vista electromagnético, en un entorno muy ruidoso dada la presencia de motores eléctricos, sistemas de ignición, emisiones de radio, etc., la comunicación establecida tiene que ser altamente robusta. Para poder ofrecer esto, se emplea un sistema de comunicación balanceada. Cuando se usa este tipo de comunicación, la información se envía variando los niveles de voltaje de dos líneas.

En este caso, las líneas utilizadas son las líneas CANH y CANL (podéis verlas en la figura anterior). En el receptor se mide la tensión diferencial de estas dos líneas, es decir, se mide la tensión CANH-CANL para determinar el bit enviado. Esto difiere de esquemas convencionales en los que el receptor extrae la información mirando el voltaje de una sola línea. En el caso de CAN concreto, un bit 0 se envía induciendo las tensiones 3,5 V a la línea CANH y 1,5 V a la línea CANL. Por lo tanto, en el receptor se tendrá una tensión diferencial CANH-CANL igual a 2 V. En el caso de querer enviarse un bit 1, las dos líneas se ponen a 2,5 V teniendo como resultado una tensión diferencial de 0 V. El objetivo para utilizar este mecanismo diferencial es reducir el ruido inducido por otras fuentes electromagnéticas. Es decir, si hay un ruido de este tipo, este será inducido del mismo modo a las dos líneas CANH y CANL. Por lo tanto, al obtener la tensión diferencial CANH-CANL, este ruido será compensado.

Ejemplo

Si se quiere enviar un 0 lógico, CANH será igual a 3,5 V y CANL a 1,5 V. Si tenemos un ruido inducido a un instante de tiempo igual a +1 V, lo que se recibe de las líneas CANH y CANL será 4,5 V y 2,5 V, respectivamente. Al obtener la tensión diferencial, sin embargo, se obtendrán los 2 V nominales correspondientes a un bit 0.

El tipo de conector utilizado en CAN es el mismo que el utilizado en el puerto serie, es decir, se utiliza el conector DB9. Sin embargo, es importante remarcar que no hay compatibilidad entre estos dos sistemas. En la tabla siguiente, se indica la señal que trae cada pin en el caso de CAN.

Número de pin	Utilización
1	No utilizado
2	CANL
3	GND (masa)
4	No utilizado
5	No utilizado
6	GND (masa)
7	CANH
8	No utilizado
9	Alimentación opcional

Hay que comentar que este tipo de puerto se encuentra muy a menudo en dispositivos utilizados en entornos industriales. Por lo tanto, es una opción muy adecuada si la aplicación del sistema empotrado es de este tipo. Sobre todo en caso de que trabaje en un entorno muy hostil en términos de ruido e interferencia, y la limitación de velocidad no sea un impedimento para la aplicación concreta.

3.2. Comunicación inalámbrica

Las comunicaciones inalámbricas, tal como indica su nombre, son aquellos tipos de comunicaciones que posibilitan la interconexión de dos o más equipos sin la necesidad de usar cableado. Su utilización es cada vez más común en todos los tipos de aparatos, debido a la comodidad que ofrecen. En el caso de un sistema empotrado, este tipo de comunicaciones simplifica enormemente la interoperabilidad y las tareas de comunicaciones. Hay que decir también que los sistemas empotrados pueden trabajar en escenarios con muy poca accesibilidad o en entornos en los que el uso de cableado represente un gran impedimento (por ejemplo, en aplicaciones de monitorización del medio ambiente o de la fauna). Por lo tanto, las comunicaciones inalámbricas ofrecen la posibilidad de que el sistema empotrado se pueda comunicar con el exterior de una manera eficiente e, incluso, pueden ser la única solución viable en algunas circunstancias.

3.2.1. IRDA

Este tipo de tecnología está basada en el uso de comunicación vía infrarrojos. Tal como se verá a continuación, sus características difieren bastante del resto de las tecnologías presentadas en esta sección, que trabajan con comunicaciones vía radio.

La estandarización Infrared data association (IRDA) va ligada a un consorcio de empresas que se estableció en 1993 para promover una tecnología de comunicación de bajo coste y basada en el uso de infrarrojos. El origen de esta tecnología proviene de los enlaces de comunicaciones por infrarrojo que Hewlett-Packard incorporó a sus calculadoras.

Para la transmisión de los datos, hay tres categorías:

- Serial infrared (SIR) con velocidades hasta 115,2 kBps.
- Medium speed infrared (MIR) con velocidades máximas de 1.152.
- MBps y fast infrared (FIR) que puede llegar a los 4,0 MBps.

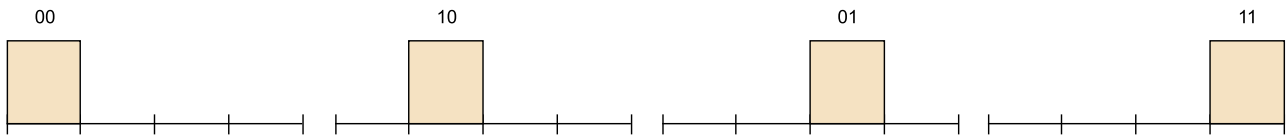
Para llevar a cabo la comunicación de datos en los casos SIR y MIR, esta se hace en serie y mediante la modulación de los bits con pulsos de luz tal como se describe a continuación:

- Si se quiere enviar un 0, el tiempo de transmisión se divide en dos partes: una inicial con un tiempo igual a 3/16 del tiempo de bit total y una segunda parte con el resto del tiempo de bit. Durante la primera parte se emite un pulso de luz de intensidad continua, mientras que en la segunda no se emite nada.

- Si se quiere enviar un 1, no se emite luz durante el tiempo de transmisión.

Sin embargo, en caso de que se transmita a 4 MBps, la modulación tiene una pequeña variación. En este caso, el tiempo de transmisión se divide en cuatro partes y se utiliza para enviar dos bits simultáneamente. Según los bits que se quieran transmitir, solo una de las cuatro fracciones se utiliza para emitir luz, tal como se describe en la figura siguiente:

Pulsos utilizados en IRDA trabajando a 4 MBps

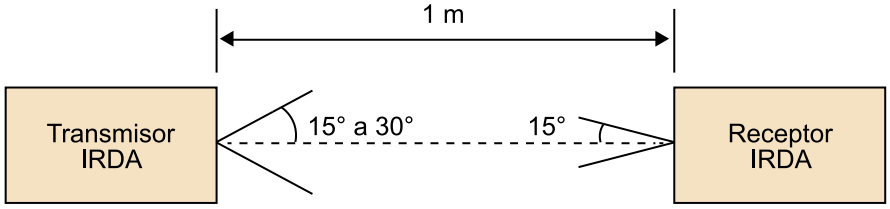


Cada tiempo de transmisión se utiliza para enviar 2 bits y hay cuatro combinaciones diferentes de pulsos.

En cuanto a la cobertura, se encuentra bastante limitada (tal como se presenta en la figura siguiente).

Por lo tanto, la utilidad de este sistema está principalmente orientada a evitar el uso de un cable para conectar dos dispositivos cercanos. Básicamente, su motivación es evitar los típicos problemas encontrados cuando se quiere hacer una conexión en un momento determinado y no se puede hacer porque o bien no se dispone de un cable o bien uno de los equipos no está bien configurado. En algunas situaciones, esta cobertura limitada, en cuanto a ángulo y necesidad de visión directa, se puede ver como una garantía de seguridad al enlace de la comunicación, puesto que se dificulta que un intruso intercepte datos.

Esquema de transmisión de IRDA y ángulos de visión



Este tipo de tecnología se encuentra presente principalmente en ordenadores portátiles, teléfonos móviles, calculadoras y PDA, y disfrutó de gran popularidad en la década de 1990 y comienzos de la década del 2000. Posteriormente, su utilización disminuyó hacia el uso de otras tecnologías radio, como Bluetooth y Wireless, que disfrutaban de más cobertura y evitaban la necesidad de visión directa. Sin embargo, según la aplicación, IRDA puede ser un mejor candidato debido a su nivel de seguridad y en entornos en los que las interferencias representen un problema. Además, es una tecnología que fue diseñada para poder ser implementada a un coste muy bajo y esto se debe, en parte, al hecho de que el transmisor y receptor se basan en el uso de un LED y un detector fotodiodo, respectivamente.

3.2.2. Bluetooth

Este sistema de comunicaciones radio fue ideado por la compañía Ericsson en el año 1995 con el principal objetivo de proveer un estándar radio capaz de conectar multitud de dispositivos de manera sencilla y evitar tareas de configuración. Es decir, la idea básicamente es conectar dos dispositivos equipados con Bluetooth, que se detecten automáticamente y que se pueda establecer conexión entre ellos de una manera inmediata. El objetivo, por lo tanto, es similar al del caso de IRDA, pero en este caso se evitan los problemas observados en cuanto a necesidad de visión directa y cobertura escasa al trabajar con un sistema de comunicaciones radio.

Por otro lado, también fue diseñado con el objetivo de poder crear pequeñas redes (conocidas como **picorredes**) entre los diferentes dispositivos y facilitar, por lo tanto, el intercambio de datos. Actualmente, este estándar recibe el apoyo del Bluetooth Special Interest Group, consorcio formado por más de 1.900 empresas. En los últimos años, el sistema Bluetooth ha sido principalmente destinado a los periféricos, PDA y telefonía móvil, puesto que se ve como un sistema alternativo al uso del cable en entornos relativamente cercanos y de área personal.

¿De dónde viene el nombre *Bluetooth*?

El origen del nombre *Bluetooth* proviene del rey escandinavo Harold Blåtand (siglo X), quien se hizo famoso por unificar multitud de tribus, bastante disonantes entre ellas, de las actuales Noruega y Dinamarca. La traducción de Blåtand al inglés es Bluetooth. Ericsson tomó este nombre para reflejar el carácter unificador de Bluetooth en el sentido de usarse para conectar multitud de dispositivos diferentes vía radio.

Bluetooth es un estándar de comunicaciones radio que permite la transmisión de voz y datos que opera en la banda de frecuencias libre de 2,4 GHz (una de las bandas Industrial, Scientific and Medical o ISM). Este estándar entraría en la categoría de redes inalámbricas de carácter personal (o *wireless personal*

area networks, WPAN), puesto que está diseñado para proveer de dispositivos de baja potencia, tamaño y coste. Concretamente, los dispositivos Bluetooth se clasifican en diferentes clases según la limitación en potencia:

- Clase 1: potencia máxima de 100 mW y cobertura de unos 100 m.
- Clase 2: potencia máxima de 2,5 mW y cobertura de unos 10 m.
- Clase 3: potencia máxima de 1 mW y cobertura cerca de 1 m.

Hay que mencionar que estas coberturas se refieren a entornos interiores, en los que suelen operar los dispositivos Bluetooth, puesto que están enfocados a redes tipo WPAN. Aparte de los diferentes niveles de potencia según la clase del dispositivo, el estándar también define un sistema de control de potencia, en el que el transmisor adapta la potencia para transmitir solo con la estrictamente necesaria que asegure que la señal llega al receptor con un nivel de potencia aceptable. Esto se hace determinando una serie de umbrales de potencia en el receptor y el resultado obtenido es que se optimiza la duración de la batería.

Bandas ISM

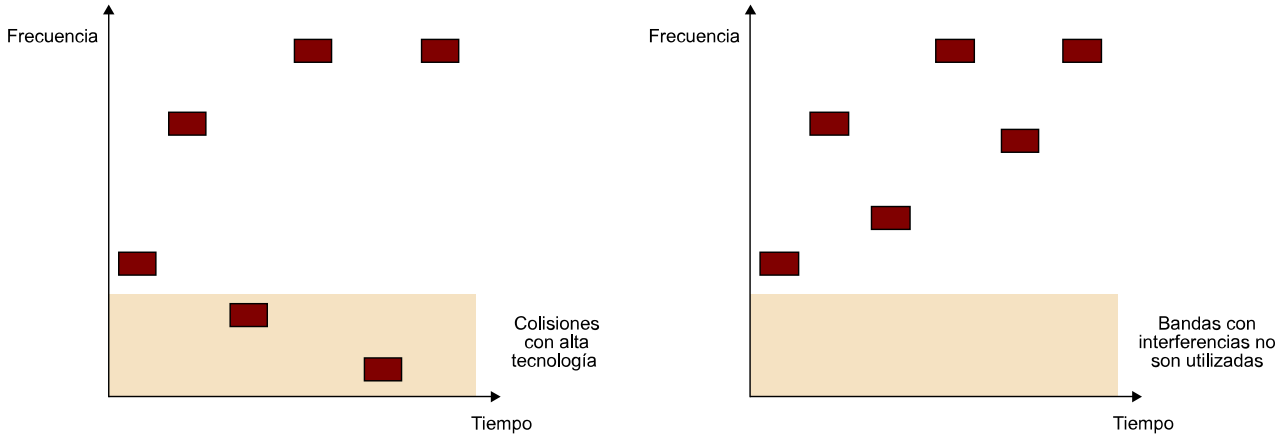
Las bandas ISM son bandas reservadas internacionalmente para el uso de sistemas industriales, científicos y médicos. La más popular es la banda libre a 2,4 GHz (comprendida entre los 2.400 y 2.500 MHz), puesto que se utiliza para multitud de sistemas radio de carácter personal (WPAN, *wireless personal area networks*) o local (WLAN, *wireless local area networks*), como algunos de los comentados en esta sección. Esto viene propiciado por el hecho de que, a diferencia de otras bandas, esta es una banda libre en el sentido de que no se necesita ninguna licencia para poder transmitir. Cuando son necesarias, estas licencias son concedidas por la entidad reguladora del país donde se está trabajando.

La velocidad de transmisión depende de la versión del estándar, que cubre desde 1 MBps ofrecido para la versión 1.2 del estándar hasta los 3 MBps de la versión 2.0 + Enhanced Data Rate (EDR). El signo + de esta última versión sirve para reflejar que EDR es un modo opcional del estándar 2.0 que se activa para llegar a los 3 MBps. En caso de que no se active, el dispositivo continúa siendo compatible con la versión anterior 1.2. Hay que mencionar que existe una versión 3.0 + High Speed (HS) que puede llegar hasta los 24 MBps. Sin embargo, esta versión usa tecnología Wi-Fi para enviar los datos a esta velocidad, mientras que Bluetooth se reserva para establecer la conexión.

Tal como se ha mencionado, Blueetoth trabaja en una banda libre donde multitud de dispositivos operan. Por lo tanto, existe un alto nivel de interferencias procedentes de otros equipos. Para proveer de un sistema robusto frente a estas interferencias, Bluetooth emplea un mecanismo de transmisión denominado *frequency-hopping spread spectrum*. La idea de este mecanismo es simple: se divide la banda utilizada por Bluetooth (entre 2.402 y 2.480 MHz) en 79 canales de 1 MHz y se va cambiando de canal a lo largo del tiempo. Concretamente, se cambia de canal 1.600 veces por segundo y esto reduce el impacto de interferencias de otros sistemas, que raramente ocuparán la banda ISM entera. Para llevar a cabo esta tarea con más éxito, se incluyó en la versión 1.2 del estándar una variante más sofisticada de esta técnica denominada AFS (*adaptive*

frequency-hopping spread spectrum). Esta variante detecta cuáles son los canales utilizados por otros sistemas con el fin de evitarlos en la secuencia de saltos de canales. Este concepto se puede observar en la figura siguiente:

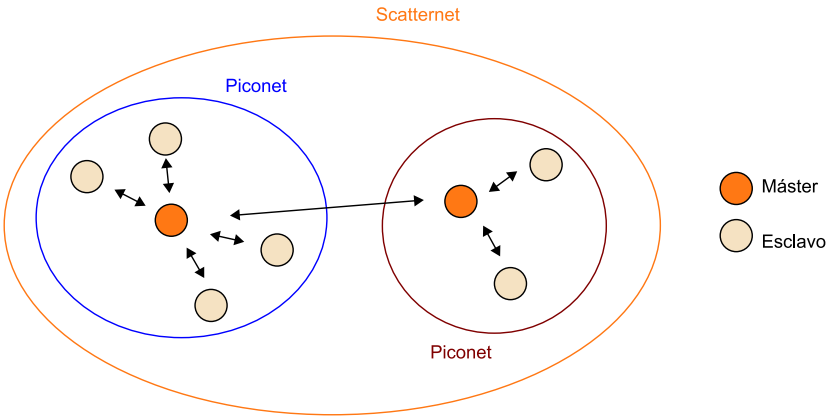
Esquema de funcionamiento de *frequency-hopping spread spectrum* y su variante adaptativa AFS



Espectro ensanchado por salto de frecuencia **Espectro ensanchado por salto de frecuencia adaptativo**

El tipo de red utilizado en Bluetooth, referida como picorred, se basa en el concepto *máster-esclavo*, en el que un nodo actúa como máster, de modo que se encarga de la sincronización de la red, y los otros toman el papel de esclavos. En una picorred, puede haber hasta 7 esclavos y varias picorredes se pueden enlazar entre ellas y formar lo que se conoce como *red dispersa* (*scatternet*) (podéis ver la figura 72). Dentro de una picorred, la transmisión se divide en *slots* de 625 µs, en los que el inicio de cada *slot* viene marcado por el reloj interno del máster. El máster utiliza los *slots* pares para transmitir paquetes y los impares para recibirlos, mientras que los esclavos transmiten y reciben los *slots* impares y pares, respectivamente. También se permite la transmisión de paquetes de 3 y 5 *slots* de duración. En este caso, para respetar la regla comentada en el caso de paquetes de un *slot* de duración, el máster debe iniciar las transmisiones en los *slots* pares, a pesar de que la transmisión dure más de un *slot*, y los esclavos, en los impares.

Ejemplo de *scatternet* Bluetooth formada por dos picorredes



En el caso de sistemas empotrados, Bluetooth puede ser una buena opción para llevar a cabo comunicaciones inalámbricas debido al bajo coste, tamaño y bajo consumo de los chips. Sin embargo, dependiendo del tipo de aplicación, la elección de este estándar podrá finalmente cambiar por los sistemas radio que se presentarán a continuación: Wi-Fi y 802.15.4/Zigbee. Por un lado, veremos que Wi-Fi ofrece más coberturas y velocidades de transmisión. Además, el número de estaciones base, para hacer de pasarela (o *gateway*) a Internet, por ejemplo, está más extendido en esta tecnología que en el caso de Bluetooth. En cuanto al coste de integración y consumo de batería, la tecnología 802.15.4/Zigbee ofrece una alternativa más atractiva, puesto que fue diseñada con el objetivo principal de ofrecer dispositivos radio con un coste, tamaño y consumo extremadamente bajos.

3.2.3. Wi-Fi

Este sistema de comunicaciones radio está basado en la familia de estándares IEEE 802.11 y se encuentra bajo la tutela de la WIFI Alliance, que es un consorcio de empresas encargadas de su certificación para asegurar la interoperabilidad de los diferentes dispositivos Wi-Fi. Las principales ventajas de Wi-Fi son:

1) la madurez de la tecnología en el sentido de que asegura la interoperabilidad entre los diferentes fabricantes y un buen comportamiento en dispositivos pequeños;

2) la instalación y el mantenimiento sencillos de los equipos;

3) el hecho de que opere en las bandas libres de 2,4 GHz (como otros sistemas radio de características similares) y 5 GHz (donde no hay un solapamiento tan alto de sistemas y las interferencias son más bajas), y

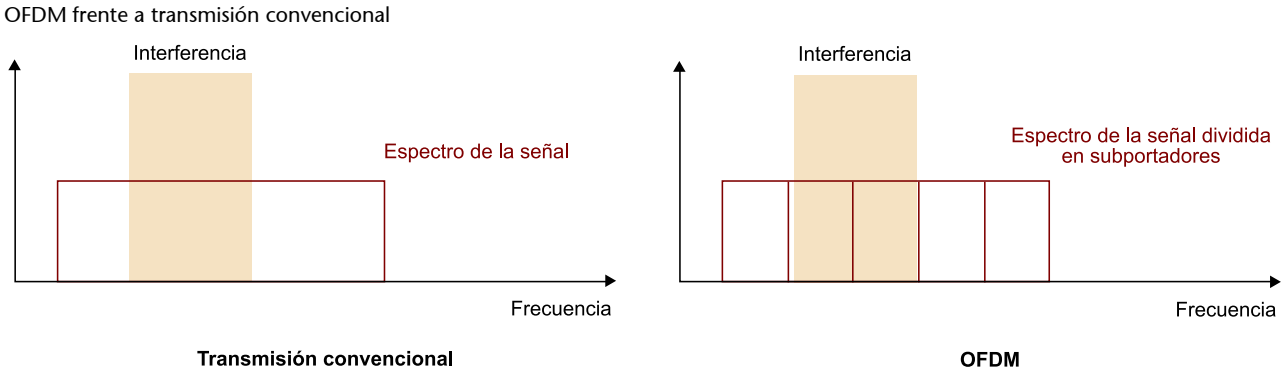
4) su gran incursión en el mercado. Concretamente, Wi-Fi se puede ver como un caso similar al observado con Ethernet en el caso de redes cableadas.

En Wi-Fi también hay diferentes versiones con distintas velocidades de transmisión. Sin embargo, estas versiones además tienen grandes diferencias en términos de implementación de la capa física. Las más destacables son las basadas en los estándares siguientes:

- **IEEE 802.11b**: opera en la banda de 2,4 GHz, que se divide en 14 canales de 22 MHz comprendidos entre los 2.401 y 2.495 MHz (algunos canales se encabalgan). Se pueden conseguir velocidades de transmisión en el aire de 11 MBps (para distancias de 30 m en interiores) y 1 MBps (si la distancia crece a 90 m). Esta variación de velocidad viene causada por el hecho de que el sistema dispone de un sistema de adaptación que consiste en ajustar la velocidad según el nivel de señal recibida en el receptor. Es decir, se aumenta o se reduce la velocidad si la calidad de la señal es buena o mala. Por otro lado, tal como sucede en Bluetooth, el sistema incluye una

técnica de transmisión orientada a reducir el impacto de las interferencias existentes en la banda 2,4 GHZ, que se encuentra saturada de tecnologías diferentes. En este caso, se utiliza lo que se conoce como *direct sequence spread spectrum* (DSSS). Esta técnica consiste en distribuir la energía de la señal en un ancho de banda más grande que el necesario. Por lo tanto, si se recibe una interferencia, su impacto será inferior, puesto que no afecta a la señal total, sino únicamente a una porción. Finalmente, cabe señalar que esta modalidad de Wi-Fi es la más utilizada y se encuentra muy presente en los hogares, donde la estación base Wi-Fi se suele emplear también como puerta de enlace a Internet.

- **IEEE 802.11g**: este sistema también opera en la banda de 2,4 GHz, que se encuentra dividida en este caso en trece canales de 22 MHz. Concretamente utiliza los trece primeros canales del 802.11b (entre 2.401 y 2.483 MHz). En este caso, las velocidades de transmisión pueden aumentar hasta 54 MBps, dado que se usa una técnica de transmisión más sofisticada conocida con el nombre de multiplexado por división ortogonal de frecuencia (OFDM, *orthogonal frequency division multiplexing*). Esta técnica consiste en dividir la transmisión en varias transmisiones paralelas, donde cada una de ellas utiliza un ancho de banda más reducido. El concepto se presenta en la figura siguiente. Tal como se puede ver, tanto el sistema de transmisión convencional como la OFDM utilizan el mismo ancho de banda total; por lo tanto, transmiten con la misma velocidad. Aun así, si se tiene una atenuación grande o una interferencia en alguna banda frecuencial (como se muestra en la figura siguiente), la señal convencional se encuentra totalmente distorsionada. En el caso de OFDM, solo unas pocas subbandas son distorsionadas, de manera que se reduce el impacto puesto que los datos enviados al resto de las subbandas no se pierden. Por esta razón, la cobertura del 802.11g es mayor que en el caso de 802.11b y se pueden llegar a los 200 m. En este caso concreto, la OFDM también se utiliza como alternativa a DSSS para llevar a cabo la tarea de reducir el impacto ocasionado por la presencia de otros equipos en la banda libre (solo en los modos de transmisiones más lentos, de 1 y 2 MBps, se mantiene el DSSS para transmitir la señal). Esta variante del estándar nació para cubrir la alta demanda existente de los usuarios en cuanto a necesidad de sistemas inalámbricos y ofrecer velocidades altas. Por lo tanto, es un sistema que se encuentra bastante implantado, como el caso del 802.11b. Concretamente, existen multitud de dispositivos equipados con ambas variantes (dispositivos 802.11b/g).



- **IEEE 802.11a**: esta versión del estándar opera en la banda de los 5 GHz; por lo tanto, una de las primeras ventajas que presenta es que se encuentra en una banda que no está tan saturada de interferencias como la banda de 2,4 GHz. Las bandas de trabajo concretas dependen de la regulación de cada país. En el caso de Europa, se utilizan dieciocho canales de 20 MHz distribuidos en el rango de frecuencias 5.180 MHz–5.680 MHz. También utiliza la OFDM, debido a las mejores prestaciones en términos de velocidad ofrecido con una velocidad máxima de 54 MBps. A pesar de que la banda de 5 GHz ofrece un escenario menos saturado de interferencias, su utilización presenta el problema de que la atenuación en función de la distancia es más grande. Esto se puede ver repasando la fórmula de Friis, que nos da el nivel de potencia recibida, P_r (expresada en vatios), en un sistema de comunicación inalámbrica, donde se supone que la transmisión se realiza en el espacio libre (es decir, no hay obstáculos entre el transmisor y el receptor y se consideran negligibles los posibles rebotes de la señal transmitida):

$$P_r = P_t\left(\frac{c}{4\pi f d}\right)^2$$

donde P_t es la potencia transmitida (expresada en vatios), c es la velocidad de la luz ($3 \cdot 10^8$ m/s), f es la frecuencia portadora (expresada en Hz) utilizada para transmitir la señal y d es la distancia en metros entre el transmisor y el receptor. Como se puede ver, la potencia recibida disminuye de manera cuadrática en función de la distancia. Por lo tanto, la cobertura se encuentra bastante más limitada. Si además se tuvieran en cuenta los posibles obstáculos entre el transmisor y receptor, la calidad de la señal se vería bastante más degradada. Una manera sencilla que se utiliza para modelar las pérdidas por propagación en entornos diferentes al espacio libre es adaptar la fórmula de Friis de la manera siguiente:

$$P_r = P_t\left(\frac{c}{4\pi f}\right)^2 \frac{1}{d^\gamma}$$

donde γ es el parámetro que modela las pérdidas de propagación del entorno, que varía normalmente entre 2 (espacio libre) y 4-5 (en entornos interiores con muchos obstáculos). Ciertamente, modelar la potencia recibida en un entorno real implica el uso de modelos más complejos donde se deben tener también en cuenta los parámetros y no-idealidades del hard-

ware relacionado con el dispositivo radio (eficiencia de las antenas, diagrama de radiación, factor de ruido de los amplificadores, etc.), pero esta aproximación puede servir para hacer un primer acercamiento al problema. Además, hay que señalar que la capacidad de salvar obstáculos también disminuye a medida que aumenta la frecuencia. Por lo tanto, esta variante del estándar se suele utilizar para desplegar radioenlaces siempre que se pueda asegurar la visión directa entre ellos. Básicamente, el objetivo es unir redes Wi-Fi remotas mediante un radioenlace. El hecho que el radioenlace trabaje en la banda de 5 GHz evita que el sistema induzca o reciba interferencias a otras u otros sistemas. Según el entorno y la calidad del enlace, las coberturas en exteriores pueden llegar hasta los 500-1.000 m. Algunos experimentos revelan coberturas mayores, pero los transmisores y receptores se emplazaban en lugares elevados para asegurar un buen enlace. En el caso de entornos interiores, la señal se degrada considerablemente, lo que provoca que el sistema no sea una opción viable para este tipo de escenarios.

El tipo de red utilizado en las variantes comentadas es de topología en estrella, donde un dispositivo actúa de estación base y el resto de los dispositivos se conectan para llevar a cabo las comunicaciones. Además, es la estación base la que define cuál será el canal (de los disponibles) que se usará. Como se trata de un medio inalámbrico, el acceso al medio de los diferentes dispositivos es compartido y se lleva a cabo mediante el protocolo de acceso múltiple con escucha de portadora y evitamiento de colisión (CSMA/CA, *carrier sense multiple access with collision avoidance*). Este protocolo es parecido al CSMA/CD comentado en el caso de Ethernet, pero hay algunas diferencias, dado que no se trabaja en un entorno cableado. Básicamente, se llevan a cabo las mismas tareas: los dispositivos que utilizan este protocolo escuchan el medio antes de transmitir sus paquetes para evitar colisiones. Sin embargo, en el momento en el que un dispositivo está transmitiendo la información, este no puede llevar a cabo la tarea de *collision detection*. Es decir, no puede escuchar el medio a la vez que está transmitiendo. Por lo tanto, el mecanismo de acceso varía un poco al llevar a cabo un proceso de *collision avoidance*. Básicamente, si el dispositivo que quiere transmitir detecta que el medio está siendo utilizado, se espera un tiempo aleatorio hasta que vuelve a probar a enviar. Este tiempo aleatorio permite que si dos dispositivos intentan acceder al medio a la vez y se tienen que esperar, se esperan tiempos diferentes y, de este modo, se reduce la probabilidad de solapamiento de las próximas transmisiones.

A pesar de que la madurez de la tecnología ha llegado a un punto en el que se dispone de hardware muy optimizado, el coste de estos dispositivos es mayor que el de los dispositivos Bluetooth o Zigbee. Por lo tanto, su utilización en sistemas empotrados vendrá determinada por el grado de compromiso entre complejidad/coste frente a velocidad de datos. Es decir, este sistema será una opción viable si se quieren conseguir velocidades de transmisión altas. Por otro lado, también dependerá de la cobertura requerida; en este sentido, Wi-Fi tiene una ventaja frente a sus competidores. Hay que señalar también que

en los últimos años ha proliferado la implementación de módulos Wi-Fi, que facilitan la integración de este estándar en cualquier sistema, lo cual simplifica su integración con sistemas empotrados.

3.2.4. IEEE 802.15.4 / ZIGBEE

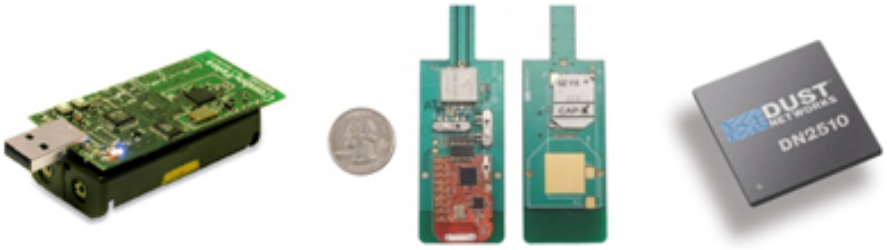
El estándar IEEE 802.15.4 surgió con el objetivo claro de ofrecer un sistema de comunicaciones radio de muy bajo coste, de un consumo extremamente bajo y que permitiera el uso de dispositivos de tamaño muy reducido. Esta motivación venía, en parte, determinada por la proliferación de redes basadas en el uso de dispositivos muy sencillos que transmiten información con velocidades de transmisiones muy bajas, que son conocidas como *low-rate wireless personal area networks* (LR-WPAN).

Algunos ejemplos de estos tipos de redes son las utilizadas para llevar a cabo tareas de monitorización, como pueden ser la monitorización de temperatura en edificios, de condiciones climáticas en bosques y montañas, control de la fauna, etc. En estos tipos de aplicaciones se requieren multitudes de dispositivos empotrados de muy baja complejidad y bajo coste que lleven a cabo tareas muy sencillas (por ejemplo, medir la temperatura y enviarla). También es importante que el grado de mantenimiento sea el mínimo posible y la duración de la batería sea bastante alta. Por lo tanto, hay que tener un sistema de comunicación que no encarezca la solución, no aumente en gran medida el consumo de batería y no condicione el tamaño del dispositivo. Hay que mencionar que estos tipos de redes inalámbricas orientadas a tareas de monitorización, donde cada nodo de la red actúa como sensor de algún acontecimiento físico, están proliferando en los últimos años gracias a los intereses suscitados en la industria, dada la reducción de los costes de los dispositivos, y son conocidas como **redes de sensores inalámbricos**[2].

[(2)]Comúnmente referidas con el término inglés: *wireless sensor networks* (WSN).

En la figura siguiente, se presentan algunos ejemplos de nodos o sensores de estos tipos de redes. Estos nodos son sistemas empotrados en los que se integra todo el hardware necesario para su funcionamiento autónomo, como el sensor en sí, el microcontrolador, memorias, fuente de alimentación, equipamiento radio, etc. Como se puede observar, el grado de integración es bastante elevado.

Ejemplos de sensores de WSN comerciales



Volviendo al estándar IEEE 802.15.4, las principales características de este estándar son que trabajan a velocidades de transmisión reducidas (hasta 250 kBps) y que se utilizan diferentes bandas de trabajo, que pueden variar según el país de aplicación. Las disponibles en Europa son:

- **868-868.6 MHz**: dividida en tres canales con velocidades de 20 kBps, 100 kBps o 250 kBps.

- **2.400-2.483.5 MHz**: dividida en dieciséis canales con velocidades de 250 kBps.

Cabe indicar que también existen versiones del estándar con velocidades superiores (del orden de 27 MBps), pero estas versiones se fundamentan en el uso de la tecnología *ultra wide band* (UWB). Esta tecnología se basa en la transmisión de pulsos extremadamente estrechos; por lo tanto, hay que usar canales con anchos de banda muy grandes (superiores a 500 MHz y por eso se denominan así). Sin embargo, en el mercado no hay todavía soluciones competitivas que trabajen con esta tecnología. A su vez, UWB trabaja con anchos de banda muy grandes pero a expensas de reducir la potencia de transmisión de manera drástica y estos sistemas están principalmente destinados para trabajar a distancias muy pequeñas.

En las versiones del estándar operativas en Europa (y en la mayoría de las versiones disponibles), se utiliza DSSS como medio para combatir las interferencias de otras tecnologías. En cuanto a la cobertura de esta tecnología, depende del entorno, que varía desde los 10 metros en entornos interiores hasta los 100 metros en entornos exteriores con buenas condiciones.

Las topologías de redes permitidas son las de tipo estrella y las de igual a igual, en las que todos los nodos se pueden comunicar con el resto (podéis verlo en la figura siguiente). En todo caso, la red necesita que un nodo actúe como coordinador. Dadas las restricciones en términos de complejidad de los escenarios donde IEEE 802.15.4 trabaja, se definen dos tipos de nodos:

- **Nodos con funcionalidad completa**[3]: estos nodos pueden llevar a cabo operaciones con cierta carga computacional y pueden retransmitir paquetes a otros nodos.

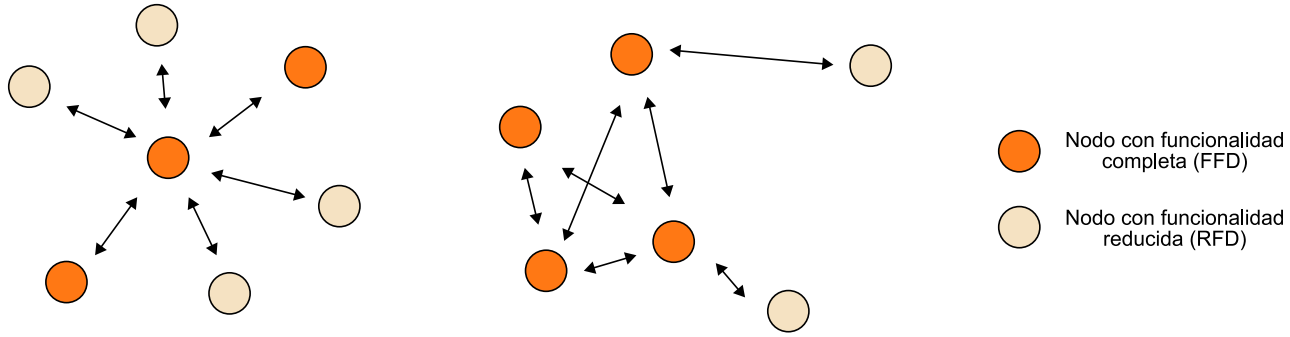
- **Nodos con funcionalidad reducida**[4]: estos nodos tienen funciones limitadas y únicamente son capaces de comunicarse con nodos FFD.

(3)Normalmente conocidos como *full functional devices* (FFD).

(4)Normalmente conocidos como *reduced functional devices* (RFD).

Por lo tanto, los coordinadores de la red deben ser FFD forzosamente. Es decir, es necesario que haya un nodo de este tipo para establecer una red. Igualmente, para poder establecer redes de igual a igual, es necesario que haya nodos de esta naturaleza en la red para poder crear las diferentes conexiones tal como se refleja en la figura.

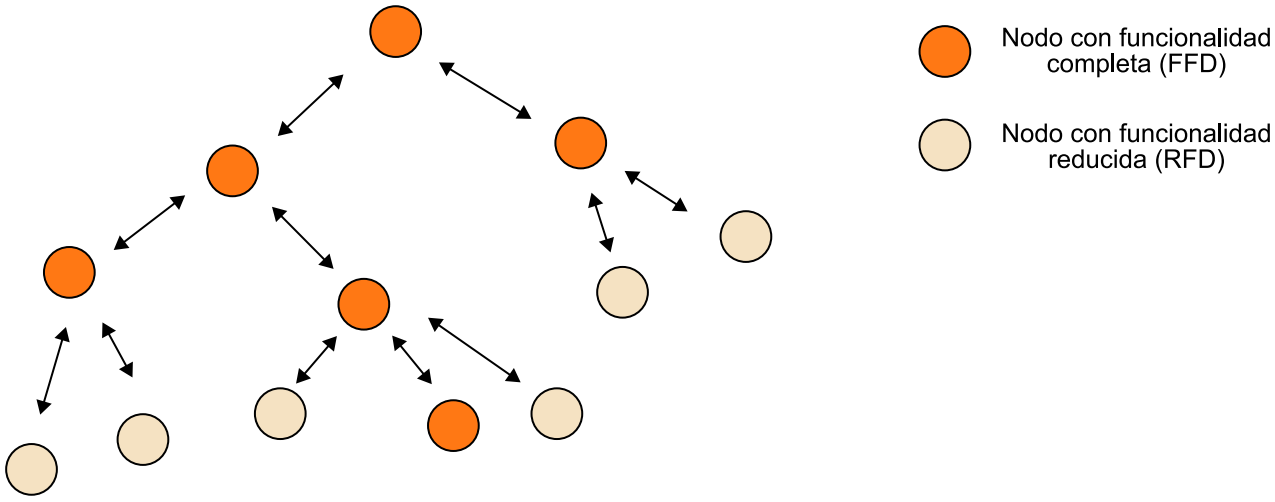
Topologías utilizadas en IEEE 802.15



Izquierda: topología en estrella. Derecha: topología de igual a igual

Hay que decir que una variante de topología de igual a igual muy utilizada en aplicaciones basadas en IEEE 802.15.4 es la conocida como *topología en árbol*, que se muestra en la figura siguiente. Esta topología se utiliza mucho en entornos de monitorización donde hay un nodo que actúa como recolector de datos y el resto de los nodos de la red le van enviando los datos medidos. Para ahorrar energía, se transmite a baja potencia para comunicarse localmente usando los nodos más cercanos y llegar al nodo recolector mediante un sistema de comunicación multisalto.

Topología en árbol



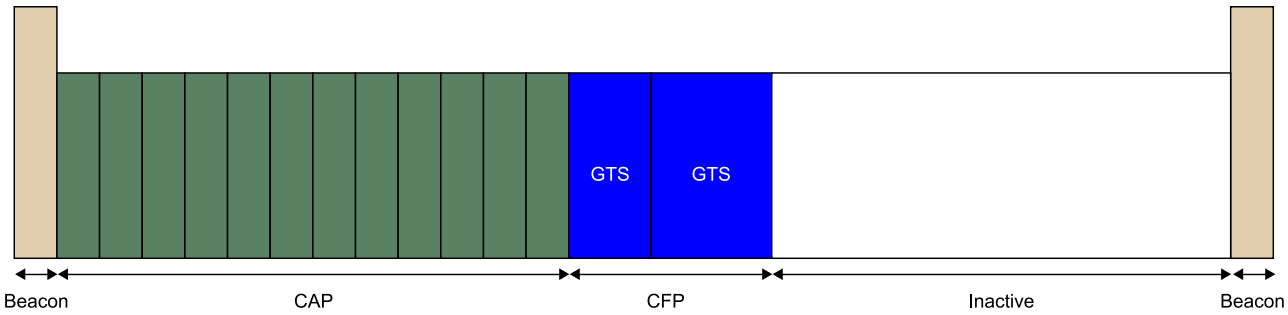
El acceso al medio se puede llevar a cabo usando dos modos de operación diferentes: modo *non-beacon* y modo *beacon*. Las características de cada uno de estos modos son las siguientes:

- **Modo *non-beacon***: en este modo los nodos de la red no se encuentran sincronizados y acceden al medio usando CSMA-CA. La ventaja de este modo es que la complejidad de la red es bastante reducida.

- **Modo *beacon***: en este modo el coordinador de la red va enviando una serie de paquetes conocidos como *beacons*. Estos *beacons* sirven para sincronizar el resto de los nodos, de manera que se les indica cuál es el inicio de una supertrama, que es la unidad de transmisión utilizada en este estándar (podéis verlo en la figura siguiente). Esta supertrama, por su parte, se divide en tres períodos: el período CAP (*contention acces period*), el período CFP (*contention free period*) y un período inactivo (*inactive period*). En el período CAP, los nodos utilizan una versión modificada de CSMA-CA, conocida como CSMA-CA con ranuras (o *slotted* CSMA-CA), donde se continúa usando la misma estrategia de acceso compartido. Pero las transmisiones de los nodos se deben ajustar a una serie de *slots* temporales definida en la supertrama. En el período CFP, por su parte, el coordinador de la red asigna unos *slots* determinados[5] a los nodos que han pedido acceso para asegurar que podrán enviar datos. Esto se suele utilizar cuando se debe asegurar un ancho de banda determinado a una serie de nodos de la red. Finalmente, el período inactivo se introduce para reducir la actividad de los nodos y, por lo tanto, alargar la duración de las baterías.

[5] En inglés, *guaranteed time slots* (GTS).

Supertrama IEEE 802.15.4 en modo *beacon*



Normalmente, al estándar IEEE 802.15.4 se le asocia el término *Zigbee*. Zigbee no es otra cosa que una alianza de empresas que se encarga de asegurar la interoperabilidad de sus dispositivos basados en el uso del IEEE 802.15.4. Dado que IEEE 802.15.4 define solo las capas física y de enlace, Zigbee se encarga de definir las capas superiores para ofrecer esta interoperabilidad. Esto no significa que todos los sistemas que trabajen con IEEE 802.15.4 utilicen Zigbee, puesto que hay otras opciones y algunos fabricantes usan las suyas propias. Sin embargo, esta es una de las opciones más famosas.

Finalmente, hay que mencionar que, claramente, este tipo de comunicación es de los más adecuados para ser implementados en sistemas empotrados con restricciones en cuanto a tamaño, consumo y coste. En el caso de WSN, es la solución preferentemente utilizada. Sin embargo, puede ser que la aplicación

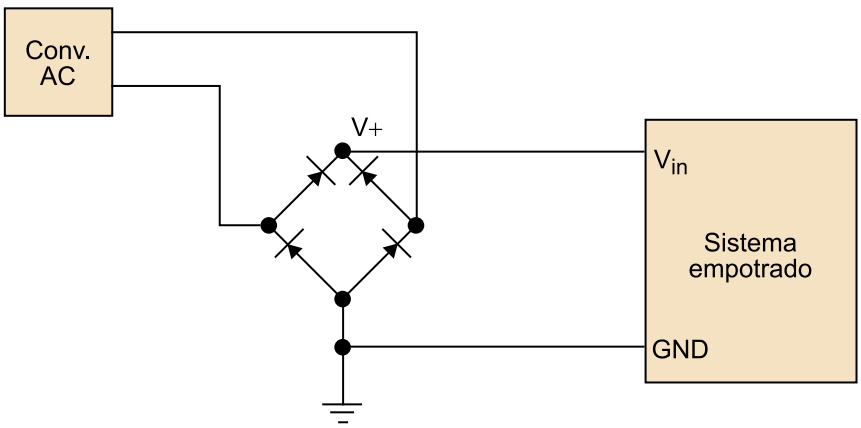
concreta requiera velocidades más altas o más cobertura. En este caso, las otras opciones radio mencionadas en esta sección pueden presentar una buena alternativa.

4. Subsistema de alimentación

Lógicamente, cualquier dispositivo electrónico necesita una fuente de alimentación para poder funcionar. Dependiendo del tipo de dispositivo, la selección del tipo de fuente y el diseño del subsistema de alimentación varía.

En caso de que el dispositivo no necesite ser portable y disponga de una toma de corriente cercana, la mejor opción es conectarlo directamente a la red eléctrica. Al trabajar la red eléctrica en corriente alterna (ca) y con altos niveles de voltaje (230 V en el caso de Europa), se debe transformar para que pueda ser útil para el sistema empotrado, que requiere corriente continua (cc) y un voltaje bastante inferior. Una de las opciones más viables para hacer esto, en términos de simplicidad y bajo coste, es el uso de un adaptador ca, el cual es fácilmente adquirible por cualquier distribuidor. Concretamente, es como el que se utiliza en el caso de los teléfonos móviles y otros dispositivos de informática de consumo. Estos dispositivos transforman la corriente alterna de entrada en corriente continua con un voltaje que suele estar entre los 5 y 12 V. Una cosa que se debe tener en cuenta con el uso de este tipo de adaptadores es que el conector que tienen de salida (que se conecta al sistema empotrado) puede tener diferente polaridades según el fabricante. Por lo tanto, antes de conectarlo al sistema empotrado, hay que asegurarse de que la polaridad sea la correcta para no provocar desperfectos en el sistema empotrado. Esto se puede verificar mediante el uso de un **multímetro**. Otra opción es proveer el sistema empotrado de un **puente de diodos**. Un puente de diodos, o puente rectificador, es un circuito que da siempre la misma polaridad a la salida, independientemente de la polaridad de entrada (podéis ver la figura siguiente, en la que las entradas V_{in} y GND se refieren a las entradas de alimentación y masa, respectivamente). Esto se debe a la característica propia de los diodos, que solo dejan pasar corriente eléctrica en una única dirección.

Esquema del subsistema de alimentación diseñado



En el resto de los casos, la opción más común es el uso de baterías. En primer lugar, se debe hacer una buena selección de la batería para asegurar que ofrece el voltaje correcto y un nivel de corriente suficiente. En caso contrario, el sistema puede funcionar erróneamente o directamente no funcionar. Por otro lado, se debe considerar, aparte del nivel de corriente media que ofrece la batería, cuál es la corriente de pico que puede ofrecer.

Ejemplo

Algunos sistemas empotrados pueden necesitar solo una corriente media de 20 mA, pero requerir en algunos momentos corrientes de pico de 100 mA. Esto suele suceder en sistemas con memorias flash, que requieren altas corrientes cuando llevan a cabo operaciones de escritura.

En segundo lugar, se debe tener un sistema muy diseñado, puesto que esto puede condicionar bastante la duración de la batería, que puede ir desde cuestiones de minutos (si está diseñado bastante mal) hasta algunos años, según el tipo de hardware del sistema empotrado y la aplicación. Para llevar a cabo este diseño, el primer paso es tener un sistema empotrado que utilice dispositivos de bajo consumo. El consumo de potencia entre chips diferentes puede variar mucho, y muy a menudo hay versiones de bajo consumo del dispositivo que se quiere integrar. Por otro lado, muchos periféricos y chips de memoria entran de manera automática en modo de bajo consumo cuando no están en uso. Otros pueden ser puestos en este modo al accionar una entrada digital o mediante una orden de software. Dependiendo de la aplicación, una opción también puede ser implementar una solución en la que se separen los circuitos de alimentación de algunos dispositivos del sistema para poderlos desconectar, vía control de software, cuando estos no sean utilizados. Finalmente, algunos dispositivos de muy bajo consumo, como los sensores, necesitan muy poco nivel de corriente para funcionar. Por lo tanto, una opción es alimentarlos directamente con las líneas de entrada y salida propias del microcontrolador o microprocesador del sistema. Es decir, se utiliza una línea de entrada y salida para alimentar el dispositivo, puesto que estas líneas pueden dar niveles de corrientes suficientes (20 mA en algunos casos) y, simultáneamente, se aprovecha esta línea para activar o desactivar el dispositivo cuando sea necesario.

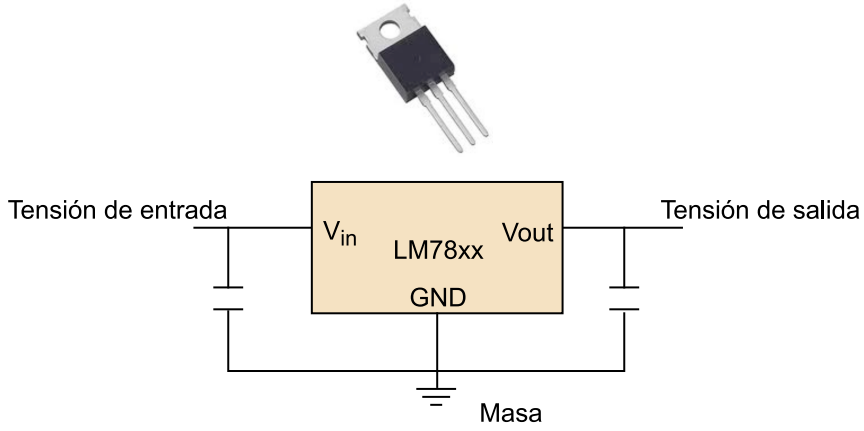
Dejando ahora de lado el tipo de fuente de alimentación que utilice el sistema, es recomendable que el subsistema de alimentación del sistema empotrado use reguladores de voltaje. Un regulador de voltaje es un dispositivo que asegura un nivel fijo de tensión continua a su salida a pesar de que la tensión a la entrada adopte diferentes valores o sufra variaciones o ruido. Esta opción es recomendable para asegurar que el nivel de tensión que se aplica a los componentes del sistema empotrado es la deseada y se mantiene constante. A pesar de que muchos de los componentes son reacios a variaciones del nivel de tensión, otros son muy sensibles, como por ejemplo los convertidores analógico-digitales que hacen la conversión de la entrada analógica comparando el nivel de tensión de entrada con el nivel de tensión de alimentación del mismo convertidor. Es decir, la cuantificación de la señal de entrada se efectúa tomando la tensión de alimentación como referencia. Por otro lado, si el sistema

empotrado trabaja con baterías, el regulador puede ser un mecanismo eficaz para combatir las variaciones de tensiones ocasionadas por el mismo consumo de la batería. En la figura siguiente se muestra un ejemplo de regulador comercial, en el que las tres patas se corresponden con la tensión de entrada (referida normalmente como V_{in}), la tensión de salida (V_{out}) y la conexión de masa (GND). Este regulador concreto se conoce con el nombre de LM78xx, lo fabrican varias compañías (como Fairchild y ST Microelectronics) y xx se refiere al voltaje de salida. En la parte derecha de la misma figura se muestra el circuito que se suele utilizar para su integración, que se conoce como *regulador lineal*. Tal como se puede ver, se utiliza un condensador a la entrada y otro a la salida. En esta configuración, los condensadores se denominan *condensadores desacopladores* y su objetivo es filtrar el ruido electromagnético presente en la línea de alimentación. Hay que mencionar también que algunos reguladores tienen entradas adicionales que actúan como interruptores para la activación o desactivación del circuito. Esto puede simplificar el proceso de optimización de consumo comentado en el párrafo anterior en cuanto a desactivar dispositivos que no estén siendo utilizados.

Ejemplo

En el caso de LM7805, la salida sería igual a 5 V.

Regulador comercial (arriba) y circuito de un regulador lineal (abajo)



Otro tema que se debe tener en cuenta a la hora de diseñar el subsistema de alimentación es que las mismas líneas del subsistema pueden causar interferencias a las señales que usan los otros componentes del sistema empotrado, lo cual provoca la aparición de datos corruptos o, incluso, el mal funcionamiento del sistema. Este efecto es básicamente el ruido electromagnético comentado en secciones anteriores y algunas posibles pautas que hay que seguir para reducirlo son las siguientes:

- **Reducir el área del bucle de corriente**: el bucle de corriente es el camino descrito por la corriente que va alimentando cada dispositivo, que sale de la fuente por la línea de alimentación y vuelve por la línea de masa. En este bucle se generan campos magnéticos, que son inducidos por las corrientes que circulan en sentidos opuestos por ambas líneas. Estos campos

magnéticos causan ruido electromagnético y una manera de reducirlo es colocando las líneas de alimentación y de masa cercanas, puesto que, de este modo, los campos magnéticos generados por cada línea se cancelan entre ellos. Básicamente, el objetivo de tener un buen diseño es reducir el área del bucle de corriente mediante la minimización de la longitud de las líneas y acercando estas líneas lo máximo posible. Una opción para hacer esto es utilizando un plan de masa. Es decir, utilizar una superficie conductora grande para enlazar todas las conexiones de masa y minimizar, de este modo, el camino de regreso.

- **Utilización de condensadores desacopladores**: la solución propuesta en el punto anterior puede ser difícilmente implementable en algunos sistemas porque la tarea de distribuir la alimentación, a la vez que se asegura un bucle de corriente con área reducida, puede ser complicada. Una solución al problema es poner en la entrada de alimentación de cada componente del sistema un mecanismo para conducir el ruido presente en la línea hacia masa. Esto se puede hacer poniendo condensadores desacopladores en las entradas de cada circuito. Básicamente, una de las patas del condensador se conecta con la línea de alimentación y la otra a masa. Teniendo en cuenta el principio de funcionamiento de un condensador, si la tensión de entrada es continua, el condensador actúa como circuito abierto (es como si no hubiera nada conectado a la línea de alimentación). Si hay alguna fluctuación de corriente, el condensador se va cargando y descargando, de manera que actúa como si condujera el ruido a tierra, puesto que se mantiene el nivel de voltaje constante. De este modo, el ruido puede ser eliminado del subsistema de alimentación. A pesar de que esta solución es bastante antigua, se continúa usando porque se considera eficaz.

5. Detección de errores de hardware

Asegurar la fiabilidad de los sistemas empotrados es de gran importancia, dados los tipos de escenarios en los que trabajan estos dispositivos. En comparación con los sistemas informáticos convencionales, donde normalmente trabajan en entornos adecuados, los sistemas empotrados encuentran usualmente entornos de operación con altas temperaturas, humedad, suciedad y ruido electromagnético, aparte de que pueden estar alimentados con un sistema de distribución pobre. Por lo tanto, la fiabilidad del hardware debe ser realmente alta para poder operar en estas condiciones. Además, muchos de los dispositivos pueden ser empleados por usuarios con muy poco conocimiento en cuanto a una manipulación adecuada del dispositivo. Los diseños deberán tener en cuenta posibles malos usos del sistema y acciones inesperadas, ya sean intencionadas o no.

A pesar de que los sistemas actuales disfrutan de unos procesos de diseño y verificación cuidadosos y llevados a cabo para reducir los errores de funcionamiento drásticamente, estos errores se pueden producir siempre. Además, muchos de los sistemas empotrados se usan en localizaciones remotas, de difícil acceso y/o se montan con la idea de disfrutar de un mantenimiento limitado o prácticamente nulo. Por lo tanto, es de vital importancia que el sistema de diseño tenga en cuenta acciones para detectar posibles errores o malos funcionamientos del sistema y se incluyan mecanismos de recuperación autónomos.

Una técnica que se utiliza muy a menudo para detectar errores del sistema y para llevar a cabo su recuperación es el uso de sistemas de monitorización hardware y software conocidos como *temporizadores de vigilancia* (***watchdogs***).

Temporizadores de vigilancia

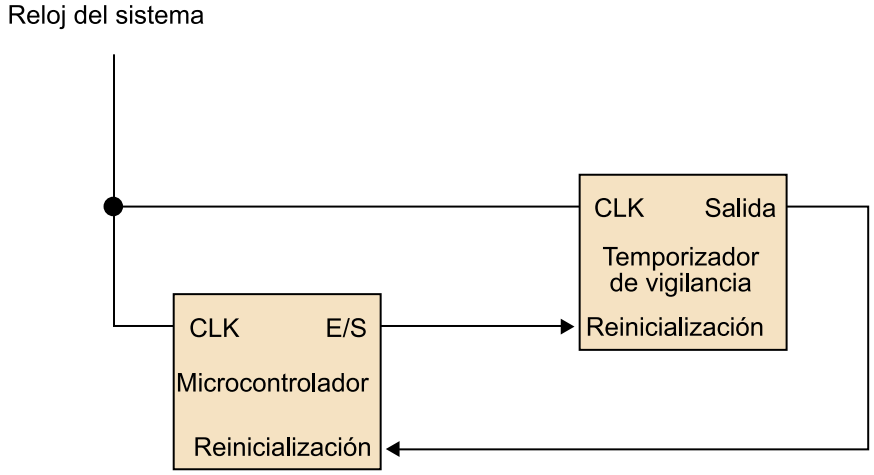
Los temporizadores de vigilancia (*watchdog*, en inglés; 'perro guardián' sería la traducción literal) hacen referencia a un sistema hardware o software encargado de vigilar si el sistema funciona correctamente. Normalmente, se basa en el uso de contadores temporales y en el control de que eventos esperados vayan apareciendo con normalidad. Si sucede una anomalía, el temporizador de vigilancia se encarga de reinicializar el sistema.

Hay que mencionar que temporizadores de vigilancia para software van incluidos en varios sistemas operativos, como lo son los casos de Linux y TinyOs, que son sistemas operativos bastante utilizados en sistemas empotrados y en las redes WSN comentadas anteriormente.

En cuanto a las opciones de hardware, estas consisten en la inclusión de un contador digital y de un circuito de control adicional al sistema. En la figura siguiente se muestra un ejemplo de configuración. Aquí, el temporizador de vigilancia dispone de un contador interno que cuando llega a cero activa su salida. Tal como se puede ver, esta salida está conectada con el RESET del microcontrolador del sistema empotrado; por lo tanto, su función es reinicializarlo. Al mismo tiempo, el microcontrolador tiene una de sus líneas de entrada/salida (I/O) conectadas al RESET del temporizador de vigilancia. Básicamente, la idea es que el microcontrolador vaya reinicializando periódicamente el contador del temporizador de vigilancia para evitar que llegue a cero. Si el

sistema sufre algún mal funcionamiento, el reinicio del contador no se llevará a cabo y, por lo tanto, el temporizador de vigilancia acabará reinicializando el sistema empotrado.

Esquema de temporizador de vigilancia de hardware



El uso de una opción de software presenta, por lo tanto, una opción menos compleja de implementar y con un coste más reducido. El problema es que no tienen una fiabilidad tan alta como las de hardware y se puede dar el caso de que no reinicialicen el sistema cuando sea necesario.

Además de los temporizadores de vigilancia, también hay sistemas de monitorización del estado del hardware basados, por ejemplo, en la medición de la temperatura de algunas partes del sistema o del voltaje en algún punto. En el momento en el que se detecta algún sobrecalentamiento o sobrevoltaje, el sistema lleva a cabo las acciones más adecuadas. Estos sistemas proporcionan una solución interesante para poder actuar de una manera eficaz e, incluso, prever malos funcionamientos del sistema, pero requieren siempre hardware adicional para poder implementar el sistema de medida y monitorización de los eventos físicos.

Resumen

En este módulo se han estudiado los diferentes elementos que forman los sistemas empotrados, tanto desde el punto de vista del hardware como de las interfaces del software. Así, se introduce, en primer lugar, la arquitectura básica de cualquier sistema empotrado con sus elementos esenciales, que son: procesador, memoria, dispositivos de entrada y dispositivos de salida. En segundo lugar, se detallan los diferentes tipos de procesador que hay (microprocesador, microcomputador, microcontrolador o DSP) y se analiza cómo interactúan con el resto de los elementos del sistema por medio de los buses de comunicaciones (direcciones, datos y control). También ha sido objetivo de estudio la interacción entre máquina y hombre en el ámbito de la programación, es decir, se han visto cuáles son las operaciones que un procesador puede llevar a cabo (conjunto de instrucciones), cómo se indican (código máquina) y qué forma tienen dentro del sistema (instrucciones binarias). Además, se han estudiado las diferentes formas de ejecución de estas instrucciones (entre otras, la secuencial, en lazo o la llamada a subrutinas). A partir de aquí, se ha puesto especial énfasis en el diseño del subsistema de memoria, puesto que es una parte esencial de cualquier sistema empotrado.

A continuación, se ha dividido la memoria en los tipos RAM o de acceso aleatorio, y ROM o de solo lectura. Posteriormente, se ha ampliado la clasificación para intentar dar cabida a los principales tipos de memoria que podemos encontrar en el mercado. Después de esto, se ha trabajado la interacción entre memoria y procesador, y se han detallado las señales que intervienen en ella, así como la temporización que siguen. Después, se han estudiado dos casos prácticos de diseño que pretenden recoger la mayor parte de los problemas con los que nos podemos encontrar en una situación real. En particular, se ha implementado un sistema de memoria basado en SRAM y otro basado en DRAM. La última parte dedicada a la memoria ha incidido en algunos de los mecanismo de *caching* más destacados, que permiten traspasar pequeñas cantidades de datos de la memoria principal a una memoria más pequeña y mucho más rápida para operar y agilizar, así, las operaciones del procesador.

Finalmente, la parte de componentes de hardware concluye con el estudio de piezas más pequeñas pero no por ello menos importantes para los sistemas empotrados, que son los diferentes tipos de registros, los contadores y divisores y el reloj del sistema. Los registros pueden servir, entre otras cosas, como pequeñas memorias temporales (para guardar una sola palabra), para hacer conversiones serie a paralelo y viceversa y también como memorias de entrada serie y salida serie. Además, tienen gran uso en los generadores de números pseudoaleatorios. Los contadores y divisores, que se estudian juntos, puesto que utilizan una arquitectura común con finalidades diferentes, nos sirven para acumular eventos, medir retardos temporales, establecer cuentas atrás u

obtener relojes con diferentes bases temporales a partir de un único reloj de sistema, entre otras cosas. Para acabar, se repasan los diferentes tipos de relojes que hay, pues es muy importante elegir bien sus características (estabilidad y precisión) para asegurar el funcionamiento correcto del sistema. En realidad, es el reloj quien marca el ritmo de ejecución de las diferentes operaciones en el sistema y se debe asegurar que entre dos golpes de reloj haya siempre suficiente tiempo para hacer las tareas encomendadas.

Seguidamente, se han presentado los mecanismos de comunicación que pueden ser utilizados para comunicar los sistemas empotrados con el exterior. Concretamente, se han enumerado las opciones más comunes y, por cada caso, se han discutido las ventajas y desventajas que se encuentran al aplicarse en sistemas empotrados. Para hacer esto, se han dividido los mecanismos en dos grupos de acuerdo con la estrategia utilizada para llevar a cabo la comunicación: mediante puertos de comunicación (con cableado) o usando estándares de comunicaciones inalámbricas. En el primer caso, se ha hecho un repaso de las estrategias más clásicas (como son el puerto serie, paralelo y Ethernet) y de las que se encuentran de manera más común en los últimos años (como USB y Firewire). También se ha tenido en cuenta el puerto CAN, que es uno de los puertos más utilizados en entornos industriales. En el caso de las comunicaciones inalámbricas, se ha empezado la discusión presentando la tecnología IRDA, que se basa en infrarrojos y fue muy utilizada a finales de la década de 1990 y a comienzos de la del 2000. A continuación, se han mencionado sistemas de comunicaciones inalámbricas basados en estándares radio, concretamente los casos más adecuados para los tipos de escenarios encontrados en los sistemas empotrados. Estos estándares son el Bluetooth (muy utilizado en redes tipo WPAN), el Wi-Fi (opción predominante en las WLAN) y el IEEE 802.15.4/Zigbee (solución cada vez más presente, dada en parte la proliferación de las WSN).

Después de esto, se ha presentado el subsistema de alimentación, que es el subsistema encargado de alimentar los diferentes componentes de un sistema empotrado. Se han diferenciado aquellos casos en los que se usa la red eléctrica de aquellos que optan por el uso de baterías y se han enumerado algunas estrategias enfocadas a optimizar la distribución de alimentación del sistema. Además, se ha hecho referencia al ruido electromagnético provocado por el propio subsistema de alimentación, que puede interferir en los diferentes componentes del subsistema. En esta dirección, se han descrito técnicas para reducir este ruido basadas en la reducción del bucle de corriente del subsistema (para reducir directamente el nivel de ruido generado) o en la utilización de condensadores desacopladores en las entradas de los diferentes componentes (para filtrar el ruido presente).

Finalmente, se ha dirigido el problema de detección de errores de hardware durante el funcionamiento del sistema. Como muchos de los sistemas empotrados se usan en localizaciones remotas o de difícil acceso, es necesario que dispongan de mecanismos de recuperación autónomos. En este módulo, se ha

hecho referencia a una de las opciones más utilizadas en sistemas empotrados: los temporizadores de vigilancia. Tal como se ha comentado, un temporizador de vigilancia no es otra cosa que un sistema que se encarga de vigilar si este funciona correctamente y puede ser de tipo software o hardware; el segundo caso es una solución más compleja pero más robusta.

Actividades

En este apartado se proponen algunos ejercicios que ilustran las explicaciones que se han dado en este módulo.

1. Se quiere programar la función void funcion1(int *a, int *b, int c) en código máquina. Considerad el fragmento de código y el estado de la pila en el momento de la llamada de la función que aparecen a continuación e identificad los errores existentes.

```
funcion1:      MOV BP, SP
               PUSH CX
               PUSH BX
               PUSH AX

               ... (Cuerpo de la función) ...

               POP BX
               POP AX
               RET
```

2. Disponemos de bloques de memoria de 2 MB SRAM para construir la memoria de un sistema empotrado que trabaja con palabras de 32 bits y necesita poder dirigir 8 millones de palabras en total. Con esta información, diseñad el subsistema de memoria.

3. Se pretende diseñar el temporizador de vigilancia de un sistema empotrado que trabaja con un reloj de 1 MHz. El temporizador de vigilancia se tiene que activar aproximadamente cada 500 ms si antes no se ha activado la entrada de *reset* correspondiente. Para construirlo, se dispone de un contador síncrono de 9 bits con entrada de *reset* asíncrona y de los biestables y puertas lógicas necesarias. Diseñad el temporizador de vigilancia en cuestión.

4. Un sistema empotrado se encarga de almacenar y transmitir los datos de una estación meteorológica situada en alta montaña. Durante el funcionamiento normal, se transmiten los datos por el Wi-Fi. Sin embargo, se dota al sistema de una tarjeta SD de 32 GB de capacidad para almacenar los datos en el caso de no poder establecer la comunicación vía radio. El sistema incorpora, además, un puerto USB para poder volcar los datos a un disco duro externo si es necesario. Teniendo en cuenta que el sistema es capaz de leer la SD a una velocidad de 125 K palabras por segundo y que la longitud de palabra es de 8 bits, diseñad el módulo de conversión paralelo a serie necesario para extraer los datos si el reloj del sistema es de 1 MHz.

5. En una aplicación de videovigilancia se quiere conectar una videocámera digital a un sistema empotrado encargado de almacenar los vídeos grabados. La videocámara digital trabaja a 30 FPS (*frames per second* o imágenes por segundo) con una resolución de 640 × 480 píxeles, en la que cada píxel es codificado con 24 bits. En este escenario os pedimos seleccionar un mecanismo de comunicación para conectar el sistema empotrado con la videocámara. Se debe tener en cuenta que la videocámera se encuentra emplazada a 4 metros del sistema empotrado y que se quiere asegurar que este reciba los vídeos de la videocámara con un flujo constante de transmisión, que debe ser bastante alto para operar con el mínimo retardo. Además, esto se quiere hacer con el mínimo coste posible y garantizando que la transmisión de vídeo no represente un trabajo extra para el microprocesador del sistema.

6. Considerad una aplicación en la que se quieren conectar dos sistemas empotrados que están separados 60 m. Para llevar a cabo la comunicación, se opta por el uso de tecnología radio basada en el estándar IEEE 802.15.4 que opera en la banda de 868 MHz. Concretamente, se usan dispositivos con una potencia de transmisión igual a 0 dBm y una sensitividad en el receptor de –82 dBm, donde la sensibilidad se define como la potencia mínima necesaria para asegurar que la señal se puede recibir correctamente. Suponiendo que el escenario se pueda modelar con la fórmula de Friis usando un valor de γ igual a 3, justificad si el uso de esta configuración radio es adecuada.

7. Continuando con el escenario del problema anterior, proponed una alternativa de tecnología radio que cumpla los requisitos presentados. Es decir, se supone un escenario en el que los únicos dispositivos IEEE 802.15.4 disponibles son los mencionados en el problema anterior pero, sin embargo, se puede usar algún dispositivo de otro tipo de tecnología radio. El hecho de que se tenga que cambiar de banda frecuencial no implica ningún problema y se continúa suponiendo que, independientemente de la tecnología seleccionada, la sensibilidad del receptor continúa siendo igual a –82 dBm. Tened en cuenta que la consideración inicial de utilizar IEEE 802.15.4 venía determinada por el hecho de que se quiere una solución de bajo coste y de bajo consumo.

8. Se quiere diseñar el subsistema de alimentación de un sistema empotrado. Este sistema usa la red eléctrica como fuente de alimentación y está formado por un microcontrolador, una

memoria ROM, una memoria DRAM, una memoria SRAM y un convertidor analógico-digital. Dibujad el esquema del subsistema sabiendo que el convertidor necesita una entrada estable a 5 V y teniendo en cuenta que se quiere conseguir un sistema robusto a ruido electromagnético y otras posibles anomalías.

Ejercicios de autoevaluación

1. El formato de una instrucción en código máquina debe contener...

a) el código de operación y la dirección de memoria donde se indica la ubicación de los operandos.
b) solo el código de operación.
c) las direcciones de los operandos y la dirección donde se encuentra la instrucción.
d) el código de operación y las direcciones de los operandos, habitualmente entre 1 y 3.

2. En un procesador, la ALU...

a) se encarga de hacer los cálculos numéricos y evaluar las operaciones lógicas.
b) se encarga de la interacción entre el procesador y la memoria.
c) se encarga de tomar las instrucciones y descodificarlas.
d) no forma parte de los procesadores en general, solo de los DSP.

3. Las memorias flash son...

a) memorias de tipo EEPROM. Por lo tanto, la lectura es más rápida que la escritura.
b) memorias de tipo SRAM, puesto que no necesitan refresco para mantener la información que almacenan.
c) memorias DRAM que incorporan un subsistema de alimentación para conservar la información.
d) memorias RAM capaces de mantener la información durante unas horas en caso de que falle el subsistema de alimentación.

4. Indicad cuál de las afirmaciones siguientes es incorrecta en lo referente a la memoria caché.

a) Solo tiene sentido utilizarla cuando sabemos que la actividad del programa se centra en una pequeña zona de la memoria durante un período de tiempo.
b) Trabaja siempre con bloques de la memoria principal.
c) Siempre hay que estar pendiente de los cambios que se producen en caché para después reflejarlos en la memoria principal.
d) Es la que se utiliza en los registros internos del procesador para las operaciones básicas.

5. Las señales CAS y RAS...

a) son típicas de las memorias SRAM.
b) se utilizan en las memorias DRAM, típicamente más grandes, para indicar la dirección del dato en dos veces.
c) solo se utilizan en algunas memorias ROM antiguas.
d) hoy en día están en desuso.

6. Respecto a un contador de Johnson con N registros, podemos afirmar que...

a) el período es de $2N$ y a veces aparecen estados prohibidos.
b) el período es de $2N - 2N$ y no siempre aparecen estados prohibidos.
c) el período es de $2N - 2N$ y siempre hay $2N$ estados prohibidos.
d) el período es de $2N$ con $2N - 2N$ estados prohibidos.

7. Una de las principales ventajas del contador de Johnson respecto al contador síncrono con el mismo número de variables de estado es que...

a) no tiene ninguna ventaja, puesto que el período es inferior y además hay que controlar los estados prohibidos.
b) es más sencillo de implementar.
c) permite descodificar las salidas asegurándonos de que no tendremos transitorios indeseados.
d) se puede utilizar como divisor de frecuencia.

8. Los registros LFSR son...

a) registros de desplazamiento que se utilizan para retrasar una señal.
b) registros de desplazamiento realimentados según un polinomio de conexiones que se utilizan para generar secuencias pseudoaleatorias.
c) un tipo de convertidor serie a paralelo.
d) un tipo de convertidor paralelo a serie.

9. Respecto a la pila del programa, no es cierto que...

a) sea una parte de memoria a la que se accede exclusivamente siguiendo una política LIFO.
b) se utilice para la llamada de subrutinas.
c) tenga un puntero de pila exclusivamente dedicado a apuntar la última posición ocupada.
d) dentro de una subrutina, la podamos emplear para guardar los registros del procesador y evitar así efectos colaterales.

10. Respecto a la representación binaria de una palabra...

a) la representación es única; siempre el bit de más a la derecha es el de más peso.
b) la representación es única pero el bit de más a la derecha es el de menos peso.
c) hay dos representaciones posibles, que son Big Endian, en la que el bit de más a la derecha es el de menos peso, y Little Endian, en la que el bit de más a la derecha es el de más peso.
d) hay dos representaciones posibles, que son Big Endian, en la que el bit de más a la derecha es el de más peso, y Little Endian, en la que el bit de más a la derecha es el de menos peso.

11. Indicad cuál de las afirmaciones siguientes es cierta respecto a modos de direccionamiento.

a) El modo de direccionamiento indirecto es más rápido que el directo.
b) En primera opción, utilizaremos siempre el modo de direccionamiento inmediato.
c) El modo indexado es adecuado para acceder a estructuras de datos.
d) En los sistemas basados en i8086, podemos utilizar el puntero de pila para acceder a los datos de esta usando el modo de direccionamiento indirecto de registro.

12. La memoria SDRAM es...

a) un tipo de memoria SRAM de altas prestaciones.
b) una memoria DRAM que se convierte en estática gracias al hecho de que implementa el refresco de manera interna.
c) una memoria RAM síncrona y, por lo tanto, más rápida que la DRAM convencional.
d) la memoria que hay en las tarjetas de memoria SD.

13. Indicad qué afirmación es correcta.

a) En el estándar Firewire 800, se utilizan normalmente conectores de 4 pines, en los que los 4 pines son utilizados para enviar datos.
b) En el estándar Firewire 800, se utilizan normalmente conectores de 6 pines, en los que 5 pines son utilizados para enviar datos y el pin restante para proveer alimentación.
c) En el estándar Firewire 800, se utilizan normalmente conectores de 6 pines, en los que 4 pines son utilizados para enviar datos y los 2 restantes para proveer alimentación.
d) En el estándar Firewire 800, se utilizan normalmente conectores de 9 pines, en los que 6 pines son utilizados para enviar datos y los 3 restantes para proveer alimentación.

14. Indicad qué afirmación es falsa.

a) En una red de dispositivos USB formada por tres dispositivos, en la que dos trabajan con la versión USB 2.0. y uno con la versión 1.1, la velocidad máxima que podrán utilizar los dispositivos USB 2.0 será de 12 MBps.
b) En una red de dispositivos Firewire formada por tres dispositivos, en la que dos trabajan con la versión Firewire 800 y uno con la versión 400, la velocidad máxima que podrán utilizar los dispositivos Firewire 800 será aproximadamente de 800 MBps.
c) En una red de dispositivos Firewire formada por tres dispositivos, en la que dos trabajan con la versión Firewire 400 y uno con la versión 800, la velocidad máxima que podrán utilizar los dispositivos Firewire 800 será aproximadamente de 800 MBps.
d) En una red de dispositivos USB formada por tres dispositivos, en la que dos trabajan con la versión USB 1.1. y uno con la versión 2.0, la velocidad máxima que podrá utilizar el dispositivo USB 2.0 será de 12 MBps.

15. La variante de Ethernet a 10 MBps más utilizada es...

a) 10Base2 con par trenzado y 185 m de longitud máxima de cable.
b) 10Base2 con cable coaxial y 100 m de longitud máxima de cable.
c) 10BaseT con par trenzado y 100 m de longitud máxima de cable.
d) 10BaseT con cable coaxial y 185 m de longitud máxima de cable.

16. Indicad cuál de las afirmaciones siguientes relacionadas con Ethernet es correcta.

a) Un concentrador retransmite el paquete recibido a todos los dispositivos que tiene conectados y un conmutador reenvía solo el paquete recibido al dispositivo de destino.
b) Un conmutador reenvía solo el paquete recibido al dispositivo de destino y un encaminador retransmite el paquete recibido a todos los dispositivos que tiene conectados.
c) Un conmutador retransmite el paquete recibido a todos los dispositivos que tiene conectados y un encaminador reenvía solo el paquete recibido al dispositivo de destino.
d) Un concentrador reenvía solo el paquete recibido al dispositivo de destino y un conmutador tiene la habilidad de encaminar el paquete por la mejor salida en términos de la salida que ofrece el camino más corto y menos congestionado.

17. Los dispositivos que trabajan con CAN utilizan como método de acceso al medio...

a) CSMA/CD.
b) CSMA/CD+AMP.
c) CSMA/CA.
d) CSMA/CA+AMP.

18. El sistema IRDA es adecuado cuando...

a) se quieren conectar dos sistemas empotrados que se encuentran a 3 m.
b) se quieren conectar dos sistemas empotrados separados por una pared y que se encuentran a una distancia de 50 cm el uno del otro.
c) se quiere transmitir información de manera segura.
d) se quiere transmitir información a alta velocidad.

19. Indicad cuál es la afirmación más acertada.

a) Bluetooth es una estrategia útil para llevar a cabo comunicaciones entre dos sistemas empotrados con altas limitaciones de batería, que están separados 200 m y equipados con dispositivos de clase 1.
b) Bluetooth es una estrategia útil para llevar a cabo comunicaciones entre dos sistemas empotrados con altas limitaciones de batería, que están separados 10 m y equipados con dispositivos de clase 1.
c) Bluetooth es una estrategia útil para llevar a cabo comunicaciones entre dos sistemas empotrados con altas limitaciones de batería, que están separados 10 m y equipados con dispositivos de clase 2.
d) Bluetooth es una estrategia útil para llevar a cabo comunicaciones entre dos sistemas empotrados con altas limitaciones de batería, que están separados 10 m y equipados con dispositivos de clase 3.

20. Indicad cuál de las afirmaciones siguientes es falsa.

a) Wi-Fi basado en IEEE 802.11a trabaja con OFDM y es útil para unir redes remotas, siempre que haya un enlace de visión directa entre ellas.
b) Wi-Fi basado en IEEE 802.11g puede utilizar DSSS.
c) Wi-Fi basado en IEEE 802.11g es una opción aconsejable para sistemas empotrados situados dentro de una nave industrial con una aplicación que requiere conexión con Internet de alta velocidad.
d) Wi-Fi basado en IEEE 802.11b opera con OFDM y puede ofrecer conexiones a 30 m de distancia.

21. Un conjunto de sistemas empotrados que trabajan con IEEE 802.15.4 y formados por nodos tipos RFD pueden...

a) formar una red de igual a igual.
b) formar una red en estrella.
c) usar DSSS como técnica de transmisión.
d) formar una red en árbol.

22. Indicad cuál de las afirmaciones siguientes es falsa.

a) Una red de sistemas empotrados que utilizan IEEE 802.15.4 en modo *beacon* no usa CSMA-CA.
b) Una red de sistemas empotrados que utilizan IEEE 802.15.4 en modo *non-beacon* usa CSMA-CA.
c) Una red de sistemas empotrados que utilizan IEEE 802.15.4 en modo *beacon* puede usar CSMA-CA o tener una serie de *slots* asignados por el coordinador.
d) Una red de sistemas empotrados que utilizan IEEE 802.15.4 en modo *beacon* puede tener una serie de *slots* asignados por el coordinador.

23. Indicad qué estrategia no es adecuada para llevar a cabo el diseño del subsistema de alimentación de un sistema empotrado.

a) Incluir un puente de diodos a la entrada de alimentación del sistema empotrado cuando este se alimenta mediante un adaptador ca.
b) Alimentar todos los dispositivos con la fuente de alimentación, incluidos los dispositivos de bajo consumo, como los sensores y los LED.
c) Utilizar un plan de masa.
d) Incluir condensadores desacopladores en las entradas de alimentación de los componentes del sistema.

24. En un sistema empotrado, un temporizador de vigilancia...

a) es un sistema software que se encarga de vigilar si los datos enviados a un bus de datos contienen errores.
b) es un sistema hardware encargado de asegurar que todos los circuitos tienen un nivel de energía adecuado.
c) presenta una solución menos compleja cuando se implementa en el nivel del software, pero presenta una respuesta menos robusta que su versión hardware.
d) presenta una solución más compleja cuando se implementa en el nivel del software, pero presenta una respuesta más robusta que su versión hardware.

Solucionario

Actividades

1. En este caso, nos encontramos con una función que no devuelve nada y que tiene tres parámetros de entrada, dos de los cuales se pasan por referencia y el otro directamente. Observando la pila, vemos que todo es correcto, es decir, en el momento de la llamada tenemos las direcciones de los dos parámetros pasados por referencia y el valor del parámetro c. Además, la dirección de regreso a la función que ha llamado funcion1 ya se ha apilado.

Como tendremos que poder acceder a los parámetros que nos han pasado mediante la pila y SP no nos sirve (no lo podemos usar para direccionamientos indirectos), copiamos el contenido de SP al registro BP. Esto es correcto. Sin embargo, podría ser que la función que nos ha llamado necesite el contenido de este registro para continuar su ejecución y, por lo tanto, conviene apilarlo antes de hacer la copia.

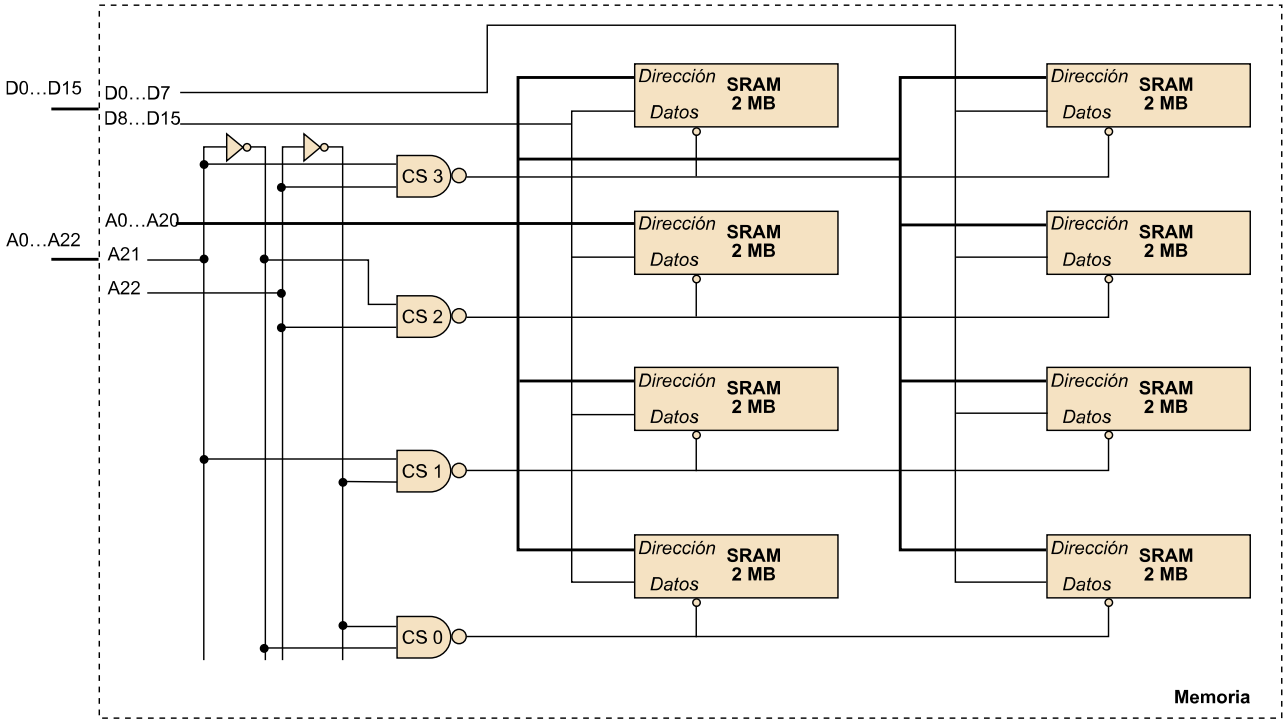
Después, apilamos los registros AX, BX y CX. Esto quiere decir que los usaremos durante el cuerpo de la función. Antes de acabar la función, por lo tanto, se deben restaurar. Aquí hay un primer error, puesto que los hemos de restaurar con el orden inverso al de apilado. Tal como está, pondríamos el valor anterior de AX a BX y el valor anterior de BX a AX. Además, al ejecutar RET volveríamos a la posición que indica CX y no a la que toca. Por lo tanto, falta restaurar CX. Finalmente, teniendo en cuenta que hemos apilado BP, también lo restauraremos. El código correcto es:

```
funcio1:    POP BP
            MOV BP, SP
            PUSH CX
            PUSH BX
            PUSH AX

            ... (Cuerpo de la función) ...

            POP AX
            POP BX
            POP CX
            POP BP
            RET
```

Fijémonos, además, en que dentro del cuerpo de la función accederemos a la dirección de *a* haciendo [BP + 4], a la dirección de *b* haciendo [BP + 6] y al parámetro *c* haciendo [BP + 8].

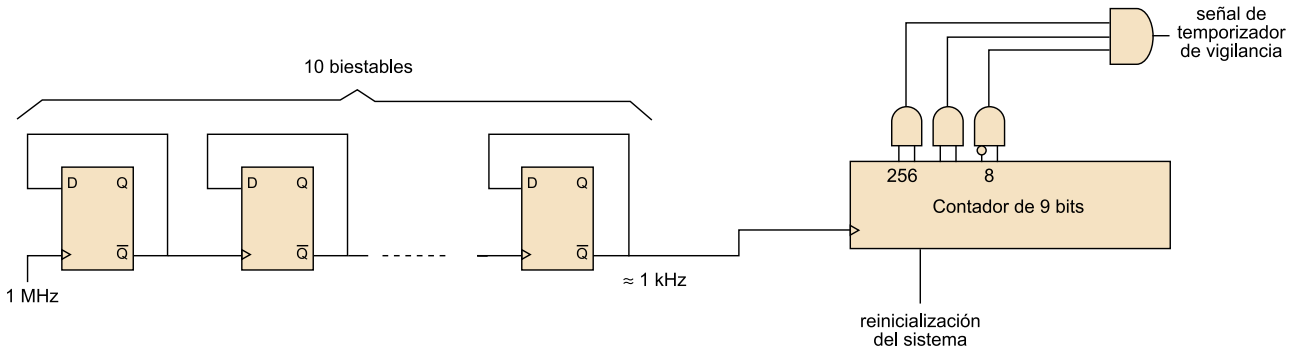
2. En primer lugar, hemos de saber cómo está organizada la memoria que tenemos disponible. Si no nos dicen nada más, supondremos que son memorias de 2 millones de palabras de 8 bits cada una. Por lo tanto, agrupando cuatro bloques de memoria tendremos 2 millones palabras de 32 bits. Finalmente, si queremos conseguir los 8 millones de palabras de memoria, tendremos que repetir esta estructura cuatro veces. Finalmente, en función de la dirección, deberemos distinguir en cuál de estas estructuras guardamos la palabra en cuestión. Para hacerlo, descodificaremos los dos bits de más peso de la dirección, lo cual nos permitirá activar solo una de las estructuras en cada ciclo de memoria. El diseño resultante se puede ver en la figura siguiente:

Diseño resultante de la actividad 2



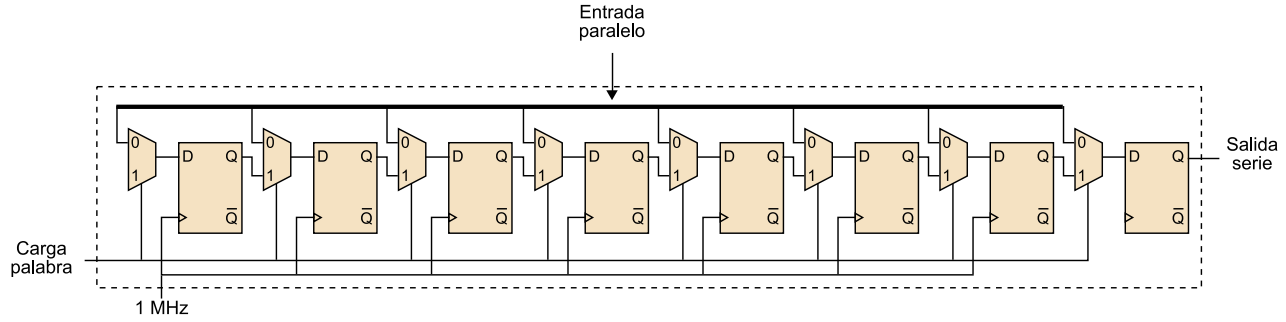
3. Lo primero que hemos de hacer es ver cuál es la resolución temporal del reloj del sistema. En este caso, 1 MHz corresponde a un millón de ciclos de reloj por segundo. Por lo tanto, si contamos medio millón de ciclos de reloj, conseguimos nuestro objetivo. Para hacerlo, necesitaríamos un contador de 19 bits que no tenemos. La solución es implementar un divisor de frecuencia utilizando biestables. En este caso, podemos construir un contador asíncrono, puesto que lo usaremos exclusivamente como reloj. Así pues, si utilizamos 10 biestables, dividiremos la frecuencia por 210 = 1024, con lo cual obtendremos un reloj de 976.5625 kHz, en el que un período son 1,024 ms. Para contar 500 ms con esta nueva base temporal, necesitamos aproximadamente $^{500\ \text{ms}}/_{1{,}024\ \text{ms}} \approx 488$ golpes de reloj. Para acabar, descodificamos el número 488 (111101000 en binario) a la salida del contador, que será la salida del temporizador de vigilancia. De hecho, podemos observar que en este caso, descodificando los seis bits de más peso, será suficiente para cumplir nuestro objetivo. El diseño resultante se puede ver en la figura siguiente:

Diseño resultante de la actividad 3



4. En este caso, hemos de ver que el flujo de datos en paralelo de 125 K palabras por segundo corresponde a 1 MBps en una conexión serie. Ahora nos falta convertir los datos de paralelo a serie, para lo cual utilizamos un registro como el de la figura siguiente. Hay que señalar que, en este caso, el reloj del sistema de 1 MHz ya es el que necesitamos para alimentar los biestables.

Diseño resultante de la actividad 4



5. El primer paso para resolver el problema es calcular cuál es la velocidad requerida para trabajar con la señal de vídeo generada por la videocámara. La videocámara opera con imágenes de 640 × 490 píxeles de 24 bits. Por lo tanto, cada fotograma tiene un número de bits igual a:

$$640\ x\ 490\ x\ 24\ =\ 7\ 372\ 800\ \text{bits}$$

Para obtener la velocidad requerida, se debe tener en cuenta que la videocámara trabaja a 30 FPS:

$$v_{req} = 7\ 372\ 800\ \text{bits/fotogrames}\ x\ 30\ \text{fotogrames/s} = 221.18\ \text{Mbits/s}$$

Es decir, se tiene que proveer al sistema de un mecanismo de comunicación capaz de asegurar velocidades de 221,18 MBps. Este sistema, a la vez, tiene que ser de bajo coste. Repasando los mecanismos de comunicación discutidos en el módulo, estas velocidades se pueden asegurar solo con mecanismos cableados. De entre los disponibles, los que aseguran una velocidad de esta magnitud con un coste reducido, son:

- USB 2.0: 480 MBps con 5 m de longitud de cable.
- Firewire 400: 400 MBps con 4,5 m de longitud de cable.

Al mismo tiempo se observa que ambas tecnologías cumplen el requisito de poder establecer una conexión a 4 m de distancia.

Finalmente, para acabar de decidir qué solución usar, se debe tener en cuenta el requisito del sistema en cuanto a asegurar un flujo constante y un trabajo mínimo por parte del microprocesador del sistema empotrado. En este sentido, Firewire 400 es la mejor opción, puesto que es una tecnología puramente de igual a igual, capaz de asegurar velocidades sostenidas mayores que en el caso de USB y donde no se genera carga de trabajo adicional al microprocesador.

A pesar de que USB es una solución más extensa comercialmente, es en situaciones particulares como esta (en la que se pide el diseño de una solución empotrada con unas condiciones muy particulares) cuando el uso de Firewire resulta la opción más adecuada.

6. En este problema, se pide comprobar si, con la configuración propuesta, el receptor puede recibir potencias superiores a –82 dBm. Como se comenta que el escenario se puede modelar con la fórmula de Friis usando un valor de igual γ a 3, se tendrá que calcular la potencia recibida con la fórmula siguiente:

$$P_r = P_t\left(\frac{c}{4\pi f}\right)^2 \frac{1}{d^3}$$

y, una vez hecho esto, comprobar si $P_r > P_s$, donde P_s es la sensibilidad mencionada anteriormente (expresada en W, en este caso). Como esta fórmula trabaja en lineal, es decir, las potencias se expresan en W, y el valor de potencia transmitida que ofrece el enunciado del problema está expresado en dBm (tal como sucede a la práctica), lo primero que se debe hacer es pasar el valor de potencia en dBm a su equivalente lineal en W. El término dBm, por su parte, se refiere al nivel de potencia que se tiene en escala logarítmica cuando esta potencia se referencia a 1 mW. Por lo tanto, antes de pasarlo a lineal se tendrá que hacer la transformación de dBm a dBW, puesto que este último término es el equivalente referenciado a 1 W y su uso simplifica el proceso de transformación a lineal. Para hacer esto, se debe tener en cuenta que para expresar una potencia, P, en dBm y en dBW, las fórmulas matemáticas utilizadas son:

$$P(\text{dBm}) = 10log_{10}(P(\text{W})/1\text{mW}) = 10log_{10}(P(\text{W})/1{\cdot}10^{-3}\text{W})$$
$$P(\text{dBW}) = 10log_{10}(P(\text{W})/1\text{W})$$

de las que se puede ver fácilmente que:

$$P(\text{dBm}) = 10log_{10}(P(\text{W})/1{\cdot}10^{-3}\text{W}) = 10log_{10}(P(\text{W})/1W) - 10log_{10}(1/10^{-3})$$
$$= P(\text{dBW}) + 30\text{dB}$$

Aplicando estas igualdades, quedará la transformación siguiente:

$$P_t(\text{dBm}) = 0\ \text{dBm} \Rightarrow P_t(\text{dBW}) = 0\ \text{dBm} - 30\ \text{dB} = -30\ \text{dBW}$$

y con este valor se puede hacer ya el paso de dBW a W de la manera siguiente:

$$P_t(\text{W}) = 10^{P_t(\text{dBW})/10} = 10^{-30/10} = 1{\cdot}10^{-3}\ W$$

En cuanto al resto de los parámetros de la fórmula, c se refiere a la velocidad de la luz, $c = 3 \cdot 10^8$ m/s; f es la frecuencia de trabajo de la tecnología radio, $f = 868 \cdot 10^6$ Hz, y $d = 60$ m, tal como menciona el enunciado del problema. Utilizando estos valores tenemos que:

$$P_r = P_t\left(\frac{c}{4\pi f}\right)^2 \frac{1}{d^3} = 1{\cdot}10^{-3}\left(\frac{3{\cdot}10^8}{4\pi 868{\cdot}10^6}\right)^2 \frac{1}{60^3} = 3.5{\cdot}10^{-12}\text{W}$$

Para comparar esta potencia recibida con la sensibilidad, se convierte el resultado anterior a dBm, puesto que el uso de la escala logarítmica ofrece una manera más cómoda de trabajar cuando se opera con estas magnitudes de potencia. Haciendo la transformación se obtiene que:

$$P_r(\text{dBm}) = 10log_{10}\left(3.5{\cdot}10^{-12}\text{W}\right) + 30\text{dB} = -84.55\text{dBm}$$

Tal como se puede ver, la potencia recibida, P_r(dBm) = –84.55 dBm, es inferior a la sensibilidad del receptor, igual a P_s(dBm) = –82 dBm. Por lo tanto, el uso de esta configuración no es adecuada para el escenario del problema.

7. Tal como se ha visto en la solución del problema anterior, la configuración propuesta basada en IEEE 802.15.4 no es adecuada, puesto que la potencia recibida a 60 m no es mayor que la sensibilidad del receptor. Como el enunciado dice que la única configuración disponible para IEEE 802.15.4 es esta, hay que buscar una alternativa tecnológica. Teniendo en cuenta que se busca una solución de bajo coste y consumo, el candidato siguiente es Bluetooth. Tal como se ha visto en teoría, puede haber dispositivos Bluetooth de tres clases según la potencia de transmisión:

- Clase 1: 100 mW de potencia.
- Clase 2: 2,5 mW de potencia.
- Clase 3: 1 mW de potencia.

Dado que más potencia implica más consumo, interesa elegir un dispositivo con el mínimo nivel de potencia posible. Entre las tres clases posibles, el dispositivo de clase 3 queda directamente descartado, puesto que la configuración base, basada en IEEE 802.15.4, también operaba a 1 mW. Además, en el caso de Bluetooth, la potencia recibida todavía será más baja porque se opera en la banda de 2.400 MHz. Por lo tanto, la opción siguiente que hay que verificar es un dispositivo de la clase 2. Como en el problema anterior, se utiliza la fórmula de Friis, pero en este caso se debe tener en cuenta que $P_t = 2{,}5 \cdot 10^{-3}$ y $f = 2.400 \cdot 10^6$ Hz:

$$P_r = P_t\left(\frac{c}{4\pi f}\right)^2 \frac{1}{d^3} = 2.5{\cdot}10^{-3}\left(\frac{3{\cdot}10^8}{4\pi 2400{\cdot}10^6}\right)^2 \frac{1}{60^3} = 1.14{\cdot}10^{-12}\text{W}$$

si se pasa a dBm:

$$P_r(\text{dBm}) = 10log_{10}\left(1.14\cdot10^{-12}\text{W}\right) + 30\text{dB} = -89.43\ \text{dBm}$$

Tal como se puede observar, el valor de potencia no llega a superar la sensibilidad. De hecho, la potencia recibida en este caso es inferior que en el caso del dispositivo IEEE 802.15.4. A pesar de que la potencia de transmisión es mayor, la frecuencia de trabajo también lo es y por este motivo la potencia de señal decae de manera más pronunciada con la distancia.

Si se analiza ahora el caso de clase 1, se ve que la potencia recibida es:

$$P_r = P_t\left(\frac{c}{4\pi f}\right)^2\frac{1}{d^3} = 100\cdot10^{-3}\left(\frac{3\cdot10^8}{4\pi2400\cdot10^6}\right)^2\frac{1}{60^3} = 45.8\cdot10^{-12}\text{W}$$

que al transformarlo a dBm da:

$$P_r(\text{dBm}) = 10log_{10}\left(45.8\cdot10^{-12}\text{W}\right) + 30\text{dB} = -73.39\ \text{dBm}$$

Y, por lo tanto, esta sí es una alternativa adecuada para el escenario del problema, puesto que la señal recibida está por encima de la sensibilidad del receptor.

8. Para diseñar el subsistema de alimentación, lo primero que se debe tener en cuenta es que el sistema empotrado del enunciado es alimentado por la red eléctrica. Por lo tanto, se tiene que usar un convertidor ca y a su salida se pondrá un puente de diodos para asegurar que el sistema será resistente a los cambios de polaridad.

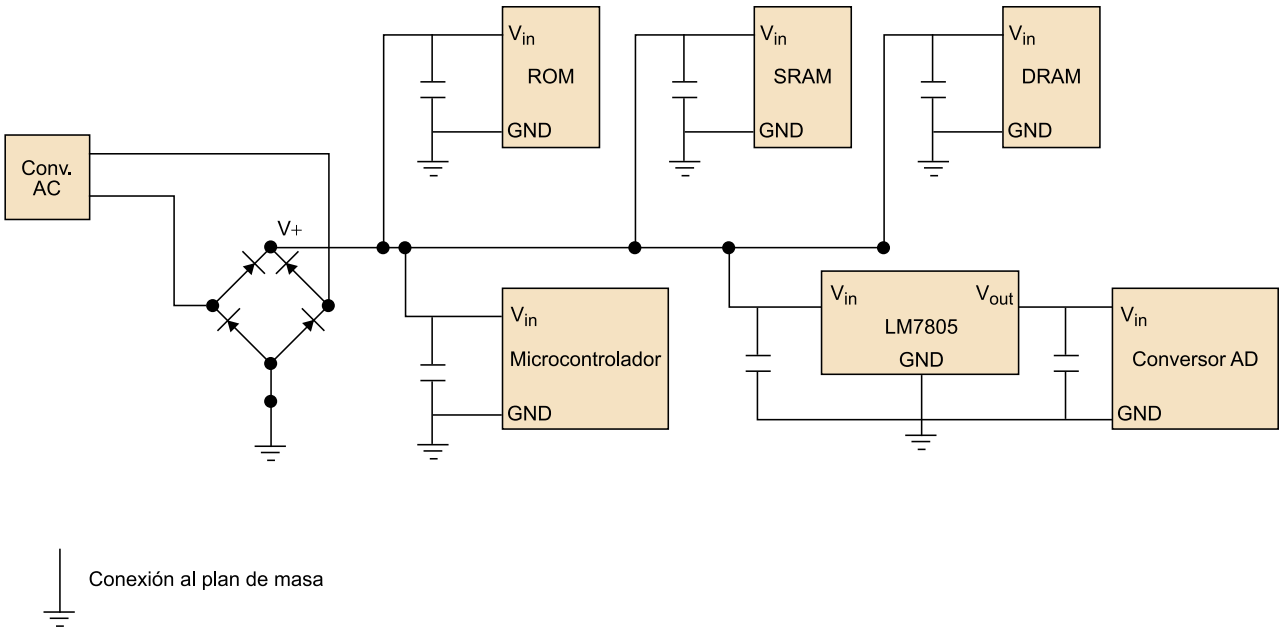
Hecho esto, el paso siguiente es conectar el microcontrolador a la línea de alimentación. Para hacer el sistema resistente al ruido electromagnético, se incluirá un condensador desacoplador en la línea. Por otro lado, se usará un plan de masa para conectar todas las masas del circuito y de este modo se reducirá el bucle de corriente.

Para alimentar las diferentes memorias se seguirá el mismo procedimiento que el utilizado en el caso del microcontrolador.

En el caso del convertidor analógico-digital, se usará un regulador para asegurar que el componente se alimenta con una tensión a 5 V. Para ello, se puede usar el regulador comercial LM7805. Por otro lado, para evitar el ruido electromagnético, se utilizará el esquema basado en el regulador lineal con dos condensadores desacopladores.

En la figura siguiente se muestra el esquema del diseño resultante, en el que las entradas V_{in} y GND se refieren a las entradas de alimentación y masa, respectivamente. Observad que solo se han dibujado las líneas de alimentación del sistema, puesto que el problema se refería solo al diseño del subsistema de alimentación.

Diseño resultante de la actividad 8



Ejercicios de autoevaluación

1. d

2. a

3. a

4. d

5. b

6. d

7. c

8. b

9. a

10. c

11. c

12. c

13. d

14. c

15. c

16. a

17. b

18. c

19. b

20. d

21. c

22. a

23. b

24. c

Bibliografía

Alcubilla, R.; Pons, J.; Bardés, D. (2004). *Diseño digital. Una perspectiva VLSI-CMOS*. Barcelona: UPC.

Catsoulis, J. (2005). *Designing Embedded Hardware*. Sebastopol, California: O'Reilly and Associates.

Peckol, J. K. (2008). *Embedded Systems: A Contemporary Design Tool*. Hoboken, Nueva Jersey: Wiley.

Yaghmour, K. (2003). *Building Embedded Linux Systems*. Sebastopol, California: O'Reilly and Associates.