6. Arquitecturas hardware para tiempo real

Contenido

6.1 Introducción

Existen una serie de factores que hacen necesario un conocimiento del hardware para la programación y utilización de los sistemas de tiempo real.

El comportamiento temporal de un sistema viene condicionado por las características del hardware. No se puede suponer un comportamiento determinista si los dispositivos físicos no lo pueden asegurar. Por tanto, no todas las arquitecturas se pueden utilizar para realizar aplicaciones de tiempo real pues, como es de suponer, los mecanismos software que se utilizan en la programación de estas aplicaciones han de estar soportadas por los dispositivos físicos.

En muchas ocasiones los mecanismos tradicionales de programación no ofrecen las prestaciones necesarias para cumplir las restricciones temporales que puede exigir un sistema. En tal caso, no hay más remedio que utilizar la programación a bajo nivel y esta está siempre estrechamente relacionada con el hardware.

A menudo, los dispositivos de entrada / salida utilizado en los sistemas de control son específicos y, por tanto, no son periféricos de propósito general sobre los que existan herramientas de programación estándar. En tal caso, no hay más remedio que programar el hardware directamente, siendo obligado tener en cuenta las prestaciones físicas de estos elementos para que el sistema siga siendo determinista.

Las políticas de planificación requieren que los trabajos más prioritarios puedan interrumpir a los menos prioritarios. A menudo este mecanismo de prioridades debe ser apoyado por el hardware y se debe utilizar con exactitud para que sea coherente con la política de planificación.

Cuando se interconectan varios dispositivos, el mecanismo de transmisión de información es determinante para el comportamiento temporal. Tanto en la comunicación a través de buses como utilizando redes es necesario que el medio de transmisión tenga un comportamiento predecible y ofrezca mecanismos para realizar correctamente la planificación de las tareas que utilizan el medio.

Por todo ello, se hace necesario un conocimiento del hardware y su influencia en el comportamiento temporal de los sistemas informáticos.

6.2 Entrada / salida.

Una de las principales características de los sistemas empotrados es la necesidad de interactuar con dispositivos de entrada / salida de propósito especial.

Para estudiar los mecanismos que deben ofrecer los lenguajes para el manejo de periféricos, es necesario comprender primero el funcionamiento básico del hardware de entrada / salida.

Respecto a los periféricos de entrada salida, existen dos clases generales de arquitectura de ordenadores:

- Una en la que la memoria y los periféricos utilizan buses independientes a nivel lógico (ej. arquitectura Intel).

- La otra en la que la memoria y los periféricos comparten el mismo bus (ej. arquitectura Motorola). En este caso ciertas direcciones dan acceso a la memoria física y otras a los periféricos. Este modelo se llama Entrada / Salida mapeada en memoria.

El interfaz con un periférico normalmente es a través de un conjunto de registros.

En ambos modelos de arquitectura suelen existir instrucciones especiales para acceder a los registros de los periféricos.

- Cuando la memoria y los periféricos utilizan buses separados el ordenador debe disponer de dos juegos de instrucciones, uno para acceder a la memoria y el otro para acceder a los registros de los periféricos.

- Cuando comparten el mismo bus las instrucciones especiales permiten manejar un tamaño de registro distinto al tamaño de la palabra de memoria (normalmente los registros de los periféricos son de 8 bits mientras que la memoria suele tener un tamaño de palabra mayor).

Para manejar un periférico de entrada / salida existen dos mecanismo generales distintos; estos son el control por supervisión del estado y el control por interrupción.

6.2.1 Control por supervisión del estado

En este tipo de mecanismo de control, el programa debe realizar tests explícitos para determinar el estado de un periférico dado. Una vez se ha determinado el estado del periférico, el programa ya puede llevar a cabo las acciones apropiadas.

Típicamente, hay tres tipos de instrucciones hardware que soportan este tipo de mecanismo:

- Las operaciones de test que permiten a un programa determinar el estado de un periférico dado.

- Las operaciones de control que conducen a realizar acciones dependientes del periférico que no son de transferencia, tal como seleccionar registros de lectura.

- Operaciones de entrada / salida que realizan la transferencia actual de datos entre el periférico y la CPU.

Aunque los periféricos por control de supervisión de estado eran muy comunes hasta hace poco debido a su menor costo, con motivo de la reducción del precio del hardware y la evolución de los componentes, están siendo reemplazados por periféricos con control por interrupción. No obstante, el control por interrupción puede ser sustituido por el control por supervisión del estado en la mayoría de los periféricos.

Las interrupciones presentan el inconveniente que introducen un comportamiento no determinista en los sistemas de tiempo real, lo que puede resultar prohibitivo en entornos de gran seguridad.

6.2.2 Control por interrupción

En el mecanismo de control por interrupción, existen varias posibilidades dependiendo de la forma en que se realiza la transferencia, de como se inicializa y de como se controla. Existen tres variaciones del control por interrupción:

- Controlada por programa
- Inicializada por programa
- Controlada por canal programable

Interrupciones controladas por programa

En este caso, un periférico solicita una interrupción como resultado de un determinado evento, por ejemplo la llegada de un dato. Cuando la petición es reconocida, el procesador suspende la ejecución del proceso e invoca al manejador de interrupción que ésta tenga asociado, el cual realiza la acción apropiada en respuesta de la interrupción. Cuando el manejador de la interrupción finaliza, se restaura el estado del procesador al estado previo de su activación, y se devuelve el control al proceso interrumpido.

Iniciada por programa

Este tipo de mecanismo de entrada / salida se denomina habitualmente acceso directo a memoria (DMA). El dispositivo DMA va situado entre el dispositivo de entrada / salida y la memoria principal. El dispositivo DMA toma el lugar del procesador en la transferencia de datos entre el dispositivo de entrada/salida y la memoria. Aunque la operación de entrada / salida se inicializa por programa, el control de una transferencia de datos es realizada por el DMA (en forma de bloques). Para cada palabra de datos que se transfiere, el dispositivo DMA realiza una petición de un ciclo de memoria y la transferencia se realiza cuando se satisface esta petición. Cuando ha finalizado la transferencia de un bloque entero, el DMA realiza un petición de interrupción de transferencia finalizada. Esta interrupción es manejada utilizando un mecanismo de interrupción controlada por programa

Controlada por canal programable

La entrada/salida controlada por canal programable es una extensión de la iniciada por programa, pero eliminando lo máximo posible la participación del procesador central en el proceso de transferencia de datos. Este mecanismo está constituido por tres elementos básicos:

- El hardware que constituye el canal y los periféricos conectados.
- El programa que define las acciones del canal (denominado guión o script).
- Las instrucciones de entrada/salida para manejar el canal.

La ejecución del programa del canal es iniciada por la aplicación. Este programa se ejecuta en paralelo con el procesador principal por una CPU independiente incluida en el canal. Una vez ha finalizado la operación se da un aviso a la CPU principal por medio de una interrupción. La principal ventaja de este mecanismo frente al anterior es que permite realizar operaciones de entrada / salida más complejas sin intervención del procesador central (un ejemplo serían las tarjetas de comunicación serie multipuerto).

En muchos aspectos, el hardware del canal se puede ver como un procesador independiente que comparte memoria con el procesador central. La memoria compartida es utilizada pos ambos procesadores para comunicarse los datos enviados o recibidos.

Su efecto sobre los tiempos de acceso a memoria son impredecibles, al igual que ocurre con los dispositivos que utilizan DMA.

6.3 Programación a bajo nivel

6.3.1 Requerimientos del lenguaje

Tradicionalmente, las programación de los periféricos se ha realizado con un componente grande de lenguaje ensamblador, pero los lenguajes de alto nivel como son el C, Modula, y Ada tienden a ofrecer mecanismos que permiten realizar estas funciones de bajo nivel. Esto permite que el software de manejo de los periféricos, bien por supervisión de estado o por manejadores de interrupción, sea más fácil de leer, escribir y de mantener.

No es fácil decidir cuales son las características más importantes que debe ofrecer un lenguaje para la programación de los periféricos de forma adecuada. De todas formas parecen importantes las siguientes:

- Facilidades de encapsulación
- Abstracción

6.3.1.1 Facilidades de modularidad y encapsulamiento

La interfaz para el manejo de los dispositivos de entrada / salida es, necesariamente, dependiente del hardware y, por tanto, poco portable. En el software de un sistema, es importante separar las secciones de software portables de las que no los son. Por tanto, siempre que sea posible, es aconsejable encapsular el software que es dependiente de la máquina en unidades que sean fácilmente identificables. Por ejemplo, en modula 2 existe un tipo de módulo llamado '**device module**'. En Ada se utilizan los paquetes y los tipos protegidos. En C se utilizan ficheros de fuente independientes.

6.3.1.2 Modelos abstractos para el manejo de periféricos

Un periférico puede verse como un procesador que realiza una tarea fija. Un sistema informático puede considerarse entonces como una serie de procesos en paralelo. Existen varios modelos según los cuales un proceso asociado a un periférico puede comunicarse y sincronizarse con el proceso que se ejecuta en el procesador principal. Todos estos modelos deben ofrecer:

1) *Facilidades para representar, direccionar y manipular los registros del periférico*. Un registro de periféricos se puede representar como una variable de programa, un objeto o incluso como un canal de comunicación.

2) *Una representación adecuada de las interrupciones*. Son posibles las siguientes representaciones:

    a) **Procedimiento**. Una interrupción se ve como una llamada a un procedimiento. La llamada se realiza de forma remota por el proceso del periférico. Cualquier

comunicación y sincronización debe programarse en el procedimiento asociada al manejador de la interrupción. En este procedimiento se puede acceder a los datos locales y a las variables globales del sistema.

b) **Proceso esporádico**. La interrupción se ve como un proceso que es activado de forma esporádica. Este proceso puede tener variables locales persistentes y acceder a las variables globales. Puede utilizar los mecanismos de comunicación y sincronización entre procesos.

c) **Evento asíncrono**. La interrupción se ve como un evento asíncrono asociado a un proceso. Desde el manejador de este evento se puede tener acceso a las variables locales del proceso y a las variables globales. Están disponibles tanto el modelo de reanudación como el de terminación.

d) **Condición de sincronización por variable compartida**. La interrupción se ve como una condición de sincronización incluida dentro del mecanismo de sincronización de las variables compartidas; por ejemplo la operación de señalización en un semáforo o la operación 'send' en una variable condición o en un monitor. El manejador puede acceder tanto al estado local del proceso monitor como al estado global.

e) **Sincronización basada en paso de mensaje**. La interrupción se ve como un mensaje que es recibido por un canal de comunicación. El proceso de recepción puede acceder al estado local del proceso y al estado global.

No todos estos modelos se suelen encontrar en los lenguajes y sistemas operativos de tiempo real. El más popular es el modelo de procedimiento ya que requiere un soporte reducido partiendo de los mecanismos que suele ofrecer el hardware. Usualmente, los sistemas operativos implementados utilizando el C y C++ utilizan este modelo, representando los registros de los dispositivos como variables de programa.

En el lenguaje Ada el modelo es un híbrido entre el modelo de procedimiento y el modelo de sincronización por variable compartida. La interrupción es mapeada sobre un procedimiento protegido y se accede a los registros por medio de variables de programa.

El KMOS adopta el modelo de sincronización basada en paso de mensajes. Las interrupciones se traducen en mensajes que son enviados a un mailbox.

6.3.2 Ada

En Ada 83, las interrupciones se representan por llamadas a entry de tareas generadas por el hardware. En la versión actual de Ada 95, esta facilidad se considera obsoleta y probablemente se quitará en futuras versiones.

El método preferido en el manejo de los periféricos es encapsular sus operaciones en una unidad protegida. Una interrupción se puede manejar por medio de una llamada a un procedimiento protegido.

6.3.2.1 Manipulación de registros de periféricos

Ada ofrece al programador un conjunto de facilidades para especificar la implementación de los tipos de datos. A estas se conocen como **cláusulas de representación**, e indican como los tipos del lenguaje se mapean sobre el hardware utilizado. Un tipo sólo puede tener una única representación, ahora bien, esta se puede dejar a elección del compilador o se pueden utilizar unos indicadores adicionales para indicar el tipo de representación. La representación específica del tipo se indica de forma separada a la representación lógica del tipo.

Las cláusulas de representación son un compromiso entre las estructuras abstractas que se utilizan en la programación y las concretas que se utilizan en la representación. Hay disponibles tres formas distintas de especificación para los tipos:

- **Cláusulas de representación de enumeración**. Se da valores internos concretos a los literales de los tipos enumerados.

- **Cláusulas de representación de registros**. Permite asignar desplazamientos y longitudes a los componentes de un registro en el interior de una única unidad de almacenamiento.

- **Cláusulas de definición de atributos**. Permite seleccionar varios atributos de un objeto, tarea o subprograma; por ejemplo, el tamaño en bytes de los objetos, la alineación en la representación en memoria, el tamaño máximo del espacio de almacenamiento de una tarea o la dirección de un objeto.

Para ilustrar estas cláusulas de representación, consideremos las siguientes declaraciones de tipos:

```
type Mode_select is new integer range 0..5;
type Format_select is (Latch, MSB_only, LSB_only, LSB_and_MSB);
type Counter_select is new integer range 0..2;
type Timer_control is record
      BCD : Boolean;
      Mode: Mode_select;
      Format: Format_select;
      Counter: Counter_select ;
end record;
```

Si deseamos utilizar la estructura `Timer_control` para manejar un registro de un periférico con los campos que en ella se definen, habrá que hacer coincidir las posiciones de cada campo y los posibles valores de los datos que en ellos se guardan, con los utilizados por el periférico.

La implementación por defecto que utiliza el Ada no puede abarcar todas las posibilidades utilizadas en los periféricos. Más aún, posiblemente no coincidirá con ninguno pues los lenguajes procuran utilizar una implementación que optimice el uso de la estructura sobre memoria, normalmente separando cada campo en una unidad de memoria (32 bits en un micro de 32 bits), mientras que los registros de los periféricos tienden a aprovechar al máximo los bits de una palabra.

Para hacer coincidir la implementación de la estructura con representación en el periférico se utilizan las cláusulas de representación. En nuestro caso utilizaremos las siguientes:

Para dar los valores adecuados a los literales de los campos del tipo `Format_select` usaremos la cláusulas de representación de enumeración:

```
for Format_select use (Latch => 0, MSB_only => 1,
                       LSB_only >= 2, LSB_and_MSB => 3);
```

La cláusula de representación de registro nos permite indicar para cada campo de una estructura su orden y, su tamaño y su posición en bits. El byte de posición en el registro se numera a partir de 0 y el rango en la cláusula de cada componente especifica el tamaño en número de bits que ocupa.

En nuestro caso usaremos la siguiente representación:

```
for Timer_control use record
      BCD     at 0 range 0..0;
      Mode    at 0 range 1..3;
      Format  at 0 range 4..5;
      Counter at 0 range 6..7;
end record;
```

Se pueden definir algunos atributos para el tipo de datos que indican la forma de representar la estructura, como por ejemplo:

```
for Timer_control'Size use 8;
for Timer_control'Alignement use 1;
for Timer_control'Bit_Order use Low_Order_First;
```

Finalmente, una instancia de un registro necesita direccionarse en la posición que es visible el periférico. Esto se hace también definiendo un atributo, pero esta vez en una variable en lugar del tipo:

```
Timer_control_Reg: Timer_control;
for Timer_control_Reg use System.Storage_Element.To_Address(16#43#);
```

Una vez se ha construido la representación abstracta del registro de forma adecuada, y se situado la variable en la dirección correspondiente al hardware, se puede manipular el contenido del registro como si se tratara de una variable normal:

```
Timer_control_Reg := (BCD => False, Mode =>3,
                      Format => Latch, Counter => 0);
```

y hacer uso de su contenido de forma análoga:

El objeto `Timer_control_Reg` es, por lo tanto, un grupo de variables compartidas, las cuales son compartidas por la tarea de control del periférico y el periférico en si. Para hacer un uso fiable y eficaz es necesario utilizar exclusión mutua entre los dos procesos paralelos. Esto en Ada se hace utilizando los objetos protegidos.

6.3.2.2 Manejadores de interrupción

En Ada existen tres mecanismos que permiten la sincronización y la comunicación entre procesos:

- El mecanismo de cita
- Utilizando unidades protegidas
- A través de variables compartidas

En general, en Ada se asume que el periférico y el programa tienen acceso a los registros del periférico en memoria compartida.

La representación principal de una manejador de interrupción en Ada es un procedimiento protegido sin parámetros. Cada tipo de interrupción tiene un único identificador discreto que es ofrecido por el sistema. Debido a que este identificador único es dependiente de la implementación, este puede ser, por ejemplo, la dirección del periférico que la ha generado o el número de vector hardware asociado a la interrupción. Este identificador es utilizado para programar el manejador de una determinado interrupción.

Para asociar una interrupción al procedimiento protegido se usa una sentencia `pragma`:

```
pragma Attach_Handler(procedimiento, identificador);
```

Donde el procedimiento no debe tener parámetros y el identificador determina la interrupción a utilizar, normalmente usando el número de interrupción.

6.3.2.3 Instrucciones especiales para acceder a los periféricos

Ada permite la utilización de código ensamblador integrado en los programas, en el caso de que sean necesarias instrucciones especiales que no se pueden codificar con el propio lenguaje.

El mecanismo de inserción de código permite a los programadores escribir código Ada que contiene objetos visibles que no están escritos en Ada.

La inserción de código ensamblador no está permitido hacerlo de forma indiscriminada, lo que restaría claridad a los programas, sino que obliga a hacerlo de forma controlada. Si un subprograma contiene sentencias de código en ensamblador, entonces no se pueden utilizar sentencias Ada, sólo sentencias ensamblador.

El subprograma con código ensamblador dispondrá de una parte visible con el formato Ada de definición de subprogramas, lo cual permite utilizar este código desde los programas.

Como es de esperar, los detalles y características de la utilización de código ensamblador son muy dependientes de la implementación; junto con el código se suelen utilizar pragmas y atributos para imponer restricciones particulares en la utilización de objetos definidos con códigos de instrucciones.

Un ejemplo de subprograma implementado en código ensamblador, sin entrar en detalles, sería:

```
Procedure In_Op; pragma Inline(In_op);

procedure In_Op is
          use System.Machine_code;
begin
          My_Machine_Format'(Code => In_instruction,
                                Reg => 1, Port => 1);
          My_Machine_Format'(Code => SAVE, Reg => S'Address);
end;
```

El `pragma Inline` indica al compilador que al utilizar el procedimiento inserte el código completo en lugar de hacer la llamada.

6.3.3 C, C++

Los lenguajes C y C++ no soportan la multitarea y, por tanto, no poseen un mecanismo que permita comunicar los manejadores de interrupción con las tareas de usuario. Lo normal es programar sobre un entorno que ofrezca la multitarea, bien sea una librería o un sistema operativo. En ambos casos es este entorno el que debe ofrecer la posibilidad de que un manejador de interrupción se comunique con un proceso.

La llamada a un manejador de interrupción se hace de forma distinta a un procedimiento normal. En general, un procedimiento en C asume que ciertos registros los puede modificar sin salvarlos, lo que no puede ocurrir en un manejador de interrupción que a su salida debe dejar inalterados todos los registros de la CPU. En la vuelta de un manejador se debe restaurar el contenido del registro de flags, lo que no se suele hacer al volver de una función. En algunos procesadores se realiza un cambio de stack y/o un cambio de modo para los manejadores, lo que no ocurre en los procedimientos.

Algunos compiladores permiten distinguir la definición de un manejador de una interrupción de la de un procedimiento, encargándose el compilador de generar el código adecuado en cada caso. Por ejemplo, los compiladores de C y C++ de BORLAND definen un manejador de la forma:

```
void interrupt handler(void)
{
          ...
}
```

Si se trabaja en un entorno multitarea, posiblemente éste tenga prevista una forma de instalar un procedimiento como un manejador de interrupción. En realidad el manejador lo construirá este entorno y desde este se llamará a nuestro procedimiento.

Si no hay otro remedio, se deberá construir el manejador en código ensamblador y realizar en él una llamada a un procedimiento escrito en el lenguaje de alto nivel.

El lenguaje C, al igual que el Ada, permite definir estructuras a nivel de bit para acceder a los registros de los periféricos. La forma de hacerlo es con la sentencias `field`. Veamos un ejemplo y como utilizarla para acceder al periférico:

```
typedef struct {
     unsigned char bcd              : 1;
     unsigned char mode             : 3;
     unsigned char format           : 2;
     unsigned char counter          : 2;
} Timer_control;

enum { Latch = 0, MSB_Only = 1, LSB_Only = 2, LSB_and_MSB = 3 };

Timer_control *register, shadow;

shadow.bcd = 0;
shadow.mode = 3;          /* Onda cuadrada */
shadow.format = LSB_and_MSB;
shadow.counter = 0;

*register = shadow;
```

Los compiladores de C y C++ más modernos permiten incluir sentencias de código ensamblador en los ficheros de fuente. Hay dos formas de hacerlo, con la sentencia `asm` y con la pareja: `#pragma asm` y `#pragma endasm`. La primera forma permite incluir sentencias de código máquina junto con sentencias de C en la implementación de un procedimiento. La utilización del `#pragma` permite programar en ensamblador dentro de un fichero en C.

Para ganar en claridad conviene utilizar las sentencias `asm` en `#define`'s, construyendo macros para poder realizar en ensamblador funciones que no es posible realizarlas con el lenguaje de alto nivel. Un ejemplo podría ser provocar generar una instrucción que genere una interrupción software. Por ejemplo:

```
#define INT_DOS asm(“ int $0x21”);
#define disable() asm(“ cli”);
```

6.3.4 Planificación y drivers de periféricos

Muchos sistemas de tiempo real incluyen componentes de entrada / salida, por tanto, es importante que el análisis de la planificación incorpore algunas características particulares que tengan en cuenta la programación a bajo nivel.

Ya se ha comentado que los dispositivos con DMA y con canales programables normalmente presentan un comportamiento temporal no predecible, lo que hace imposible su inclusión en la planificación. Nosotros vamos a centrarnos en el control de periféricos por interrupción y por supervisión de estado.

Cuando una interrupción pone en ejecución un procesos esporádico, aunque este tenga menor prioridad que los procesos no esporádicos, la activación de su manejador interrumpe también a los procesos no esporádicos y, por tanto, se puede considerar que actúa como un proceso de mayor prioridad.

Este es un ejemplo de inversión de prioridad. Si la interrupción tiene la finalidad de lanzar un proceso esporádico, idealmente, debería tener su misma prioridad, y no interferir a los procesos de prioridad mayor.

Desafortunadamente, la mayoría de las plataformas hardware requieren que las interrupciones tengan mayor prioridad que los procesos software. Entonces, para modelar un manejador de interrupción se añade un proceso 'extra' en el test de planificabilidad. A este nuevo proceso se le asignará el mismo periodo que el proceso esporádico, una prioridad igual a la prioridad hardware (mayor que cualquier prioridad software) y un tiempo de ejecución igual al más largo que puede durar el manejador.

En los periféricos controlados por supervisión de estado, el código de control se puede hacer de manera usual. No obstante, estos dispositivos introducen una dificultad particular. A menudo, el protocolo seguido para utilizar el periférico es el siguiente: lanzar la lectura, esperar a que el hardware realice la lectura, acceder al registro que proporciona la lectura al programa. El problema es como manejar el retardo mientras se toma la medida.

Dependiendo de la duración del retardo, existen tres posibilidades:

- Espera ocupada observando un flag de 'listo'.
- Suspender el proceso hasta un tiempo futuro.
- En procesos periódicos, aplazar la acción hasta el siguiente periodo partiéndola en dos.

En retardos muy pequeños, la espera ocupada es aceptable. Desde el punto de vista de la planificación, el 'retardo' es un tiempo de cálculo añadido al proceso y, si el retardo esté acotado, no afecta al método de análisis. Como protección contra el fallo del periférico (o sea, que nunca active el bit de listo), se puede utilizar un algoritmo de espera ocupada con time-out.

Si el retardo es estimable, conviene suspender el proceso y, durante este tiempo, realizar otras acciones hasta que la lectura esté lista al final del retardo. Por ejemplo, si el tiempo para realizar al lectura son 30 mseg. El código podría ser:

```
begin
     -- programar la lectura
     delay 0.03
     -- tomar la lectura y utilizarla
end;
```

Desde la perspectiva de la planificación, esta estructura tiene tres implicaciones significativas:

1) El tiempo de respuesta no resulta fácil de calcular. Se debe analizar por separado cada mitad en que queda dividido el proceso por el retardo. El tiempo de respuesta total se obtiene añadiendo el valor del retardo a los dos subtiempos de cálculo de cada mitad. Aunque exista un retardo en el proceso, este se ignora en lo referente al impacto sobre los procesos de menor prioridad.

2) El tiempo de computación extra utilizado en detener al proceso para realizar el retardo y para activarlo al final de este, debe añadirse al tiempo de ejecución del peor de los casos del proceso.

3) Existe un impacto en los tiempos de bloqueo, en el caso de que el proceso que realiza el retardo tenga bloqueado un recurso cuando se realice el retardo, y este se utilice en procesos

de mayor prioridad. Esto es usual en el caso de compartirse el periférico pues se suelen utilizar recursos para garantizar acceso exclusivo durante las operaciones que se realizan sobre él.

Existe una forma sencilla de no interferir en la planificación. Esta es, realizar la lectura en el siguiente periodo de activación., siempre que esto sea aceptable para la aplicación. Para asegurarse que existe un hueco suficiente entre el final de la ejecución de un periodo y el principio del siguiente, si S es el tiempo mínimo que requiere el periférico para ofrecer una lectura, se debe de cumplir que: $S £ P - D$.

6.4 Sincronización por interrupciones

La sincronización por interrupciones puede presentar diferentes características según la implementación. Entre las cuestiones que caracterizan un sistema particular podemos enumerar:

1) ¿Cuantas líneas de interrupción existen?

2) ¿Como responde la CPU a una interrupción?

3) ¿Como determina la CPU la fuente de una interrupción en el caso que el número de fuentes supere al número de entradas?

4) ¿Como distingue la CPU entre las interrupciones más o menos importantes?

5) ¿Como y cuando el sistema de interrupciones se activa o desactiva?

Existen muchos tipos de respuestas para estas cuestiones. En cualquier caso, lo que se pretende es siempre servir lo más rápido posible la interrupción y devolver el control cuanto antes al programa interrumpido.

El número de entradas de interrupción en el chip de la CPU determina el número de respuestas distintas que puede producir la CPU sin la utilización de hardware o software adicional. Cada entrada provocaría una respuesta interna distinta, es decir, se activaría un manejador distinto, que estará asociado a un vector de interrupción..

Pero lo normal es que en un sistema, y más aún en los sistemas de control de tiempo real, se precise tratar más interrupciones que el número de líneas de interrupción que posee. Será necesario emplear algunos mecanismos que nos permita utilizar una misma línea para más de un tipo de interrupción. Los más utilizados es el polling (software) y la vectorización (hardware).

6.4.1 Polling e interrupciones vectorizadas

La solución más sencilla para que varios periféricos compartan la misma línea de interrupción es conectando la salida de interrupción de cada periférico a la misma entrada de la CPU con una puerta OR. En este caso, cualquier periférico activará el mismo manejador y este deberá verificar el estado de todos los periféricos que pueden haberlo activado para tratar la interrupción correspondiente.

Para poder usar el **mecanismo de polling** el periférico debe poder indicar, en alguno de sus registros de estado, si está provocando interrupción.

Una vez se ha descubierto el periférico que provoca la interrupción se tratará y se indicará que borre la interrupción.

Finalmente el manejador puede seguir buscando periféricos que hayan producido interrupción y terminar devolviendo el control al programa. Si quedara alguna interrupción por tratar se volvería a activar el manejador cuando se volvieran a activar las interrupciones y este volvería a buscar el periférico que la provoca.

Si se desea optimizar el tiempo en que el procesador está bajo interrupción, es preferible volver a buscar otro periférico con la interrupción pendiente a la salida del manejador. De esta forma se evita el restaurar y guardar el estado del proceso interrumpido cuando coinciden más de una interrupción.

El orden en que se revisan los periféricos para hacer el polling puede ser importante, pues se está dando prioridad a los primeros que se miran. Por ejemplo, si una misma línea de interrupción es compartida por un periférico de entrada y otro de salida, es preferible mirar primero el de entrada para no perder información que llega desde fuera por retrasos producidos en tratar una interrupción de salida. Si la salida quedara retrasada lo único que ocurriría es que saldría más lentamente.

El mecanismo de polling se utiliza de forma análoga en un mismo periférico cuando este es capaz de producir varios tipos de interrupción. Por ejemplo, un periférico de entrada/salida serie puede producir interrupciones de entrada, de salida, de líneas de control de errores y de tratamiento de errores.

Existe un mecanismo hardware para evitar la pérdida de tiempo en la realización del polling que permite distinguir que periférico ha provocado una interrupción, de los que comparten una misma línea, y activar manejadores distintos. A este mecanismo se le llama **interrupciones vectorizadas**.

Cuando el procesador recibe una petición de interrupción vectorizada activa otra línea de reconocimiento. En este momento el periférico indica en el bus da datos un valor que identificará su interrupción. A este valor se le denomina vector y suele corresponder con el número de vector de interrupción que se usará para activar el manejador.

Los periféricos que pueden provocar interrupciones vectorizadas son algo más complejos, pues normalmente el número de vector es programable en un registro y deben realizar la operación de poner en el bus de datos el vector cuando es reconocida una interrupción.

Es evidente que, antes de permitir las interrupciones del periférico, se debe programar el número de vector en el periférico y el manejador correspondiente en este vector.

6.4.2 Controladores de interrupción

El algunas familias de microprocesadores el propio fabricante proporciona una pastilla denominada **controlador de interrupciones** (PIC = *Programmable Interrupt Controler*). Su misión fundamental es aumentar el número de interrupciones de la CPU y proporcionar un método para distinguir al periférico solicitante, normalmente empleando interrupciones vectorizadas.

Poseen una línea independiente por cada interrupción que controlan y utilizan una sola línea de interrupción de la CPU. Están conectados también al bus de datos para poder ser programados y para enviar a la CPU los vectores de interrupción.

Otras funciones que suelen ofrecer es poder enmascarar individualmente cada interrupción y establecer prioridades en las interrupciones. Para ello, la CPU debe activar las interrupciones en el interior de un manejador para que se puedan procesar las de menor nivel.

La arquitectura de los PC utiliza dos controladores de interrupción 8259A accesibles en los puertos 0x20 (el master) y 0xA0 (el slave)

6.5 Transferencia de datos. DMA

El acceso directo a memoria (**DMA** = *Direct Memory Access*) es un método de entrada / salida que se distingue por la transferencia rápida de bloques entre memoria y periféricos en ambos sentidos.

Su funcionamiento está basado en la transferencia de información situada en posiciones de memoria consecutivas. De esta forma, el control de transferencia puede ser realizado por un dispositivo diferente de la CPU sin la necesidad de que los datos tengan que pasar por ella.

El dispositivo especial que se encarga de realizar la transferencia se denomina controlador de DMA. Este dispositivo puede acceder a la memoria a golpes de ciclo de reloj y, por tanto, será más rápido que la CPU que, para realizar la misma operación, debe leer y decodificar las instrucciones de lectura/escritura sobre la memoria y el periférico.

Por otro lado, la CPU puede seguir funcionando a la vez que lo hace el controlador de DMA, con lo que el rendimiento del procesador cuando el volumen de entrada/salida es elevado. Ahora bien, como la memoria es compartida por la CPU y el DMA, se debe evitar el acceso simultáneo al bus de datos y de direcciones que la manejan.

Para la coordinación del proceso de transferencia por DMA, es preciso que se realicen las siguientes funciones:

- La CPU debe indicar (previamente a la transferencia) al controlador de DMA las posiciones inicial y final del bloque a transferir, o bien la posición inicial y el tamaño del bloque.

- Debe establecerse algún tipo de protocolo entre la CPU y el controlador de DMA con el fin de coordinar el momento de comienzo y final de la transferencia. El comienzo podría tener lugar bajo control del programa o como consecuencia de una interrupción procedente del periférico que interviene en la transferencia.

- Debe establecerse también un cierto protocolo entre el controlador de DMA y el periférico con el fin de armonizar las velocidades de ambos de forma que no haya pérdida de datos.

6.5.1 Sincronización en la utilización del BUS

Durante las transferencias por DMA el bus de direcciones, de datos y la línea R/W debe estar aislados del microprocesador. En consecuencia, éste no podrá realizar operaciones que impliquen el uso de los buses. Existen diversos métodos para que la CPU y el controlador de DMA compartan la utilización del bus. Estos métodos se pueden clasificar en los siguientes:

- **Parada del procesador**. Durante el proceso de DMA el procesador queda parado, es decir, sin realizar ningún tipo de operación interna o externa. Este es el método más utilizado pues permite aprovechar al máximo la velocidad propia de las transferencias por DMA. Muchos microprocesadores pueden entrar, a petición de una línea hardware o mediante la ejecución de una instrucción adecuada, en un estado de parada (denominado HALT o STOP, etc.) en el que se congela toda actividad y los buses quedan en estado de alta impedancia.

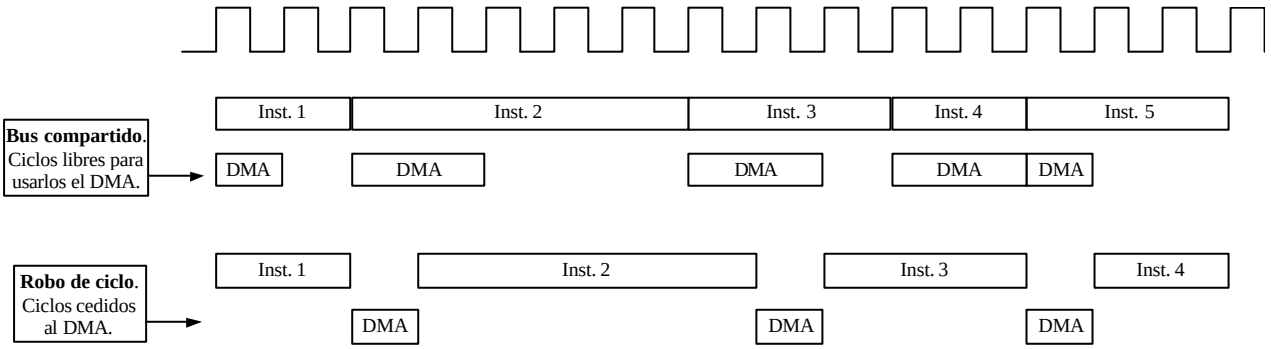
- **Robo de ciclo**. La CPU continua trabajando, pero cada vez que finaliza la ejecución de una instrucción cede los buses al proceso de DMA durante uno o varios ciclos. De este modo las dos actividades se realizan simultáneamente, quizás perdiéndose la virtud fundamental del DMA, es decir, velocidad de transferencia muy alta.

- **Bus compartido**. Durante la ejecución de un programa se alternan ciclos de reloj en los que la CPU hace uso de los buses y ciclos de trabajo interno. Este procedimiento aprovecha estos ciclos para que sea el controlador de DMA el que utilice los buses. La ejecución del programa de la CPU no pierde velocidad, pero simultáneamente tiene lugar una transferencia por DMA. Lo mismo que ocurría en el robo de ciclo, se pierde la alta velocidad de transferencia propia del DMA.

- **Por demanda**. El periférico inicia la transferencia por DMA pero devuelve control a la CPU cuando, de momento, no tiene más datos disponibles. Retoma el control cuando vuelve a tener datos y así sucesivamente hasta que finaliza la transferencia completa del bloque.

- **Dato a dato**. Cada vez que el periférico solicita DMA se transfiere un único dato y se devuelve control a la CPU. El proceso finaliza cuando se ha transferido todo el bloque previsto. Este método es útil para simultanear la ejecución de un programa con la recepción o envío a velocidades relativamente bajas, como en el caso de datos transmitidos vía serie. La CPU no tiene que ocuparse para nada del proceso y sigue ejecutando su programa casi sin pérdida de velocidad, mientras que simultáneamente se realiza la transferencia de datos.

Las utilización de uno u otro procedimiento dependerá de las prestaciones deseadas y de las características propias del microprocesador que se utilice.



DMA en bus compartido y robo de ciclo

6.5.2 Retardos introducidos por el DMA.

Según el método utilizado por el controlador del DMA para compartir el bus, afecta de forma distinta.

En los casos de la parada del procesador y del robo de ciclo el controlador de DMA detiene o relentiza a la CPU. El problema está en determinar el número de ciclos que precisa el DMA para realizar la transferencia. Si el número de ciclos dependiera sólo del tamaño del bloque el retardo que se introduce sería predecible, pero lo normal es que el DMA se tenga que sincronizar con el periférico y, a menudo, los tiempos que este detiene al DMA no son predecibles. Por ejemplo, si se utiliza el DMA para E/S a un

disco, el tiempo utilizado dependerá del tiempo de acceso a la posición del disco, y esto dependerá de la situación del cabezal en el momento de comenzar la transferencia.

En los demás casos en los que el controlador de DMA solicita el bus al procesador puede ocurrir que, una vez tomado el control del BUS, utilice más ciclos de los que no utiliza la CPU, introduciendo el consiguiente retardo.

Para aplicaciones de tiempo real, es importante que el tiempo necesario para realizar la transferencia sea conocido, o al menos el tiempo máximo, y que se conozca el retardo que introduce al procesador.

Pero aunque sea conocido, puede ocurrir que dificulte en gran medida el test de planificabilidad del sistema, por ello, no es frecuente utilizar DMA en los sistemas de tiempo real estricto.

6.6 Servicios de tiempo

En los sistemas de tiempo real la gestión del tiempo es fundamental. A menudo el control del tiempo lo realiza el soporte de ejecución del lenguaje o el sistema operativo.

La gestión del tiempo debe ofrecer los siguientes servicios:

- Actualizar un calendario, dando soporte a la hora del día y la fecha
- Activación de tareas en un cierto instante y/o con un cierto periodo
- Retardar la ejecución de una tarea durante un tiempo
- Gestionar los time-out en las situaciones de espera
- Envío de mensajes o señales a una hora determinada
- Activación de funciones de forma asíncrona
- Obtener medidas de tiempo

Para realizar correctamente estas funciones se precisa un apoyo hardware. Los dispositivos hardware que ofrecen funciones de tiempo se denominan **timers**.

La forma de utilizar un timer para realizar la gestión de tiempos de un sistema de tiempo real es programándolo para que provoque interrupciones periódicas con periodo igual a la mínima unidad de tiempo o resolución temporal que deba ofrecer el sistema.

En cada interrupción el manejador del timer incrementará la hora actual y actualizará el calendario. También debe manejar una lista de las acciones ordenadas por tiempo que se utilizará para realizar los servicios anteriores que se deben hacer cuando el tiempo alcance un determinado valor.

La frecuencia de las interrupciones dará la granularidad en la programación de las actuaciones y retardos. Las necesidades del sistema y el overhead que introduce el manejador de las interrupciones nos darán el valor óptimo para esta frecuencia.

6.6.1 Timers

Los timer son circuitos que pueden ser programados para realizar funciones controladas por tiempo. Las funciones que se suelen realizar con estos circuitos son:

- Generar un pulso de una determinada duración.
- Generar una onda cuadrada
- Provocar una interrupción al cabo de cierto tiempo
- Provocar interrupciones periódicas

Las dos últimas funciones son equivalentes a las dos primeras conectando el terminal de salida del timer a una línea de interrupción.

Los timers funcionan con un contador interno que se decrementa por los pulsos generados por un reloj de frecuencia constante. La variación de la señal de salida o la generación de interrupción se produce cuando el contador sobrepasa el valor 0. De este modo, variando el valor inicial del contador se consigue varia el tiempo que tarda el timer en actuar. Normalmente los timers se pueden programar para que se detengan cuando realicen una acción o que estén funcionando de forma periódica.

Para realizar el control del tiempo en un sistema, el timer se programa para que produzca interrupciones periódicas de forma que funcionará como un reloj, generando un tick de reloj en cada interrupción.

La obtención de la hora en un sistema se puede calcular en base al número de interrupciones producidas desde la puesta en marcha y la hora en que se arrancó el sistema. Se puede obtener mayor precisión el la hora si se tiene en cuenta el valor del contador del timer (fracción de tick) en el instante de la lectura de la hora.

La arquitectura de los PC utiliza el timer 8254-2 con un reloj de 1.193.180 Hz accesible en el puerto 0x40. Este posee tres timers; el primero es el utilizado por el sistema, el segundo se utiliza para generar los sonidos del zumbador y el tercero para las operaciones de refresco de la memoria dinámica DRAM.

El MS-DOS programa el timer con un valor inicial de 0x10000, lo que hará que se produzcan interrupciones con una frecuencia de 18.2 interrupciones por segundo.

6.6.2 Circuitos de Watch-dog

Los circuitos de watch-dog son similares a los timer pero la acción que realizan cuando expira su tiempo es provocar un reset del sistema y reiniciar todo el software.

Se utilizan como elementos de seguridad en sistemas empotrados contra errores software para evitar que, ante un error, el sistema queda inoperativo de forma indefinida.

En el funcionamiento normal el programa, o alguna tarea de la aplicación, debe estar continuamente reprogramando el momento de actuación del watch-dog, con un periodo inferior a su tiempo de actuación. El software se debe diseñar para que cualquier fallo que no sea recuperable acabe siempre, en ultima

instancia, evitando que se continúe reprogramando el watch-dog y, por tanto, el sistema se reiniciaría desde el principio.

Una técnica que permiten muchos sistemas operativos es diseñar un proceso que vigile el funcionamiento del resto de procesos (proceso monitor), utilizando el watch-dog para detectar una fallo en el proceso monitor.

El los sistemas de tiempo real estricto no siempre es posible la utilización del watch-dog de la forma anterior pues, si en mitad del proceso normal se provocara un reset, seguramente esto conllevaría a la violación de las restricciones temporales. Lo normal en estos casos es que el watch-dog provoque un cambio de modo o ponga en funcionamiento un sistema auxiliar.

6.7 Buses

Un bus es un elemento incorporado a un sistema informático que permite conectar diferentes elementos de forma que puedan transferirse información. Esta transferencia de información implica un intercambio de datos y una sincronización. Por ejemplo, a través de un bus se pueden interconectar varios procesadores, dispositivos de almacenamiento, periféricos de entrada / salida, etc.

Muchos sistemas empotrados se implementan en diseños monotarjeta y, por tanto, no necesitan de un bus externo. Ahora bien, si el sistema adquiere cierta complejidad o se desea realizar un diseño modular, se hace necesario la utilización de un bus que conecte los distintos elementos del sistema.

Debido a que un bus es un elemento compartido por distintos elementos, puede ocurrir que distintos procesos requieran simultáneamente su utilización, bien dentro del mismo procesador (por ejemplo, varias tareas desean acceder simultáneamente a algún dispositivo externo conectados al mismo bus, aunque no sea el mismo dispositivo), en distintos procesadores (cuando se va a acceder al bus está ocupado) o con periféricos que realizan transferencia de datos por DMA.

En todos los casos se producirá un retraso en la ejecución de los procesos debido a la sincronización para la utilización del bus, que influirá en los tiempos de ejecución de las tareas.

En los sistemas de tiempo real es imprescindible que los tiempos de ejecución estén acotados, por tanto, los buses que se utilicen deben ofrecer mecanismos que permitan acotar los tiempos de espera y utilización.

Los factores a tener en cuenta en los buses serán, principalmente:

- **Información temporal** a través del bus precisa
- **Respuesta rápida y predecible** para los eventos urgentes
- Un **grado alto en la planificabilidad**. La planificabilidad es el límite en el grado de utilización de recursos por debajo del cual se pueden garantizar las cotas temporales de las tareas de tiempo real. La planificabilidad en el bus se puede equiparar al número de transacciones por segundo aceptables bajo ciertas asumciones.
- **Estabilidad bajo sobrecargas** transitorias. Cuando el sistema se sobrecarga de eventos, y es imposible cumplir todos los límites temporales, el sistema debe cumplir al menos los límites

temporales de las tareas críticas. Naturalmente, el sistema debe ser suficiente para soportar todas estas tareas críticas ejecutándose ellas solas en el sistema.

Como ejemplo de buses utilizados en sistemas de tiempo real veremos el bus VME.

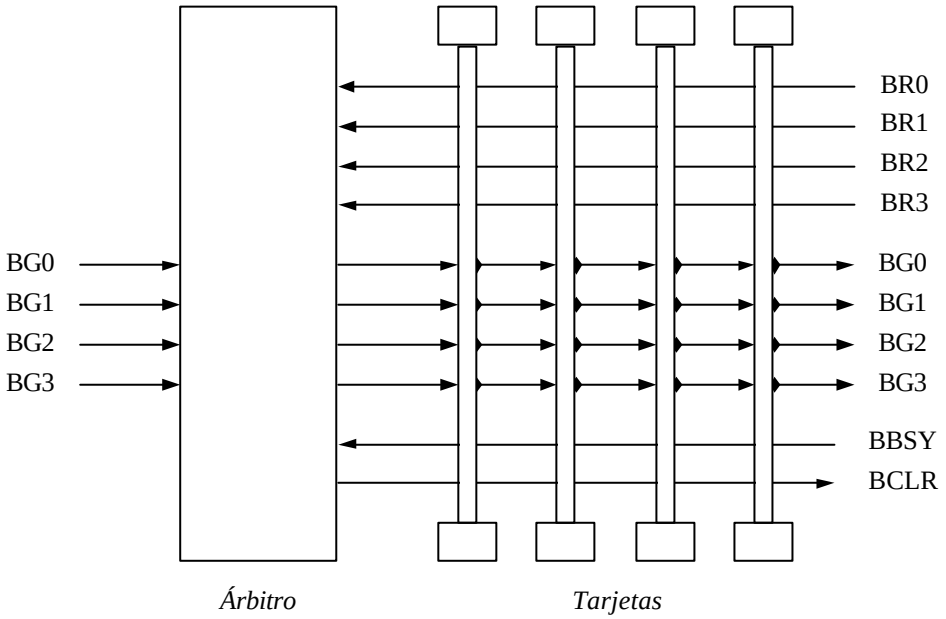
6.7.1 VME

El VME es un bus estándar muy difundido en los sistemas de control para tiempo real. Está diseñado para el microprocesador de Motorola MC68000 de 16 bits y existen en el mercado una amplia gama de componentes que utilizan este bus, con distintas configuraciones en cuanto a elementos procesadores y dispositivos de entrada/salida.

6.7.1.1 Arbitraje

La arbitración del bus es una técnica que consiste en los siguientes pasos: solicitud, concesión y reconocimiento. En el bus VME esto se realiza con dos elementos:

- El árbitro del bus. Recibe las peticiones del bus y gestiona su concesión.
- Varios solicitadores. Realiza las solicitudes para la utilización del bus y el reconocimiento de su conceción.



Para solicitar el bus existen cuatro líneas de petición BR0, BR1, BR2 y BR3 (activas bajo), que corresponden a 4 niveles de prioridad, siendo BR3 la línea de mayor prioridad.

El árbitro del bus lee estas 4 líneas junto con la línea BBSY (activo bajo) que indica si el bus está siendo utilizado por algún módulo del bus.

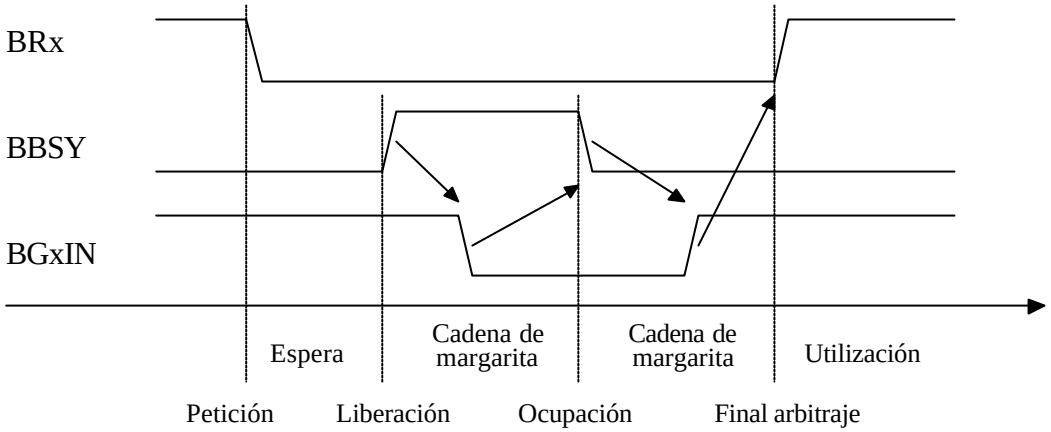
Cuando el bus está libre, el árbitro activa una de las señales BG0IN, BG1IN, BG2IN o BG3IN (activas bajo) para conceder el uso del bus según la petición más prioritaria que se está realizando.

El módulo que realiza la solicitud del bus recibe la concesión del bus y debe procesarla en una cadena de margarita (daisy chain), esto es, si es éste módulo el que está haciendo la petición del nivel

recibido, entonces activará la señal BBSY, si no pasará la concesión del bus al siguiente módulo en la cadena de margarita con la señal BG0OUT, BG1OUT, BG2OUT o BG3OUT correspondiente.

Cuando el árbitro reciba la activación de BBSY desactivará la señal BGxIN que había activado, quedando listo para una nueva petición de bus. Esta desactivación se propagará a través de la cadena de margarita hasta que llegue al módulo que ha solicitando el bus, desactivando la señal BRx que había usado para solicitar el bus. A partir de este momento el módulo será el poseedor del bus.

Cuando el módulo que posee el bus termine de utilizarlo, entonces desactivará la señal BBSY, con lo cual el árbitro podrá procesar otras peticiones.



Arbitraje con una sola línea de prioridad

El árbitro del bus puede gestionar la señal BCLR (activa bajo) cuando detecte una petición de bus de mayor prioridad cuando el bus está siendo utilizado por una petición de menor prioridad. Esta señal debe ser monitorizada por los módulos que pueden utilizar el bus, y liberarlo lo antes posible para que pueda ser procesada la petición más urgente, evitando así una inversión de prioridad motivada por el bus. La monitorización de esta señal puede hacerse por polling o generando una interrupción que bloquee al procesador y libere el bus hasta que se vuelva a ser el dueño.

6.7.1.2 Tiempo real

Las prestaciones de tiempo real se pueden conseguir utilizando adecuadamente la gestión de prioridades en el arbitraje del bus.

Como el número de niveles de prioridad es reducido, se deben de organizar las prioridades software y hardware para que los eventos prioritarios sean atendidos dentro de sus límites temporales.

Lo normal es que un determinado módulo del bus realice sus peticiones de bus con un determinado nivel de prioridad, de forma que todas las tareas de este módulo que utilicen el bus lo harán con este nivel. Por el diseño del bus, es importante el orden que ocupa el periférico en la cadena de margarita pues, ante peticiones simultáneas, se atenderá antes al periférico que esté más cercano al árbitro del bus.

Para la transmisión del tiempo a través del bus se dispone de la línea opcional SYSCLK que distribuye para todos los módulos un señal de reloj de 16 MHz. Para utilizar un tiempo distribuido cada módulo debería implementar un reloj tomando como base de tiempos esta señal y prever un mecanismo de sincronización de los relojes por medio de paso de mensajes entre los módulos.

6.8 Redes. Capas inferiores

Al igual que ocurre con los buses, en las redes que se utilizan para interconectar sistemas de tiempo real deben de cumplir ciertas propiedades para que se puedan seguir garantizando los límites temporales.

Las redes, como todos los sistemas de comunicación, introducen retardos que viene determinados tanta por el hardware como por los protocolos de comunicación.

Los factores que se deben tener en cuenta son:

- **Ancho de banda**. Indica la cantidad máxima de información por unidad de tiempo que se puede enviar a través de la red. Influye en el tiempo de transmisión de un mensaje y en el tiempo de espera hasta que un mensaje comienza a enviarse cuando hay sobrecarga.

- **Niveles de prioridad**. Los niveles de prioridad permiten que los mensajes más urgentes no sean retardados por el envío de mensajes menos urgentes. Los niveles de prioridad deben ser soportados por los niveles más bajos del protocolo de comunicaciones de forma que los niveles de prioridad se tratan globalmente en todos los nodos, es decir, que un mensaje prioritario de un nodo tenga preferencia sobre otro mensaje menos prioritario de otro nodo.

- **Tiempo máximo de latencia**. Indica el tiempo máximo que puede retardarse la salida de un mensaje más prioritario desde que se decide enviarlo hasta que realmente comienza a enviarse. Si el protocolo no permite interrumpir mensajes menos prioritarios, este tiempo está directamente relacionado con el tiempo de transmisión del mensaje más largo de prioridad inferior.

- **Mecanismo de recuperación de errores**. Los medios de comunicación siempre están sujetos a errores y, por tanto, es necesario prever mecanismos de recuperación de detección y recuperación de errores. Los mecanismos usuales implican, a menudo, detección de errores por time-out y el reenvío de los mensajes perdidos hasta un cierto número de veces, repercutiendo notablemente en el tiempo máximo que puede tardar en enviarse un mensaje en el caso de producirse algún error.

No todas las redes son adecuadas para utilizarlas en los sistemas de tiempo real, en el sentido que resulta difícil obtener una cota satisfactoria en el tiempo de envío de los mensajes. Por ejemplo, la red ethernet transmite sobre un bus común en la que todos los nodos pueden tomar la iniciativa de transmisión en cualquier momento. Esto puede provocar colisiones que se resuelven reenviando el paquete en el que se detecta colisión después de un retardo aleatorio. En este caso no existe ningún mecanismo que limite el número de colisiones consecutivas y, por tanto, que ofrezca una cota superior en el envío de los mensajes.

En las redes, el término protocolo se utiliza para hacer referencia a un conjunto de reglas y formatos que permiten establecer una comunicación entre procesos en orden a realizar una tarea determinada. La especificación de un protocolo tiene dos partes importantes:

- La especificación de la secuencia de mensajes y el orden en el que se deben intercambiar.

- La especificación del formato de los datos en los mensajes.

Los protocolos se pueden estructurar en capas, en los que cada capa se basa en la interface que proporciona la capa inferior.

El estándar OSI (Open System Interconnection) propone un modelo basado en siete capas. La funcionalidad de cada uno de los niveles se describe en la siguiente tabla.

Nivel	**Descripción**
Físico	Comprende los circuitos y el hardware que forman la red. Transmite secuencias de datos binarios por medio de señales analógicas, utilizando modulación en frecuencia o en amplitud de señales eléctricas (en circuitos de cable), señales luminosas (sobre fibra óptica) o señales electromagnéticas (en circuitos de radio o microondas).
Enlace	Es responsable de la transmisión libre de errores entre ordenadores que están conectados directamente. En una WAN las conexiones son entre pares PSEs (packets-switches exchanges). En una LAN la conexión es entre pares de hosts.
Red	Transfiere paketes de datos entre ordenadores en una red específica. En una WAN o red internet esto supone la generación de una ruta entre PSEs o ruters. En una LAN sencilla no es necesario el enrutamiento.
Transporte	Este es el nivel más bajo en el que se manejan mensajes (en vez de paquetes). Los mensajes se direccionan a puertos de comunicaciones. Los protocolos en este nivel pueden ser orientados a conexión o sin conexión.
Sesión	En este nivel se establece la comunicación entre procesos y se realiza la recuperación de errores. No es necesario para la comunicación sin conexión.
Presentación	A este nivel el protocolo transmite datos en la representación de red que es independiente de la representación utilizada en los ordenadores individuales, las cuales pueden ser distintas. La encriptación se realiza en este nivel en caso de ser necesaria.
Aplicación	Los protocolos son diseñados para responder a los requerimientos de un aplicación específica, a menudo definiendo el interfaz para un servicio.

Normalmente, el soporte para tiempo real viene dado por los niveles inferiores del protocolo de red, normalmente en los tres niveles inferiores, de forma que sobre estos niveles se pueden utilizar protocolos generales que estén soportados en niveles superiores, si los niveles inferiores ofrecen un interface común. De esta forma se pueden utilizar protocolos estándar junto con protocolos que hagan uso de las prestaciones de tiempo real.

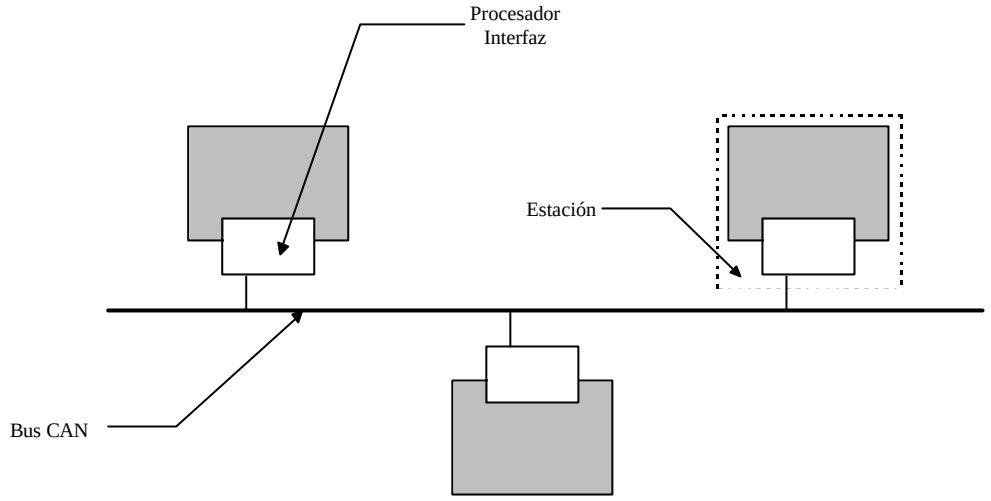
Como ejemplo de protocolos de tiempo real vamos a estudiar el bus CAN uno utilizado en sistemas industriales.

6.8.1 CAN

La red CAN (Controller Area Network) es un bus de comunicaciones diseñado para enviar y recibir mensajes de control cortos en tiempo real. El bus está diseñado para sistemas de control en un área pequeña (como por ejemplo vehículos), operando en un entorno ruidoso a velocidades de hasta 1Mbit/seg.

CAN es un bus de uno a todos (broadcast) en el que un número determinado de procesadores están conectados a un bus a través de un interface.

Una fuente de datos es transmitida como un mensaje, formado de entre 1 a 8 bytes. Una fuente de datos puede ser transmitida periódicamente, esporádicamente o bajo demanda. Por ejemplo, una fuente de datos podría ser la velocidad de circulación, codificada como un mensaje de un byte, y transmitida cada 100 mseg.



Arquitectura CAN

Comparada con el modelo de referencia ISO, la estructura de niveles de CAN es de tres niveles:

Nivel	**Descripción**
Físico	En la mayoría de los casos, el nivel físico o medio de transmisión es un bus diferencia a 2 hilos, utilizado como par trenzado.
Enlace	Viene fijado por la especificación del protocolo CAN y está implementado en gran cantidad de circuitos integrados de distintos fabricantes.
Red Transporte Sesión Presentación	Niveles vacíos
Aplicación	Se implementa según las necesidades del usuario.

Según las prestaciones del controlador podemos distinguir dos tipos:

- FullCAN. En controlador CAN almacena todos los mensajes, accediendo a ellos a través de una memoria RAM de doble puerto compartida con la CPU. La CPU no precisa de realizar un trabajo excesivo para recibir los mensajes, pues prácticamente todo lo realiza el controlador, la CPU le indica los mensajes que desea leer y este avisa a la CPU cuando recibe alguno de los mensajes deseados.

- BasicCAN. Los mensajes se almacena en la memoria de la CPU, y la CPU debe realizar todo el trabajo para obtener los mensajes deseados. La CPU debe sincronizarce con el controlador por medio de manejadores de interrupción para ‘enviar un mensaje’ y detectar un ‘mensaje recibido’.

En la versión CAN 1.0, a la fuente de datos se le asigna un identificador, representado como un número de 11 bits (esto proporciona un total de 2032 identificadores, pues CAN prohibe los identificadores con los 7 bits más significativos a ‘1’). El identificador cumple dos propósitos: filtrar los mensajes en la recepción y asignar una prioridad a los mensajes.

La versión 2.0 de CAN prevé 29 bits para los identificadores, ofreciendo un número de niveles más que suficientes.

Una estación en un bus CAN es capaz de recibir un mensajes basado en su identificador: por ejemplo, si un procesador necesita obtener la velocidad de circulación, entonces se lo indicaría la interfaz CAN por medio de su identificador. Sólo se recibirán los mensajes con los identificadores deseados y serán entregados al procesador principal.

La utilización como la prioridad del mensaje es la parte más importante del bus CAN en relación a las prestaciones de tiempo real.

En la red ethernet cada estación espera a que no haya tráfico en el bus para comenzar la transmisión. Si más de una estación intentan transmitir a la vez, entonces todas ellas lo detectan, esperan un tiempo aleatorio e intentan retransmitir cuando el bus esté libre. La ethernet es un ejemplo de bus con transmisión a todos con detección de portadora, pues cada estación espera a que el bus esté libre (sin presencia de portadora) y monitoriza su propio tráfico para detectar colisiones. CAN es también un bus de transmisión a todos con detección de portadora, pero utiliza un técnica más eficaz para realizar la contención en el caso de detectar una colisión. El campo identificador de un mensaje CAN se utiliza para el control de acceso al bus cuando hay una colisión aprovechando ciertas características eléctricas.

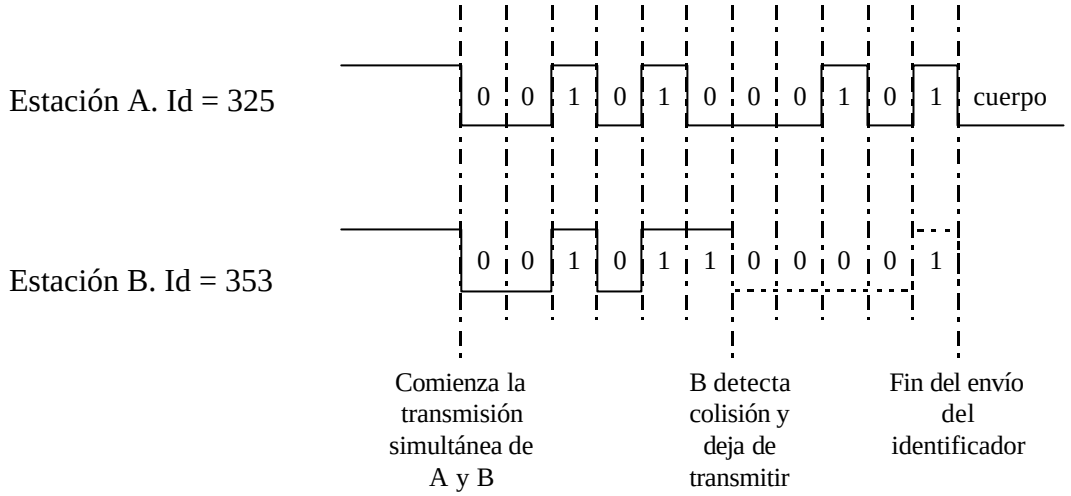
En el bus CAN, si más de una estación transmiten simultáneamente y una de ellas transmite un bit '0', entonces todas las estaciones que monitorizan el bus ven un '0'. O lo que es lo mismo, sólo cuando todas las estaciones transmiten un '1' entonces en el bus se ve un '1'.

En la terminología CAN, al bit '0' se le denomina *dominante* y al bit 1 *recesivo*. En efecto, el bus CAN se comporta como una gran puerta AND, en la que cada estación dispone de una entrada de la puerta y puede ver su resultado.

Este comportamiento se utiliza para detectar y resolver colisiones: cada estación espera hasta que el bus esté libre (al igual que en la ethernet). Cuando se detecta silencio en el bus cada estación comienza a transmitir el mensaje más prioritario de los que tenga en la cola de transmisión. El mensaje se codifica de manera que el bit más significativo del identificador se envía primero.

Si una estación transmite un bus recesivo pero recibe un bit dominante, esto significa que se ha producido una colisión. La estación sabe entonces que el mensaje que está transmitiendo no es el más prioritario en el sistema y detiene la transmisión, esperando a que el bus vuelva a quedar libre.

Si una estación transmite un bit recesivo y recibe un bit recesivo, entonces significa que está transmitiendo el mensaje más prioritario, y prosigue con la transmisión en el bit siguiente del identificador. Como el bus CAN requiere que los identificadores sean únicos en el sistema, cuando una estación transmita el último bit del identificador (menos significativo) sin detectar colisión, significará que ella está transmitiendo el mensaje de mayor prioridad y, por tanto, podrá continuar enviando el cuerpo del mensaje.



Arbitraje en un identificador de 11 bits

Conviene hacer algunas observaciones generales en este protocolo de arbitración. Primero, un mensaje con identificador de valor inferior corresponde a un mensaje de mayor prioridad. Segundo, el mensaje de mayor prioridad no recibe ninguna perturbación en el proceso de arbitraje y, por tanto, el cuerpo de su mensaje será transmitido sin interrupción.

Con estas observaciones, el tiempo más largo desde que el mensaje de mayor prioridad es encolado hasta que es recibido, se puede calcular fácilmente: es el tiempo más largo en que una estación puede estar esperando hasta que el bus esté en silencio, que corresponde a la máxima duración en la transmisión de un mensaje. Si el mensaje más largo son 8 bytes, más 2 del identificador, a una velocidad de 1 Mbits/seg. y suponiendo un tiempo de relleno de 5 bits por byte, esto corresponde a un tiempo de 130 microsegundos (10 bytes por 13 bits).

Para los mensajes de menor prioridad, el tiempo máximo de respuesta no es tan sencillo de calcular, pero también se puede acotar.

Las propiedades del bus CAN lo hacen adecuado para su utilización en sistemas de tiempo real.