Aplicación de los Sistemas Operativos en Tiempo Real y los Protocolos de Comunicación al Diseño de un Sistema de Adquisición de Datos Modular de Bajo Coste y Bajo Consumo.

José Antonio Pérez Castellanos[1] y Antonio Barbancho Concejero[2]

1: Departamento de Lenguajes y Sistemas Informáticos. Facultad de Informática y Estadística. Universidad de Sevilla. Avda Reina Mercedes s/n. 41011 Sevilla. Email: jperez@lsi.us.es

2: Departamento de Tecnología Electrónica. Escuela Universitaria Politécnica de Sevilla. Universidad de Sevilla. C/ Virgen de África, 7. 41012 Sevilla. Email: ayboc@cica.es

***Resumen**: En este artículo presentamos una aplicación de los sistemas operativos en tiempo real y los protocolos de comunicación al diseño de un sistema de adquisición de datos modular de bajo coste y bajo consumo. Se describe la arquitectura básica del sistema, consistente en una red tipo bus, basada en los estándares RS-485 y SPI y una serie de dispositivos dedicados que se conectan a ella. Se muestran algunos aspectos destacables del sistema operativo y de los protocolos. Entre los objetivos marcados al iniciar el proyecto, destacamos como más importantes el bajo consumo de potencia (funcionamiento autónomo), bajo coste, manejo de gran número y tipo de sensores, flexibilidad, escalabilidad, gran capacidad de almacenamiento y funcionamiento en tiempo real. Los prototipos montados hasta ahora y las medidas realizadas sobre ellos demuestran que se han cumplido dichos objetivos, y que el camino marcado permite ir aún más allá en nuestras pretensiones.*

1. Introducción

Los sistemas de adquisición de datos tradicionales (SAD) de bajo consumo se basan en un sistema informático (típicamente un microprocesador y alguna memoria y circuitería de control) que gobierna una cadena de adquisición analógica. Si se requiere la adquisición de señales digitales periódicas o de tipo pulso, es necesario añadir circuitería (contadores), con el problema añadido de que este tipo de señales no es posible multiplexarlas, por lo que el coste crece proporcionalmente con cada canal añadido. Por tanto, no es extraño ver sistemas con 16 entradas analógicas, ampliables hasta 128 o más y sólo 4 canales de entrada de pulsos, no ampliables. Por otro lado, si es necesario adquirir señales de distinto tipo (tensión, intensidad, resistencias, etc.) la circuitería se hace más compleja.

Esto lleva a que algunos sistemas pueden ser demasiados caros para ciertas aplicaciones, ya que hay que dimensionar algunos componentes respecto de la aplicación más exigente. O todo lo contrario, caso que se da cuando la aplicación evoluciona con el paso del tiempo, pero el SAD no es capaz. La mayor parte de la veces, el fabricante diseña una línea completa de sistemas desde el más simple y de bajo coste, hasta el más potente, y por supuesto, más caro. Sin embargo, el usuario debe elegir en el momento de la compra qué modelo elige.

Consideraciones parecidas pueden hacerse en referencia con el consumo de potencia y coste del equipo. Un sistema sobredimensionado puede satisfacer ciertas necesidades de ampliación futura, pero está aumentando el gasto inicial y el consumo del sistema. Otro problema fundamental es la distribución de los sensores. Sobre todo en ambientes industriales, es habitual encontrar procesos en los que las señales a adquirir están diseminadas por un espacio relativamente amplio. Las soluciones que se plantean son:

- adquirir las señales de forma local y transmitirlas hacia un centro de control. El problema que suele presentar este tipo de solución es que puede aumentar bastante el coste del

sistema;
- transformar la señal para que pueda ser transmitida a grandes distancias, normalmente en forma de bucles de corriente de 4 a 20 mA [1], con el problema añadido del consumo.

Con estos problemas en mente, numerosos sistemas los han enfrentado con éxito. Los fabricantes han diseñado equipos ampliables de forma fácil, pero con frecuencia a costa de perder otras características, como la de bajo consumo. Así, no es difícil ver sistemas distribuidos, principalmente usando buses industriales, que permiten solucionar los problemas antes citados. Sin embargo, dichos buses se pensaron para entornos en los que el consumo no es un problema, o al menos no es principal. Entre los principales problemas detectados en algunos de estos sistemas, podemos citar:

- necesidad de que la señal vaya modulada, lo que aumenta la complejidad del sistema y aumenta el consumo;
- multiplexión de datos y alimentación sobre el mismo canal. Estos sistemas no permiten grandes velocidades de transmisión, ni elementos que necesiten picos importantes de consumo si no es con cableado adicional;
- empleo de protocolos simples de comunicación (tales como comunicación asíncrona RS-232 o RS-485) que desperdician ancho de banda del canal y pueden incrementar el consumo.

Podríamos resumir todos los requerimientos anteriores, enunciando lo que, a nuestro entender, deben ser las capacidades de un moderno sistema de adquisición de datos de propósito general:

- manejar un elevado número de sensores de diversa naturaleza;
- permitir la conexión de sensores situados a grandes distancias;
- permitir el funcionamiento de manera autónoma, alimentado mediante baterías;
- adecuarse a las necesidades de la aplicación;
- facilitar las tareas de instalación y mantenimiento;
- almacenar una gran cantidad de datos;
- permitir ampliaciones del sistema de forma fácil;
- facilitar la interacción del usuario, presentando una interface simple e intuitiva;
- realizar operaciones en tiempo real;

y, por supuesto, todo ello al menor coste.

En este artículo se describe la aplicación de los sistemas operativos en tiempo real y los protocolos de comunicación al diseño de MIDAS, un sistema de adquisición de datos de propósito general, de bajo coste y bajo consumo, pero que al mismo tiempo pretende responder a las necesidades de aplicaciones de muy diversa índole (industria, meteorología, control, monitorización, etc.). MIDAS (*MultIpurpose Data Acquisition System*, Sistema de Adquisición de Datos de Propósito Múltiple) cumple todos los requisitos antes enu nciados. MIDAS es un sistema de bajo coste y bajo consumo y se basa en una red de comunicación y un sistema operativo en tiempo real. Sus principales características son:

- La red soporta hasta 256 elementos (sensores, actuadores o cualquier otro dispositivo).
- Los elementos se conectan a un bus compuesto por 8 líneas. Por tanto, sólo se necesitan 8 líneas para alimentar y controlar todos los dispositivos, facilitando la instalación y mantenimiento del sistema.
- Bajo consumo de corriente que permite alimentar todo el sistema con una batería y un panel solar.
- Todos los sensores disponen de una interface inteligente, posibilitando un tratamiento uniforme y simplificando el manejo de sensores de naturaleza diferente.
- Todos los dispositivos son *plug and play*, es decir, su configuración es llevada a cabo por el sistema de forma automática. El sistema permite, además, la conexión en caliente.

- Sistema operativo en tiempo real.
- Programación flexible.

Aunque MIDAS no es un sistema comercial, ha sido diseñado para servir de base a una serie de sistemas comerciales que se están desarrollando en la actualidad. En este sentido, no sólo se han tenido en cuenta criterios puramente técnicos a la hora del diseño, como empleo de ciertas metodologías, sino que se ha atendido a otras necesidades relacionadas con los productos comerciales, como el costo, facilidad de diseño, depuración, fabricación y teste, disponibilidad de los recursos *hardware* y *software* y comodidad de empleo para el usuario final.

En la sección 2 se describe la arquitectura básica de MIDAS. La sección 3 se centra en los detalles e implementación del sistema operativo en tiempo real. Los datos relacionados con los protocolos de comunicaciones se muestran en la sección 5. Los resultados obtenidos tras los primeros prototipos se discuten en la sección 6. Finalmente la sección 7 presenta las conclusiones y las líneas de investigación seguidas actualmente.

2. Arquitectura del Sistema

En la figura 1 se muestra la arquitectura de MIDAS. El sistema se basa en una red tipo bus en tiempo real. Los dispositivos que se conectan a dicha red se clasifican en maestros (*masters*) y esclavos (*slaves*). En la actualidad sólo puede existir un *master* o dispositivo principal en la red, aunque futuras versiones obviarán esta limitación. El dispositivo principal se encarga de asignar direcciones de red y configurar cada esclavo. Al mismo tiempo, gobierna la operación del sistema y actúa como recolector de datos (*datalogger*).

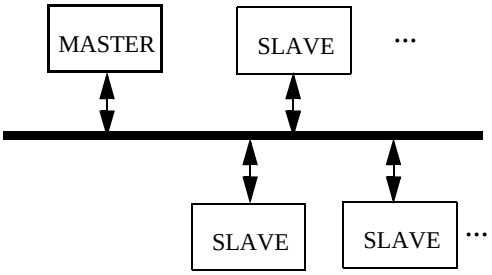

Fig. 1: Arquitectura del sistema

El resto de los dispositivos en la red (sensores inteligentes, sistemas de almacenamiento de datos, acondicionadores de señal, interfases de usuario, etc.) son esclavos. Dichos dispositivos presentan una interface estándar a la red. En caso de necesidad se puede conectar cualquier sensor tradicional a la red a través de un módulo adaptador, permitiendo una transición no traumática desde sistemas convencionales. Puesto que el *master* no presenta ningún tipo de interface analógica con el mundo exterior, no está fijando ninguna limitación en cuanto a las características de ésta. Al mismo tiempo, al recaer las tareas de acondicionamiento de señal en dispositivos remotos especializados, la precisión, resolución y otros aspectos son los adecuados para cada aplicación, y no hay pérdida de información en la transmisión de la señal, puesto que ésta es digital.

El principal problema que puede presentar este tipo de soluciones es un mayor coste, puesto que al coste del sensor y acondicionador, hay que añadir la digitalización local y su transmisión al dispositivo principal. Sin embargo, tal como se discute en [1], en muchos casos el coste de los sensores inteligentes puede ser aún menor, si se emplean ciertas técnicas:

- La linealización, filtrado y otras tareas se puede hacer de manera digital, simplificando el acondicionador.

- Muchas sistemas de medida son intrínsecamente digitales, por lo que no necesitan estos tratamientos.
- Existen técnicas para digitalizar las medidas de sensores sin necesidad de un CAD [1].

2.1. Arquitectura de la red

El modelo de comunicación de MIDAS se basa en una simplificación del modelo OSI [2], de forma que sólo implementa los niveles físico, enlace de datos y aplicación.

El nivel físico se encarga de la trasmisión de bits sobre el canal de comunicaciones [3]. Con objeto de realizar un sistema de bajo costo, nuestro diseño se basa en los estándares RS-485 y SPI (*Serial Peripheral Interface*). El canal de comunicaciones es un bus que transporta 2 líneas de alimentación (V+ y masa) y tres pares de líneas de datos (CLK para el reloj, DATA para datos, y SS para selección de esclavo). Puesto que se usa el protocolo RS-485, la comunicación es balanceada y necesita dos líneas para cada señal de datos. Esto permite comunicar a distancias de hasta 1200 metros sin repetidores. La comunicación es *halfduplex* síncrona y controlada por el *master*. La velocidad de trasmisión es controlada por CLK, y puede ser de hasta 10Mbps. A pesar de que el protocolo RS-485 especifica que se pueden conectar hasta 32 cargas[1] en un bus, nuestro diseño emplea *transceivers* con la octava parte de la carga, permitiendo hasta 256 dispositivos. Se eligió el MAX3082 debido a ello, junto con su bajo consumo, bajo costo y su característica "a prueba de fallos" (*fail-safe*), que permite apagar todos los emisores RS-485 para reducir el consumo cuando el canal está desocupado. Cuando un receptor MAX3082 "ve" una línea flotante, coloca a su salida en un estado conocido (alto).

Se han previsto dos tipos de cables en el bus: cable plano de 10 vías tipo IDC (*Insulation Displacement Conection*, conexión por desplazamiento del aislante) para distancias cortas y manguera de 4 pares trenzados tipo 24 AWG para distancias grandes y para especial protección a las interferencias electromagnéticas y las inclemencias del tiempo.

La tarea principal del nivel de enlace de datos es aprovechar el canal de comunicaciones creado por el nivel físico y transformarlo de forma que el nivel superior vea una línea libre errores [3]. Se encarga, así mismo, de la identificación de los esclavos, para lo cual asigna un número único (dirección de red) a cada uno de ellos durante la fase de configuración.

El nivel de aplicación es una colección de protocolos que proporcionan soluciones específicas. Ejemplos son la transferencia de ficheros, operación remota y gestión de esclavos, que se encarga de la identificación y configuración automática (*Plug'n'Play*) y la asignación de direcciones de red.

Los protocolos de comunicación se describen con más detalle en la sección 4.

2.2. Arquitectura del Maestro

El *master* se encarga de realizar las siguientes tareas:

- Gestionar el protocolo del nivel de aplicación, tal como se describe en la secciones 2.1 y 4.
- Realizar las tareas de adquisición. Esta labor se implementa ejecutando un programa de usuario, tal como se describe más adelante.

Desde el punto de vista del *hardware*, el dispositivo principal consiste en un microcontrolador, una interface de red, un enlace serie RS-232, un reloj de tiempo real, un generador de NMI (*Non Maskable Interrupt*, interrupción no enmascarable), una ROM (*Read Only Memory*, memoria de sólo lectura) que contiene el código empotrado, y un banco de memoria SRAM (*Static Random Access Memory*, memoria estática de acceso directo) no

1. Una carga (*load*) es la impedancia de entrada de un *transceiver* RS-485 y se establece en 12KΩ en el estándar.

volátil, tal como se describe en la Fig. 2. Todo el sistema puede alojarse en una placa de circuito impreso poco mayor a una tarjeta de crédito.

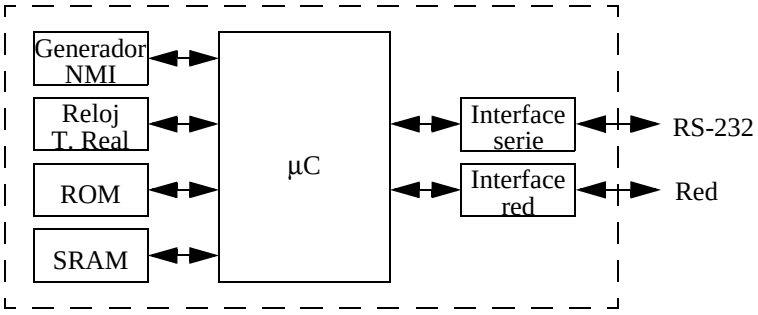

Fig. 2: Arquitectura del master

El banco de memoria RAM permite alojar hasta 512 Kbytes de memoria local rápida y se usa tanto para uso interno del dispositivo, como para el almacenamiento de datos. La capacidad de almacenamiento puede ser incrementada vía la conexión a la red de un dispositivo externo, permitiendo almacenamiento masivo. Dicha memoria y el reloj en tiempo real pueden ser alimentados por una batería de litio interna, o cualquier otra fuente externa ante la falta de alimentación principal, permitiendo la retención de programas y datos y el mantenimiento de la fecha y hora del sistema.

La interface serie RS-232 se ha incluido para facilitar la carga y descarga de programas, la depuración y tareas frecuentes como la extracción de datos o la monitorización del sistema.

El master se basa en un microcontrolador de la familia H8/500 de Hitachi [6]. La elección de este dispositivo se basó en su naturaleza de 16 bits, rico juego de periféricos dentro del propio chip y su bajo costo. Los primeros prototipos usan un H8/520, con encapsulado PLCC de 64 pines, 32 Kbytes de EPROM OTP[1] y 1 Kbyte de RAM. Un par de periféricos de tipo USART en el propio integrado permiten su conexión a un puerto RS-232 y a la red (SPI).

Desde el punto de vista del *software*, el *master* contiene cuatro módulos independientes:

- *Kernel* en tiempo real, que implementa la multitarea y la política de planificación en tiempo real. Este *kernel* sirve de soporte básico a otros servicios software, tal como se describe en la sección 3.
- Protocolos para los niveles físico y de enlace de datos, y protocolo de nivel de aplicación para el master. Dichos protocolos se describen en la sección 4.
- Interface de usuario y enlace serie RS-232. Se usa un simple protocolo Xon-Xoff para manejar las comunicaciones.
- Intérprete del programa de usuario. El master implementa una máquina virtual de adquisición de datos que ejecuta un programa suministrado por el usuario. En este artículo no se va a describir este intérprete.

2.3. Arquitectura del Esclavo

Puesto que el *master* no dispone de capacidad de entrada/salida (tanto digital como analógica), excepto lo que se refiere a los enlaces de red y serie RS-232, dicha capacidad de interacción con el mundo exterior debe recaer en los esclavos. Por tanto, la arquitectura del dispositivo esclavo es muy dependiente de la aplicación. Sin embargo, siempre debe incluir una conexión a la red. Dicha interface ha sido diseñada para ser muy simple, permitiendo un bajo costo y bajo consumo (ver “2.1. Arquitectura de la red”).

La interface de red y otras tareas comunes descansan sobre un microcontrolador de la familia Z8 de Zilog [5]. Se escogió un Z8E06 por su muy bajo coste, muy bajo consumo, pequeño encapsulado y capacidad de actuar como esclavo SPI. Una característica única de este

1. *One-Time Programmable*, programable una sola vez.

dispositivo es la capacidad de actuar bajo el modo de comparación. En este modo, el microcontrolador está en modo de ultrabajo consumo (1μA típico). El puerto SPI es controlado externamente por el *master*. Una transacción comienza con la activación de la línea SS (*Slave Select*, selección de esclavo). Entonces, el reloj CLK gobierna la recepción de 8 bits de datos por la línea DATA. Este byte es comparado con otro almacenado en un registro especial del microcontrolador. Si los datos no coinciden, el Z8 ignora los datos hasta que se desactiva la señal SS. Si los datos son iguales, el microcontrolador sale del modo de bajo consumo y continúa con la transacción. Este esquema es el que inspira el modo de funcionamiento de los protocolos de acceso al medio de MIDAS.

Se ha añadido además a los esclavos capacidad de interrupción. Dicho mecanismo se emplea en la actualidad únicamente cuando se conecta un esclavo nuevo al sistema, y cuando un esclavo dispone de información que solicitó el master y que no era posible suministrar en ese momento (p.e., medidas lentas).

3. Diseño del Sistema Operativo en Tiempo Real

Puesto que la complejidad principal de MIDAS recae en el *master*, es en el software de éste donde se presenta el mayor reto del diseño. Se ha realizado un importante esfuerzo en la fase de ingeniería con objeto de producir un código empotrado flexible, reusable, fácil de mantener y extender. Tal como se adelanta en la sección 2.2, la arquitectura del *software* asociado al *master* consiste en cuatro módulos levemente acoplados:

- *Kernel* en tiempo real, que implementa la multitarea y la política de planificación en tiempo real.
- Protocolos de comunicación.
- Interface de usuario para el enlace RS-232.
- Máquina intérprete del programa de usuario.

Con objeto de reducir el acoplamiento entre módulos al máximo, cada uno de ellos presenta una interface estándar perfectamente definida, consistente en un conjunto de tipos abstractos de datos (ADT, *Abstract Data Types*) y un conjunto de operaciones que se pueden realizar sobre ellos. Cada módulo se modela como uno o más ADTs y la funcionalidad se modela como operaciones que se pueden realizar sobre esos ADTs. Esta técnica permite modificar completamente un módulo sin efectos colaterales sobre los demás.

El mayor reto desde el punto de vista del *software* fue el *kernel* en tiempo real. Para comprender esto, recordemos que uno de nuestros principales objetivos era reducir el consumo del sistema tanto como fuera posible. El *kernel* debe ser capaz de programar todas las tareas pendientes, de forma que si se detecta un prolongado período de tiempo de inactividad hasta la activación de la siguiente tarea, debe poner la CPU en modo *standby*. Ahora bien, cada tarea, en cualquier caso, debe ser capaz de cumplir sus restricciones temporales. Sería inaceptable por ejemplo, que el sistema perdiese parte de la información que le llega a través de la red porque los datos que llegan por la misma “sorprenden” en *standby* a la CPU. La posibilidad de activar el modo de bajo consumo no depende sólo de que todas las tareas se encuentren en estado “ocioso” en un momento dado, sino que depende de:

- nivel de prioridad de la tarea activa más prioritaria;
- duración del período de *standby*.

Esto resulta ser problemático, teniendo en cuenta que el proceso de salida de la CPU de este modo puede llevar hasta 10 milisegundos (llamado tiempo de recuperación de *standby*, el cual depende de la frecuencia de reloj de la CPU), y teniendo en cuenta también que mientras la CPU está en estado de *standby* no puede responder a más eventos que la activación de la línea NMI.

Para dar solución a estos problemas, se desarrolló un sistema operativo en tiempo real para ser usado en aplicaciones empotradas de la familia H8/500 de Hitachi. Dicho sistema operativo ha sido bautizado con el nombre de LOPORTOS, acrónimo de su nombre en inglés, *LOw POwer Real Time Operating System*, sistema operativo en tiempo real de bajo consumo.

Dicho sistema operativo se basa en la arquitectura microkernel [4] y se estructura en tres módulos:

- gestión de tareas;
- gestión de memoria; y
- gestión de comunicación entre procesos.

La gestión de memoria es muy simple debido a que la arquitectura de la familia H8/500 no implementa protección de memoria, ni dispone de modo usuario o supervisor. Por consiguiente, existe un único espacio de memoria que gestionar, el cual se manipula mediante una lista enlazada de bloques libres [7]. Puesto que no se dispone de soporte hardware para acelerar la reubicación de memoria, no se pudo implementar un recolector automático de basura.

La gestión de la comunicación interproceso es también muy simple. Una única cola de mensajes se asigna a cada proceso, idenficándola a través del identificador del proceso (pid). Por otra parte, los mensajes pueden ser enviados a una cola de forma síncrona (el emisor se bloquea hasta que el mensaje es leído) o asíncrona.

La característica de tiempo real recae en el gestor de tareas. Este modulo asigna prioridades de acuerdo con los principios del análisis *Rate Monotonic*, asignando prioridades mayores a las tareas que sirven eventos de mayor frecuencia. Las interrupciones se tratan como tareas: cada tarea en el sistema debe declarar qué interrupciones maneja. Cuando se produce una interrupción, el *kernel* la captura, identificando su tarea asociada, le envía un mensaje identificando la interrupción y la coloca en estado lista para ejecución.

Como se apuntó anteriormente, el bajo consumo también es gestionado por el gestor de tareas. La CPU puede programar un generador de NMI que proporciona una secuencia fija, aunque programable, de interrupciones con objeto de “despertar” la CPU del estado *standby*. Cada proceso dispone de una cola de mensajes a través de la cual interaccionan con otros procesos y con el sistema, y expresan su estado desocupado intentando leer de dicha cola.

En LOPORTOS, cada proceso tiene asociada una prioridad dinámica (puede cambiar durante su vida), calculada a partir del máximo tiempo permisible de respuesta a un evento. En las tareas que sirven eventos periódicos, este tiempo viene determinado por el periodo mínimo entre eventos que requieran servicio. Cuando un proceso trata de leer un mensaje de su cola, y ésta se encuentra vacía, el proceso se marca como “desocupado”. Cuando el *kernel* detecta que todos los procesos se encuentran desocupados, comprueba si es “razonable” programar el generador de NMI para despertar la CPU antes del vencimiento de la espera del siguiente proceso, teniendo en cuenta el tiempo de recuperación de *standby* por parte de la CPU. Si se comprueba que es posible, es decir, el tiempo de expiración de la siguiente tarea menos el tiempo de recuperación es mayor que un tiempo mínimo fijado, se programa el generador de NMI con el valor adecuado (tiempo de expiración - tiempo de recuperación) y se coloca la CPU en estado de *standby*. Esto mantiene la CPU tanto tiempo como sea posible en estado de ultra bajo consumo, respetando la ejecución en tiempo real de todos los procesos.

Para que este mecanismo sea verdaderamente eficiente en cuanto a consumo, es necesario que los procesos puedan reasignar su prioridad durante su propio tiempo de ejecución. Esto se debe a que un mismo proceso que puede estar involucrado en tareas críticas en un momento de su vida, puede pasarse el resto del tiempo de ejecución en espera de eventos no críticos. Valga de ejemplo el proceso que controla la interface serie de usuario. Cuando este proceso comienza a ejecutarse, ha de esperar a detectar que el usuario conecta el cable en el conector para establecer la conexión. El tiempo de respuesta a este evento no es crítico, pudiéndose tardar perfectamente hasta un segundo en establecer la conexión tras conectarse el cable. Ahora bien,

una vez creada la conexión, el tiempo máximo de respuesta viene determinado por el tiempo que tardan en llegar los caracteres, el cual es lógicamente de varios órdenes superior. Por consiguiente, el proceso de control de la interface serie aumenta su prioridad cuando se crea una conexión (haciendo que el sistema aumente su consumo, pero garantizando su correcto funcionamiento) y baja su prioridad cuando el usuario retira el cable, cerrando la conexión (disminuyendo nuevamente de forma drástica el consumo del sistema).

Dentro de este esquema, un cuidadoso cálculo de las prioridades en cada instante garantiza una relación eficiente entre el consumo del sistema, y el tiempo máximo de respuesta del mismo ante los distintos eventos a los que se ha de responder.

4. Diseño de los Protocolos de Comunicación.

Debido a la falta de espacio, en esta sección se describe de forma somera los protocolos de comunicación del sistema. La presentación se centrará por su mayor interés en los protocolos de nivel físico y subnivel de acceso al medio, ya que soportan el resto de los niveles y afectan directamente a los parámetros que se desean optimizar (costo, consumo, velocidad, etc.).

Como se apuntó anteriormente, la red de MIDAS se ha diseñado con cuatro objetivos primordiales:

- bajo coste. El hardware asociado debe ser simple y de fácil acceso;
- bajo consumo. Puesto que se requiere que el sistema opere alimentado por baterías, el consumo en los momentos de inactividad debe ser mínimo;
- velocidades de transmisión medias a altas. La red debe soportar tanto dispositivos lentos, como los sensores inteligentes o interfaces de usuario, como rápidos, como los sistemas de almacenamiento masivo, pasando por otros de velocidad media, como las interfaces de comunicaciones con el exterior (módem).
- cableado simple y barato. La conexión entre elementos del sistema debe ser mediante el menor número de hilos posible, huyendo cuando se pueda de configuraciones caras.

En la sección 2.1, puede encontrarse la arquitectura de la red. En la siguientes secciones nos centraremos en la descripción de los protocolos de comunicación.

4.1. Nivel físico

Para permitir un cableado lo más simple posible, se eligió una comunicación serie. Existe un gran número de enlaces serie en el mercado, muchos de ellos estandarizados por diversos organismos. De hecho, este tipo de comunicación está en gran auge en la actualidad, debido a el enorme número de ventajas que presenta, incluido una mayor velocidad de transmisión a distancias superiores a las de una placa de circuito impreso. Algunos ejemplos son: RS-232, RS-485, Ethernet e I^2C. La elección final recayó en la conexión RS-485 por diversos motivos:

- Es un estándar que permite la conexión de elementos en topología de bus (a diferencia de RS-232 que es una conexión punto a punto).
- Si bien el estándar establece que se deben de poder conectar un mínimo de 32 *transceivers* en el bus, existen en el mercado numerosos dispositivos con impedancia de entrada superior, que permiten la conexión de hasta 256 elementos en el bus.
- La transmisión se realiza con niveles de tensión compatibles con 5V, de forma que no es necesario complicar la fuente de alimentación para obtener otras tensiones.
- La comunicación es balanceada, por lo que su inmunidad al ruido es muy buena.
- La velocidad de transmisión puede llegar hasta los 10 Mbps, igualando a las actuales Ethernet.
- La línea puede ser de hasta 1200 metros sin repetidores, pero con velocidades de transmisión menores.

En cuanto al protocolo de nivel físico, se optó por una comunicación síncrona de tipo SPI (*Serial Peripheral Interface*, interface de periféricos serie), que supone una gran cantidad de ventajas:

- No utiliza modulación, por lo que se simplifica la circuitería.
- El hardware necesario está disponible integrado en un sinnúmero de microcontroladores.
- La comunicación es síncrona y manejada por un dispositivo especial, llamado *master*. La sincronización se lleva a cabo con una señal de reloj (CLK) que es distribuida por el bus. Aunque dicha distribución se realiza por una línea distinta de la de datos, aumentando de esta forma la complejidad del cableado y conectores, permite eliminar la circuitería necesaria para la recuperación del reloj.
- Se eliminan los bits de inicio y fin propios de transmisiones asíncronas, aumentando el aprovechamiento del canal.

4.2. Nivel de enlace de datos

El nivel de enlace de datos utiliza como base el canal físico descrito en la sección anterior y presentar al nivel superior una línea libre de errores. Debe garantizar que los mensajes o tramas llegan al destinatario correcto, sin fallos ni repeticiones.

Dentro de este nivel, es de especial importancia el subnivel de acceso al medio (MAC, *Medium Access Sublayer*). En un sistema en el que el canal es común a todos los subsistemas, el protocolo MAC garantiza que la comunicación es ordenada [3]. El protocolo es muy simple debido a que se establece una relación maestro-esclavo. Sólo el maestro tiene permiso para gobernar el canal y los esclavos acceden a éste bajo petición expresa del master.

Todo dispositivo conectado al canal dispone una dirección de red, compuesta por un número de 8 bits. Dado que la dirección 255 se ha reservado para las trasmisiones *broadcast*, sólo puede haber 255 dispositivos en el bus con dirección propia. El *master* tiene asociado la dirección 0. El resto de las direcciones son dinámicas y las asigna a cada elemento del bus durante la fase de configuración, que tiene lugar automáticamente al inicializar el sistema.

La trama de nivel de enlace de datos tiene la siguiente forma: un byte inicial reservado, que debe valer 0. Un byte de dirección destino. Un byte de cuenta que indica el número de bytes del campo siguiente (si este campo es 255, entonces los dos bytes siguientes indican el número de bytes del campo de datos). Por último el campo de datos, de longitud variable y un par de bytes para el CRC (*Cyclic Redundancy Code*, código de redundancia cíclica). Si se desea transmitir una trama a todos los elementos del sistema, por ejemplo, Inicialización, se debe incluir como código el 255. Dado que es el master el que gobierna la transacción, se ha implementado un simple protocolo de parada y espera.

4.3. Nivel aplicación

El nivel de aplicación reúne un conjunto de protocolos tal como se adelanta en la sección 2.1. Se han implementado protocolos para la transferencia de ficheros, gestión de esclavos, configuración automática.

La transferencia de ficheros es necesaria puesto que se ha dotado a MIDAS de un sistema de almacenamiento masivo (inicialmente 32 Mbytes). Dicho sistema está formado por memoria de estado sólido (SRAM), que permite una operación rápida, fiable y de bajo consumo. Está destinada inicialmente al almacenamiento de los datos recolectados por el datalogger.

Muy relevante, desde el punto de vista del usuario final es el protocolo que se encarga de la configuración automática. Éste permite que el sistema detecte automáticamente los dispositivos conectados así como sus capacidades, sin intervención del usuario. Cuando se inserta un nuevo dispositivo en la red que no dispone de dirección, éste solicita una interrupción al master. Cuando la interrupción es reconocida, el protocolo le asigna una nueva dirección de red. Para poder identificar cada dispositivo mientras tanto, éstos disponen de un número único

de 32 bits que se asigna en fábrica.

5. Resultados

Puesto que ya se han construido algunos prototipos de MIDAS, se han realizado algunas pruebas y mediciones. El coste aproximado unitario de producción para grandes series es:

Master: 7.000 Ptas.
Interface de usuario (Pantalla alfanumérica LCD y teclado numérico): 5.000 Ptas.
Sensor de dirección o velocidad de viento en aluminio anodizado/plástico: 15.000/4.000 Ptas.
Sensor de temperatura (ambiente o de suelo): 5.000 Ptas.
Sensor de precipitación (lluvia): 10.000 Ptas.
Módulo de almacenamiento masivo de 32 Mbytes: 50.000 Ptas.
Módulo de entrada analógica de 16 canales de 16 bits: 5.000 Ptas.
Módulo de entrada/salida digital de 32 canales: 4.000 Ptas.

El consumo medio de una configuración para aplicación de estación meteorológica, compuesta de un master, sensores de temperatura ambiente y de suelo, sensores de dirección y velocidad de viento, sensor de lluvia, sensor de humedad relativa y sensor de presión atmosférica, resultó ser inferior a 5 mA.

6. Conclusiones y Trabajo Futuro

Se ha presentado la aplicación de los sistemas operativos en tiempo real y los protocolos de comunicación al diseño de un sistema de adquisición de datos modular de bajo coste y bajo consumo. Los resultados obtenidos de los primeros prototipos muestran que disponer de varios sensores de bajo consumo conectados a través de una red a un dispositivo principal en el que recae la tarea de recolección de datos, es una forma interesante de reducir costes, consumo de potencia y complejidad del sistema, incrementando por otro lado la flexibilidad, escalabilidad y robustez. Nuestro esfuerzo en el futuro se centrará en añadir nuevas capacidades al sistema, así como en mejorar otras:

- programación orientada a objetos;
- sistemas con varios master;
- multiplexión de varias redes lógicas en un mismo medio físico;
- protocolo de operación remota, que permita operar el sistema desde interfaces de usuarios múltiples.
- aumentar la flexibilidad y velocidad del proceso de salida del estado de *standby*, con objeto de aumentar el rendimiento del sistema, reduciendo simultáneamente su consumo.

7. Referencias

[1] Ramón Pallás Areny, *Sensores y Acondicionadores de señal*. Ed. Marcombo, 1.994. ISBN: 84-267-0989-3.
[2] Day and Zimmermann, *The OSI Reference Model.*
[3] Andrew S. Tanenbaum, *Computer Networks*. Ed. Prentice-Hall, 1996, 3rd edition. ISBN 0-13-349945-6.
[4] Andrew S. Tanenbaum et al., *Operating Systems: Design And Implementation*. Ed. Prentice-Hall, 1997. 2nd edition. ISBN 0-13-638677-6.
[5] *Discrete Z8 Microcontrollers. Product Specifications Databooks*. Zilog, Inc.
[6] *H8/520 Hardware Reference Guide*. Hitachi
[7] Willian Stallings, *Operating Systems*. Ed. Prentice-Hall, 1997, 2nd edition. ISBN 0-13-180977-6.