XIV Conferencia Latinoamericana de Informática
17avas. Jornadas Argentinas de Informática
e Investigación Operativa
Buenos Aires, Setiembre 1988.

SOFTWARE PARA CONTROL DE TIEMPO REAL:

LA CONCEPCION ESTRUCTURADA DE UN BIOS

H. Fabián Torres. (*)
Marcelo Cardós. (*)
José Lizzoli. (*)
Armando De Giusti. (**)

(*) Alumno avanzado de la Licenciatura en Informática.

(**) Profesor Titular de la Fac. de Cs. Exactas de la UNLP.
Investigador Independiente del CONICET.

RESUMEN:

Se presenta el desarrollo de un BIOS para microcomputadora, orientado a ser utilizado en aplicaciones de tiempo real.-

A partir del estudio detallado de un BIOS convencional, se analiza su implementación en lenguaje C y la posibilidad de extender y adecuar sus posibilidades para sistemas de control dedicados.-

Se presentan medidas de rendimiento del software desarrollado y las extensiones que permiten controlar hardware "no convencional" desde el mismo BIOS.-

INTRODUCCION:

El desarrollo de sistemas de tiempo real, y especialmente de control dedicado, basados en microprocesadores ha caracterizado el área electrónica desde mediados de la década del 70, en nuestro país y en el mundo (Ref. 1,2,3,4).-

La primer etapa tecnológica relacionada con estos desarrollos se caracterizó por los equipos "a medida" desde el punto de vista del hardware y la utilización de lenguaje assembler desde el punto de vista del software (Ref. 5,6).-

Simultáneamente el desarrollo explosivo del mercado de microcomputadoras personales (PC's) y el establecimiento de arquitecturas estándar produjo un notorio abaratamiento de los costos de este tipo de hardware y el desarrollo de una serie de productos de software de carácter general (Sistemas Operativos, Lenguajes, utilitarios) permitieron extender sus aplicaciones originales de cálculo y procesamiento de información al desarrollo de sistemas dedicados basados en arquitecturas tipo PC (Ref. 7,8).-

Una rápida caracterización de un sistema de tiempo real nos permite apreciar tres "capas" o niveles tanto de hardware como de software:

1- La capa más "externa" está constituida por la interfaz entre el fenómeno de tiempo real tratado y el sistema digital que lo procesa. En ella habitualmente existe algún tipo de conversión de tipo a nivel de señal y las soluciones son dedicadas según las características del fenómeno que se está tratando.-

2- La capa de "comunicación" es la que permite resolver la recolección y trasmisión de los datos al procesador y de éste al subsistema externo. Habitualmente es una capa "crítica" desde el punto de vista de los tiempos involucrados y dadas las velocidades muy diferentes de los distintos procesos que pueden estar

en curso, es característico que dicha comunicación se base en interrupciones asincrónicas.-

3- Por último la capa más "interna" está constituída por el procesamiento puro de la información ya convertida. Habitualmente el software desarrollado para ella no se diferencia de cualquier software de aplicación, salvo en características de contexto como pueden ser la memoria disponible o el hecho de que la versión definitiva debe residir en ROM y por lo tanto debemos generar un módulo ejecutable que sea reubicable a partir de una dirección específica.-

Si se analiza desde el punto de vista de software, normalmente el trabajo de mayor envergadura está en la capa de comunicación:

** Es necesario un conocimiento detallado del hardware involucrado y del manejo de interrupciones del procesador.-

** Las soluciones son difíciles de generalizar para diferente tipo de sistemas de tiempo real.-

** La especificación formal y mucho más la validación de esta capa de software constituye un tema aún no resuelto en forma general (Ref. 9,10,11,12).-

Asimismo la utilización de hardware "PC like", que simplifica notablemente el costo de desarrollo de una parte importante del equipamiento, pero sobre todo el costo de desarrollo del software "de procesamiento", se encuentra limitada por la capa de más bajo nivel (BIOS), que fue desarrollada "a medida" para un subconjunto especial y restringido de periféricos, y que no es muy fácil de extender ni mucho menos de portar a otro procesador central.-

Por otra parte el BIOS en la mayoría de los sistemas de microcómputo de propósito general ha sido desarrollado en assembler del microprocesador involucrado, con criterios de optimización de memoria y tiempo de ejecución, que no hacen muy fácil su modificación o re-programación (Ref.13,14).-

Nuestro trabajo consistió en el análisis sistemático de un BIOS estándar y en el desarrollo en lenguaje C de un modelo de BIOS fácilmente extendible para el control de periféricos "no convencionales" (por ejemplo un conversor analógico-digital) o para disponer de nuevas funciones (por ejemplo criptografía de información) con alta eficiencia y mayor portabilidad.-

ANALISIS DEL BIOS DE UNA PC CONVENCIONAL:

El BIOS es un conjunto de rutinas destinadas a controlar el hardware de una microcomputadora tipo PC.-

Estas rutinas residen en ROM, en la parte alta del mapa de memoria y constituyen la capa de software más relacionada con la implementación física de la arquitectura de la microcomputadora involucrada.-

De este modo los programas de mayor nivel (incluyendo al mismo sistema operativo) "ven" una máquina "abstracta" relativamente independiente de su implementación física, brindando una mayor flexibilidad e independencia y asegurando la portabilidad de estos programas a modelos de arquitectura "compatible" que no han sido implementados con idéntico hardware y/o BIOS.-

Un rápido análisis de las rutinas del BIOS nos permite apreciar:

** Atención de RESET.-
** Atención de NMI (interrupción no-enmascarable).-
** Operación de carga del Sistema Operativo mínimo.-
** Atención de Servicios (video, teclado, diskettes, impresora, puerta serie).-
** Manejo del reloj de tiempo real.-
** Atención de otras interrupciones.-

La figura 1.A nos muestra una máquina de estados del funcionamiento del BIOS en un equipo convencional y la figura 1.B expresa simbólicamente el "programa" que ejecuta este conjunto de rutinas. En la figura 2 podemos ver el diagrama de ejecución del servicio de manejo de diskettes.

Figura 1.A: Máquina de estados del BIOS.

Estados:

A: Inicialización y testeo.
B: Espera de interrupciones de la máquina BIOS.
C: Atención de una interrupción de hardware.
D: Atención de una interrupción de software.

Condiciones:

A: Conexión de la alimentación.
B: Invocación de interrupción.
C: Invocación de una interrupción de software.
D: Desconexión de la alimentación.

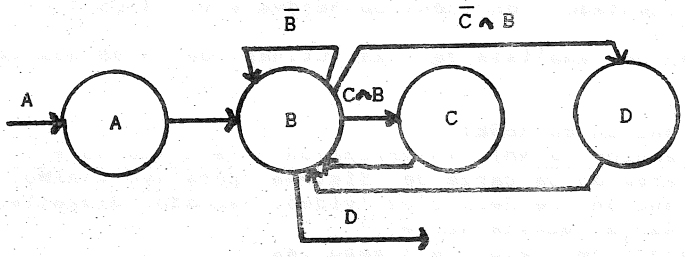


Figura 1.B: Pseudocódigo.

```
repeat
     repeat
     until  Invocación_de_Interrupción_de_la_Máquina_BIOS;
     case Interrupción of
          2           : NMI;
          5           : Prt_Scrn;
          8           : Timer_Int;
          9           : Kb_Int;
          A,B,C,D,F   : D_EOI;
          E           : Disk_Int
          10h         : Video_IO;
          11h         : Equipment;
          12h         : Memory_Size_Det;
          13h         : Diskette_IO;
          14h         : RS232_IO;
          15h         : Cassette_IO;
          16h         : Keyboard_IO;
          17h         : Printer_IO
          18h         : ROM_Basic;
          19h         : Boot_Strap;
          1Ah         : Time_of_Day;
          1Bh,1Ch     : Dummy_Return
     end;
until FALSE;
```

Figura 2: Máquina de estados del controlador de diskettes.

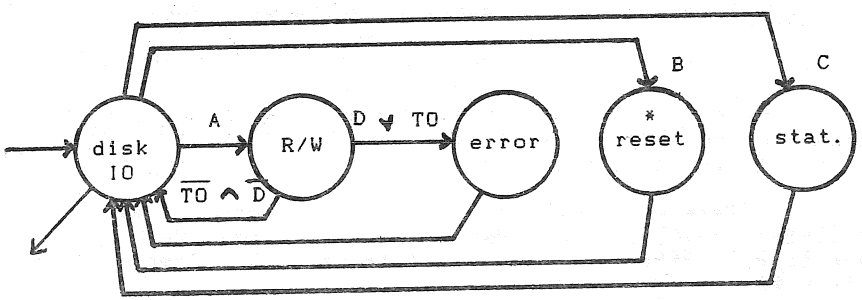
Estados:

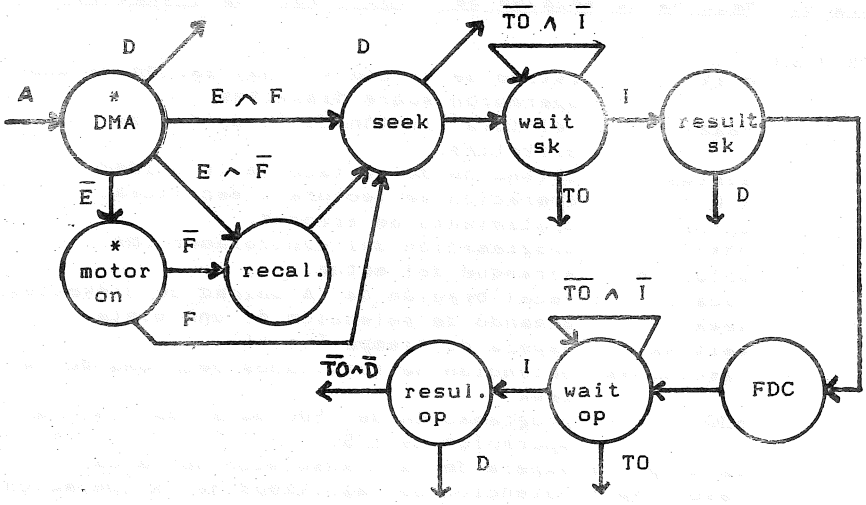
disk_IO	: estado de inicio y finalización de una operación sobre diskettes.
reset	: reinicialización de la unidad de diskettes.
status	: obtención del estado de la unidad.
R/W	: operación de lectura o escritura.
error	: tratamiento de errores.
DMA	: programación del controlador DMA.
motor on	: arranque del motor.
recal.	: recalibración de la unidad de diskettes.
seek	: comando de selección de una pista.
wait sk	: espera de tiempo 'seek'.
result. sk	: obtención de resultados del comando de seek.
FDC	: programación del FDC para realizar la operación de E/S.
wait op	: espera de la transmisión de datos.
resul. op	: obtención de resultados de la operación.

* Indica estado no interrumpible.

Condiciones:

A : solicitud de operación de lectura o escritura.
B : " " " de reinicialización.
C : " " " de informe del estado.
D : error en alguna de las operaciones que conforman la lectura o escritura.
E : motor en marcha.
F : recalibración no requerida.
TO: error producido por exceso de espera.
I : interrupción producida por el controlador de diskettes.





Las rutinas de servicio que forman parte del BIOS "normal" son:

Int 5h	-----	Servicio de vuelco de la pantalla a la impresora.
Int 10h	-----	Servicio de video.
Int 11h	-----	Servicio de listado de configuración.
Int 12h	-----	Servicio de tamaño de la memoria.
Int 13h	-----	Servicio de diskette.
Int 14h	-----	Servicio de comunicaciones.
Int 15h	-----	Servicio de cassette.
Int 16h	-----	Servicio de teclado
Int 17h	-----	Servicio de impresora.
Int 18h	-----	Servicio de activación del ROM-BASIC.
Int 19h	-----	Servicio de carga y puesta en marcha del sistema.
Int 1Ah	-----	Servicio de hora y fecha.

Para utilizar cualquiera de estos servicios, es necesario cargar los parámetros requeridos en los registros del microprocesador y luego ejecutar la interrupción de software deseada.-

Existe además un grupo de rutinas destinadas a controlar las interrupciones de hardware y previsiones para rutinas del usuario que deban ejecutarse a intervalos de tiempo determinados y la atención de 'break'.-

La implementación de estas rutinas, realizada en Assembler 8Ø88 o similar, es altamente eficiente tanto en código como en velocidad de procesamiento. Sin embargo la realización de cualquier modificación o extensión funcional de este conjunto de rutinas exige un profundo conocimiento de la arquitectura de la microcomputadora y del lenguaje Assembler del microprocesador correspondiente.-

CONCEPCION DEL LIDI-BIOS:

Del análisis del BIOS se pueden abstraer sus funciones de modo de obtener una estructura jerárquica como la que muestra la figura 3.-

Como se ve, tenemos tres grupos bien diferenciados que podemos describir como sigue:

** Las rutinas de inicialización del sistema son invocadas cuando se enciende la máquina y consisten en testeos de verificación del procesador y de memoria, inicialización de los circuitos integrados que lo requieran, inicialización de la tabla de vectores de interrupción, verificación de conexión de opcionales a la configuración básica y, si hay controlador de disco, carga del sistema operativo desde éste.

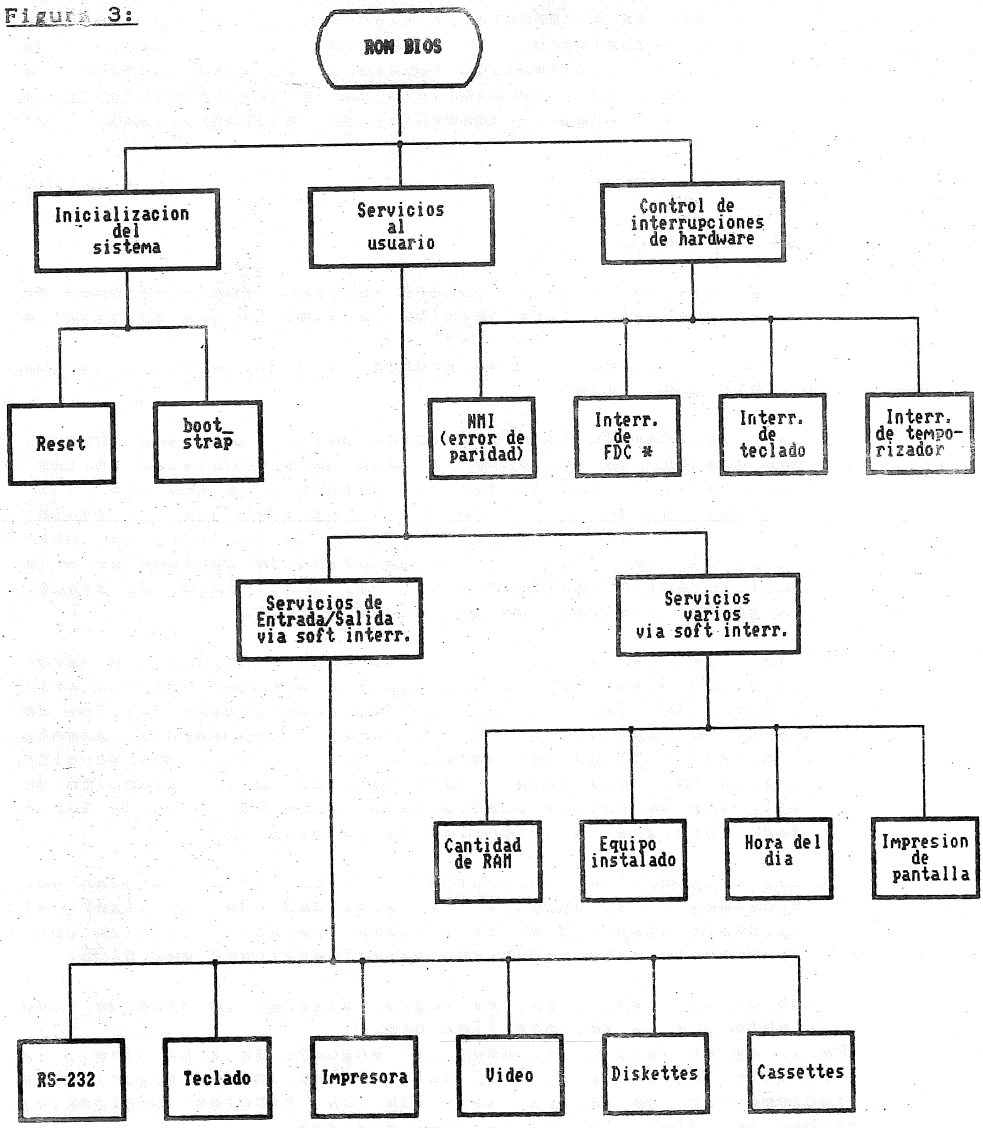
** Las rutinas de control de las interrupciones de hardware son las que se encargan de atender los requerimientos de comunicación con los subsistemas físicos de la microcomputadora y trabajan complementariamente con las rutinas de servicios al usuario. Una mención aparte hay que hacer para la rutina de atención de NMI, que se activa cuando se producen errores de paridad y provoca la detención de la máquina.

** Las rutinas de servicios al usuario se invocan por programa y atienden a la necesidad de utilizar el hardware disponible. El sistema operativo utiliza continuamente este mecanismo para "servirse" del BIOS.-

El objetivo, entonces, es lograr diseñar un BIOS de modo que sea portable y a la vez más flexible.

Para lograr esto utilizamos un lenguaje de alto nivel, ya que nos permite lograr una fácil estructuración del algoritmo e incluso implementar de manera sencilla los árboles funcionales que describen las distintas partes del sistema.

Figura 3:



* **FDC** : **Floppy Disk Controller**

El lenguaje de programación C nos permite establecer una relación casi directa con el hardware, ya que nos provee las siguientes herramientas:

** Aritmética de direcciones (punteros)

** Manejo total de la memoria
En este sentido existe la posibilidad de solapar información en una misma posición de memoria, definición de estructuras a nivel de bit y otros mecanismos que son parte del lenguaje y que otorgan gran flexibilidad en las operaciones sobre memoria.-

** Control de periféricos
Para esto contamos con operaciones que nos permiten programarlos por intermedio de sus puertas de datos y control de entrada/salida.-

** Control de interrupciones
Se pueden enmascarar y desenmascarar interrupciones, de modo de asegurar la exclusión mutua de los procesos en ejecución.-

** Control de llamadas a procedimientos de biblioteca
Podemos controlar la inclusión de procedimientos de bibliotecas, ya que éstas pueden involucrar invocaciones al sistema operativo, perdiendo además el control sobre el tamaño del código generado.

Contando con un lenguaje que posee estas características, vemos que es posible escribir un BIOS en un lenguaje de alto nivel de manera sencilla, legible y que al mismo tiempo sea razonablemente eficiente.-

Acerca de las rutinas interrupt:

Cuando fue descripta la estructura del BIOS, se vio que éste está compuesto por un conjunto de módulos de manejo de interrupción; algunos nuevos compiladores de C proveen el modificador de función 'interrupt' que le da al programa las características necesarias para el manejo de interrupciones.

En efecto estas rutinas realizan el salvado del estado de la máquina (apilado de los registros del micro) y el retorno se realiza por medio de la instrucción IRET, la que, a diferencia del retorno normal, tiene en cuenta el desapilado de las banderas de estado, que son automáticamente apiladas al producirse una interrupción (sea de software o de hardware). Además se permite el acceso a los registros salvados sobre la pila, para obtener su contenido o para modificarlos; esto es muy importante para el planteo de nuestro problema, dado que en las rutinas de servicios del BIOS el intercambio de datos entre éstas y los programas lla-

mayores se realiza por intermedio de los registros del microprocesador.

Finalmente, podemos decir que el LIDI-BIOS, así concebido, cumple con las condiciones que inicialmente habíamos planteado acerca de la portabilidad y flexibilidad, en el sentido de que con cambios mínimos es posible utilizar diferentes implementaciones del lenguaje y adaptarlo a distintos requerimientos de hardware en arquitecturas similares.

LA INSTALACION DEL LIDI-BIOS EN ROM:

Anteriormente fue dicho que el BIOS, al residir en ROM, forma una capa de software transparente al usuario que aísla a los programas de aplicación del hardware. El LIDI-BIOS fue desarrollado teniendo en cuenta la posibilidad de implementarlo en alguna memoria de lectura solamente.-

El problema que surge en forma inmediata al plantear el tema de la generación de código "romable", es la ubicación en memoria de las variables. El BIOS estándar utiliza una pequeña área de RAM de menos de 256 bytes reservada para su uso exclusivo, donde son almacenados ciertos datos que deben mantenerse entre llamadas sucesivas al BIOS, tales como direcciones de puertas de periféricos, parámetros corrientes de video, etc; las "variables" del BIOS son los propios registros del microprocesador.-

Obviamente, este esquema no es aplicable a los lenguajes de alto nivel, dado que en este caso no se posee control directo sobre los registros y las variables siempre ocupan algún área en la memoria, mayor que los 256 bytes antes mencionados.-

En la implementación del LIDI-BIOS también se planteó la existencia de alguna RAM para uso exclusivo, la cual podría estar fuera de los 640 K estándar o "quitándole" memoria al sistema haciendo que el BIOS reporte menos memoria que la real; en el trabajo citado en Ref. 15 puede encontrarse una discusión general del tema y también el detalle de la técnica de implementación utilizada en este caso.-

Las variables utilizadas por LIDI-BIOS fueron definidas como estáticas; la imposibilidad de utilizar variables automáticas está dada por ser éstas definidas sobre el segmento de pila y es muy común que los programas de aplicación y comandos del sistema no provean de suficiente espacio de pila como para, además de salvar el estado de la máquina y realizar el apilado de los parámetros en el llamado a procedimientos, definir las variables locales. De haberse pretendido utilizar variables automáticas se debería haber provisto de un código de inicialización para cada módulo que armase una pila local, éste código debería haberse re-

alizado en assembler lo cual hubiera desvirtuado un objetivo central del trabajo.

El LIDI-BIOS aún no ha sido implementado en ROM. El mecanismo utilizado consiste en reservar un espacio de memoria dividido en dos partes de direcciones conocidas; en una de éstas es cargado, mediante un comando hecho al efecto, el módulo romable "simulando" ser la ROM, la otra parte constituye la RAM dedicada. El sistema se arranca mediante un comando que le pasa el control a la rutina de reset, cuyas principales tareas son mover las constantes a la RAM, dado que serán direccionadas junto con las variables, y reapuntar los vectores de interrupción. Este último proceso es realizado con las interrupciones deshabilitadas y, al ser rehabilitadas, es el LIDI-BIOS el que realiza el control de los periféricos.

EXPERIENCIAS REALIZADAS:

Todas las experiencias se realizaron teniendo en cuenta comparaciones de rendimiento entre las rutinas del BIOS original y el LIDI-BIOS. Se corrieron programas de prueba que ejecutaban llamadas al BIOS y utilizaban el temporizador de la máquina para determinar la cantidad de ticks utilizados.

El testeo de la cuenta se realizó mediante programas escritos en assembler que involucraban únicamente las intrucciones de máquina indispensables para la generación de los lazos y la invocación a las rutinas.

1- Manejo de video

Se realizaron dos experiencias sobre la función de simulación de impresora (función 14, utilizada por el MS-DOS), que consistieron en:

A- Impresión de 25 líneas (una pantalla completa) comenzando en la esquina superior izquierda. No se consume tiempo en scrolling.
B- Impresión de 200 líneas a partir de la última fila (nótese el tiempo demorado en el scrolling).

2- Manejo de disco

Se realizaron dos experiencias:

A- Lectura secuencial de las dos caras de un disco.
B- Lectura de 720 sectores tomados al azar.

3- Manejo de impresora

Se realizaron dos experiencias:

A- Impresión de 7000 caracteres, menos que la capacidad del buffer de la impresora utilizada.
B- Impresión de 32000 caracteres, aproximadamente 4 veces la capacidad del buffer.

	1.A	1.B	2.A	2.B	3.A	3.B
Estándar	35	340	2833	3066	169	4025
LIDI-BIOS	46	563	2694	3049	169	4025
Rendimiento	0.76	0.60	1.05	1.01	1.00	1.00

Los tres dispositivos seleccionados para estas experiencias tienen características dispares que pueden justificar los diferentes resultados obtenidos.

El control de video demanda un uso intensivo del procesador ya que la principal tarea consiste en el manejo de la RAM del adaptador. Este manejo se realiza por medio de punteros 'far' lo cual explica el menor rendimiento del programa en C.

La unidad de discos flexibles tiene piezas móviles con baja velocidad respecto del procesador, además la transferencia de información se realiza mediante DMA, lo cual no involucra uso del procesador. Debido a esto es que el programa en C puede competir con el assembler. Las diferencias a favor son debidas a que los algoritmos involucran lazos de espera optimizados por el compilador. En este punto debe hacerse notar la dificultad de los lenguajes de alto nivel para manejar retardos ya que dependen de cómo el compilador los implemente.

En el caso del control de la impresora no se notan diferencias por tratarse de un dispositivo lento. La sincronización del envío de información es realizada por medio de lazos de espera prolongados donde no se encuentra ninguna ventaja en los programas escritos en assembler.

EXTENSION DE LAS PRIMITIVAS DEL BIOS

Tal como lo hemos expresado a lo largo de este trabajo, uno de los objetivos centrales ha sido la extensión de las prestaciones estándar del BIOS a fin de poder manejar primitivas especialmente adecuadas para sistemas de tiempo real.-

Nuestras primeras experiencias, que a continuación presentamos, se refieren a la incorporación del control de una tarjeta de conversión analógica/digital (A/D) directamente a nivel de BIOS, y a la posibilidad de disponer entre los servicios para el usuario una forma de registro y recuperación de datos en diskettes con codificación y decodificación bajo clave de seguridad.-

1- El manejo de un conversor A/D desde el LIDI-BIOS:

En lo que sigue se ejemplificará con la plaqueta conversora DT28Ø1 (Ref. 16); sin embargo los conceptos son similares para una gama de módulos de E/S.-

La plaqueta DT28Ø1 es un conversor analógico/digital que admite 8 ó 16 entradas (cada una de ellas con tensiones unipolares o bipolares de 1.25, 2.5, 5.Ø ó 1Ø.Ø Voltios) y entrega una salida de 12 bits de resolución.-
La velocidad de respuesta es de 13.7ØØ muestras por segundo (15 microsegundos por muestra para la adquisición y 25 microsegundos por muestra para la conversión propiamente dicha).-

Esta plaqueta se instala directamente en el bus de una PC compatible, ocupando dos bytes en el mapa de direcciones de E/S y pudiendo ser direccionada mediante una combinación de 1Ø bits. Esta dirección es reprogramable por jumpers.-

Los comandos de conversión A/D son 4:

** READ A/D inmediato: realiza una única conversión sobre una entrada y devuelve el valor binario equivalente.-

** SET A/D : permite fijar la ganancia, el o los canales sobre los que se realizarán las conversiones y el número de muestras a tomar. Es un comando previo a todo READ A/D.-

** READ A/D: este comando inicia una secuencia sincrónica de conversiones en los canales especificados por un previo SET A/D.-

** READ A/D con DMA: se trata del mismo comando anterior, pero los datos convertidos son transferidos a memoria vía DMA, es decir sin intervención del procesador.-

En una configuración de PC estándar, el manejo de una plaqueta como la DT28Ø1 debe hacerse mediante un programa en BASIC, FORTRAN o ASSEMBLER. Esto significa un importante overhead respecto del tiempo de muestreo propio de la plaqueta, y además restringe los lenguajes "de procesamiento" seleccionables para las aplicaciones de la plaqueta a aquéllos que permiten el direccionamiento absoluto del mapa de I/O. Más aún, si las muestras deben grabarse en diskette, muchas veces el tiempo de proceso excede lo aceptable para la aplicación.-

En nuestro caso se optó por incorporar a los servicios del LIDI-BIOS la atención de una placa "tipo DT28Ø1"; esto significó un reducido bloque en lenguaje C que implementa la activación de los 4 comandos mencionados anteriormente (Ref. 2Ø). La ventaja que inmediatamente se obtiene es que desde cualquier lenguaje de programación que pueda invocar al BIOS se puede manejar este servicio con altísima eficiencia.-

Lógicamente una extensión como la mencionada anteriormente es muy especial y "dedicada"; sin embargo éste era precisamente el objetivo buscado: poder personalizar mediante las funciones del BIOS un hardware estándar a fin de utilizarlo en aplicaciones dedicadas.-

2- Codificación/Decodificación directa:

Es indudable la utilidad e importancia de poder guardar información "protegida" en un ámbito compartido (por ejemplo un soporte magnético accesible por diferentes usuarios).-

Una primer aproximación, muy común a nivel de sistema operativo, es el control del acceso a la información a través de "palabras clave". Sin embargo esta solución es relativamente segura frente a las diferentes técnicas de "lectura física" o de "revisión exhaustiva" del contenido de un soporte magnético.-

Un nivel mayor de seguridad podemos obtenerlo si la información a preservar es "codificada" antes de grabarse en el soporte, por ejemplo agregando redundancia mediante un código lineal o no lineal (Ref. 17) y utilizando como generador del código un polinomio "exclusivo" del usuario.-

La recuperación de la información sólo podrá realizarse conociendo el polinomio "clave" y toda copia de la información no decodificada sería inútil.-

Desde el punto de vista del LIDI-BIOS, se trata simplemente de agregar una rutina de servicio en el árbol de la figura 3. Esta rutina de servicio es muy semejante a de lectura/grabación de diskettes, sólo que recibe como parámetro

adicional los coeficientes del polinomio (en nuestro caso 4 números hexadecimales a fin de manejar un polinomio de grado 15 o menor, que significa 16 bits de redundancia).-
Con este polinomio como dato, en la grabación se simula por software el funcionamiento del registro de desplazamiento que realiza la codificación en este tipo de códigos (Ref. 18, 19) y luego se graba el bloque de información más la redundancia.-
Recíprocamente, en la lectura se simula la función del registro de desplazamiento "decodificador" (divisor) y se recupera el dato original.-

Lógicamente, a nivel de sistema operativo o desde un lenguaje de aplicación puede explotarse esta prestación del LIDI-BIOS en forma mucho más abstracta, por ejemplo manteniendo el modo de trabajo de "palabra clave" y realizando internamente la transformación en el polinomio asociado, o teniendo un archivo con los polinomios útiles de grado 15 y permitiendo al usuario el manejo de una clave numérica directamente asociada con los elementos de este archivo.-

Por otra parte es de hacer notar que la elección de un código polinomial lineal obedece simplemente al objetivo de indicar posibilidades dentro del LIDI-BIOS; tal vez para una aplicación de seguridad en textos convendría un código orientado a carácter y no a bit (Reed-Solomon, por ejemplo).-

CONCLUSIONES:

Se ha presentado el desarrollo de un BIOS en lenguaje C y algunas extensiones para la operación de periféricos dedicados o el tratamiento especial de información en sistemas de tiempo real.-

En el LIDI (Cs. Exactas - UNLP) está disponible el software desarrollado para aquéllos que estén interesados en el mismo.-

La línea de trabajo actual es el desarrollo de un ambiente para la construcción de Sistemas Operativos dedicados a partir de un kernel de primitivas basadas en el LIDI-BIOS.-

BIBLIOGRAFIA

1- IEEE TUTORIAL on MICROPROCESSOR APPLICATIONS.
IEEE Press 1981.-

2- A. O. WILLIMAN, H. J. JELINEK.
"Microprocessor Developments".
Computer IEEE. Jun 1976.-

3- T. IWASAKA, T. KANEKO.
"Control system for elevators".
Hitachi Review. Vol 28. 1979.-

4- A. KOCHHAR, N. BURNS.
"Microprocessors and their manufacturing applications".
E. Arnold Publishers. 1983.-

5- D. LIBES.
"An introduction to microcomputer languages".
Microsystems. Spt/Oct. 1982.-

6- IEEE TUTORIAL ON PROGRAMMING LANGUAGES.
IEEE Press 1982.-

7- M. BUCHNER, I. LEFKOWITZ.
"Distributed computer control for industrial process systems: characteristics, attributes and an experimental facility".
IEEE Control systems magazine, 1982.-

8- D. BERRY.
"Distributed intelligence in process control"
Control Engineering, may 1987.-

9- S. FAULK, D. PARNAS.
"On syncronization in hard real-time systems".
Communications of the ACM Vol 31, Num.5, Mar 1988.-

10- W. QUIRK.
"Verification and validation of real time software"
Springer Verlag 1985.-

11- C. HOARE.
"Communicating sequential processes"
Prentice Hall 1985.-

12- M. WATKINS.
"A technique for testing command and control soft ware"
Communications of the ACM Vol. 25, Num. 4, 1982.-

13- IBM PC XT Technical Reference.
1986.-

14- R. JOURDAIN.
"Programmer's problem solver for the IBM PC XT & AT"
Prentice Hall 1986.-

15- H. FABIAN TORRES, A. DE GIUSTI.
"Código romable en lenguaje C".
Informe Técnico LIDI 88-0601Ø, 1988.-

16- User Manual for DT2801 Series: Single board analog and digital I/O system.-
1984.-

17- SHU LIN, D. COSTELLO.
"Error Control Coding". Prentice Hall 1983.-

18- W. PETERSON, E.J. WELDON.
"Error Correcting codes". MIT Press 1972.-

19- E.R. BERLEKAMP.
"Algebraic coding theory". McGraw Hill 1968.-

2Ø- A. DE GIUSTI, H. FABIAN TORRES.
"Comando de un conversor A/D tipo DT28Ø1 desde el LIDI-BIOS".
Informe Técnico LIDI-88-0601l, 1988.-