Escuela Politécnica Superior
Universidad de Huelva

Departamento de Ing. Electrónica,
Sistemas Informáticos y Automática

TERCER CURSO. INFORMÁTICA INDUSTRIAL II

Interface Analógica

Manuel Sánchez Raya
Versión 0.99
20 de Octubre de 2002

ÍNDICE

BIBLIOGRAFÍA:

Apuntes de Ingeniería Técnica Industrial. Universidad de Málaga.

Microcontroladores MCS-51 y MCS-251. José Matas Alcalá, Rafael Ramón Ramos Lara

C and the 8051, Tom Schultz, Prentice Hall

1.- Introducción.

En el control de procesos industriales se precisa tener el valor instantáneo de determinadas señales analógicas, de forma que se pueda aplicar un algoritmo de control que gestione y dote de ciertas características al sistema que se desea controlar. De la misma forma, en el procesado de señal también es necesario tener el valor instantáneo de la señal analógica, sobre la que se aplican ciertos algoritmos digitales como pueden ser filtros FIR' o IIR', con el fin de extraer determinadas características de la señal medida. En consecuencia, es habitual tener convertidores del tipo analógico/digital, A/D, y digital/analógico, D/A, en un sistema basado en microcontrolador, para efectuar el control de sistemas, monitorizar el estado de señales analógicas y efectuar el procesado digital de señales. Mediante los convertidores A/D se obtiene el valor discreto de la señal analógica, mientras que con los convertidores D/A se convierte un valor discreto en analógico, pues es necesario que un microcontrolador intervenga de esta manera en determinados sistemas.

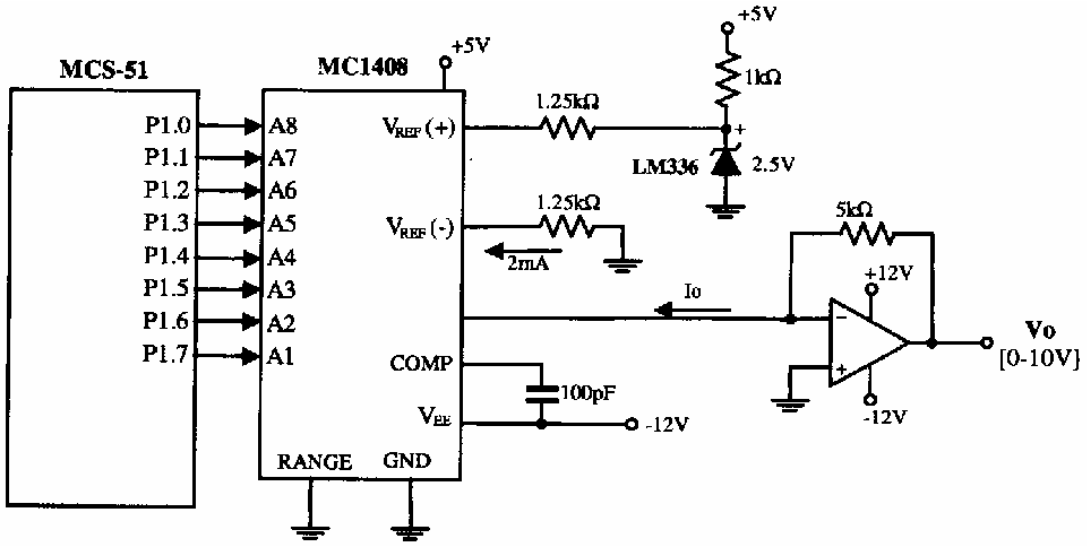
La familia MCS-51 de Intel no dispone de conversores A/D ni D/A, con la excepción de la versión 87C51GB, que tiene un conversor A/D de 8 bits de resolución con hasta 8 canales multiplexados de entrada. Por tanto, en estas familias los convertidores A/D y D/A se deben conectar externamente al microcontrolador. En este sentido, se pueden emplear los puertos del microcontrolador para enviar datos, recibir datos y realizar el control de los conversores AID y D/A. No obstante, la modulación de anchura de pulsos, PWM, que se puede generar con los microcontroladores que disponen de PCA, se puede utilizar como conversor D/A para las aplicaciones donde se requiere el control de un motor, debido a que la acción de control de un motor se suele realizar mediante el ciclo de trabajo de una señal PWM.

Hay otros fabricantes de la familia MCS-51 que proporcionan versiones con convertidor A/D, como por ejemplo el SAB80C515A de Siemens, que incorpora un conversor A/D de 10 bits. Por tanto, el diseñador, en función de los costos de desarrollo, siempre puede optar por conectar conversores A/D y D/A comerciales al microcontrolador, o bien, por emplear versiones que ya los tengan incorporados.

Este capítulo se centra en la conexión de convertidores A/D y D/A al microcontrolador, válida para la MCS-51, así como para otros microcontroladores del mercado. También, se expone la implementación de varias técnicas de conversión A/D de bajo coste, en las que se utilizan algunos de los recursos internos del microcontrolador.

2.- Conexión de un convertidor D/A.

La figura siguiente muestra la conexión del convertidor digital/analógico MC1408DAC de 8 bits de Motorola, que es un convertidor sencillo y de bajo costo. El convertidor está basado en una estructura en escalera de resistencias del tipo R-2R. De esta manera, la corriente de salida es proporcional a la palabra digital de entrada del convertidor, D0-D8. La corriente finalmente obtenida se convierte en tensión mediante el amplificador operacional de salida, cuya ganancia puede ajustarse a fondo de escala, o sea, con todas las entradas a 1 lógico, con la resistencia RT del amplificador, para que la tensión de salida tenga un valor determinado.



En el circuito de la figura, el diodo zener de 2.5V y las resistencias de 1.25 kohm se emplean para establecer una fuente de corriente precisa de 2 mA. La corriente Io de salida del convertidor D/A se convierte en tensión mediante el amplificador operacional, de manera que la tensión de salida de este amplificador es:

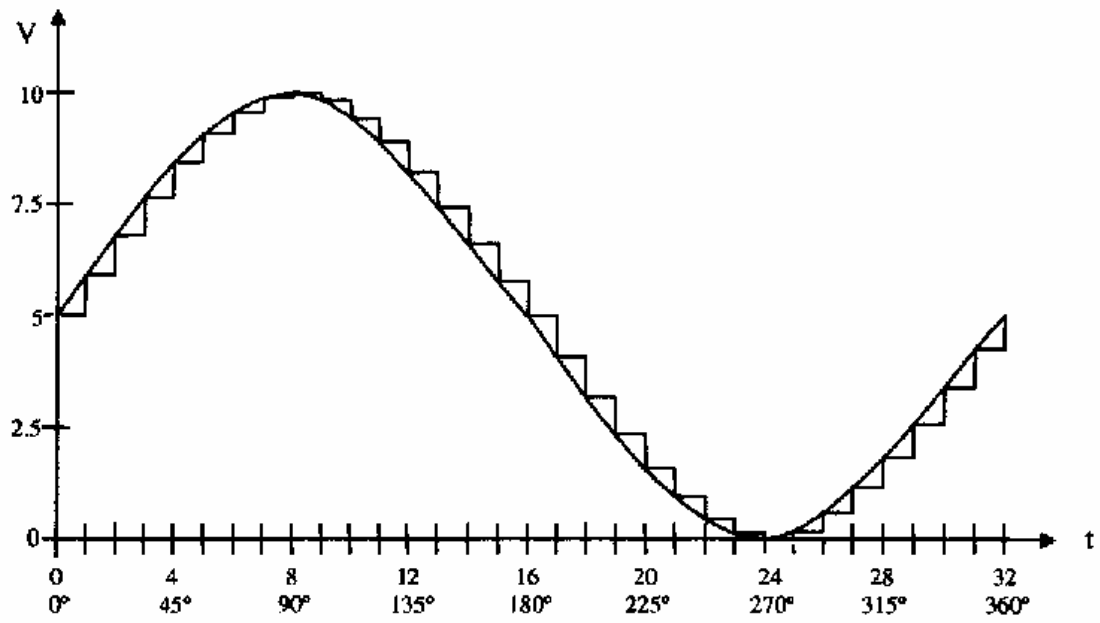
$$vo = 2mA \cdot \frac{\text{Entrada binaria}}{256} \cdot 5k\Omega$$

Con esta fórmula, la comente a fondo de escala, es decir, con la palabra binaria con todos los bits a 1 lógico, es de 2mA; por tanto, la tensión máxima a fondo de escala será de 1OV. Este valor se puede modificar mediante la resistencia RT de realimentación del amplificador operacional.

El tiempo de respuesta del convertidor DIA es un parámetro que determina su máxima velocidad de conversión; el MC1408DAC tiene un tiempo de establecimiento “settling time” de 300ns. Y, la resolución del MC1408DAC, para el fondo de escala de l0V, es de 39.l mV, que equivale al incremento de la tensión de salida que se produce cuando el bit de menor peso de entrada del DAC cambia de valor.

Ejemplo 1 Generación de una senoide mediante el MC1408DAC.

En este ejemplo, se trata de realizar un programa para la MCS-51 con el circuito de la figura anterior, que sea capaz de sintetizar una señal senoidal de 1.25kHz de frecuencia. Para ello, se debe tener en cuenta que la tensión de salida Vo del circuito es unipolar, y que se debe especificar un número de muestras de salida para que la señal tenga una forma adecuada. En este caso, se obtienen 32 muestras de la señal (figura siguiente), es decir, 16 muestras por cada semiperíodo, para representar la señal senoidal.



La señal senoidal tiene una componente continua de 5V, de manera que su valor oscilará entre 0V y 10V, según la siguiente expresión:

$$Vo = 5V + 5V \cdot \sin(\theta)$$

, donde θ es el ángulo de cada muestra de la senoide. En la tabla siguiente se presenta el valor de cada una de las muestras de salida del convertidor D/A, en voltios, y el valor binario correspondiente a la entrada del convertidor. En el programa de este ejemplo se debe implementar la tabla de datos, D7-D0, que debe extraer por el puerto del microcontrolador. El Timer 1, mediante interrupción, determinará la base de tiempos de extracción de los datos por el puerto P1.

Para generar una frecuencia de 1.25kHz el periodo de la señal debe ser de 0.8ms, y el de cada muestra de salida de 0.8ms/32, es decir, cada muestra se debe extraer cada 25µs. Este período se puede generar con el Timer 1, configurado en el modo 2 de 8 bits con autorrecarga. Con una frecuencia de reloj de 12MHz, y con el Timer contando pulsos internos, bit C/T a 0 lógico, se puede hacer que el Timer interrumpa cada 25 µs, poniendo el registro TH1 al valor E7H, es decir, a la diferencia entre el valor de rebasamiento del Timer y 25 µs (256µs -25µs).

θ	sin(θ)	Vo (V)	D7-D0	θ	sin(θ)	Vo (V)	D0-D7
0°	0	5	80H	180°	0	5	80H
11.25°	0.195	5.975	99H	191.25°	-0.195	4.024	67H
22.5°	0.382	6.913	B1H	202.5°	-0.382	3.086	4FH
33.75°	0.555	7.777	C7H	213.75°	-0.555	2.222	39H
45°	0.707	8.535	DAH	225°	-0.707	1.464	25H
56.25°	0.831	9.157	EAH	236.25°	-0.831	0.842	13H
67.5°	0.923	9.619	F6H	247.5°	-0.923	0.380	08H
78.75°	0.980	9.904	FDH	258.75°	-0.980	0.096	02H
90°	1	10	FFH	270°	-1	0	00H
101.25°	0.980	9.904	FDH	281.25°	-0.980	0.096	02H
111.5	0.923	9.619	F6H	292.5°	-0.923	0.380	08H
123.75°	0.831	9.157	EAH	303.75°	-0.831	0.842	13H
135°	0.707	8.535	DAH	315°	-0.707	1.464	25H
146.25°	0.555	7.777	C7H	326.25°	-0.555	2.222	39H
157.5°	0.382	6.913	B1H	337.5°	-0.382	3.086	4FH
168.75°	0.195	5.975	99H	348.75°	-0.195	4.024	67H

El programa para generar la señal senoidal en la MCS51 se muestra a continuación:

```
/*************************************************************/
/* Generación de una señal senoidal periódica de 32 muestras */
/*************************************************************/

#include <reg51.h>
#define uchar unsigned char

uchar code Tab_Seno[32] = { 0x80,0x99,0x0B1,0x0C7,0x0DA,0x0EA,0x0F6,0x0FD,
                            0x0FF,0x0FD,0x0F6,0x0EA,0x0DA,0x0C7,0x0B1,0x99,
                            0x80,0x67,0x4F,0x39,0x25,0x13,0x08,0x02,
                            0x00,0x02,0x08,0x13,0x25,0x39,0x4F,0x67};
uchar cuenta;

void RSI_Timer1(void) interrupt 3 using 1 {   /* 8*n+3 */
      P1=Tab_Seno[cuenta];      /* Saca al puerto valor de la funcion */
      cuenta=(cuenta++)&0x1F;   /* incrementa indice,pone a cero si >31 */
}

void main(void) {

      ET1 = 1;          /* Habilita interrupción del Timer 1 */
      PT1 = 1;          /* Asigna prioridad alta al Timer 1 */
      EA = 1;           /* Habilita bit de interrupción general */
      TMOD = 0x20;      /* Timer 1 en el Modo 2, GATE=0 y C/T=0 */
      TL1 = 0x0E7;      /* (256-25) valor inicial para 1.25kHz */
      TH1 = 0x0E7;      /* valor inicial para 1.25khz */
      TR1 = 1;          /* Pone en marcha el Timer 1 */
      cuenta = 0;       /* inicializa contador */

      for(;;);          /* Bucle sin fin */
}
```

La rutina principal de este ejemplo consiste en un bucle infinito sin ninguna función definida, puesto que la señal senoidal se genera totalmente por interrupción. La variable cuenta se utiliza como un contador entre 0 y 31, que índica el orden del dato a leer en la tabla definida por la subrutina Tab_seno. Este contador se pone a cero cada vez que se ha extraído la última muestra de la señal senoidal.

```
C:0x0019    00      NOP
C:0x001A    00      NOP
C:0x001B    02001E  LJMP    RSI_Timer1(C:001E)
C:0x001E    C0E0    PUSH    ACC(0xE0)
C:0x0020    C083    PUSH    DPH(0x83)
C:0x0022    C082    PUSH    DPL(0x82)
C:0x0024    C0D0    PUSH    PSW(0xD0)
C:0x0026    75D008  MOV     PSW(0xD0),#0x08
    15: void RSI_Timer1(void) interrupt 3 using 1 {     /* 8*n+3 */
    16:       P1=Tab_Seno[cuenta];      /* Saca al puerto valor de la funcion */
C:0x0029    E510    MOV     A,cuenta(0x10)
C:0x002B    900043  MOV     DPTR,#Tab_Seno(0x0043)
C:0x002E    93      MOVC    A,@A+DPTR
C:0x002F    F590    MOV     P1(0x90),A
    17:       cuenta=(cuenta++)&0x1F;  /* incrementa indice,pone a cero si >31 */
C:0x0031    AF10    MOV     R7,cuenta(0x10)
C:0x0033    0510    INC     cuenta(0x10)
C:0x0035    EF      MOV     A,R7
C:0x0036    541F    ANL     A,#0x1F
C:0x0038    F510    MOV     cuenta(0x10),A
    18: }
C:0x003A    D0D0    POP     PSW(0xD0)
C:0x003C    D082    POP     DPL(0x82)
C:0x003E    D083    POP     DPH(0x83)
C:0x0040    D0E0    POP     ACC(0xE0)
C:0x0042    32      RETI
```

Para determinar el instante preciso en el cual se extrae por el puerto P1 una muestra de la señal senoidal, se debe tener en cuenta el tiempo en el que se genera la interrupción, el tiempo que tarda el microcontrolador en saltar a la rutina de RSI y el tiempo que tarda la rutina en poner el dato correspondiente en el puerto P1. La única forma de conocer esto cuando trabajamos con el compilador de C es echar un vistazo al código fuente que genera.

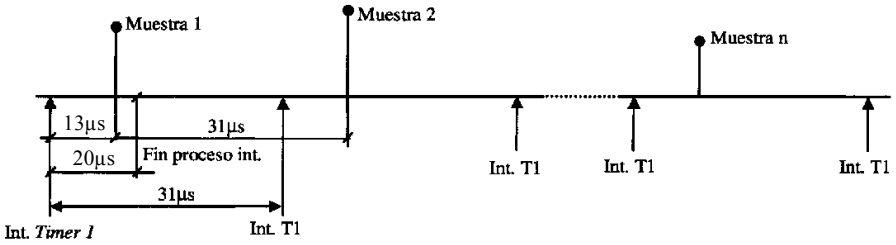
Podemos observar en el código fuente generado cosas bastante extrañas, sin embargo, el código ensamblador generado por el compilador es funcional. El Timer 1 causa una interrupción cada 31 µs, el tiempo mínimo en saltar a la rutina de RSI es de 3 ciclos máquina, o sea, de 3 µs para una frecuencia de reloj de 12 MHz, y la rutina, para poner el dato leído en el puerto, debe ejecutar 10 instrucciones, lo que supone un tiempo de 18 µs. En definitiva, desde que se produce la interrupción hasta que la muestra aparece en el puerto P1 transcurren 20 µs. Por tanto, cada muestra aparecerá 20 µs después de que se produzca la interrupción del Timer (figura siguiente).

Una vez sacada la muestra por el puerto P1, la rutina de RSI puede tardar 11 instrucciones más en terminar de ejecutarse por completo, lo que, en el peor caso, supone un tiempo de 16us adicionales. En consecuencia, el tiempo en que se finaliza el proceso de interrupción es de 34 µs, por lo que la interrupción del Timer 1 no puede producirse por debajo de este tiempo.

Este hecho supone que, si se quiere modificar la frecuencia de la señal senoidal, el período de cada muestra de la señal no puede ser inferior al proceso de interrupción; es posible generar, para este caso, 34 µs por muestra, como son 32 muestras, da un total de 1088 µs de periodo, por tanto una señal senoidal con una frecuencia máxima de 919Hz. No obstante, esta frecuencia se puede disminuir reduciendo la cantidad de muestras de la señal, o bien aumentando la frecuencia de la señal de reloj del microcontrolador. En nuestro caso resulta necesario modificar la rutina para que tarde como mucho 32 µs por muestra.

Podemos codificar en ensamblador nuestra rutina, con lo cual esta quedaría de la siguiente forma:

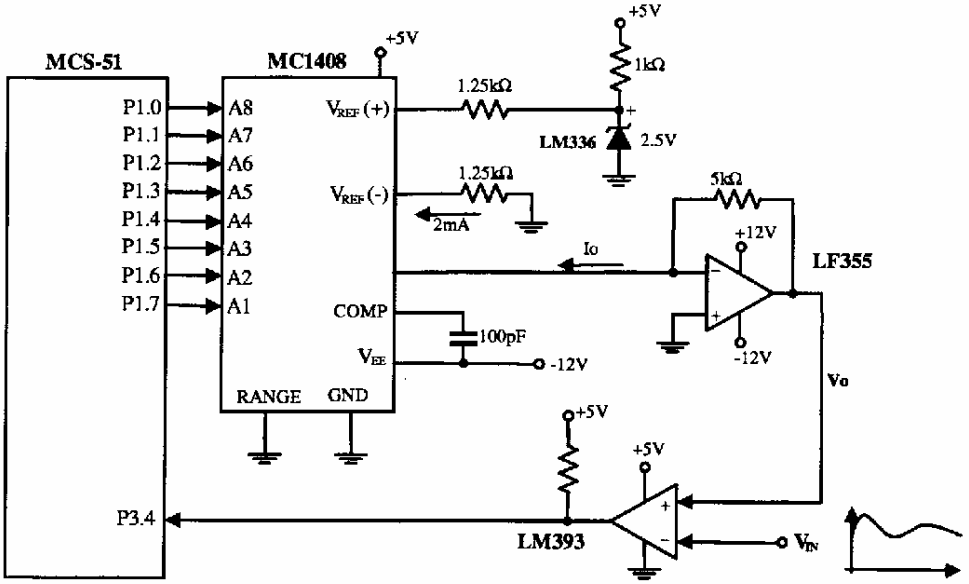
```
#pragma ASM
CSEG   AT     01BH
       MOV    PSW,#8        ;Cambia banco de registros (2 us)
       MOV    A,R7          ;Pone R7 en A, (1us)
       CALL   Tab_Seno      ;Llamada a subrutina de tabla, (2us)
       MOV    P1,A          ;Pone dato leído en A, (1 us)
       CJNE   R7,#31,Borra ;Mira si contador R7 supera máx. (2us)
       INC    R7            ;Incrementa contador R7, (1 us)
       MOV    PSW,#0        ;Restaura banco registros (2 us)
       RETI                 ;Fin de rutina. TF1 se borra, (2us)
Borra:
       MOV    R7,#0         ;Borra R7, (1 us)
       MOV    PSW,#0        ;Restaura banco registros (2 us)
       RETI                 ;Fin de rutina. TF1 se borra, (2us)
Tab_Seno:
       INC    A             ;Incrementa índice de lectura de tabla, (1us)
       MOVC   A,@A+PC       ;Lectura indexada de la tabla, (2us)
       RET                  ;Retorno de subrutina, (2us)
;Tabla de datos
       DB 80H, 99H, B1H, C7H, DAH, EAH, F6H, FDH
       DB FFH, FDH, F6H, EAH, DAH, C7H, B1H, 99H
       DB 80H, 67H, 4FH, 39H, 25H, 13H, 08H, 02H
       DB 00H, 02H, 08H, 13H, 25H, 39H, 4FH, 67H
#pragma ENDASM
```



En este caso, la señal tarda en salir 3+10 µs y la rutina RSI termina en 7 µs más, por lo que el tiempo total ahora es de 20 µs, la máxima frecuencia que se puede generar con 32 muestras es ahora mucho mayor: 1562 Hz.

3.- Convertidor A/D de bajo coste mediante aproximaciones sucesivas.

El convertidor digital/analógico anterior, MC1408DAC, se puede emplear, junto con un comparador y con el microcontrolador, para realizar un convertidor analógico/digital de bajo coste (figura siguiente), basándose en la técnica de aproximaciones sucesivas.

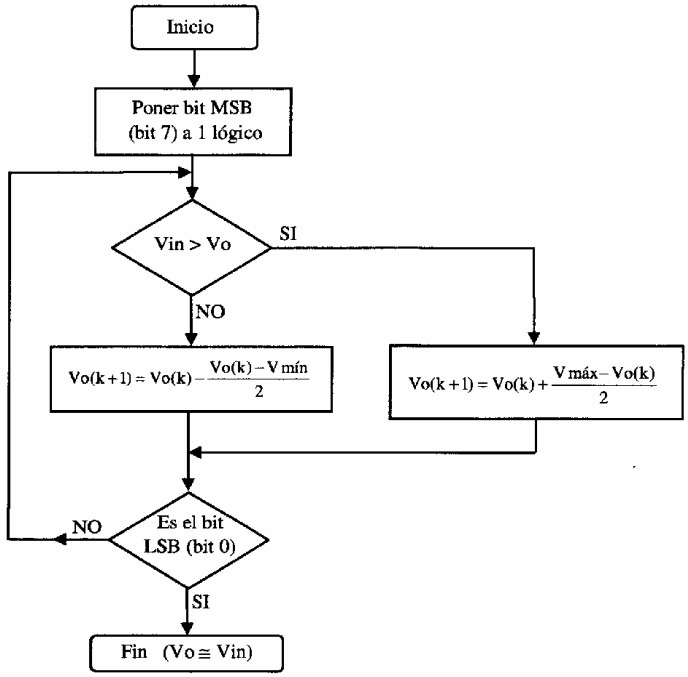


La técnica de aproximaciones sucesivas se centra en la búsqueda binaria, por parte del microcontrolador, de un valor de tensión de entrada V_{in} desconocido, comparándolo, para ello, con valores conocidos de tensión, obtenidos a partir de un convertidor D/A y de una tensión de referencia V_{ref}. Como el convertidor D/A y el circuito empleado en la figura anterior son los mismos, prácticamente, que en el apartado anterior, las tensiones máxima y mínima que se pueden generar en la salida Vo son 10V y 0V, respectivamente. El valor máximo se corresponde con la palabra digital 1111 1111 y el valor mínimo con 0000 0000 del puerto P1.

La búsqueda binaria se inicia poniendo V_o a la mitad del fondo de escala, o sea, a $(V_{max}-V_{min})/2$, por lo que en el puerto P1 se debe poner el bit más significativo a 1 lógico y el resto de bits a 0 lógico (1000 0000). La tensión V_o generada por el D/A se compara con la señal V_{in}, de forma que la salida del comparador es 0 lógico si V_{in} es mayor que Vo (Vin>Vo), y 1 lógico si V_{in} en menor que V_o ($V_{in}<V_o$). El microcontrolador debe leer el estado de la comparación, y si V_{in} es mayor que V_o, entonces le debe sumar a la tensión actual V_o la mitad del margen superior que queda hasta llegar al fondo de escala, es decir, $(V_{max}+V_o)/2$, lo que se consigue poniendo a 1 lógico el bit 6 del puerto P1 (P1=1100 0000). Si, al contrario, resulta que V_{in} es menor que V_o, entonces le debe restar a la tensión actual V_o la mitad del margen inferior que queda hasta llegar al mínimo de tensión, 0V, es decir, $(V_o-V_{min})/2$, lo que se consigue poniendo a 0 lógico el bit 7 y

a 1 lógico el bit 6 del puerto P1 (P1=0100 0000). Este procedimiento se continúa de forma sucesiva, hasta llegar a determinar el estado del bit 0 del puerto P1. La conversión de la señal se lleva a cabo de esta manera en tan sólo 8 pasos, empezando por el bit más significativo y terminando por el bit menos significativo.

La figura siguiente muestra el algoritmo que hay que realizar con la técnica de aproximaciones sucesivas. En este algoritmo $V_o(k)$ representa el valor de V_o actual y $V_o(k+l)$ representa el valor de V_o futuro a realizar. La conversión finaliza en 8 iteraciones, de forma que, partiendo del inicio, si tomamos el bit más significativo (MSB) como bit b(i), con i=7, se pueden dar dos situaciones en el algoritmo, dependiendo del resultado de la comparación entre V_{in} y V_o. En el caso de que V_{in} sea mayor que V_o, $V_{in}>V_o$, se ejecuta la segunda ecuación del algoritmo (ecuación 2), que en realidad se lleva a cabo simplemente forzando el bit b(i-1) a 1 lógico, es decir, para este caso, poniendo el bit b(6) a 1 lógico. Al contrario, si V_{in} es menor que V_o, $V_{in}<V_o$, se ejecuta la primera ecuación del algoritmo (ecuación 1), que se lleva a cabo poniendo el bit b(i) a 0 lógico y el bit b(i-1) a 1 lógico, es decir, poniendo en este caso el bit b(7) a 0 lógico y el bit b(6) a 1 lógico. Este algoritmo se iterará hasta llegar a evaluar el bit b(0).



La subrutina que realiza la conversión para la MCS51 se muestra a continuación:

```
/*************************************************************************/
/*     Subrutina CONV de aproximaciones sucesivas                        */
/*************************************************************************/
#include <reg51.h>
#include <stdio.h>
#define uchar unsigned char

sbit MENOR = P3^4;

uchar conv(void) {
       char i;
       uchar tmp;

       P1=tmp=0;
       for (i=7;i>=0;i--) {
              tmp=P1=tmp|(1<<i);
              if (MENOR) P1=tmp&(~(1<<i));
       }
       return  tmp;
   }
```

```
void main(void) {

        SCON = 0x52; /* Configura comunicación serie en modo 1 */
        TCON = 0x40;    /* TCON */
        TMOD = 0x20; /* Timer 1 en Modo 2, GATE=0 y C/T=0 */
        TH1 = 0x0FD; /* Valor para 9600 baudios */

        /* Bucle sin fin */
        for (;;) printf ("La conversion es %u.\n",(unsigned)conv());
}
```

```
    10: uchar conv(void) {
    11:         char i;
    12:         uchar tmp;
    13:
    14:         P1=tmp=0;
C:0x03ED    E4      CLR     A
C:0x03EE    FF      MOV     R7,A
C:0x03EF    F590    MOV     P1(0x90),A
    15:         for (i=7;i>=0;i--) {
C:0x03F1    7E07    MOV     R6,#tmp(0x07)
    16:             tmp=P1=tmp|(1<<i);
C:0x03F3    7401    MOV     A,#0x01
C:0x03F5    A806    MOV     R0,i(0x06)
C:0x03F7    08      INC     R0
C:0x03F8    8002    SJMP    C:03FC
C:0x03FA    C3      CLR     C
C:0x03FB    33      RLC     A
C:0x03FC    D8FC    DJNZ    R0,C:03FA
C:0x03FE    FD      MOV     R5,A
C:0x03FF    4F      ORL     A,R7
C:0x0400    F590    MOV     P1(0x90),A
C:0x0402    FF      MOV     R7,A
    17:             if (MENOR) P1=tmp&(~(1<<i));
C:0x0403    30B405  JNB     MENOR(0xB0.4),C:040B
C:0x0406    ED      MOV     A,R5
C:0x0407    F4      CPL     A
C:0x0408    5F      ANL     A,R7
C:0x0409    F590    MOV     P1(0x90),A
    18:         }
C:0x040B    1E      DEC     R6
C:0x040C    C3      CLR     C
C:0x040D    EE      MOV     A,R6
C:0x040E    6480    XRL     A,#P0(0x80)
C:0x0410    9480    SUBB    A,#P0(0x80)
C:0x0412    50DF    JNC     C:03F3
    19:         return  tmp;
```

El tiempo de conversión de esta rutina, dependerá del valor de la señal analógica V_{in}. No obstante, se pueden determinar el peor y mejor caso a ejecutar por la subrutina. El peor caso consistirá en un valor de V_{in} que obligue la ejecución de todas las instrucciones de la subrutina.

$$tpeor = 4 + (1+2+1+1+2+(2*8)+4+2+4+7)*8 = 324\ \mu seg$$

$$tmejor = 4 + 1+2+1+1+2+2+4+2+4+7 = 29\ \mu seg$$

Se puede optimizar en ensamblador la rutina anterior de la siguiente forma:

```
#pragma ASM
CONV:
      MOV   P1,#0       ;Borra puerto P1, (2 us)
      SETB  P1.7        ;Pone a 1 lógico el bit 7, (1us)
      JNB   P3.4,B1     ;Lee el estado de la comparación, (2us)
      CLR   P1.7        ;Borra el bit 7, (1us)
B1:   SETB  P1.6        ;Pone a 1 lógico el bit 6, (1us)
      JNB   P3.4,B2     ;Lee el estado de la comparación, (2us)
      CLR   P1.6        ;Borra el bit 6, (1us)
B2:   SETB  P1.5        ;Pone a 1 lógico el bit 5, (1us)
      JNB   P3.4,B3     ;Lee el estado de la comparación, (2us)
      CLR   P1.5        ;Borra el bit 5 , (1us)
B3:   SETB  P1.4        ;Pone a 1 lógico el bit 4, (1us)
      JNB   P3.4,B4     ;Lee el estado de la comparación, (2us)
      CLR   P1.4        ;Borra el bit 4, (1us)
B4:   SETB  P1.3        ;Pone a 1 lógico el bit 3, (1us)
      JNB   P3.4,B5     ;Lee el estado de la comparación, (2us)
      CLR   P1.3        ;Borra el bit 3, (1us)
B5:   SETB  P1.2        ;Pone a 1 lógico el bit 2, (1us)
      JNB   P3.4,B6     ;Lee el estado de la comparación, (2us)
      CLR   P1.2        ;Borra el bit 2, (1us)
B6:   SETB  P1.1        ;Pone a 1 lógico el bit 1, (1us)
      JNB   P3.4,B7     ;Lee el estado de la comparación, (2us)
      CLR   P1.1        ;Borra el bit 1, (1us)
B7:   SETB  P1.0        ;Pone a 1 lógico el bit O, (1us)
      JNB   P3.4,B8     ;Lee el estado de la comparación, (2us)
      CLR   P1.0        ;Borra el bit 0, (1us)
B8:   RET
#pragma ENDASM
```

Teniendo en cuenta un reloj de 12MHz para el microcontrolador, la subrutina tiene una primera instrucción que borra el puerto P1 y que tarda 2 p en ejecutarse. El resto de las instrucciones son una secuencia de puesta a 1 lógico, lectura del comparador y borrado, que tarda 4 p , es decir, 1 p para la puesta a 1 lógico, 2 p para la lectura del comparador y lps para el borrado. Esta secuencia se repite hasta ocho veces, por lo que el tiempo de conversión, en el peor de los casos, es:

$$tpeor = 2 + (1 + 2 + 1) \cdot 8 + 2 = 44\mu s$$

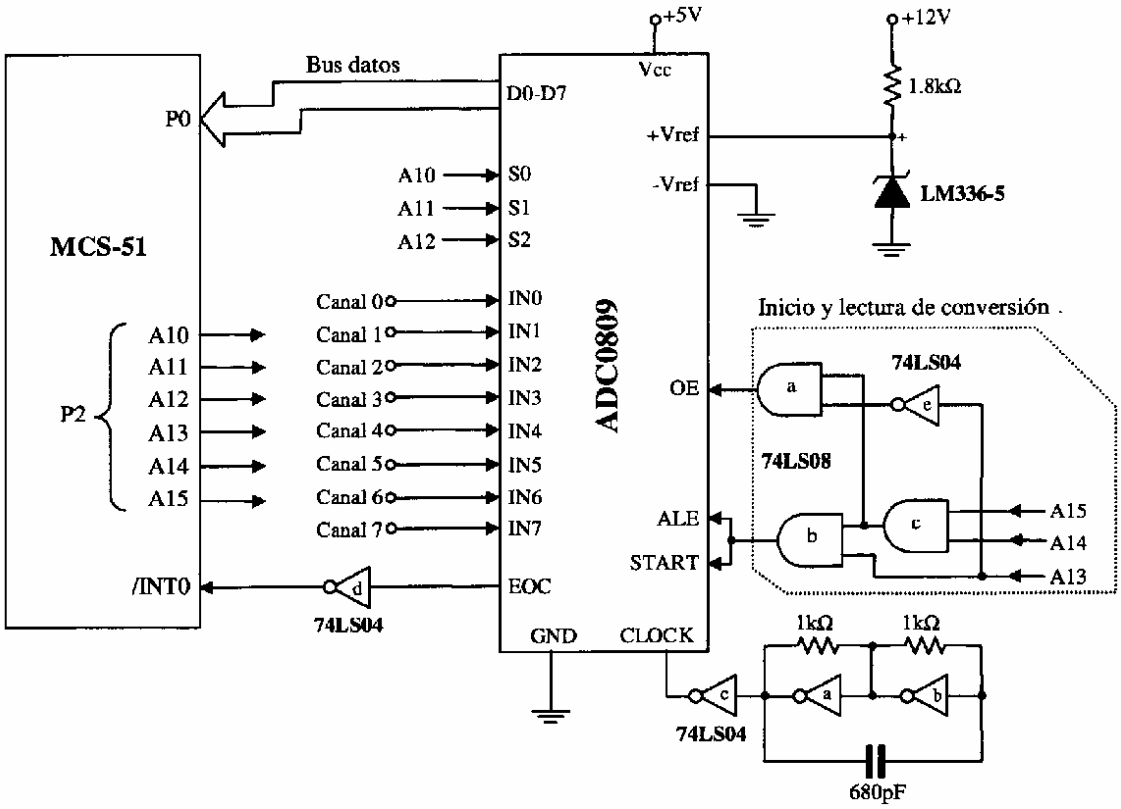
El mejor de los casos consiste en una señal V_{in} tal que no se ejecute ninguna de las instrucciones CLR, por lo que el tiempo de conversión es:

$$tmejor = 2 + (1 + 2) \cdot 8 + 2 = 28\mu s$$

En consecuencia, con la subrutina mostrada y el circuito de la figura anterior el tiempo de conversión estará comprendido entre 28 μs y 44 μs, claramente superior al mostrado cuando se codifica la rutina en lenguaje “C”.

4.- Conexión de un convertidor A/D.

La figura siguiente muestra la conexión entre el convertidor analógico/digital ADC0809 de 8 bits de National Instruments y un microcontrolador de la familia MCS-51. El ADC0809 puede leer la señal procedente de hasta 8 canales de entrada, seleccionables mediante tres líneas de dirección, S2-S0, que actúan sobre un multiplexor interno. Este convertidor está basado en la técnica de conversión por aproximaciones sucesivas.

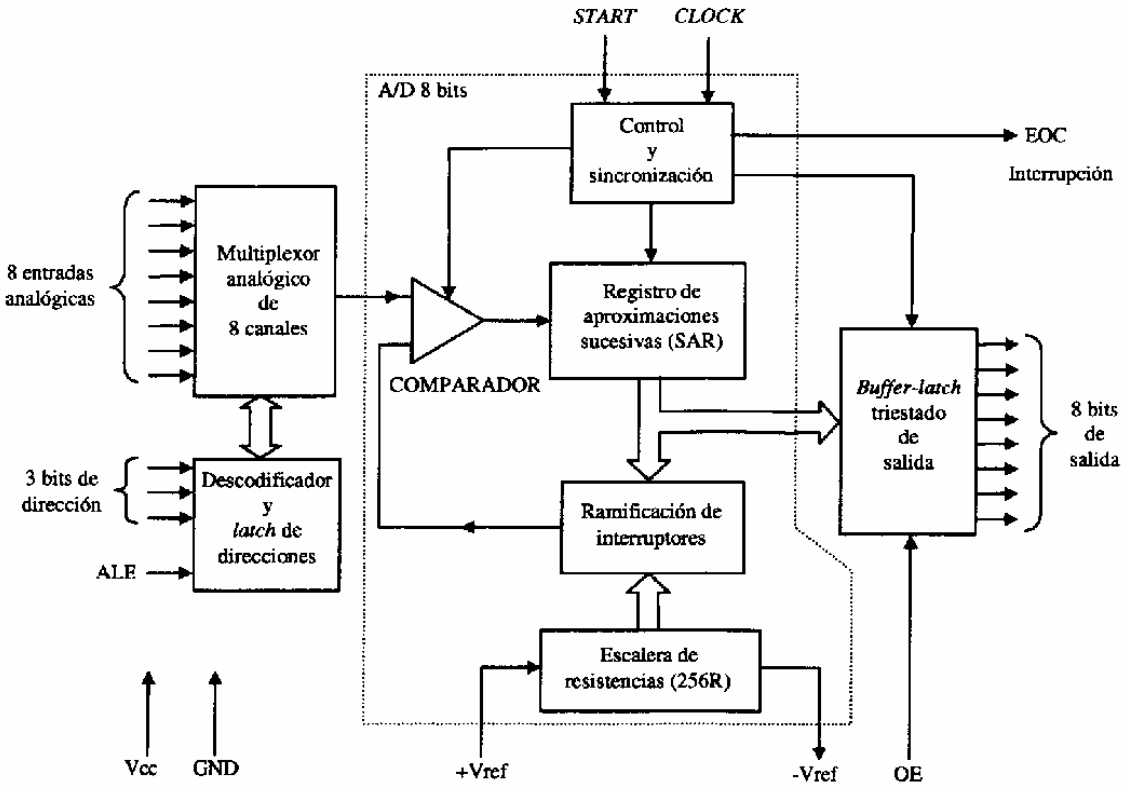


Las salidas del convertidor A/D son en triestado, D0-D7, por lo que se pueden conectar directamente al bus de datos de un sistema microprocesador. El convertidor precisa de una tensión de referencia externa de 5V y de una señal de reloj comprendida entre 10KHz y 1280kHz. El tiempo de conversión del convertidor es de 64 períodos de reloj desde el momento en que se inicia la conversión, por lo que para una frecuencia de 640kHz el tiempo de conversión es de unos l00 μs, aproximadamente, y de 50 μs para una frecuencia de 1280kHz.

La figura siguiente muestra el diagrama de bloques del convertidor, que está compuesto por un multiplexor analógico de 8 canales, un comparador, un registro de aproximaciones sucesivas (SAR), una escalera de resistencias R-2R y una serie de interruptores que actúan sobre la escalera de resistencias.

El proceso de conversión del convertidor empieza poniendo la línea de control ALE a 1 lógico, lo que hace que se carguen las direcciones S2-S0 en el latch interno de direcciones, seleccionando el canal analógico de entrada. Aplicando un pulso en la señal de START se inicia la conversión de la señal, se borran los registros internos del convertidor en el flanco de subida, y se inicia la conversión en el flanco de bajada. La señal EOC, "End of conversión", pasa a 0 lógico en los primeros 8 períodos de reloj desde el flanco de subida de la señal START, lo que indica el final de la conversión en el flanco de subida de EOC. Una vez producido este flanco, se puede leer el dato obtenido por la conversión, para lo cual debe aplicarse un pulso positivo a

la señal OE, "Ouput Enable", de forma que el dato se sitúe en las 8 líneas del bus de datos. Las salidas del convertidor permanecen en estado triestado hasta que se pone el dato convertido en el bus de datos.



Según la figura de conexionado con el μC, las salidas D0-D7 del convertidor se conectan directamente al bus de direcciones del microcontrolador, puerto P0. El inicio de la conversión se realiza aplicando un pulso en las líneas START y ALE del convertidor. Estas entradas están conectadas a las líneas A15, A14 y A13 del bus de direcciones, mediante dos puertas AND, de forma que estarán a 1 lógico sólo cuando A15, A14 y A13 estén a 1 lógico. Al mismo tiempo, las líneas A12, Al1 y A10 del bus de direcciones, están conectadas a las entradas de selección del canal analógico, I2, I1 y I0, respectivamente. De esta forma, el inicio de la conversión en el convertidor y la selección del canal analógico pueden efectuarse situando una dirección concreta en el bus de direcciones, que se corresponda con los rangos de direcciones de la tabla siguiente.

Rango de direcciones	Función
C000H-DFFFH	Lectura del dato convertido
E000H-E3FFH	Inicio conversión, selección canal 0
E400H-E7FFH	Inicio conversión, selección canal 1
E800H-EBFFH	Inicio conversión, selección canal 2
EC00H-EFFFH	Inicio conversión, selección canal 3
F000H-F3FFH	Inicio conversión, selección canal 4
F400H-F7FFH	Inicio conversión, selección canal 5
F800H-FBFFH	Inicio conversión, selección canal 6
FC00H-FFFFH	Inicio conversión, selección canal 7

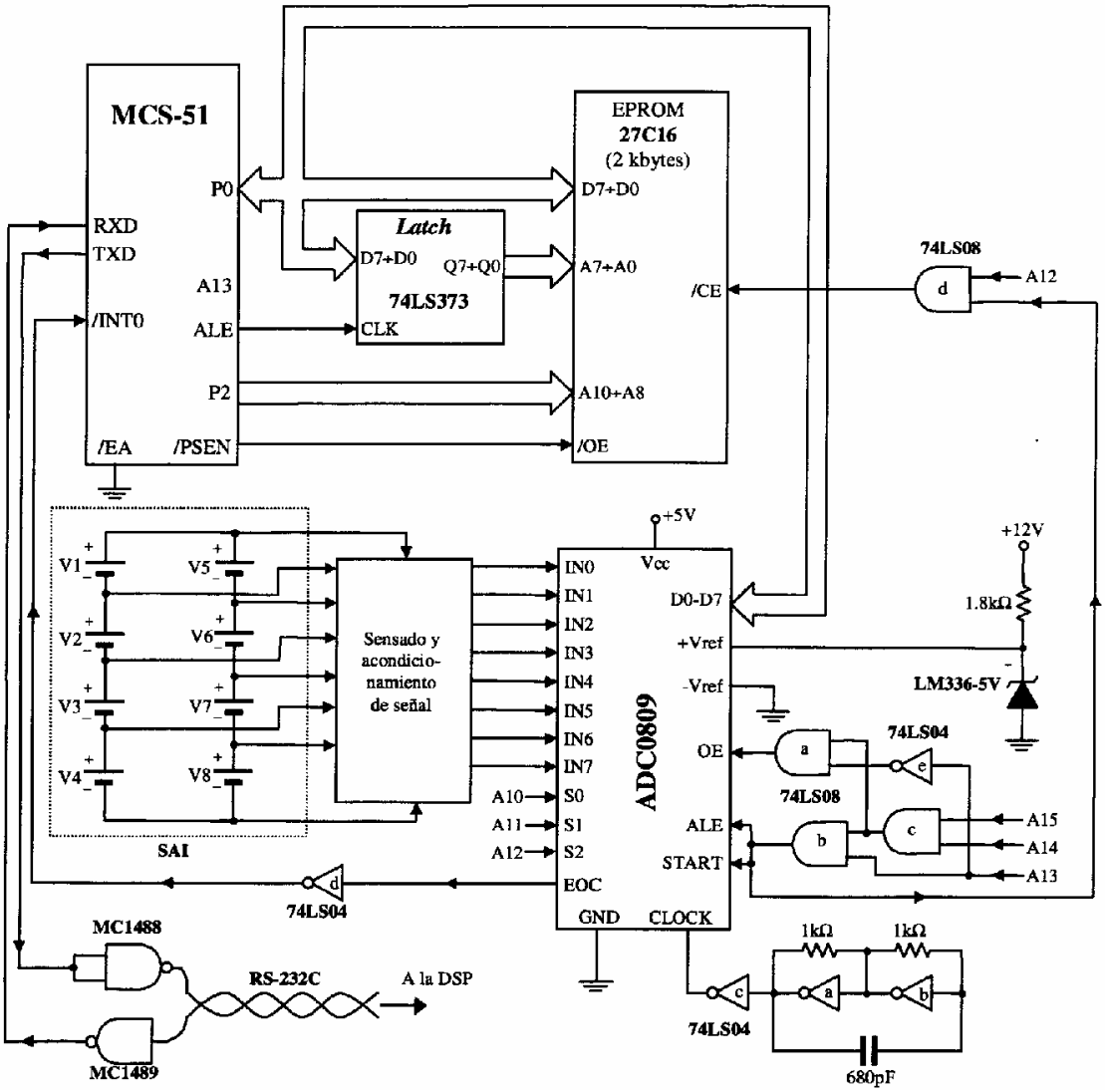
Según la tabla anterior, escribiendo cualquier dato en el bus de datos por medio de la instrucción MOVX y escogiendo el canal analógico de entrada con una de las direcciones de la tabla, puede iniciarse el proceso de conversión del convertidor. La señal EOC está conectada mediante una puerta inversora a la entrada de interrupción /INT0 del microcontrolador; en consecuencia, el flanco de subida en EOC, que indica el final de conversión del convertidor, aparece como un

flanco de bajada a la entrada de /INT0, que causará una interrupción si /INT0 está habilitada por flanco descendente.

La señal OE del convertidor está conectada también a dos puertas AND y a una puerta NOT, de manera que OE estará a 1 lógico si se sitúa una dirección con la combinación 1, 1 y O lógicos, en las líneas A15, A14 y A13, respectivamente. En definitiva, el dato convertido se puede leer, en la rutina de RSI de /INT0, ejecutando una lectura en cualquiera de las direcciones comprendidas en el rango C000H-DFFFH, mediante la instrucción MOVX.

Ejemplo 2 Monitorización del estado de las baterías de un SAI.

Los equipos de alimentación ininterrumpida, SAI, se utilizan para mantener la tensión de la red eléctrica en los equipos informáticos, en el caso de que se produzcan cortes repentinos de la red. Estos equipos tienen una serie de baterías que, en el caso de producirse un corte, aportan la energía necesaria para que, durante un intervalo de tiempo, sea posible cerrar los sistemas y guardar la información más importante. El estado de las baterías se debe comprobar regularmente para poder detectar anomalías en el proceso de carga y descarga, e indicar el momento en el cual una batería es defectuosa.



En este ejemplo se quiere utilizar un microcontrolador de la familia MCS51 para monitorizar el estado de las baterías, es decir, para leer su valor y proporcionárselo al sistema de control del SAI, que suele llevarse a cabo por medio de un procesador digital de señal DSP. Según la figura, el equipo SAI tiene un total de ocho baterías de 12V, que forman dos ramas de cuatro baterías cada una conectada en paralelo, y que entrega la energía necesaria para mantener la tensión de la red eléctrica en los equipos informáticos.

El microcontrolador debe leer la tensión de cada una de las baterías cuando la DSP se lo indique mediante la transmisión del carácter ASCII 05H vía RS-232C. Una vez finalizada la lectura de todos los canales del A/D, el microcontrolador debe transmitir a la DSP la lectura efectuada de cada batería vía RS-232C para ello debe transmitir el carácter 11H (XON), que se tomará por la DSP como carácter de inicio de la transmisión, los datos correspondientes a la lectura efectuada en cada batería y el carácter 13H (XOFF) de final de transmisión.

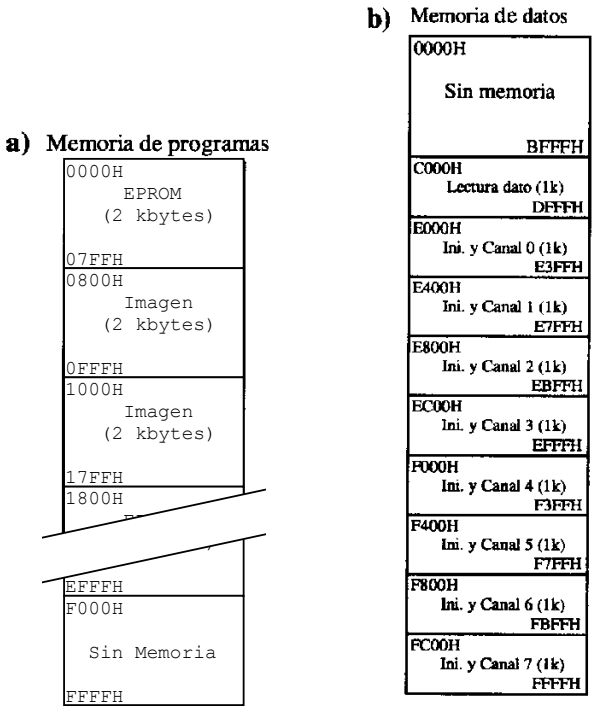
En el circuito de la figura tan sólo se utiliza una memoria EPROM de 2kbytes de capacidad para albergar el programa. No es necesario utilizar ningún circuito integrado de memoria RAM externa, pues se usa la memoria interna del microcontrolador. La memoria EPROM se selecciona mediante las líneas A12, A13, A14 y A15 del bus de direcciones, que están conectadas a una serie de puertas AND (figura), que hacen que la memoria esté seleccionada en el rango de direcciones comprendido entre 0000H y 0FFFH, o sea, en las primeras 4kdirecciones del mapa de memoria (figura); aparece, por tanto, una zona imagen en el rango [0800H-0FFFH] de 2kdirecciones. El convertidor A/D utiliza los mismos rangos de direcciones para el inicio de la conversión y para la selección del canal que los mostrados en la tabla.

La rutina de monitorización utiliza el registro R7 como indicador del final de conversión del convertidor A/D. El tiempo de conversión del convertidor es de unos 100 µs. La salida EOC del convertidor A/D (figura) está conectada a la entrada de interrupción /INT0 del microcontrolador a través de una puerta NOT. El final de la conversión se indica con un flanco de subida en EOC, que aparece en /INT0 como un flanco de bajada y provoca una interrupción. La rutina de RSI de /INT0 se limita a poner R7 al valor 0lH, indicando de esta manera que la conversión ha terminado.

De la misma forma que R7, el registro R6 se emplea por la rutina de RSI del puerto serie para indicar que se ha recibido, vía RS-232, el carácter ASCII 05H, que corresponde a la orden de lectura del A/D. La rutina de RSI, al recibir este carácter, pone el registro R6 al valor 01H, para que en la rutina principal se proceda a la lectura de todos los canales del convertidor A/D.

La rutina principal tiene un bucle de espera de recepción, de la orden de lectura del convertidor A/D, que está pendiente del valor del registro R6. Cuando la rutina detecta que R6 se ha puesto a 0lH, ésta pasa a leer cada uno de los canales del convertidor A/D. Para ello, inicia la conversión mediante la escritura con MOVX de cualquier dato en una de las direcciones de inicio y de selección de canal que aparece en la tabla anterior. Luego, la rutina pasa a la espera del final de la conversión mediante un bucle que examina continuamente el estado del registro R7. Finalmente, la rutina pasa a leer el dato obtenido por el convertidor efectuando una lectura en la dirección C000H con la instrucción MOVX. Terminada la lectura de todos los canales, la rutina principal pasa a transmitir cada uno de los datos obtenidos a la DSP vía RS-232C.

La velocidad de transmisión de los datos es de 9.600 baudios; para ello se realiza la comunicación asíncrona del puerto serie en el modo 1, en el que se transmiten 10 bits: 1 bit de start, 8 bits del dato y 1 bit de stop. El Timer 1 del microcontrolador se utiliza como base de tiempos para la transmisión, por lo que debe estar configurado en el modo 2 de 8 bits con autorrecarga, con el bit C/T a O lógico, con el valor FDH de recarga en el registro THl y con 11.059MHz de frecuencia de reloj.




```
/*****************************************************************
  Monitorización de las baterías de un SAI para la MCS-51
 *****************************************************************/

/* Registros de uso general y funciones E/S */
#include <reg51.h>
#include <stdio.h>
#include <absacc.h>
/* unsigned char */
#define uchar unsigned char
/* Posiciones de memoria de datos externa XDATA de registros de E/S */
#define Conv_Canal_0      XBYTE [0x0E000]  /* inicio y selección 0 */
#define Conv_Canal_1      XBYTE [0x0E400]  /* inicio y selección 1 */
#define Conv_Canal_2      XBYTE [0x0E800]  /* inicio y selección 2 */
#define Conv_Canal_3      XBYTE [0x0EC00]  /* inicio y selección 3 */
#define Conv_Canal_4      XBYTE [0x0F000]  /* inicio y selección 4 */
#define Conv_Canal_5      XBYTE [0x0F400]  /* inicio y selección 5 */
#define Conv_Canal_6      XBYTE [0x0F800]  /* inicio y selección 6 */
#define Conv_Canal_7      XBYTE [0x0FC00]  /* inicio y selección 7 */
#define Lectura_AD        XBYTE [0x0C000]  /* Lectura del A/D */
/* Definicion de banderas de datos globalmente accesibles */
bit          Inicio_Conversion,Fin_Conversion;      /* banderas de estado */
/* Constantes */
#define XON  0x11
#define XOFF 0x13
#define ORDEN_LECTURA 'a'

/*****************************************************************
  Rutina de servicio de INT0
 *****************************************************************/
void int0_int (void) interrupt 0 using 1
{
      Fin_Conversion=1;         /* para indicar fin de conversion */
}

/*****************************************************************/
/* Rutina de servicio del puerto serie  */
/*****************************************************************/
serial_int () interrupt 4 {
uchar tmp;
```

```
if (RI) {                   /* Si ha sido recepción */
       tmp=SBUF;
       if (tmp==ORDEN_LECTURA) Inicio_Conversion=1;
       RI=0;
  }
}

/*****************************************************************
  Subrutinas
 *****************************************************************/

void Transmite_Datos(uchar p1,uchar p2, uchar p3,
                     uchar p4, uchar p5, uchar p6, uchar p7,uchar p8)
{
       uchar code molde[]={"%u-%u"};

       putchar(XON);
       printf(molde,p1,p2);
       putchar('-');
       printf(molde,p3,p4);
       putchar('-');
       printf(molde,p5,p6);
       putchar('-');
       printf(molde,p7,p8);
       putchar(XOFF);
}

void Lectura_Conversor(void)
{
uchar  ca0,ca1,ca2,ca3,ca4,ca5,ca6,ca7; /* almacena lecturas conversores */

       Conv_Canal_0=0; /* Inicia la conversión escribiendo en la dirección */
       while (!Fin_Conversion);  /* Espera fin de conversión */
       Fin_Conversion=0;
       ca0=Lectura_AD;                          /* Guarda en memoria interna */
       Conv_Canal_1=0; /* Inicia la conversión escribiendo en la dirección */
       while (!Fin_Conversion);  /* Espera fin de conversión */
       Fin_Conversion=0;
       ca1=Lectura_AD;                          /* Guarda en memoria interna */
       Conv_Canal_2=0; /* Inicia la conversión escribiendo en la dirección */
       while (!Fin_Conversion);  /* Espera fin de conversión */
       Fin_Conversion=0;
       ca2=Lectura_AD;                          /* Guarda en memoria interna */
       Conv_Canal_3=0; /* Inicia la conversión escribiendo en la dirección */
       while (!Fin_Conversion);  /* Espera fin de conversión */
       Fin_Conversion=0;
       ca3=Lectura_AD;                          /* Guarda en memoria interna */
       Conv_Canal_4=0; /* Inicia la conversión escribiendo en la dirección */
       while (!Fin_Conversion);  /* Espera fin de conversión */
       Fin_Conversion=0;
       ca4=Lectura_AD;                          /* Guarda en memoria interna */
       Conv_Canal_5=0; /* Inicia la conversión escribiendo en la dirección */
       while (!Fin_Conversion);  /* Espera fin de conversión */
       Fin_Conversion=0;
       ca5=Lectura_AD;                          /* Guarda en memoria interna */
       Conv_Canal_6=0; /* Inicia la conversión escribiendo en la dirección */
       while (!Fin_Conversion);  /* Espera fin de conversión */
       Fin_Conversion=0;
       ca6=Lectura_AD;                          /* Guarda en memoria interna */
       Conv_Canal_7=0; /* Inicia la conversión escribiendo en la dirección */
       while (!Fin_Conversion);  /* Espera fin de conversión */
       Fin_Conversion=0;
       ca7=Lectura_AD;                          /* Guarda en memoria interna */
       Transmite_Datos(ca0,ca1,ca2,ca3,ca4,ca5,ca6,ca7);
}

/*****************************************************************
 Rutina de inicio
```

```
*****************************************************************/
void main(void) {

       IT0=1; /* Interrupción INT0 por flanco descendente */
       EX0=1; /* Habilita interrupción /INT0 */
       ES=1;  /* Habilita interrupción puerto serie */
       SM1=1; /* Configura comunicación serie en modo 1 */
       REN=1; /* Habilita recepción por el puerto serie */
       TMOD=0x20;    /* Timer 1 en Modo 2, GATE=0 y C/T=0 */
       TL1=0x0FD;    /* Valor de TL1, velocidad de 9600 baudios */
       TH1=0x0FD;    /* Valor recarga Timer 1, 9600 baudios */
       EA=1;         /* Habilita bit de interrupción general */
       TR1=1;        /* Pone marcha el Timer 1 */
       TI=1;         /* Habilita comienzo de transmisión */
/*****************************************************************
  Rutina Principal
  *****************************************************************/
       Inicio_Conversion=Fin_Conversion=0;
  while (1) {

       while (!Inicio_Conversion);      /* Bucle espera peticion de dato */
       Inicio_Conversion=0;
       Lectura_Conversor();             /* Realiza la lectura del A/D */
       }
}
```