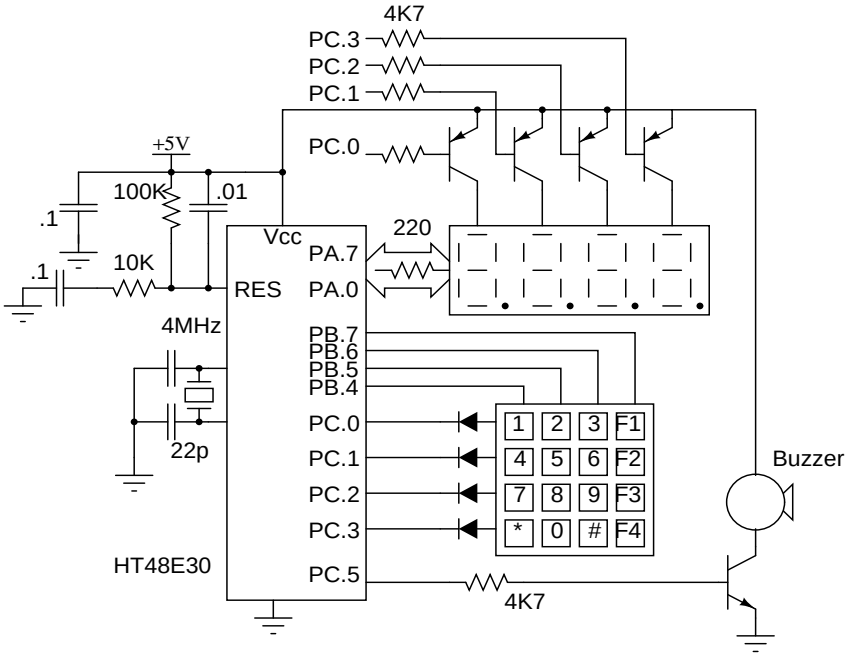
Nota de Aplicación: CAN-079
Título: **Manejo de displays de 7 segmentos con Holtek**
Autor: Sergio R. Caprile, Senior Engineer

Revisiones	Fecha	Comentarios
0	23/03/07	
1	02/07/07	optimización de findk()

La presente nota de aplicación desarrolla un esquema de control para displays de 7 segmentos, basada en la serie HT48E de Holtek. Todo el multiplexado es desarrollado por el micro, mediante software, con soporte para realizar parpadeo individual de los dígitos.
El código desarrollado es assembler, pero se proveen ejemplos de uso del mismo tanto desde C como desde assembler. Aquellos lectores no interesados en el funcionamiento interno del código, pueden saltear dicha parte y acceder directamente a los ejemplos de uso. Para una explicación detallada de los requisitos a cumplir para utilizar C y assembler simultáneamente, se recomienda la lectura del tutorial CTU-010.

Hardware

Utilizaremos el port A completo para los segmentos, y algunos pines del port C para la selección de los dígitos. La parte alta del port B la utilizaremos para retorno del teclado, el cual escaneamos aprovechando el multiplexado de los displays; el pull-up interno nos polariza convenientemente las entradas. El dígito más significativo del display puede opcionalmente reemplazarse por LEDs individuales.



Software

La rutina básica de multiplexado, que funciona como interrupt handler del timer es la siguiente:

En la misma, hemos tenido cuidado de no utilizar llamadas a subrutinas, de modo de permitir la utilización del esquema aún en los micros más chicos. Si el dígito más significativo ha sido reemplazado por LEDs individuales, el software saltea la conversión a 7 segmentos.

```
code7   .section INPAGE 'CODE'
T0HND:          mov softstack,A                 ; salva A
                mov A,STATUS
                mov softstack[1],A              ; salva STATUS
                mov A,MP0
                mov softstack[2],A              ; salva MP0
                mov A,TBLP
                mov softstack[3],A              ; salva MP0
                mov A,DIGSOFF                   ; desactiva scan lines
        IFDEF COMMONANODE
                cpl ACC
                orm A,dig_data
        ELSE
                andm A,dig_data
        ENDIF

segoff:                                         ; apaga segmentos
        IFDEF COMMONANODE
                set seg_data
        ELSE
                clr seg_data
        ENDIF
                snz _DISP_F.BEEP                ; beep ?
                jmp NoBeep
                sz _DISP_F.BLNKNG
                set BUZZER                      ; enciende buzzer en período de on
                snz _DISP_F.BLNKNG
                clr BUZZER                      ; apaga en período de off
NoBeep:         snz _DISP_F.DISPON
                jmp NoShow                      ; saltea si el display está pagado
                mov A,OFFSET _CGRAM             ; CGRAM base
                dec ACC                         ; -1 (offset 1..n)
                add A,CGIDX                     ; + offset
                mov MP0,A                       ; indirecto
                mov A,IAR0                      ; lee caracter
                mov tmp,A
                snz ACC.7                       ; maneja parpadeo
                jmp Noblink
                snz _DISP_F.BLNKNG              ; saltea en período de off
                jmp NoShow
Noblink:
        IFDEF USELEDS
                sz DIGIT.0                      ; saltea conversión a 7seg si LEDs (PC.0)
                jmp nodecode
        ENDIF
                and A,03Fh                      ; elimina bits de blink y dp
                add A,OFFSET BBCD27S            ; puntero tabla 7seg
                mov TBLP,A                      ; indirecto
                tabrdc ACC                      ; lee código 7seg
                sz tmp.6                        ; maneja dp
                set ACC.dp
nodecode:
        IFDEF COMMONANODE
                cpl ACC
        ENDIF
                mov seg_data,A                  ; muestra
NoShow:         mov A,DIGIT
        IFDEF COMMONANODE
                cpl ACC
                andm A,dig_data
        ELSE
                orm A,dig_data
        ENDIF
                mov A,kbd_data                  ; lee líneas de teclado
        IFDEF COMMONANODE
                cpl ACC
        ENDIF
                and A,0F0h                      ; nibble alto
                sz Z                            ; saltea si no hay ninguna activa
                jmp nokey
                or A,DIGIT                      ; agrega líneas de scan, forma código
                orm A,_LAST_K[1]                ; guarda provisorio
nokey:          set _DISP_F.DSYNC
                sdz CGIDX                       ; más dígitos ?
```

```
                jmp nofrs
; Frame sync
                mov A,_LAST_K[1]                    ; fin de cuadro
                mov _LAST_K,A                       ; actualiza valor de LAST_K
                clr _LAST_K[1]
                mov A,DIGITS                        ; resetea barrido
                mov CGIDX,A
                mov A,LMOST
                mov DIGIT,A
                set _DISP_F.FSYNC
                sdz ANIMTMR                         ; maneja sync animación (cuenta cuadros)
                jmp DMF_l1
                mov A,ANIMTM
                mov ANIMTMR,A
                set _DISP_F.ASYNC
DMF_l1:         sdz BLNKTMR                         ; maneja parpadeo (cuenta cuadros)
                jmp DMF_dn
                mov A,BLNKTM
                mov BLNKTMR,A
                mov A,BLNKMSK
                xorm A,_DISP_F
DMF_dn:         jmp DispDn
nofrs:          rrc DIGIT                           ; siguiente dígito
DispDn:
                mov A,softstack[3]
                mov TBLP,A                          ; recupera MP0
                mov A,softstack[2]
                mov MP0,A                           ; recupera MP0
                mov A,softstack[1]
                mov STATUS,A                        ; recupera STATUS
                mov A,softstack                     ; recupera A
                reti


BBCD27S: DW 03Fh,006h,05Bh,04Fh,066h,06Dh,07Dh,007h
        DW 07Fh,06Fh,07Ch,05Eh,054h,071h,079h,07Dh
        DW 073h,040h,008h,004h,000h,038h,001h,077h
        DW 050h,076h,030h,006h,03Eh

;Charset: 0-9
;10=b; 11=d; 12=n; 13=F; 14=E; 15=G; 16=P
;17=-; 18=_; 19=i; 20=Blank; 21=L; 22=-(up)
;23=A; 24=r; 25=H; 26=|(left); 27=|; 28=U
```

Las rutinas de inicialización y borrado:

```
_init7seg:
                clr _DISP_F                         ; borra flags
                mov A,BLNKTM                        ; inicializa contadores de cuadros
                mov BLNKTMR,A                       ; para parpadeo
                mov A,ANIMTM+1
                mov ANIMTMR,A                       ; y animación
                mov A,DIGITS
                mov CGIDX,A
                mov A,LMOST                         ; inicia barrido desde MSD
                mov DIGIT,A
                mov A,178                           ; 64us*78=4,992ms => 256-78=178
                mov TMR,A
                mov A,10010111b                     ; timer, prescaler=256 (4MHz/256=>64us)
                mov TMRC,A
_cls7seg:
                mov A,20                            ; código de blanco
                mov _CGRAM,A
                mov _CGRAM[1],A
                mov _CGRAM[2],A
                mov _CGRAM[3],A
                ret
```

Para traducir el código de scan a un valor numérico más fácilmente utilizable, buscamos el código en una tabla, el offset dentro de la tabla es el valor buscado:

```
code7k  .section INPAGE 'CODE'
_findk:         mov A,OFFSET tabk                   ; tabla de códigos de scan
```

```
                mov TBLP,A                  ; indirecto
                mov A,16                    ; largo de la tabla
                mov Klctr,A
fkl:            tabrdc ACC                  ; lee código y guarda en A
                xor A,findk0                ; compara con tecla en cuestión
                sz Z
                jmp kf                      ; sí!
                inc TBLP                    ; no, siguiente código
                sdz Klctr                   ; terminé ?
                jmp fkl
                ret A,0FFh                  ; sí, múltiples teclas o ya no presionada
kf:             mov A,TBLP                  ; obtiene offset: K0 => 0, etc.
                sub A,OFFSET tabk
                ret

tabk:   DW K0,K1,K2,K3,K4,K5,K6,K7,K8,K9,KF1,KF2,KF3,KF4,KA,KN
```

Esto tiene además la ventaja de ignorar códigos de scan de múltiples teclas simultáneamente.

Utilización desde C

Incluimos un header file, 7seg.h, de modo que el compilador pueda chequear la sintaxis y el usuario conozca las funciones que debe llamar.

Inicialización

Para inicializar el display, realizamos lo siguiente:

```
_pc=0;
_pac=0;                             // PA0-7 = outputs
_pbc=0xFF;                          // PB0-7 = inputs
_pbc&=~(1<<0);                      // PB1 = output
_pcc=0;                             // PC0-7 = outputs

INIT7SEG();

_intc|=(1<<0)+(1<<2);               // habilita int del timer y en gral
```

Despliegue de información

Podemos borrar el display llamando a la rutina correspondiente:

```
CLS7SEG();
```

Para mostrar un dígito (o caracter), debemos escribir en la posición de memoria que le corresponde, dentro del array *CGRAM*. La primera posición corresponde al dígito más significativo; el bit más significativo indica parpadeo, y el que le sigue (bit6) controla el punto decimal:

```
CGRAM[0]=(1+0x40);                  // muestra 1.234
CGRAM[1]=(2);
CGRAM[2]=(3+0x80);                  // el '3' parpadea
CGRAM[3]=4;
```

Para que el display esté encendido, seteamos el bit de control en la variable *DISP_F*:

```
DISP_F|=(1<<DISPON);
```

Para que haya un beep sincronizado con el parpadeo, seteamos su bit de control en la variable *DISP_F*:

```
DISP_F|=(1<<BEEP);
```

Obviamente, también es posible controlar de forma manual el buzzer, si no se setea dicho bit, y es conveniente apagarlo manualmente luego de inhibir el beep:

```
DISP_F&=~(1<<BEEP);
BUZZER=0;
```

Si se desea realizar alguna animación, se requiere sincronía con la actualización del display, o se necesita manejar software timers, se pueden observar los flags correspondientes:

```
DSYNC   se setea a cada cambio de dígito
FSYNC   se setea cuando se completan todos los dígitos (un 'cuadro')
ASYNC   se setea cuando se completa una cantidad de cuadros
```

Dichos flags son seteados por la rutina de multiplexado, debiendo ser reseteados por el usuario.
El ejemplo a continuación incrementa uno de los dígitos a un ritmo controlado por el timer de animación:

```
while(1) {
       DISP_F&=~(1<<ASYNC);          // borra flag anterior
       while(!(DISP_F&(1<<ASYNC)));  // espera nuevo
       if(++CGRAM[3]==10)            // realiza tarea
              CGRAM[3]=0;
}
```

Lectura del teclado
Para leer el teclado, disponemos del código de scan en la variable *LAST_K*. Para convertirlo a un código numérico correlativo, más manejable, podemos utilizar la función *FINDK()*:

```
key=FINDK(LAST_K);                   // 0xFF => ninguna o muchas, 0=>0, etc
```

Utilización desde assembler

Inicialización
Para inicializar el display, realizamos lo siguiente:

```
clr PC
clr PAC                              ; PA0-7 = outputs
set PBC                              ; PB0-7 = inputs
clr PBC.1                            ; PB1 = output
clr PCC                              ; PC0-7 = outputs
set PGC                              ; PG0 = input

call _init7seg

set INTC.2                           ; habilita int del timer
set INTC.0                           ; habilita interrupciones
```

Despliegue de información
Podemos borrar el display llamando a la rutina correspondiente:

```
call _cls7seg
```

Para mostrar un dígito (o caracter), debemos escribir en la posición de memoria que le corresponde, dentro del array *_CGRAM*. La primera posición corresponde al dígito más significativo; el bit más significativo indica parpadeo, y el que le sigue (bit6) controla el punto decimal:

```
mov A,(1+040h)                       ; muestra 1.234
mov _CGRAM[0],A
mov A,(2)
mov _CGRAM[1],A
mov A,(3+080h)                       ; el '3' parpadea
mov _CGRAM[2],A
mov A,(4)
mov _CGRAM[3],A
```

Para que el display esté encendido, seteamos el bit de control en la variable *_DISP_F*:

```
set _DISP_F.DISPON
```

Para que haya un beep sincronizado con el parpadeo, seteamos su bit de control en la variable *_DISP_F*:

```
set _DISP_F.BEEP
```

Obviamente, también es posible controlar de forma manual el buzzer, si no se setea dicho bit, y es conveniente apagarlo manualmente luego de inhibir el beep:

```
clr _DISP_F.BEEP
```

```
                clr BUZZER
```

Si se desea realizar alguna animación, se requiere sincronía con la actualización del display, o se necesita manejar software timers, se pueden observar los flags correspondientes:

```
DSYNC   se setea a cada cambio de dígito
FSYNC   se setea cuando se completan todos los dígitos (un 'cuadro')
ASYNC   se setea cuando se completa una cantidad de cuadros
```

Dichos flags son seteados por la rutina de multiplexado, debiendo ser reseteados por el usuario.
El ejemplo a continuación incrementa uno de los dígitos a un ritmo controlado por el timer de animación:

```
lp:             clr _DISP_F.ASYNC                   ; borra flag anterior
lp1:            snz _DISP_F.ASYNC                   ; espera nuevo
                jmp lp1
                inc _CGRAM[3]                       ; realiza tarea
                mov A,_CGRAM[3]
                xor A,10
                clr ACC
                sz Z
                mov _CGRAM[3],A
                jmp lp
```

Lectura del teclado
Para leer el teclado, disponemos del código de scan en la variable *_LAST_K*. Para convertirlo a un código numérico correlativo, más manejable, podemos utilizar la subrutina *_findk*, a la que le pasamos el código de scan como parámetro en *findk0* y devuelve el resultado en A:

```
                mov A,_LAST_K                       ; lee teclado
                mov findk0,A                        ; guarda como parámetro
                call _findk                         ; devuelve 0 si está el 0 presionado, etc
                                                    ; 0xFF si no hay ninguna (o demasiadas)
                                                    ; tecla(s) apretada(s)
```