Alarma con Interfaz DTMF de Línea Telefónica basado en PIC16F84

Propósito

En toda aplicación donde normalmente se requiera el establecimiento de una conexión telefónica se hace necesario el diseño de una interfaz de línea telefónica. En este proyecto se usa el discado por tono ó DTMF. La generación de los tonos DTMF se realiza por software programando el microcontrolador PIC16F84. La interfaz esta destinada a usos en sistemas de alarmas domésticos en donde la activación de la alarma produce la llamada a teléfonos previamente establecidos. Esta interfaz puede ser usada en conjunto con un módulo reproductor-grabador de mensajes a fin de que el mensaje previamente grabado sea escuchado en el otro extremo de la llamada o se puede programar en el mismo microcontrolador la emisión de tonos de diferentes frecuencias e intervalos que puedan identificar el tipo de alarma activada.

Teoría de operación

La codificación DTMF se muestra en la tabla 1. A cada tecla del teléfono le corresponde la emisión de dos frecuencias: una denominada baja y otra denominada alta. La suma de estas dos frecuencia produce el tono DTMF correspondiente.

Tecla	Frecuencia
1	697+1209
2	697+1336
3	697+1477
4	770+1209
5	770+1336
6	770+1477
7	852+1209
8	852+1336
9	852+1477
0	941+1336
*	941+1209
#	941+1477
A	697+1633
B	770+1633
C	852+1633
D	941+1633

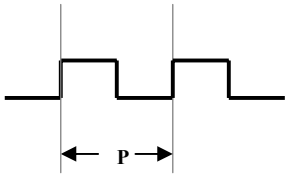
Tabla 1 Codificación DTMF para Teclado telefónico

La técnica utilizada para la generación de cada frecuencia es la modulación de ancho de pulso (PWM). Un pin del microcontrolador se utiliza como salida de frecuencias bajas y otro como salida de frecuencias altas. A la salida de cada pin se colocan sendos filtros pasabajos cuyas salidas son unidas a fin de sumar ambas frecuencias y producir el tono DTMF.

Generación de la Frecuencia deseada

Para la generación de una frecuencia determinada utilizamos un lazo de retardo el cual es repetido n veces. El lazo de retardo es producido por la ejecución de instrucciones cuya duración es de un ciclo de reloj, excepto las de salto que duran dos. Cada vez que el lazo se ejecute n veces, el pin correspondiente cambiará de estado produciéndose así medio periodo. Variando el valor de n podemos obtener la frecuencia deseada de acuerdo a la siguiente ecuación:

$$p = 1/f \qquad f = \left(\frac{1}{2 * t * n * T} \right)$$



donde:
p = periodo
f = frecuencia
t = número de ciclos de reloj del lazo
n = número de veces que se repite el lazo
T = Tiempo ciclo de instrucción (1useg @ 4Mhz)

En este caso se utilizan 18 ciclos de reloj en el lazo, así para generar la frecuencia 694.4444444 Hz, necesitamos repetir el lazo 40 veces.

$$f = \left(\frac{1}{2 * 18 * 40 * 1x10^{-6}} \right) = 694.44444Hz$$

La siguiente tabla muestra el resumen de los cálculos realizados para cada frecuencia

Tono	N	Tono Generado	Error	%
697	40	694.4444444	2.56	0.37
770	36	771.6049383	1.60	0.21
852	32	868.0555555	16.05	1.88
941	29	957.8544061	16.85	1.79
1209	23	1207.7294690	1.27	0.11
1336	21	1322.7513230	13.25	0.99
1477	19	1461.9883040	15.01	1.02
1633	17	1633.9869280	0.99	0.06

Descripción del Proyecto Propuesto

Al activarse cualquiera de las entradas de la alarma, (según su configuración), el sistema realiza llamadas a números telefónicos (máximo 3), previamente grabados en la EEPROM del microcontrolador, tantas veces como se indique, utilizando discado por tonos DTMF. Los números telefónicos son grabados utilizando la interfaz serial y un programa escrito en Visual Basic. Cada vez que el circuito es conectado a la alimentación, se enciende un LED por espacio de 10 segundos, indicando el instante en que pueden ser enviados los datos a grabar en la EEPROM. Luego de ello, el sistema queda revisando continuamente cada una de las entradas, al activarse alguna, se produce todo el proceso de discado el número de veces indicado. Luego de efectuar cada llamada, el sistema envía un bitono durante aproximadamente 20 seg.

La Memoria EEPROM de Datos

Los PIC16X8X tienen 64 bytes de memoria EEPROM de datos en donde se puede guardar datos y variables que no se pierden al desconectar la alimentación. En esta memoria almacenaremos los números telefónicos a discar así como el número de veces que deben ser discados.

La memoria EEPROM no está mapeada en la zona de memoria del Banco de Registros específicos (SFR) y el Banco de Registros de Propósito General (GPR). Para poder leerla y escribirla durante el funcionamiento normal del microcontrolador, es necesario utilizar 4 registros del banco SFR:

Registro	Dirección
EEDATA	0x8 Banco 0
EEADR	0x9 Banco 0
EECON1	0x8 Banco 1 o 88H
EECON2	0x9 Banco 1 o 89H No implementado físicamente

El registro EEADR es la dirección de la memoria EEPROM (0x00 a 0x3F) que será leída o escrita. El registro EEDATA contendrá el dato leído o el dato que se va a escribir. El registro EECON1 tiene misión de control de las operaciones tal como se describe a continuación:

EECON1

			EEIF	WRERR	WREN	WR	RD

RD: Lectura, Se pone a 1 cuando se va a realizar un ciclo de lectura, luego se pone a cero automáticamente.

WR: Se pone a 1 cuando se inicia el ciclo de escritura de la EEPROM. Cuando se completa el ciclo pasa automáticamente a cero.

WREN: Permiso de escritura. Cuando se pone a 1 se permite la escritura de la EEPROM. Cero prohibe la escritura en la EEPROM.

WRERR: Señalizador de error en la escritura. Se pone a 1 cuando una operación de escritura ha terminado prematuramente. Cero si la operación fue exitosa.

EEIF: Señalizador de fin de operación de escritura. Se pone a 1 cuando la operación de escritura es exitosa. Debe ser puesto a cero por programa. Se mantiene cero mientras la operación de escritura no se ha completado.

Una operación de escritura en la EEPROM puede tardar típicamente 10ms, que es un tiempo grande en comparación con la velocidad del microcontrolador, de tal manera que el bit EEIF debe ser verificado en un ciclo de espera hasta que pase a 1. El bit EEIF, esta conectado con el bit EEIE del registro de interrupciones INTCON y a su vez con el bit GIE, de tal manera que al terminar el ciclo de escritura, se produce una interrupción que despierta al microcontrolador luego de una instrucción `sleep`. El uso de interrupciones es más conveniente en este caso.

El registro EECON2 no esta implementado físicamente. Al leerlo todos sus bits son cero. Sólo se emplea como dispositivo de seguridad durante el proceso de escritura de la EEPROM.

Proceso de lectura

El programa siguiente muestra a manera de ejemplo el proceso de lectura de una posición de memoria EEPROM.

```
bcf     status,rpo      ;Banco 0
movlw   address         ;dirección en address
movwf   eeadr           ;dirección en registro eeadr
bsf     status,rp0      ;Banco 1
bsf     eecon1,rd       ;Lectura rd = 1
bcf     status,rp0      ;Banco 0
movf    eedata,w        ;W se carga con valor leido en EEPROM
```

Proceso de Escritura

El ciclo de escritura comienza cargando en EEADR la dirección de la posición de memoria EEPROM a escribir y en EEDATA el valor a grabar. Para escribir una posición de memoria EEPROM, el usuario debe seguir una secuencia determinada de instrucciones donde participa el registro EECON2. En este registro debe colocarse secuencialmente los valores 0x55 y 0xAA. El programa a continuación graba un dato en memoria EEPROM, donde se supone que previamente fueron cargados los registros EEADR y EEDATA. Se utiliza interrupciones para la espera de la culminación del ciclo de escritura.

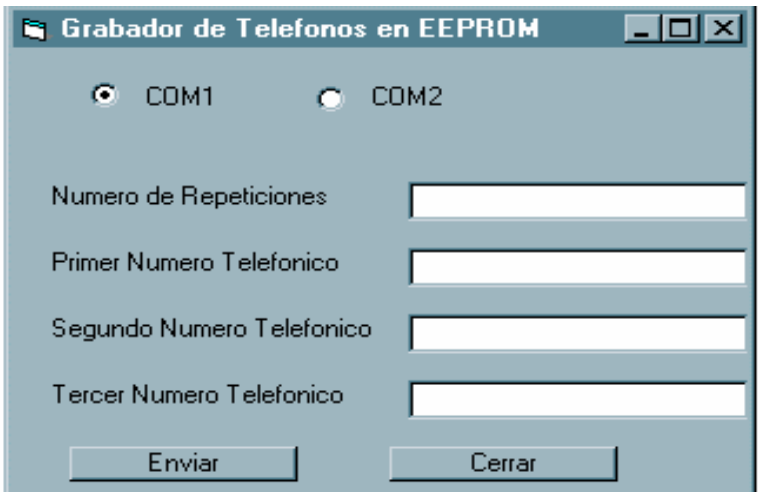
```
bsf STATUS,5          ; Banco 1
bcf INTCON, GIE       ; Desabilita las interrupciones
bsf EECON1,WREN       ; Permiso para escribir en EEPROM
movlw  0x55
movwf  EECON2
movlw  0xAA
movwf  EECON2
bsf EECON1,WR         ; Comienza ciclo de escritura


bsf INTCON,EEIE       ; Habilita interrupcion fin escritura EEPROM
sleep                 ; Espera el fin de escritura en la EEPROM
bsf INTCON, GIE       ; Habilita interrupciones
bcf EECON1,EEIF       ; limpia el flag de interrupcion EEPROM
bcf EECON1,WREN       ; Prohibe escritura en EEPROM
bcf STATUS,5          ; Banco 0
```

Todo lo anterior se facilita utilizando unas pocas instrucciones del compilador C, PCM de CCS, como veremos en el Software desarrollado.

Programa Grabador de Teléfonos en EEPROM

Este programa nos permite grabar en la EEPROM del microcontrolador 16F84 hasta tres números telefónicos los cuales serán discados secuencialmente tantas veces como lo indique el número de repeticiones. Este programa fue elaborado utilizando Visual Basic 6 y el control MSComm, para la comunicación serial. Esta configurado a 9600bits/s, 8 bit, no paridad y un bit de parada. No se contempla control de flujo.



Protocolo de Comunicación PIC-PC

Como se mencionó, para grabar los números de teléfonos en la EEPROM del microcontrolador, se utiliza la interfaz serial y el programa en Visual Basic descrito. El protocolo prevé el envío de una sola serie de datos, durante los 10 seg. siguientes al encendido, que contiene la siguiente información:

NrSepTelf1SepTelf2SepTelf3Sep

Donde:

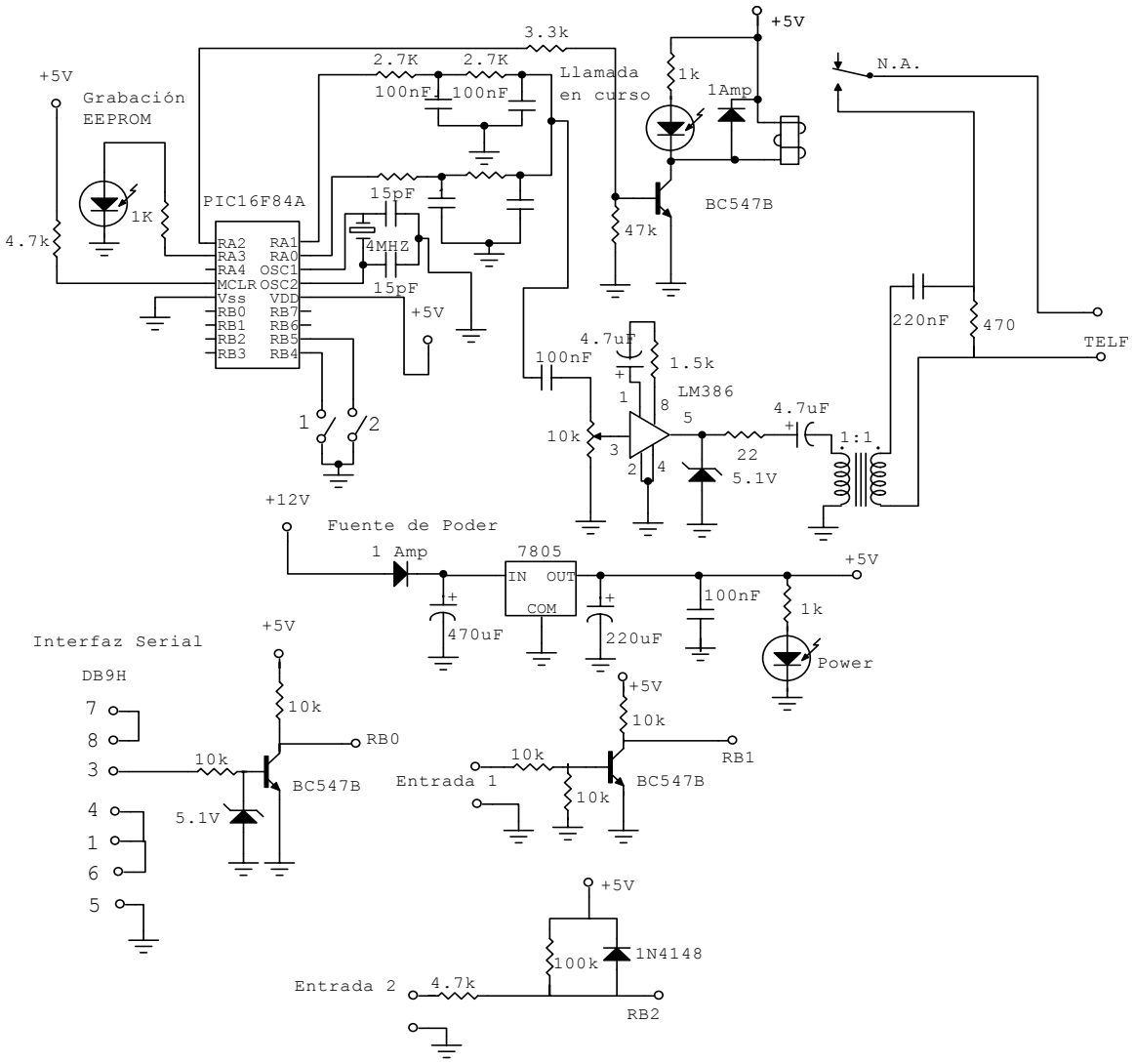
Nr = número de repeticiones (1-9)
Sep = Caracter 0x2F (“/”)
Telf1 = número de teléfono 1
Telf2 = número de teléfono 2
Telf3 = número de teléfono 3

Entradas de la Alarma

El diseño de la alarma consta de 2 entradas, la primera es una entrada normalmente abierta (N.A.) que puede ser habilitada o deshabilitada usando el DIP switch número 1. La segunda es una entrada que puede ser configurada como normalmente abierta o normalmente cerrada (N.C.) a través del DIP switch número 2. Si una entrada es N.A., se activa cerrando el circuito y si es N.C., se activa abriendo el circuito. La siguiente tabla describe las entradas en función del DIP Switch.

Switch		Descripción
0	0	Entrada 1 deshabilitada, Entrada 2 como N.A
0	1	Entrada 1 deshabilitada, Entrada 2 como N.C.
1	0	Entrada 1 habilitada, Entrada 2 como N.A.
1	1	Entrada 1 habilitada, Entrada 2 como N.C.

Hardware


+5V
3.3k
2.7K
2.7K
100nF
100nF
Llamada
en curso
1k
1Amp
N.A.
+5V
Grabación
EEPROM
1K
4.7k
PIC16F84A
RA2 RA1
RA3 RA0
RA4 OSC1
MCLR OSC2
Vss VDD
RB0 RB7
RB1 RB6
RB2 RB5
RB3 RB4
15pF
4MHZ
15pF
+5V
BC547B
47k
220nF
470
TELF
4.7uF
100nF
1.5k
LM386
1
8
5
10k
3
2
4
22
5.1V
4.7uF
1:1
1
2
+12V
Fuente de Poder
1 Amp
7805
IN OUT
COM
470uF
220uF
100nF
1k
Power
+5V
Interfaz Serial
DB9H
7
8
3
4
1
6
5
+5V
10k
RB0
10k
BC547B
5.1V
+5V
10k
RB1
10k
Entrada 1
BC547B
10k
+5V
100k
1N4148
Entrada 2
4.7k
RB2

Software

```
/*************************************************************************
*   ALARMA                                                               *
*                                                                        *
*   PIC16F84 @ 4Mhz                                                      *
*   Compilado: CCS PCM C compiler                                        *
*                                                                        *
*************************************************************************
*                                                                        *
*    Nombre:       alarma.c                                            *
*    fecha:         1/09/2001                                           *
*    Version:       1.0                                                 *
*                                                                        *
*    Autor: Carlos Narvaez                                               *
*                                                                        *
*                                                                        *
*                                                                        *
*************************************************************************
*                                                                        *
*    Notas:                                                              *
* Técnica: DTMF generador basado en Pulse width Modulation (PWM)         *
* Usa 18 ciclos por paso.                                                *
* Optimizado para crystal de 4MHz                                        *
*                                                                        *
* Este programa usa dos pines del PIC para las salidas de tono bajo y    *
* alto respectivamente. Agregue dos filtros pasa bajo uno en cada pin    *
* y una sus salidas a fin de sumar los dos tonos y obtener el tono DTMF  *
*************************************************************************/

#include <16f84.h>

#fuses XT, NOWDT, PUT, NOPROTECT
#ZERO_RAM

#use delay(clock = 4000000)
#use rs232(baud=9600,rcv=PIN_B0, parity=N, bits=8, errors)
#use fast_io(A)
#use fast_io(B)

/*************************************************************************
*          I/O
*************************************************************************/

#byte   PORTA         = 5                  // Salida
#byte   PORTB         = 6                  // Entradas
#byte   PCL           = 2

#bit Act_linea    = PORTA.2                // Activa línea telefónica
#bit LED          = PORTA.3                // LED Tiempo de grabación EEPROM
#bit IN1          = PORTB.1                // Entrada 1
#bit IN2          = PORTB.2                // Entrada 2
#bit DS1          = PORTB.4                // DIP Switch 1
#bit DS2          = PORTB.4                // DIP Switch 2
```

```
/*********************************************************************
*           Definicion de Variables
*********************************************************************/

#define HI_PIN   0x02                 // RA1  Tono alto
#define LOW_PIN  0x01                 // RA0  Tono Bajo

unsigned char temp[40];
unsigned int i;
unsigned long j;
int EEADR;
unsigned long timeout;
int NumTelf;
int ttono;
/*********************************************************************
*           Prototipos
*********************************************************************/
void delay_seg(int n);
void dtmfsend(int NumTelf, ttono);
void LeerEEprom(int EEADR);
/*--------------------------------------------------------------------*/
void main(void) {
/*--------------------------------------------------------------------*/

int NumRep;
set_tris_a(0x00);
set_tris_b(0xFF);
EEADR = 0;
LED = 1;
i = 0;
   for(j=0; j < 100; j++)
   {
    timeout =0;
       while(!kbhit()&&(++timeout<5000))
            delay_us(10);
        if(kbhit())
            temp[i++] = getchar();
    }
    LED = 0;                          // Fin tiempo lectura puerto Serial


    if(i != 0)                        // si i = 0 go to continuar.......
    {
         for(j=0; j< i; j++)
         {
               write_eeprom(EEADR,temp[j]);
               EEADR++;                       // incrementa direccion EEPROM

         }
     }
```

```
if(!DS1 && !DS2)
        while(IN2 == 1);
     else
        if(!DS1 && DS2)
           while(IN2 == 0);
         else
            if(DS1 && DS2)
               while(IN1 == 1 || IN2 == 1);
             else
                if(DS1 && DS2)
                   while(IN1 == 1 || IN2 == 0);



    EEADR = 0x00;
    LeerEEprom(EEADR);
    NumRep = NumTelf;
    do{
               i = 1;
               EEADR =0x02;
               while(i < 4){
                      Act_linea = 1;
                      delay_seg(3);
                      LeerEEprom(EEADR);
                      while(NumTelf != 0x2f) {
                             dtmfsend(NumTelf, 0x41);
                             delay_ms(300);
                             EEADR++;
                             LeerEEprom(EEADR);
               }
               delay_seg(5);
               for(j=0;j<20;j++)
               {
                      dtmfsend(12, 0xff);
                      delay_seg(1);
               }
                      i++;
                      EEADR++;
                      Act_linea = 0;
                      delay_seg(5);
               }
               Act_linea = 0;
               delay_seg(5);
               NumRep--;
        } while(NumRep !=0);
}

void dtmfsend(int NumTelf, ttono) {

int low_count;
int high_count;
int salva_low_count;
int salva_high_count;
int high_tone_count;
int low_tone_count;
int temporal;
```

```
#asm

inicio:
       movf   NumTelf,0
       andlw  0x0f                  ; key en W
       movwf  temporal              ; Salva Key en tmp
       call   tabla_low_count       ; Contador retardo tono bajo -> w
       movwf  salva_low_count       ; Salva contador retardo tono bajo
       movwf  low_count
       movf   temporal,0            ; Key -> w
       call   tabla_high_count      ; Contador retardo tono alto -> w
       movwf  salva_high_count      ; Salva contador retardo tono alto
       movwf  high_count
       clrf   low_tone_count        ; Tiempo tono 255 +
       movlw  ttono                 ; ttono en hdcnt
       movwf  high_tone_count       ;

bucle:
       clrwdt
       nop

bucle1:
       movlw   (HI_PIN + LOW_PIN)
       decfsz  low_count,1          ; Si ambos contadores (lowcnt, highcnt)
       andlw   HI_PIN               ; no han llegado a cero, w = B'00000000'
       decfsz  high_count,1         ; Si lowcnt llega a cero w = B'00000001'
       andlw   LOW_PIN              ; Selecionado RB0. Si highcnt llega a cero
                                    ; w = B'00000010' seleccionando RB1
       movwf   temporal             ; Salva w en tmp
       xorwf   PORTA,1              ; toggle RB selecionado
       movf    salva_low_count,0    ; Contador de retardo tono bajo -> w
       btfsc   temporal,0           ; Si RB0 selecionado
       movwf   low_count            ; W -> lowcnt
       movf    salva_high_count,0   ; Contador de Retardo tono Alto -> w
       btfsc   temporal,1           ; Si Rb1 selecionado
       movwf   high_count           ; W -> highcnt

       decfsz  low_tone_count,1     ; 1/2
       goto    bucle                       ; 2  3x255x hdcnt
       decfsz  high_tone_count,1    ; 1/2
       goto    bucle1               ; 2
       goto    fin

tabla_low_count:                   ; Contadores de retardo tonos bajos
       addwf PCL,1
       retlw  29
       retlw  40
       retlw  40
       retlw  40
       retlw  36
       retlw  36
       retlw  36
       retlw  32
       retlw  32
       retlw  32
       retlw  40
       retlw  36
       retlw  32
       retlw  29
       retlw  29
       retlw  29
```

```
tabla_high_count:                       ; Contadores de retardo tonos altos
       addwf PCL,1
       retlw  21
       retlw  23
       retlw  21
       retlw  19
       retlw  23
       retlw  21
       retlw  19
       retlw  23
       retlw  21
       retlw  19
       retlw  17
       retlw  17
       retlw  17
       retlw  17
       retlw  23
       retlw  19
fin:
#endasm
}

void LeerEEprom(int EEADR){
int eedata;
eedata=read_eeprom(EEADR);
       switch(eedata) {
          case 0x30:
               NumTelf = 0;
               break;
          case 0x31:
               NumTelf = 1;
               break;
          case 0x32:
               NumTelf = 2;
               break;
          case 0x33:
               NumTelf = 3;
               break;
          case 0x34:
               NumTelf = 4;
               break;
          case 0x35:
               NumTelf = 5;
               break;
          case 0x36:
               NumTelf = 6;
               break;
          case 0x37:
               NumTelf = 7;
               break;
          case 0x38:
               NumTelf = 8;
               break;
          case 0x39:
               NumTelf = 9;
               break;
          case 0x2f:
               NumTelf = eedata;
               break;
       }
}
```

```
void delay_seg(int n) {
   for(;n!=0; n--)
   delay_ms(1000);
}
```