E.T.S. de Ingeniería Industrial,
Informática y de Telecomunicación

Desarrollo de proyectos electrónicos con microcontroladores PIC

Grado en Ingeniería
en Tecnologías Industriales

Trabajo Fin de Grado

Julen Martín Carricas
Carlos Ruiz Zamarreño
Pamplona, 18 de junio de 2018

upna Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

RESUMEN:

Se han realizado los prototipos de tres proyectos distintos con el microcontrolador PIC16F876A: una central de alarmas, un Tetris y un medidor de distancia digital. Para ello se han empleado distintos sensores y actuadores tanto analógicos como digitales que estarán conectados al microcontrolador. Los proyectos se han dividido en tres partes fundamentales: el diseño de la PCB (placa de circuito impreso), para realizar dicho diseño se empleará el software DesignSpark PCB de RS. La segunda parte consiste en la programación del microcontrolador, para la cual se utilizará el software MPLAB IDE de Microchip. Por último, se procederá a realizar el montaje final; se realizará un diseño mecánico del prototipo con el software DesignSpark Mechanical, también de RS.

The prototypes for three different projects have been created with a PIC16F876A microcontroller. These projects are: the alarm centre, a Tetris and a digital meter. To carry out these prototypes various types of sensors and actuators (analogic and digital) have been used. The projects have been divided into three fundamental parts: the first part consists in the design of the PCB (printed circuit board), that will be done by using DesignSpark PCB, a software of RS. The second part consists in programming of the microcontroller, which will be done by using MPLAB IDE, a software of Microchip. To finish the project, the realization of the final montage will proceed by using DesignSpark Mechanical to create the mechanical design of the prototype.

PALABRAS CLAVE:

Microcontrolador; Central de alarmas; Tetris; Medidor de distancia; Diseño electrónico.

Microcontroller; Alarm center; Tetris; Digital meter; Electronic design.

DESARROLLO DE PROYECTOS ELECTRÓNICOS CON MICROCONTROLADORES PIC

ÍNDICE

Introducción y objetivos

Introducción

Se han realizado tres proyectos con el microcontrolador **PIC16F876A**: una central de alarmas de una casa (alarma por intruso y por escape de gas), una máquina de Tetris y un medidor de distancia digital. Para la implementación de estos prototipos se han utilizado distintos tipos de sensores y actuadores, tanto digitales como analógicos. La realización de los proyectos se divide en tres partes fundamentales: el diseño de la PCB, la programación del microcontrolador y el montaje final. Tanto para el diseño de la PCB como para el diseño mecánico (carcasa) se ha utilizado DesignSpark (DesignSpark PCB para la placa y DesignSpark Mechanical para el diseño mecánico). La programación del microcontrolador se ha realizado con el programa MPLAB, donde se programará utilizando el lenguaje C, un lenguaje de nivel medio.

Objetivos

El objetivo principal del trabajo ha sido profundizar en el diseño de sistemas electrónicos, tanto en el diseño de la placa como en la programación del microcontrolador. Concretamente se ha querido lograr los siguientes objetivos:

- Profundizar en el diseño de placas de circuito impreso.
- Profundizar en la programación de microcontroladores en lenguaje C.
- Aplicar los conocimientos obtenidos durante el grado para realizar los proyectos.
- Generar información divulgativa sobre el diseño de sistemas electrónicos.

Estado del arte

Microcontroladores

En este capítulo se describe la teoría fundamental de los microcontroladores: como seleccionar el microcontrolador adecuado, los principales fabricantes y sus entornos de programación.

Un microcontrolador es un circuito integrado que contiene todos los elementos de un procesador digital secuencial síncrono programable de arquitectura Harvard o Princeton. Debido a su bajo coste y consumo y su reducido tamaño es un elemento muy apropiado para numerosas aplicaciones. El microcontrolador está compuesto por una CPU (unidad central de procesamiento), una memoria ROM (memoria de lectura), una memoria RAM (memoria de acceso aleatorio) y entradas y salidas para la comunicación externa. Además, puede contener uno o más conversores analógicos digitales, temporizadores, módulos PWM…

Aplicaciones

Se puede encontrar un microcontrolador en cualquier aplicación en la que intervengan entradas y salidas, por lo que su rango de aplicación es muy grande; por ello se dividen los microcontroladores por gamas (baja, media y alta). Las aplicaciones de los microcontroladores van desde lo más simple (como el mando a distancia de una televisión o el aire acondicionado) hasta lo más complejo (generalmente computadoras). Existen muchas aplicaciones en la que es necesario utilizar más de un microcontrolador; por ejemplo, para el control de un automóvil se pueden llegar a utilizar hasta 50 microcontroladores. Por todo ello, se pueden llegar a producir más de 2500 millones de microcontroladores al año.

Sectores	%
Computación	30
Hogar	25
Comunicaciones	15
Industria	15
Automóvil	10
Resto	5

Tabla 1. Aplicaciones principales de los microcontroladores

Tal y como se puede observar en la tabla, la mayoría de los microcontroladores se producen para utilizarlos en el día a día (ordenadores, electrodomésticos, automóviles…).

Principales fabricantes

- **Microchip**: ofrecen soluciones para gamas de 8, 16 y 32 bits. Para programar estos microcontroladores se utiliza la aplicación MPLAB-IDE, el cual es distribuido gratuitamente por Microchip.
- **ATMEL**: fabrica los controladores de la familia AVR. La característica que diferencia a ATMEL es la memoria flash y EEPROM que incorporan los microcontroladores.

Tipos de microcontroladores

- **Gama baja**: de 4, 8 y 16 bits. Se utilizan principalmente para tareas de control (electrodomésticos, algunos periféricos de computadoras…).
- **Gama media**: de 16 y 32 bits. Para tareas de control con un grado de complejidad mayor (control del automóvil, teléfonos móviles…).
- **Gama alta**: 32, 64 y 128 bits. Dedicados fundamentalmente a procesamiento (ordenadores, videoconsolas…).

Diseño de PCB

En la PCB (placa de circuito impreso) se colocarán tanto el microcontrolador como el resto de los elementos que compongan el circuito de control. Muchas veces al microcontrolador no le valdrá el valor que llega desde un sensor, por lo que será necesario acomodarlo utilizando uno o más componentes (pueden ser tanto activos como pasivos). Existen muchas aplicaciones para realizar el diseño de la PCB; una de ellas es DesignSpark PCB, distribuida gratuitamente por RS, que será la que se va utilizar en los proyectos.

Una vez realizado el diseño de la PCB, este se manda a una impresora para que imprima la placa (sin los componentes). Generalmente se utiliza estaño para soldar dichos componentes; la soldadura se puede realizar manualmente o con una máquina; obviamente la máquina realizará una soldadura de mayor calidad y en menos tiempo, pero también será más caro; por lo que se utilizan para lotes de gran cantidad.

Empaquetado de los componentes

Como ya se ha comentado, se utilizarán varios componentes para la acomodación de la señal, estos se pueden dividir en dos grupos principales en lo que al empaquetado respecta:

- **Orificio pasante**: los componentes tienen unas patillas que se deberán pasar por los orificios de la PCB y soldarlos a ellos. La ventaja que presenta este tipo es que los componentes son mucho más fáciles de soldar que los de empaquetado SMD.

Figura 1. Componente de orificio pasante

- **Montaje superficial (SMD)**: los componentes se sueldan directamente en la placa, es decir, no tienen las patillas que tienen los componentes de orificio pasante. La principal ventaja de

este tipo de empaquetado es su reducido tamaño, ocupan por lo menos la mitad de lo que ocupa un elemento de orificio pasante; por lo que el tamaño de la PCB se reduce considerablemente. Además, permiten la automatización del ensamblaje y tienen una fiabilidad mayor.

Figura 2. Componentes SMD de distinto tamaño

1.- Central de alarmas

1.1 Introducción

Se va a realizar el prototipo de una central de alarmas que detecte tanto la entrada de un intruso como una fuga de gas o humo. La detección del intruso se realizará mediante unos sensores PIR (Passive Infrared Sensor) y de vibración; por otro lado, el humo y las fugas de gas se detectarán con sensores de la serie MQ. Para anular la cuenta atrás para que suene la alarma se deberá introducir la contraseña correcta en el teclado numérico.

1.2. Diseño esquemático

El circuito de control de la alarma se basa en el microcontrolador **PIC16F876A**, que trabaja a 5V, además se introducirá un oscilador externo de 20MHz, dado que este microcontrolador no tiene ningún oscilador interno.

Se utilizará una pila de 8.4V para alimentar la PCB, por lo que será necesario el uso de un

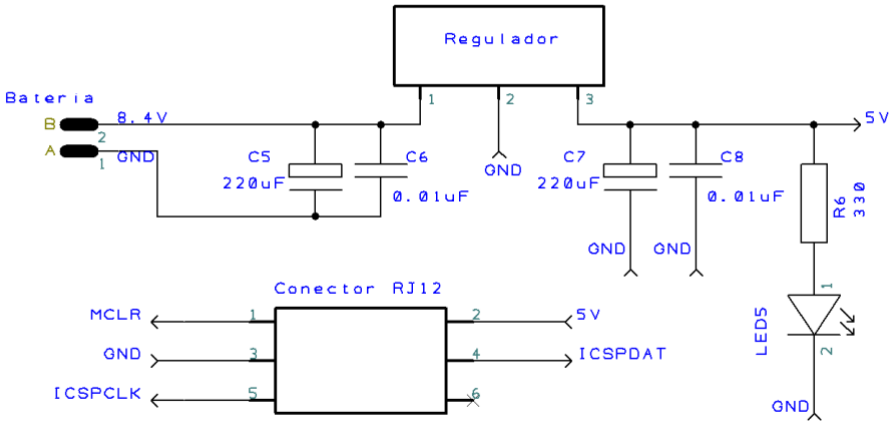


Figura 3. Diseño esquemático de la alimentación de la alarma

Con un conector tipo RJ-12 se comunicará el microcontrolador con la aplicación del ordenador, desde la cual se programará el PIC.

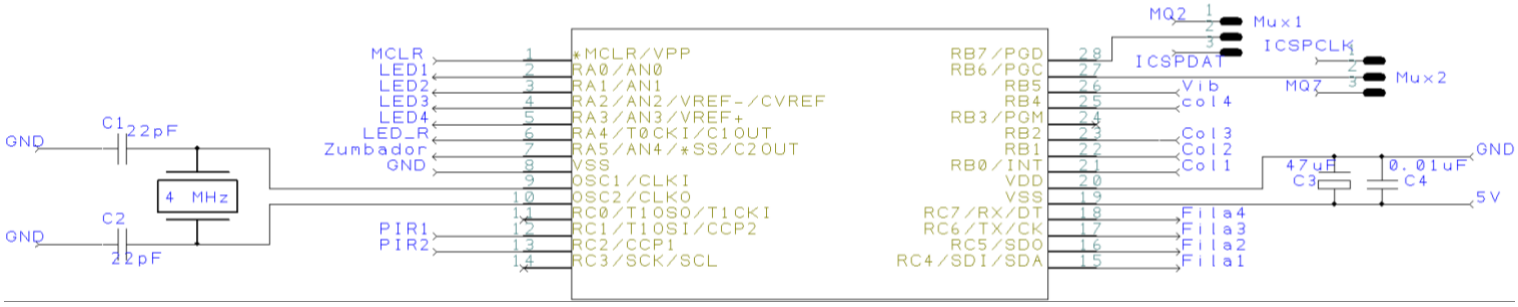


Figura 4. Diseño del circuito del microcontrolador de la alarma

La mayoría de los periféricos no requieren ningún tipo de acomodamiento, por lo que irán conectados directamente al microcontrolador. Los sensores de presencia (PIR) se componen de tres patillas: dos de alimentación (5V y GND) y la patilla de salida digital, la cual se conectará con el microcontrolador. En cuanto el sensor detecte a una persona el pin de salida digital se pondrá a 5V. Se puede ajustar tanto la sensibilidad del sensor como el tiempo entre mediciones mediante dos potenciómetros.

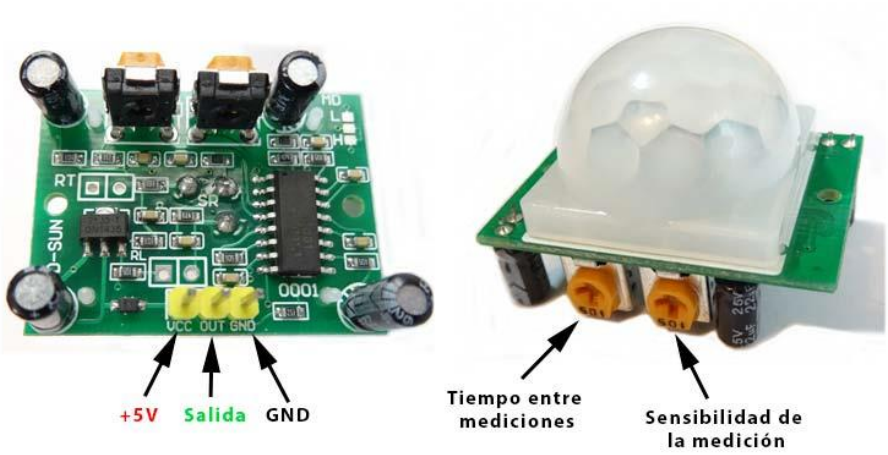


Figura 5. Sensor de presencia

Los sensores de gas y vibración (que se utiliza para que salte cuando se ha cerrado una puerta o ventana) tienen una cuarta patilla de salida analógica, aunque en este caso no se va a utilizar, por lo que las conexiones serán las mismas que las de los sensores PIR: alimentación entre 5V y GND y la patilla de salida conectada con el microcontrolador.

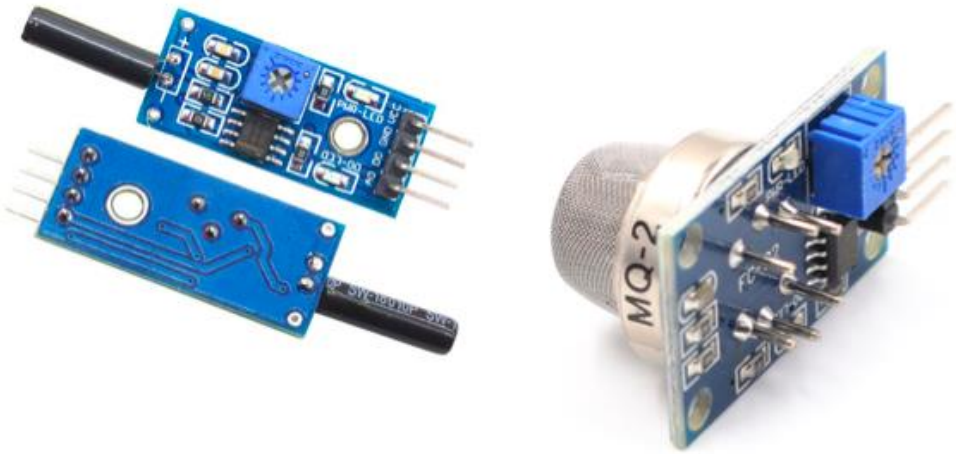


Figura 6. Sensor de vibración y MQ2

Se utiliza un teclado de 4x4 para introducir la contraseña. Con el fin de no utilizar 16 pines (uno por cada botón), se le asigna un pin a cada columna y fila, obteniendo así un total de 8 pines. Todos los botones están conectados entre una fila y una columna. Para saber qué botón ha sido presionado se deberán comprobar las filas por separado, es decir, se pondrán una por una las columnas a 0V y luego examinar el estado de los pines de las columnas. Por ejemplo, si la primera fila está a 0 y todas las demás a 1 y el microcontrolador recibe que la tercera columna está a 0, significa

que se ha pulsado el tercer botón de la primera fila (“3”). Para que esto sea posible se deberán utilizar unas resistencias pull-up en los pines del microcontrolador de las columnas.

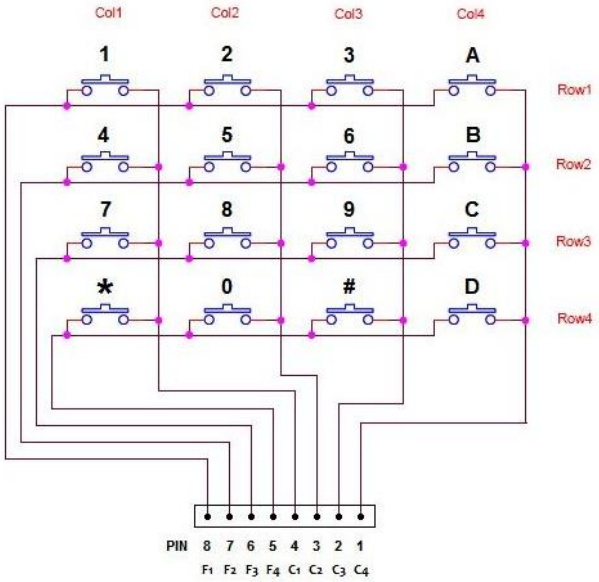


Figura 7. Circuito interno del teclado matricial

Cada vez que se pulse un botón se encenderá un LED verde. El circuito para encenderlos consiste en una resistencia de 330Ω que está entre una de las salidas del microcontrolador y el LED, por lo que cuando la salida se ponga a un “1” lógico, circulará por el LED una corriente de casi 15mA, suficiente como para que se encienda.

Para el circuito de control del zumbador se utilizará un transistor NPN, de esta forma cuando se mande un “1” desde el micro el transistor se activará y el zumbador estará entre 5V y 0V. Además, se ha introducido un jumper entre el micro y el zumbador, así se podrá desconectar el zumbador sin necesidad de alterar el programa.

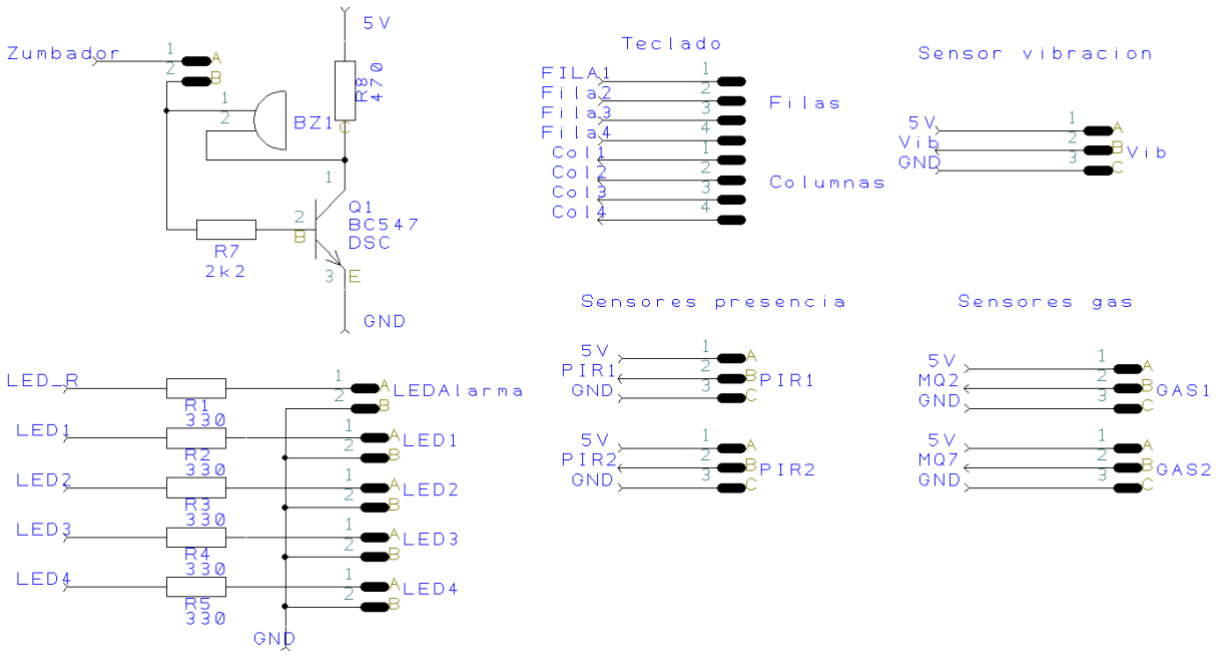


Figura 8. Diseño esquemático del control de los periféricos de la alarma

1.2.1 Listado de componentes

REFERENCIA	COMPONENTE	VALOR	EMPAQUETADO
R1-6	Resistencia	330Ω	DSC
R7	Resistencia	2.2kΩ	DSC
R8	Resistencia	470Ω	DSC
C1-2	Condensador cerámico	22pF	DSC
C3	Condensador electrolítico	47uF	DSCV
C4,6,8	Condensador cerámico	0.1uF	DSC
C5,7	Condensador electrolítico	220uF	DSCV
XTAL	Oscilador de cristal	20MHz	DIL
U1	Microcontrolador	PIC16F876A	SPDIP
U3	Conector RJ12	-	DIL
U4	Regulador	MC7800	DIL
Q1	Transistor NPN	BC547	DSC
BZ1	Zumbador piezoeléctrico	-	DIL
PL1-7	Conector 2 pines	-	DSC
PL8-14	Conector 3 pines	-	DSC
PL15-16	Conector 4 pines	-	DSC
LED1	LED rojo	-	DIL

Tabla 2. Lista de componentes de la alarma

1.3. Configuración del microcontrolador

1.3.1. Configuración del Timer 1

Se va a utilizar el Timer 1 del PIC16F876A para contar 30 segundos desde la detección de una persona hasta que suene la alarma. El número de pulsos necesarios para el desbordamiento dependerá del factor pre-divisor (P) y del periodo de los pulsos de entrada (T_i), los valores se obtendrán con la siguiente ecuación:

$$TMR1 = (65536 - N_{TMR1}) * P * T_i$$

Dado que el valor máximo del factor de división es 8 y que el microcontrolador trabaja a 20MHz, el tiempo máximo que puede llegar a contar el TMR1 es de 0,105 segundos, por lo que será necesario utilizar un contador auxiliar para poder llegar a los 30 segundos. Se ajustará el TMR1 para que cada 0,1 segundos salte el flag de desbordamiento, por lo que cuando el contador llegue a 300 habrán pasado 30 segundos.

```
T1CON=0b00110000;                       //Timer 1 como temporizador, factor pre-divisor=8
TMR1H=0x0B;                             //0.1 segundos, N=3036
TMR1L=0xDC;
```

1.3.2. Configuración resistencias pull-up

Para el correcto funcionamiento del teclado será necesario activar las resistencias pull-up de PORTB. Estas resistencias se activan modificando el registro *OPTION_REG*, poniendo a 0 el bit 7 se activarán.

```
OPTION_REG=0b01111111;        //Activar PORTB Pull-up
```

1.3.3. Generación de sonidos

Dado que los módulos CCP estaban ocupados no se ha podido implementar la generación de sonidos en uno de ellos. Si se hubiese conectado el zumbador al pin RC1 o RC2 se podría haber utilizado el módulo PWM y la generación de la señal habría sido mucho más sencilla. Por lo tanto, la señal cuadrada que irá al zumbador se generará manualmente:

```
for(k=0;k<2;k++){
        for(j=0;j<50;j++){
                RA5=~RA5;
                __delay_us(956);
        }
}
```

En este ejemplo se he generado la nota DO; para saber de cuánto debe ser el delay se busca la frecuencia de la nota en la tabla que se muestra a continuación:

Frecuencia (en Hertzios) de las notas musicales

	x								
		0	1	2	3	4	5	6	7	8
n=1	do		32.7	65.41	130.81	261.63	523.25	1046.50	2093.00	4186.01
n=2	do#		34.65	69.30	138.59	277.18	554.37	1108.73	2217.46	4434.92
n=3	re		36.71	73.42	146.83	293.66	587.33	1174.66	2349.32	4698.64
n=4	re#		38.89	77.78	155.56	311.13	622.25	1244.51	2489.02	4978.03
n=5	mi		41.2	82.41	164.81	329.63	659.26	1318.51	2637.02	5274.04
n=6	fa	21.826	43.65	87.31	174.61	349.23	698.46	1396.91	2793.83	5587.65
n=7	fa#	23.125	46.25	92.50	185.00	369.99	739.99	1479.98	2959.96	5919.91
n=8	sol	24.50	49.00	98.00	196.00	392.00	783.99	1567.98	3135.96	6271.93
n=9	sol#	25.96	51.91	103.83	207.65	415.30	830.61	1661.22	3322.44	
n=10	la	27.50	55.00	110.00	220.00	440.00	880.00	1760.00	3520.00	
n=11	la#	29.14	58.27	116.54	233.08	466.00	932.33	1864.66	3729.31	
n=12	si	30.87	61.74	123.47	246.94	493.88	987.77	1975.53	3951.07	

Tabla 3. Frecuencias de las notas musicales

Una vez obtenida la frecuencia el delay será igual a la mitad del periodo:

$$delay = \frac{1}{2 * f_{nota}} = \frac{1}{2 * 523.25} = 956 * 10^{-4}\ seg = 956\ \mu s$$

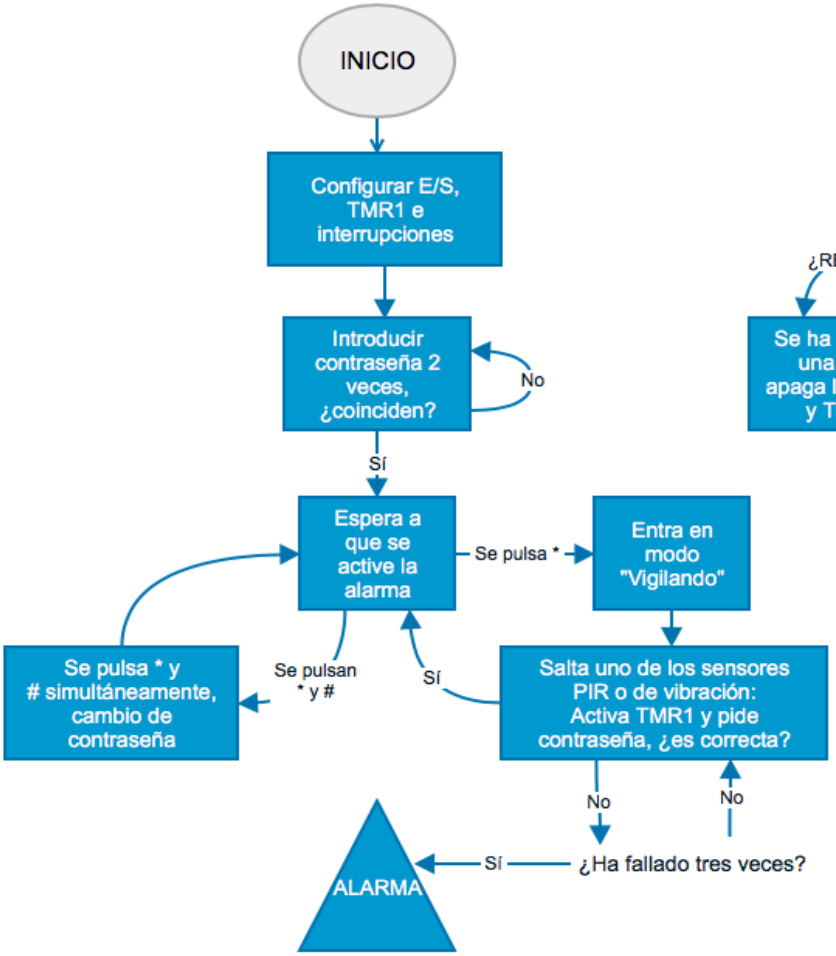
1.3.4. Interrupciones

El programa funcionará con interrupciones del TMR1 y por cambio en el nivel lógico RB4:RB7 del puerto B, de esta forma cuando se detecte una fuga de gas sonará la alarma. Los sensores de gas están conectados a RB6 y RB7, por lo que en el momento en el que tengan un cambio de estado saltará el flag de interrupción. Por otro lado, la cuarta columna del teclado está conectada a RB4, por lo tanto, también saltará el flag de interrupción cuando haya un cambio de estado; esto se aprovechará para apagar la alarma en caso de que salte; es decir, si la alarma está sonando basta con pulsar un botón de la cuarta columna para apagarlo (A, B, C o D).

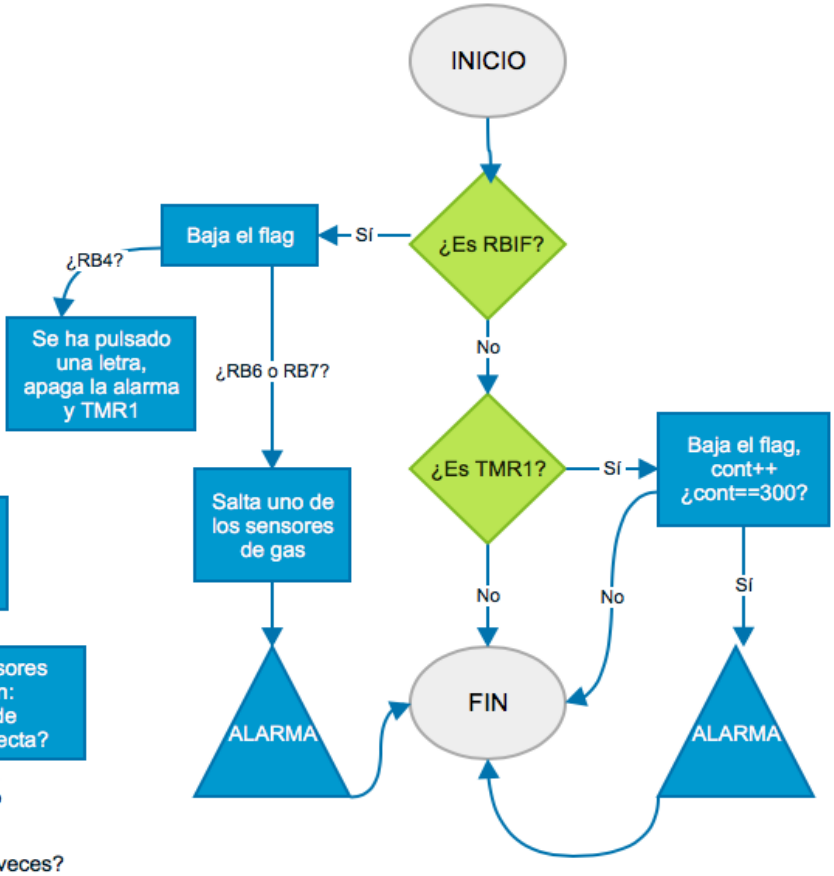
```
GIE=1;                          //Activar interrupciones globales
PEIE=1;                         //Activar interrupciones de periféricos
TMR1IE=1;                       //Interrupciones por Timer1 activadas
RBIE=1;                         //Activar interrupciones por cambio en PORTB
```

1.3.5. Funcionamiento

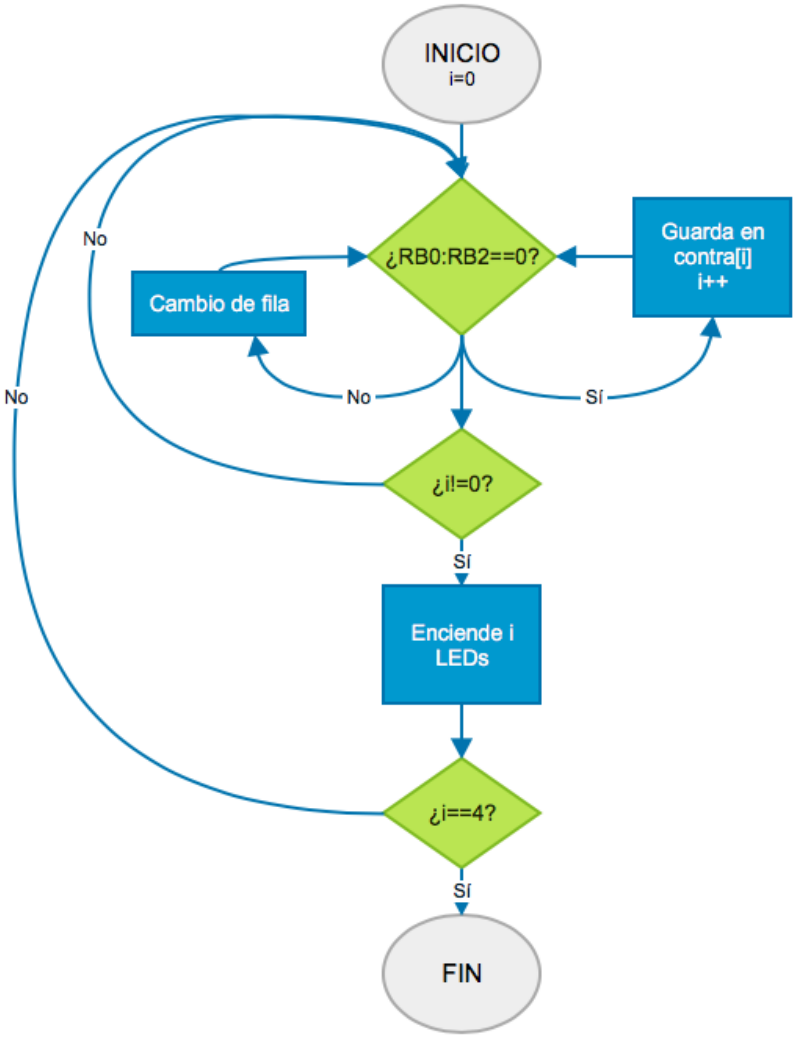
PROGRAMA PRINCIPAL



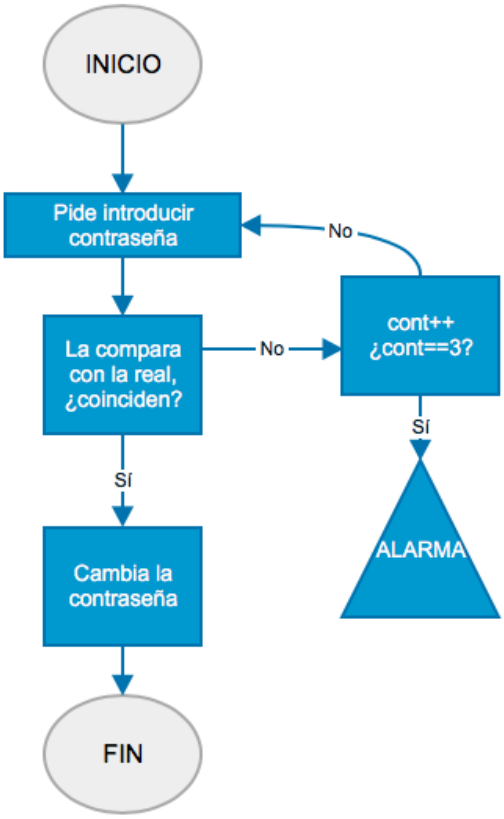
RUTINA DE ATENCIÓN A INTERRUPCIONES



RUTINA DE INTRODUCCIÓN DE CONTRASEÑA



RUTINA DE CAMBIO DE CONTRASEÑA



1.3.6. Programa fuente

PROGRAMA FUENTE

```
#include<htc.h>                        //Incluimos libreria del micro a usar
__CONFIG(WRT_OFF & WDTE_OFF & PWRTE_OFF & FOSC_HS & LVP_OFF);
#define _XTAL_FREQ 4000000          //Oscilador Interno de 4MHZ

unsigned char error, activar=0, fallos, tiempo, intruso=0;
unsigned int         cont1=0;

void contrasena(void);
void verificar(void);
void cambio(void);
void son_ok(void);
```

```
void son_fail(void);
void alarma(void);

                    //////////////////////INTERRUPCIONES//////////////////////////
static void interrupt isr(void){
        if(RBIF){                           //Salta el flag de cambio de estado en RB4:RB7
           RBIF=0;                          //Baja el flag
           if(RB6 | RB7)     intruso=1;     //Uno de los sensores de gas está activado
           if(activar && !RB4){             //Se pulsa un botón de la cuarta columna
                    while(!RB4){            //Espera a que se deje de pulsar
                      __delay_ms(20);
                    }
                    intruso=0;              //Se apaga la alarma
                    activar=0;
                    tiempo=0;
                    PORTA=0;
                    TMR1ON=0;
                    son_ok();
           }
        }
        if(TMR1IF){                         //Salta el flag del TMR1
           TMR1IF=0;                        //Baja el flag
           cont1++;
           if(cont1==330){
                    cont1=0;
                    TMR1ON=0;
                    tiempo=1;               //Han pasado 30 segundos desde la detección
           }
        }
}

                    ////////////////////PROGRAMA PRINCIPAL//////////////////////
void main(void){
        ADCON1=6;                           //Digitales
        TRISA=0;                            //PORTA salidas
        TRISB=0xFF;                         //PORTB entradas
        TRISC=0x06;                         //RC1 y RC2 entradas, resto salidas.
        GIE=1;                              //Activar interrupciones globales
        PEIE=1;
        TMR0IE=0;                           //Desactivar interrupciones por Timer0
        RBIE=1;                             //Activar interrupciones por cambio en PORTB
        RBIF=0;
        T1CON=0b00110000;                   //Timer 1 como temporizador, factor pre-divisor=8
        TMR1H=0x0B;
        TMR1L=0xDC;                         //0.5 segundos
        OPTION_REG=0b01111111;              //Activar PORTB Pull-up
        TMR1IE=1;                           //Interrupciones por Timer1 activadas
        intruso=0;

        error=1;
        while(error==1){                    //Se queda en bucle hasta que la contraseña coincida
```

```
        error=0;
        contrasena();                   //Introduce contraseña
        verificar();                    //Verifica la contraseña
        __delay_ms(500);
        if(error==1){
                son_fail();             //Sonido de error (las contraseñas no coinciden)
        }
    }
    son_ok();                           //Sonido que confirma que las contraseñas coinciden
    while(1){
        activar=0;
        while(activar==0){                      //Bucle "Modo Casa"
                PORTC=0b01110000;
                if(RB0==0 && RB2==1){           //Si se mantiene pulsado * se activa la alarma
                  __delay_ms(3000);
                  if(RB0==0 && RB2==1){
                        activar=1;
                        son_ok();               //Sonido para confirmar al usuario que se
                  }                             //ha activado la alarma
                }
                if(RB0==0 && RB2==0){           //Si se mantienen pulsados * y # simultáneamente
                  __delay_ms(3000);             //se activa el cambio de contraseña
                  if(RB0==0 && RB2==0){
                        son_ok();
                        cambio();
                  }
                }
                if(intruso){            //Si salta uno de los sensores de gas suena la alarma
                  alarma();
                }
        }
        while(activar==1){                      //Bucle "Modo Vigilando"
                if(RC1 || RC2 || !RB5){         //Salta uno de los sensores PIR o el de vibración
                  fallos=0;
                  TMR1ON=1;                     //Enciende el Timer1 para contar hasta 30 segundos
                  while(activar){
                        PORTA=0x0F;
                        verificar();            //Pide la contraseña
                        if(error==1){           //Si es incorrecta
                                son_fail();
                                fallos++;
                                if(fallos==3){          //Si se falla tres veces al introducir
                                        intruso=1;      //la clave salta la alarma
                                        alarma();
                                        fallos=0;
                                        tiempo=0;
                                }
                        }
                        if(error==0){   //Si la contraseña es correcta se pasa a "Modo Casa"
                                activar=0;
                                cont1=0;
```

```
                                TMR1ON=0;                   //Desactiva el Timer1
                                son_ok();
                            }
                        }
                    }
            }
        }
}
```

SUBPROGRAMAS DE LAS CONTRASEÑAS

```
#include<htc.h>                      //Incluimos libreria del micro a usar
#define _XTAL_FREQ 4000000          //Oscilador Interno de 4MHZ

unsigned char contra[4]=0;
unsigned char i=0,j, contador, error, intruso, tiempo;

void son_ok(void);
void son_fail(void);
void pulsa(void);
void alarma();

        ////////////////////SUBPROGRAMA PARA INTRODUCIR UNA NUEVA CONTRASEÑA//////////////////////
void contrasena(void){
        i=0;
        PORTA=0;
        while(i<4){
           PORTC=0b11100000;        //Se pone a 0 la primera fila
           __delay_us(10);
           if(RB0==0){              //Se comprueba la primera columna
                     contra[i]=1;
                     i++;
                     pulsa();              //y suena una nota al pulsar
                     while(!RB0){          //Se queda en bucle hasta que se deje de pulsar el botón
                       __delay_ms(20);     //Delay para los rebotes
                     }
           }
           if(RB1==0){                     //Segunda columna
                     contra[i]=2;
                     i++;
                     pulsa();              //y suena una nota al pulsar
                     while(!RB1){          //Se queda en bucle hasta que se deje de pulsar el botón
                       __delay_ms(20);     //Delay para los rebotes
                     }
           }
           if(RB2==0){                     //Tercera columna
                     contra[i]=3;
                     i++;
                     pulsa();              //y suena una nota al pulsar
                     while(!RB2){          //Se queda en bucle hasta que se deje de pulsar el botón
                       __delay_ms(20);     //Delay para los rebotes
```

```
            }
        }
        PORTC=0b11010000;               //Segunda fila
        __delay_us(10);
        if(RB0==0){
            contra[i]=4;
            i++;
            pulsa();                    //y suena una nota al pulsar
            while(!RB0){                //Se queda en bucle hasta que se deje de pulsar el botón
              __delay_ms(20);           //Delay para los rebotes
            }
        }
        if(RB1==0){
            contra[i]=5;
            i++;
            pulsa();                    //y suena una nota al pulsar
            while(!RB1){                //Se queda en bucle hasta que se deje de pulsar el botón
              __delay_ms(20);           //Delay para los rebotes
            }
        }
        if(RB2==0){
            contra[i]=6;
            i++;
            pulsa();                    //y suena una nota al pulsar
            while(!RB2){                //Se queda en bucle hasta que se deje de pulsar el botón
              __delay_ms(20);           //Delay para los rebotes
            }
        }
        PORTC=0b10110000;               //Tercera fila
        __delay_us(10);
        if(RB0==0){
            contra[i]=7;
            i++;
            pulsa();                    //y suena una nota al pulsar
            while(!RB0){                //Se queda en bucle hasta que se deje de pulsar el botón
              __delay_ms(20);           //Delay para los rebotes
            }
        }
        if(RB1==0){
            contra[i]=8;
            i++;
            pulsa();                    //y suena una nota al pulsar
            while(!RB1){                //Se queda en bucle hasta que se deje de pulsar el botón
              __delay_ms(20);           //Delay para los rebotes
            }
        }
        if(RB2==0){
            contra[i]=9;
            i++;
            pulsa();                    //y suena una nota al pulsar
            while(!RB2){                //Se queda en bucle hasta que se deje de pulsar el botón
```

```
                    __delay_ms(20);      //Delay para los rebotes
                  }
          }
          PORTC=0b01110000;        //Cuarta fila
          __delay_us(10);
          if(RB1==0){
                  contra[i]=0;
                  i++;
                  pulsa();              //y suena una nota al pulsar
                  while(!RB1){          //Se queda en bucle hasta que se deje de pulsar el botón
                    __delay_ms(20);     //Delay para los rebotes
                  }
          }
          if(i==1)          PORTA=0x01;
          else if(i==2)     PORTA=0x03;
          else if(i==3)     PORTA=0x07;
          else if(i==4)     PORTA=0x0F;
        }
        __delay_ms(500);
        i=0;
        PORTA=0;
}

//////////////////////SUBPROGRAMA PARA VER SI LA CONTRASEÑA COINCIDE////////////////////////////
void verificar(void){
        unsigned char contra2[4];
        i=0;
        error=0;
        for(i=0;i<4;i++){
          contra2[i]=contra[i];           //Guarda la contraseña en una auxiliar
        }
        contraseña();                     //Introducir la contraseña
        for(i=0;i<4;i++){
          if(contra[i]!=contra2[i])error=1;       //Si las contraseñas no coinciden
          contra[i]=contra2[i];           //Vuelve a guardar la contraseña en su vector
        }
}
////////////////////////////SUBPROGRAMA PARA CAMBIAR LA CONTRASEÑA//////////////////////////////
void cambio(void){
        contador=0;
        error=1;
        while(error==1){                  //Pide la contraseña antigua
          error=0;
          verificar();
          if(error==1){
                  contador++;
                  son_fail();
                  if(contador==3){      //Si se falla tres veces salta la alarma
                    intruso=1;
                    alarma();
                  }
```

```
          }
       }
       son_ok();
       if(error==0){
          error=1;
          while(error==1){                 //Pide la nueva contraseña, se queda en bucle hasta que
la contraseña coincida
                     error=0;
                     contrasena();         //Introduce contraseña
                     verificar();          //Verifica la contraseña
                     if(error==1){
                       son_fail();         //Sonido de error (las contraseñas no coinciden)
                     }
          }
       }
       son_ok();                           //Sonido que confirma que las contraseñas coinciden
}
```

SUBPROGRAMAS DE LOS SONIDOS

```
#include<htc.h>                      //Incluimos libreria del micro a usar
#define _XTAL_FREQ 4000000          //Oscilador Interno de 4MHZ

unsigned char j=0, k=0, intruso=0, tiempo=0;

       ////////////////////////////////CONTRASEÑA CORRECTA//////////////////////////////////
void son_ok(void){
       for(k=0;k<1;k++){
          for(j=0;j<50;j++){
                     RA5=~RA5;
                     __delay_us(956);
          }
       }
       __delay_ms(20);
       for(k=0;k<1;k++){
          for(j=0;j<200;j++){
                     RA5=~RA5;
                     __delay_us(638);
          }
       }
}
       ////////////////////////////////CONTRASEÑA INCORRECTA//////////////////////////////////
void son_fail(void){
       for(k=0;k<1;k++){
          for(j=0;j<10;j++){
                     RA5=~RA5;
                     __delay_us(7644);
          }
       }
       __delay_ms(50);
       for(k=0;k<1;k++){
          for(j=0;j<10;j++){
```

```
                    RA5=~RA5;
                    __delay_us(7644);
           }
        }
}

        ////////////////////////////////////ALARMA/////////////////////////////////////////////
void alarma(void){
        RC7=0;
        while(intruso||tiempo){
           RA4=1;                           //Enciende el LED
           for(k=0;k<1;k++){
                    for(j=0;j<100;j++){
                      RA5=~RA5;
                      __delay_us(7644);
                    }
           }
           RA4=0;                           //Apaga el LED
           __delay_ms(3000);
        }
}

        ////////////////////////////////////TECLA PULSADA/////////////////////////////////////////
void pulsa(void){
        for(k=0;k<1;k++){
           for(j=0;j<50;j++){
                    RA5=~RA5;
                    __delay_us(478);
           }
        }
}
```

1.4. Montaje

1.4.1. Diseño PCB

En la Figura se muestra el diseño de la placa de circuito impreso desarrollada para el control de la central de alarmas. El diseño se ha realizado utilizando el programa DesignSpark PCB. Se ha implementado el circuito a una sola cara, lo que reduce el coste y tiempo de fabricación además de facilitar su montaje. Además, se ha tenido en cuenta en el diseño el trazado de las pistas de alimentación utilizando un mayor grosor, así como la utilización de un plano de masa.

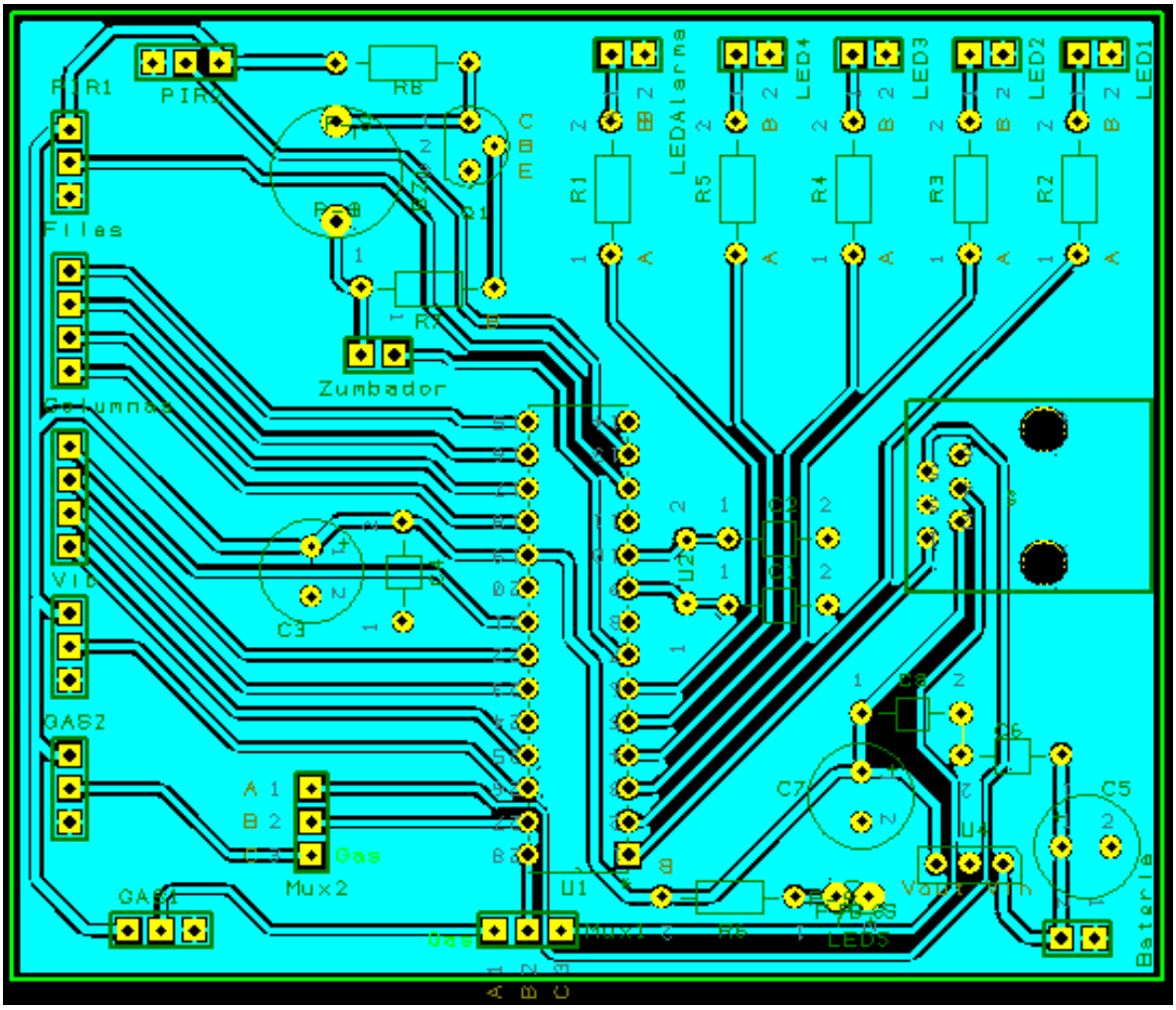


Figura 9. Diseño de la PCB de la alarma

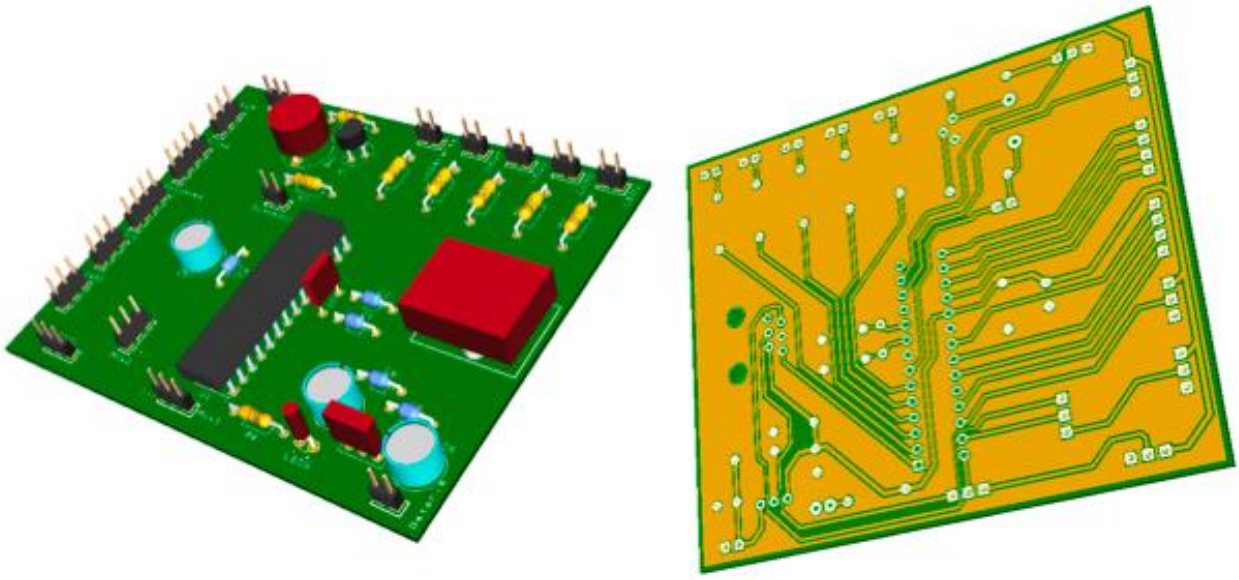


Figura 10. Diseño de la PCB de la alarma en 3D

1.4.2. Diseño mecánico

Se ha realizado una maqueta a escala reducida de una casa (dos habitaciones). En cada una de las habitaciones irá un sensor de presencia y uno de los sensores de gas; por lo que se han hecho dos agujeros del tamaño de dichos sensores. El teclado se pegará en la pared frontal junto con los LEDs. Se ha separado la pared frontal del resto de la maqueta para que el montaje sea más sencillo, una vez esté todo montado se podrá atornillar dicha pared al resto de la casa.

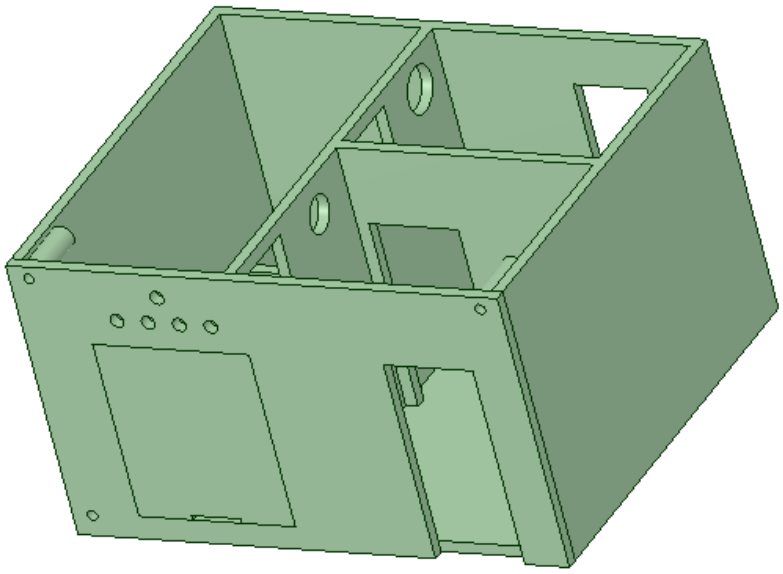


Figura 11. Diseño 3D de la maqueta de la casa

2.- Tetris

2.1 Introducción

El Tetris es un videojuego de puzzle que consiste en ir colocando unas piezas que irán cayendo de tal forma que rellenen una fila entera. La caída de las piezas es inevitable, pero se podrá variar tanto su posición como su rotación (0º, 90º, 180º y 270º). Hay 7 tipos de piezas, todas compuestas por 4 bloques: *I, J, L, O, S, T* y *Z*.

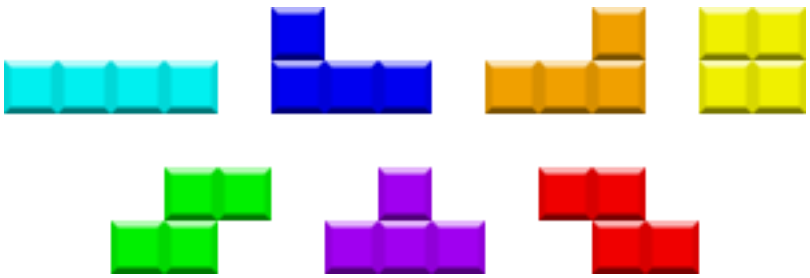


Figura 12. Piezas del Tetris

Se utilizarán dos matrices de LEDs de 8x8 como pantalla, se emplearán dos contadores Johnson para encender dichas matrices; su funcionamiento se explicará más adelante. Además, con un joystick se podrá tanto mover como rotar la pieza que esté cayendo.

2.2. Diseño esquemático

El circuito de control de la alarma se basa en el microcontrolador **PIC16F876A**, que trabaja a 5V, además se introducirá un oscilador externo de 20MHz, dado que este microcontrolador no tiene ningún oscilador interno.

Se utilizará una pila de 8.4V para alimentar la PCB, por lo que será necesario el uso de un regulador que lo convierta a 5V.

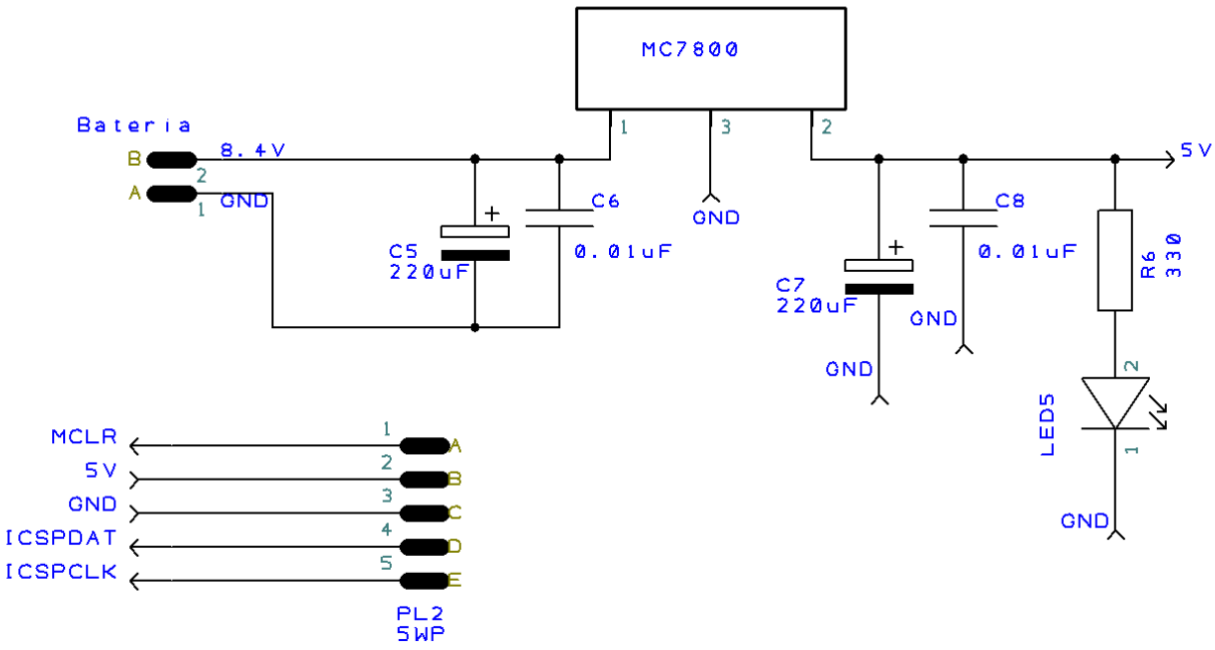


Figura 13. Diseño esquemático de la alimentación del Tetris

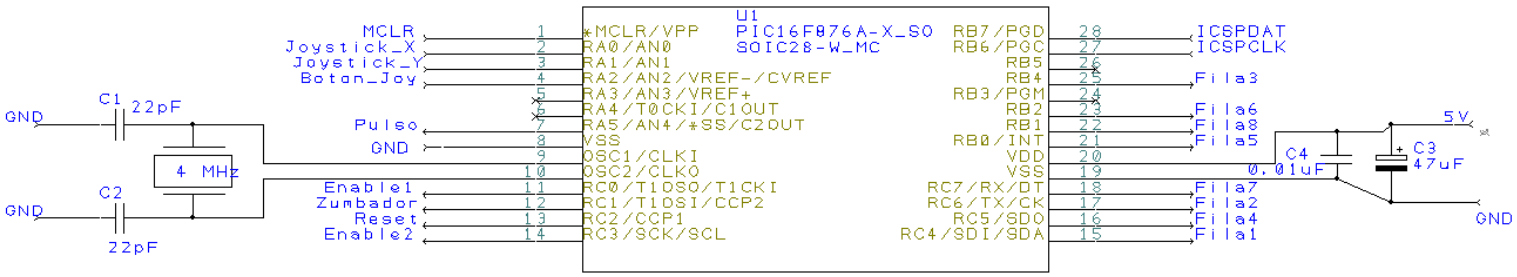


Figura 14. Diseño esquemático del circuito del microcontrolador del Tetris

La pantalla se controlará con 8 pines del microcontrolador, que estarán conectados a las columnas a través de unas resistencias de 330Ω, y dos contadores que irán iluminando las filas una por una. Las matrices de LEDs que se emplearán son las 1088AS. Como se puede observar en la figura 15, los ánodos de los LEDs se encuentran en las columnas de las matrices, mientras que los cátodos en las filas. Dado que las salidas de los contadores se ponen a 5V, estas se conectarán a los ánodos de las matrices, mientras que los pines del microcontrolador a los cátodos. Se "girarán" las matrices para que las columnas de estas sean las filas de la pantalla y viceversa.

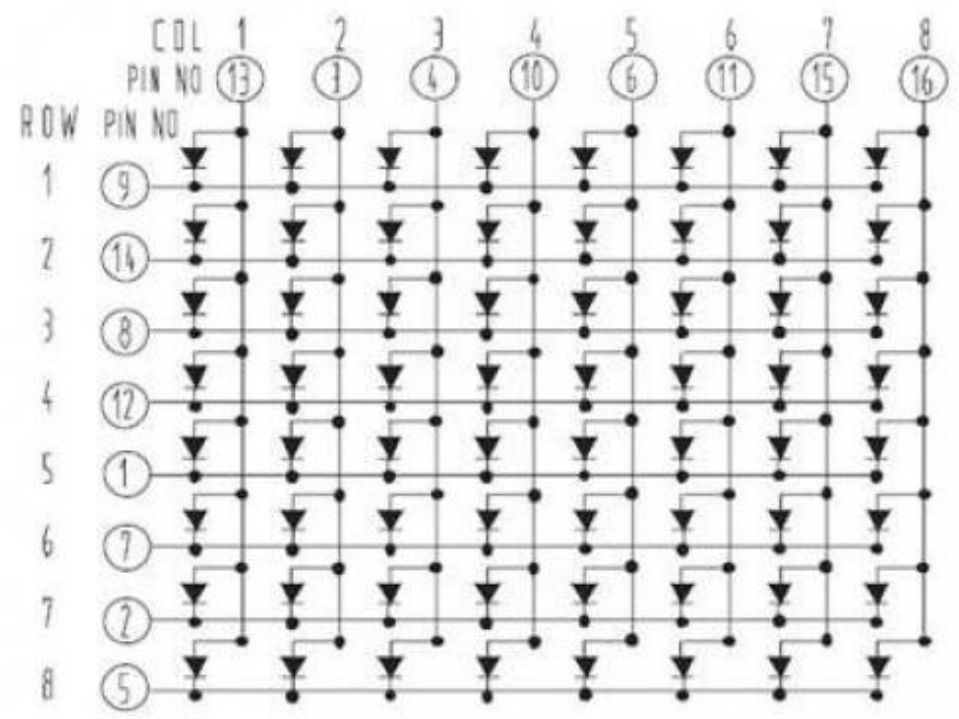


Figura 15. Circuito interior de la matriz de LEDs

Con el fin de ahorrar pines del microcontrolador, se utilizarán dos contadores Johnson para encender las matrices de LEDs; por lo tanto, en vez de tener que utilizar un pin por cada fila de la matriz, solo harán falta 4 pines para el control de los contadores. Dado que estos pueden "contar" hasta 10, el primero encenderá las 10 primeras filas de la pantalla y el segundo las 6 restantes. A continuación se puede ver cómo funciona el contador:

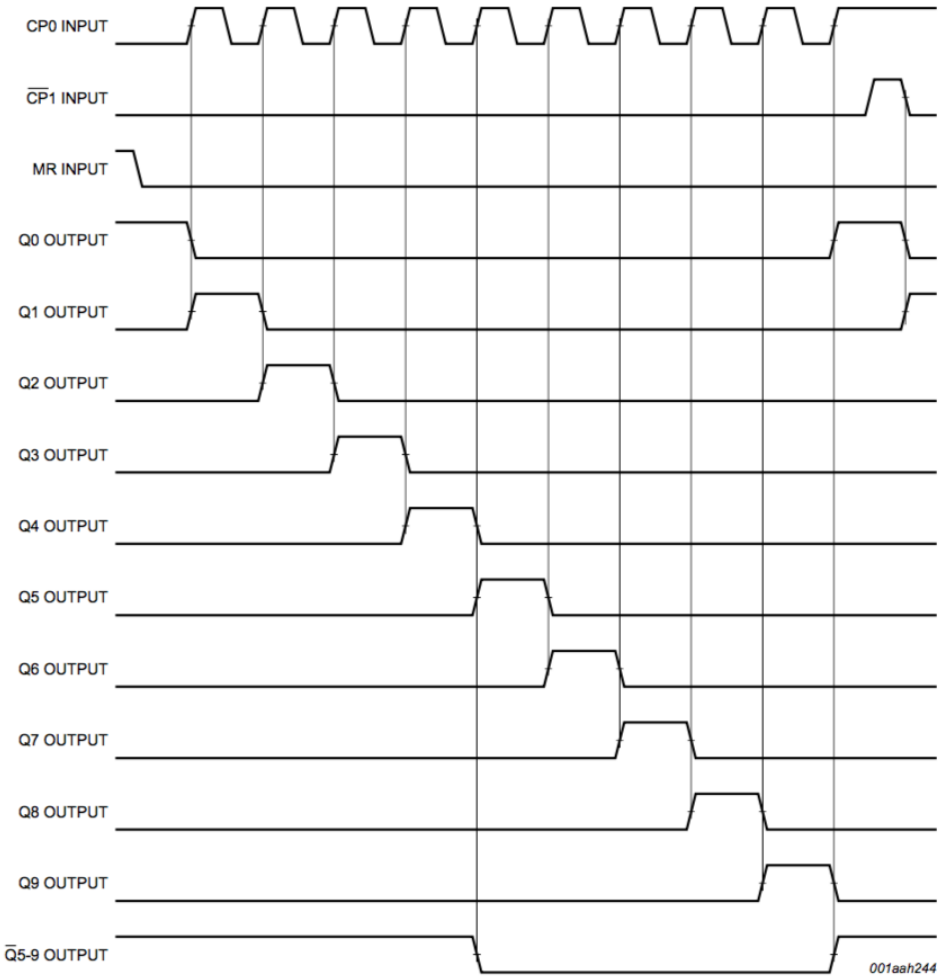


Figura 16. Funcionamiento de los contadores Johnson

También puede funcionar poniendo CP0 a 5V, de esta forma habrá un cambio cada vez que haya un flanco de bajada en CP1.

Los contadores no podrán estar encendidos simultáneamente, ya que de esta manera se iluminarían dos filas a la vez. Para poder controlar el encendido de los contadores se utilizarán dos transistores NPN, uno por contador.

El joystick que se utilizara tiene 5 pines: dos de alimentación (5V y GND), dos salidas analógicas (eje “x” y eje “y”) y una salida digital (pulsador). Las salidas analógicas variarán su valor en función de la posición del joystick:

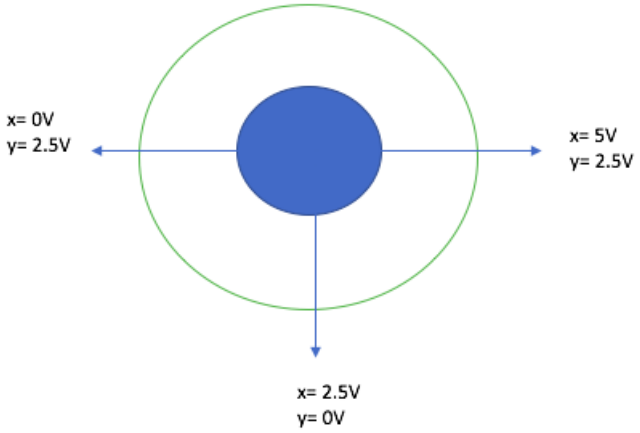


Figura 17. Voltaje de los pines analógicos en función de la posición del joystick

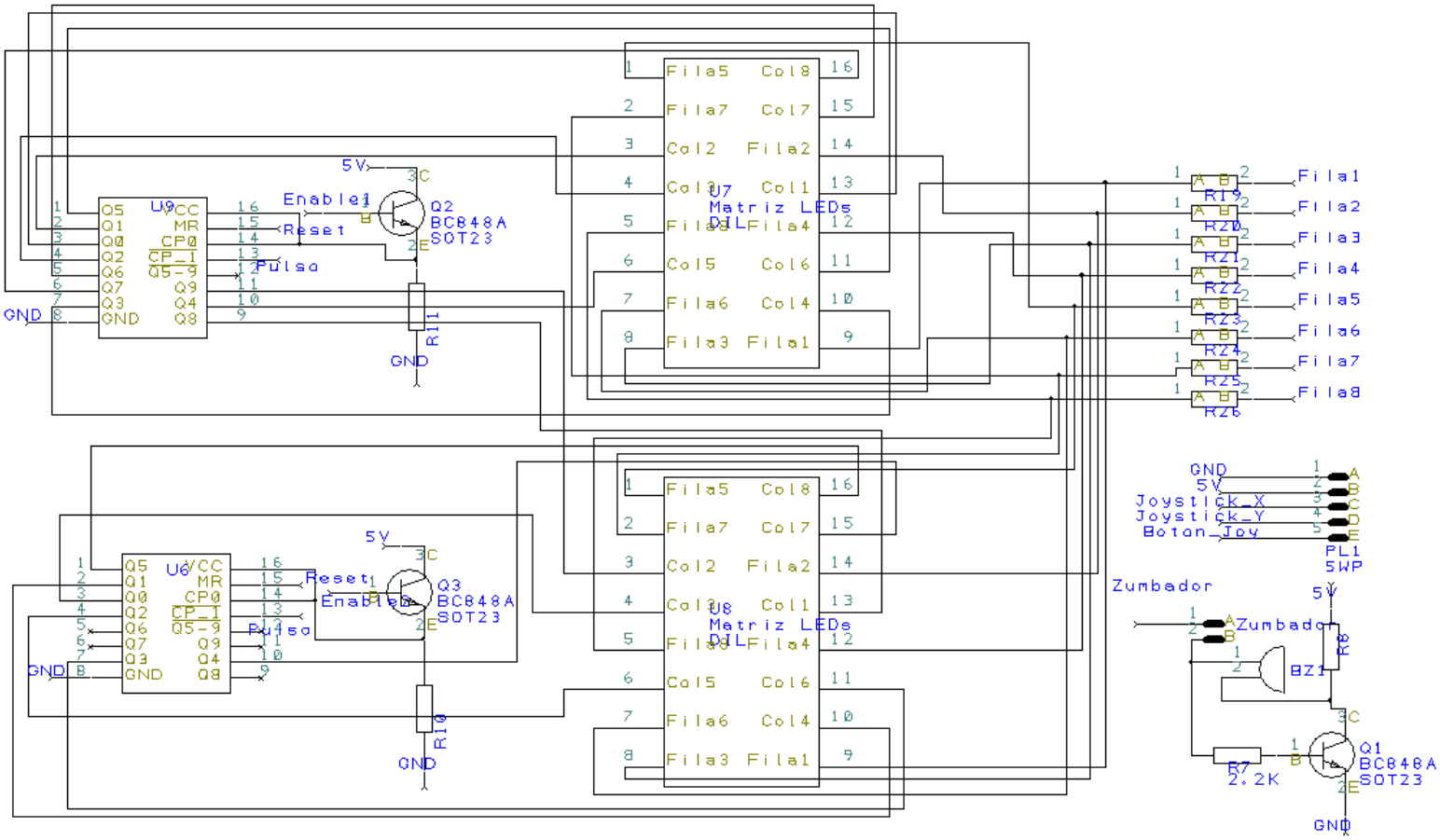


Figura 18. Circuito de control de los periféricos

Para el circuito de control del zumbador se utilizará un transistor NPN, de esta forma cuando se mande un "1" desde el micro el transistor se activará y el zumbador estará entre 5V y 0V. Además, se ha introducido un jumper entre el micro y el zumbador, así se podrá desconectar el zumbador sin necesidad de alterar el programa.

2.2.1 Listado de componentes

REFERENCIA	COMPONENTE	VALOR	EMPAQUETADO
R6	Resistencia	330Ω	SMD
R7	Resistencia	2.2kΩ	SMD
R8	Resistencia	470Ω	DSC
R19-26	Resistencia	330Ω	DSC
C1-2	Condensador cerámico	22pF	SMD
C3	Condensador electrolítico	47uF	SMD
C4,8	Condensador cerámico	0.1uF	DSC
C5,7	Condensador electrolítico	220uF	SMD
C6	Condensador cerámico	0.1uF	SMD
XTAL	Oscilador de cristal	20MHz	DIL
U1	Microcontrolador	PIC16F876A	SOIC
U4	Regulador	MC7800	SOL
U6,9	Contadores Johnson	74HC4017	SOIC
Q1-3	Transistor NPN	BC848	SMD
BZ1	Zumbador piezoeléctrico	-	DIL
PL1-2	Conector 5 pines	-	DSC
PL3-4	Conector 2 pines	-	DSC
LED1	LED rojo	-	DIL

Tabla 4. Lista de componentes del Tetris

2.3. Configuración del microcontrolador

2.3.1. Temporizadores

Para la implementación del Tetris en el microcontrolador PIC16F876A se utilizarán los 3 Timers disponibles: TMR0, TMR1 y TMR2. A continuación se mostrarán las distintas funciones para las que será necesario utilizar un temporizador:

- Módulo PWM (necesariamente TMR2).
- Duración de las notas de la melodía (duración semicorchea=100ms).
- Activación de la conversión A/D (aproximadamente 100ms).
- Tiempo para que caiga la pieza (1 segundo).
- Tiempo para el cambio de fila de los contadores Johnson.

Se utilizará en TMR1 para la conversión A/D, la caída de la pieza y la duración de las notas, se configurará de tal forma que salte el flag cada 100ms; para llegar hasta 1 segundo se empleará un contador auxiliar, de tal forma que cuando el contador llegue a 10 habrá pasado 1 segundo.

$$TMR1 = (65536 - N_{TMR1}) * P * T_i$$

Con un factor pre-divisor de 8 y una frecuencia de 20MHz el valor necesario de N_{TMR1} para que el Timer1 salte cada 100ms será 3036 (0x0BDC en hexadecimal).

```
T1CON=0x30;                             //Timer 1 como temporizador, factor pre-divisor=8
TMR1H=0x0B;
TMR1L=0xDC;                             //0.1 segundos
```

Para el cambio de fila de los contadores se utilizará el TMR0, ya que es el temporizador de menor capacidad. El valor que se debe introducir se obtiene de la siguiente ecuación:

$$TMR0 = (256 - N_{TMR0}) * P * T_i$$

Poniendo N_{TMR0} a 0 y con un factor pre-divisor de 64 el flag del TMR0 saltará cada 3.28ms, por lo que, teniendo en cuenta que la pantalla es de 16 filas, se mandará una imagen aproximadamente cada 50ms.

```
OPTION_REG=0xC3;                    //TMR0 como temporizador,factor de división 64
TMR0=0;
```

Por último, se configurará el Timer 2 para darle la frecuencia deseada al módulo PWM. El valor de PR2 necesario se obtiene de la siguiente ecuación:

$$TMR2 = P1 * (N_{PR2} + 1) * P2 * T_i$$

Donde P1 es el factor pre-divisor y P2 es el factor post-divisor; ambos factores serán iguales a 16.

```
T2CON=0xFF;                         //Post y pre-divisor=16
```

2.3.2. Módulo PWM

Para generar las notas de la melodía del Tetris se usará el módulo PWM del CCP2.

```
CCP2CON=0x0C;                       //CCP2 en modo PWM
```

El valor que se le dará a PR2 dependerá de la nota que se quiera que suene:

Nota	Frecuencia (Hz)	PR2
La menor	110	176
Si	123	156
Do	131	148
Re	147	130
Mi	165	116
Fa	175	110
Sol	196	98
La	220	86

Tabla 5. Valor de PR2 en función de la nota

Ya se ha comentado anteriormente que la duración de una semicorchea será de 100ms, por lo que se utilizará el Timer 1 para controlar la duración de las notas.

```
#define _La                 176                 //110 Hz
#define Si                  156                 //123 Hz
#define Do                  148                 //131 Hz
#define Re                  130                 //147 Hz
#define Mi                  116                 //165 Hz
#define Fa                  110                 //175 Hz
#define Sol                 98                  //196 Hz
#define La                  86                  //220 Hz

#define S                   1                   //Semicorchea
#define C                   2                   //Corchea
#define N                   4                   //Negra
#define B                   8                   //Blanca
```

2.3.3. Conversión A/D

Como ya se ha comentado anteriormente, será necesario utilizar el conversor A/D para el uso del joystick; para configurarlo se utilizarán los registros ADCON0 y ADCON1. Se configurará para que el tiempo de conversión venga dado por el oscilador RC interno, por lo que se deberán poner a 1 los bits 6 y 7 de ADCON0 (ADCS1 y ADCS0). Además, se deberá ir variando el valor del bit 3 de ADCON0 (CHS0), de esta forma la conversión analógica se realizará con los valores tomados en RA0 y RA1. Por último, se tiene que configurar cuáles son los bits analógicos y cuáles los digitales; para ello se utilizarán los 4 primeros bits de ADCON1 (PCFG3:PCFG0). Dado que se desea que RA0 y RA1 sean entradas analógicas y RA2 y RA5 digitales, PCFG3:PCFG0=0b0100.

```
ADCON1=0xC4;                //RA0, RA1 y RA3 analógicas, resto digitales
ADCON0=0xC1;                //Reloj de conversión oscilador RC interno
```

Se tomarán los dos bits más significantes: cuando esté a 0V estos serán 0b00; mientras que cuando esté a 5V los bits serán 0b11.

2.3.4. Interrupciones

Para el funcionamiento del microcontrolador se emplearán las interrupciones por los tres Timers y por la conversión A/D.

```
GIE=1;                                  //Habilitar interrupciones globales
PEIE=1;                                 //Habilitar interrupciones de periféricos
```

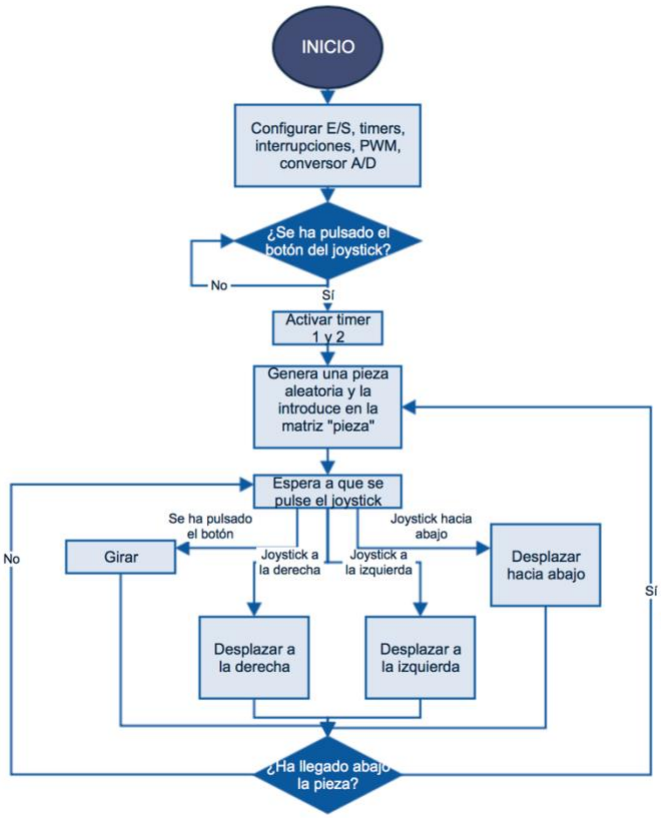
```
ADIE=1;                                 //Habilitar interrupciones por la conversión A/D
TMR0IE=1;                               //Habilitar interrupciones por Timer0
TMR1IE=1;                               //Habilitar interrupciones por Timer1
TMR2IE=1;                               //Habilitar interrupciones por Timer2
RBIE=0;                                 //Desactivar interrupciones por cambio en PORTB
```

Las interrupciones por los temporizadores no se habilitarán hasta que comience el Tetris.

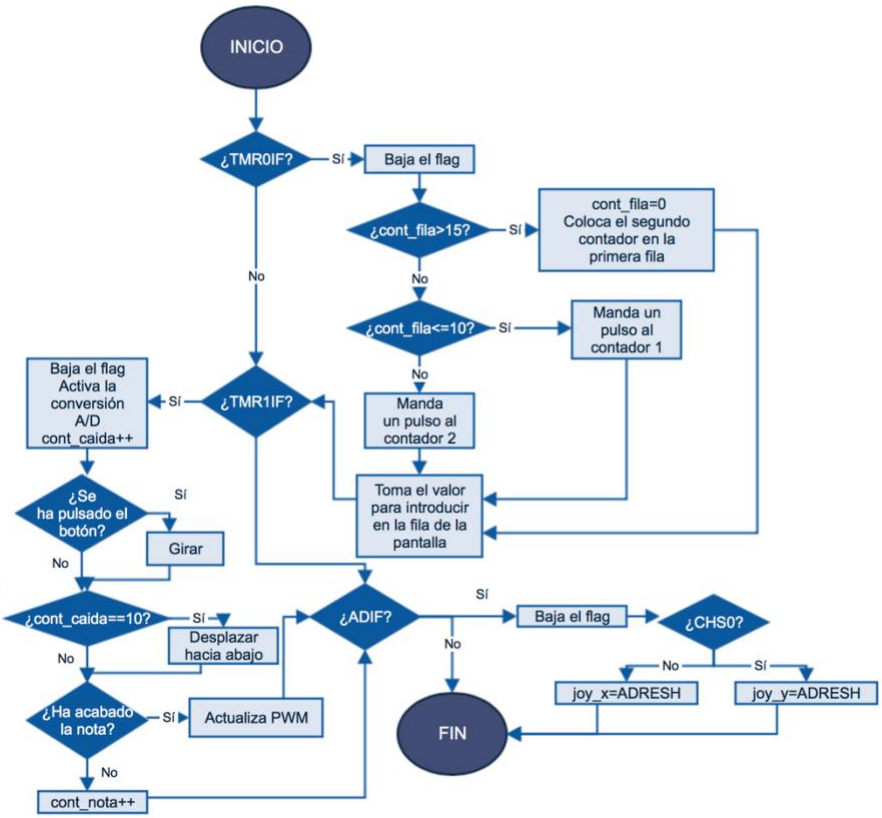
2.3.5. Funcionamiento

Se utilizará un vector de 19 filas (16 de la pantalla y 3 más encima) para guardar la matriz de la base del Tetris; por otro lado, el vector que contenga a la pieza tendrá 4 filas. Se emplearán las operaciones AND y OR para comprobar si la pieza choca con la base y sumar la pieza a la base, respectivamente. Además se utilizarán unas variables auxiliares a (columna) y b(fila) para guardar la posición de la pieza.

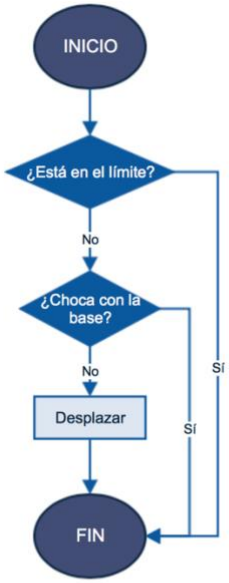
PROGAMA PRINCIPAL



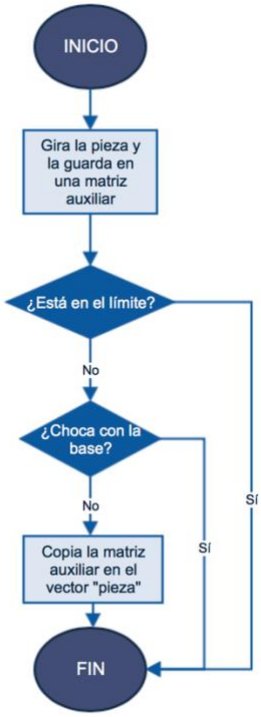
RUTINA DE ATENCIÓN A LAS INTERRUPCIONES



RUTINA DE DESPLAZAMIENTO DE PIEZA



RUTINA DE GIRO DE PIEZA



2.3.6. Programa fuente

CÓDIGO FUENTE

```
#include <htc.h>
#include <stdlib.h>
__CONFIG(WRT_OFF & WDTE_OFF & PWRTE_OFF & FOSC_HS & LVP_OFF & DEBUG_ON);
#define _XTAL_FREQ 20000000         //Oscilador Interno de 20MHZ

#define _La                 176                     //220 Hz
#define Si              157                 //246 Hz
#define Do              149                 //262 Hz
#define Re              132                 //293 Hz
#define Mi              116                 //329 Hz
#define Fa              110                 //349 Hz
#define Sol                 98                      //392 Hz
#define La              88                  //440 Hz

#define S               1                   //Semicorchea
#define C               2                   //Corchea
#define N               4                   //Negra
#define B               8                   //Blanca



const unsigned char mat_pieza[4][7]=        //Vector con las 7 piezas
```

```
{0b000000010,0b00000000,0b00000000,0b00000000,0b00000000,0b00000000,0b00000000,
 0b000000010,0b00000100,0b00000100,0b00001110,0b00000110,0b00001100,0b00000110,
 0b000000010,0b00000100,0b00000100,0b00000100,0b00001100,0b00000110,0b00000110,
 0b000000010,0b00000110,0b00000110,0b00000000,0b00000000,0b00000000,0b00000000};



const unsigned char Notas[]=
{Mi, Si, Do, Re, Mi, Re, Do, Si, _La, _La, Do, Mi, Re, Do, Si,
 Si, Si, Do, Re, Mi, Do, _La, _La, Re, Re, Fa, _La, Sol, Fa,
 Mi, Mi, Do, Mi, Re, Do, Si, Si, Si, Do, Re, Mi, Do, _La, _La, 0xFF};

const unsigned char Duracion[]=
{N, C, C, C, S, S, C, C, N, C, C, N, S, S, N, S, S, C, N,
 N, N, N, B, N, C, C, N, C, C, N, C, C, N, C, C, N, S, S,
 C, N, N, N, N, B};


unsigned char base[19];             //Matriz de la base, empieza vacía
unsigned char pieza[4];             //Matriz de la pieza
unsigned char temp[19];             //Matriz temporal

unsigned char i, j, cont_fila=0, cont_caida=0, choque_abajo=0, choque=0,
       nota=0, cont_nota=0, fila_pant,aux, a=5, b=0, desp, fila, girar=0,joy_y, joy_x;
unsigned int num_pieza;


void pantalla(void);
void cae(void);
void giro(void);
void musica(void);

                    ////////////////////INTERRUPCIONES//////////////////////////
static void interrupt isr(void){
       if(TMR0IF){                          //Salta flag TMR0 cada 1 ms
          TMR0IF=0;
          RC0=0;
          RC3=0;
          if(cont_fila>15){
                    cont_fila=0;           //Se pone el contador de filas a 0
                    RC3=1;
                    for(i=0;i<10;i++){     //Se introducen 5 pulsos al segundo contador
                      RA5=~RA5;            //para que vuelva a la primera fila
                      __delay_us(10);
                    }
                    RC3=0;
          }
          else if(cont_fila<=10){
                    RC0=1;                 //Enciende el primer contador
                    RA5=0;         //Se manda un pulso SOLO al primer contador para que avance
                    __delay_us(10);
```

```
                    RA5=1;
                    RC1=0;                          //Apaga el primer contador
          }
          else{
                    RC3=1;                 //Enciende el segundo contador
                    RA5=0;          //Se manda un pulso SOLO al segundo contador para que avance
                    __delay_us(10);
                    RA5=1;
                    RC3=0;                          //Apaga el segundo contador
          }
          if(cont_fila<b+1&&cont_fila>b-4){                      //Toma el valor que tiene que
                    if(a>6) fila=pieza[cont_fila+3-b]>>(a-7);    //tiene que introducir en la
                    else    fila=pieza[cont_fila+3-b]<<(7-a);    //fila de la matriz
                    fila_pant=(base[cont_fila+3]|fila);
          }
          else fila_pant=base[cont_fila+3];
          pantalla();
       }
       if(TMR1IF){                          //Salta flag TMR1 cada 100 ms
          TMR1IF=0;
          GO_DONE=1;                //Después de 100ms activa la conversión
          if(!RA2)   girar=1;       //Comprueba cada 100ms si se ha pulsado el botón
          cont_caida++;
          if(cont_caida==10){               //Ha pasado 1 segundo
                    cont_caida=0;
                    cae();                          //La pieza cae
                    for(i=0;i<4;i++){      //Por si hay un error al generar la pieza
                      if(pieza[i]!=0)      aux++;  //Si después de 1 segundo la matriz "pieza"
                                                   //está vacía
                    }
                    if(aux==0)     choque_abajo=1;       //Si la matriz está vacía sale del bucle
                                                         //para generar una nueva
          }
          if(cont_nota==Duracion[nota]){
                    cont_nota=0;
                    nota++;
                    musica();
          }
          else      cont_nota++;
       }
       if(ADIF){                    //Salta el flag de conversión A/D
          ADIF=0;
          if(CHS0)   joy_y=ADRESH;          //Canal 1
          else       joy_x=ADRESH;          //Canal 0
          CHS0=~CHS0;                       //Se cambia la conversión A/D de canal
       }
}

void main(void){                                    ////////////////////PROGRAMA
PRINCIPAL//////////////////////////
```

```
ADCON1=0xC4;                //RA0, RA1 y RA3 analógicas, resto digitales
ADCON0=0xC1;                //Reloj de conversión oscilador RC interno
TRISA=0x1F;                         //RA5 salida, resto entradas
TRISB=0x00;                         //PORTB salidas
TRISC=0x04;                         //PORTC salidas menos RC2
GIE=1;                                      //Activar interrupciones globales
PEIE=1;                                     //Activar interrupciones de periféricos
ADIE=1;
TMR0IE=0;                           //Activar interrupciones por Timer0
RBIE=0;                                     //Desactivar interrupciones por cambio en PORTB
OPTION_REG=0xC3;                    //TMR0 como temporizador, factor de división 64
TMR0=0;
T1CON=0x30;                         //Timer 1 como temporizador, factor pre-divisor=8
TMR1H=0x0B;
TMR1L=0xDC;                         //0.1 segundos
TMR1IE=0;                           //Interrupciones por Timer1 desactivadas
T2CON=0xFF;                         //Post y pre-divisor=16
TMR2=0xC3;                          //Salta el flag cada 10 ms
CCP2CON=0x0C;                       //CCP2 en modo PWM
srand ((int)TMR0);



while(RA2);                         //Se queda esperando hasta que se pulse el botón
RA5=1;
RC0=1;
__delay_us(20);
for(i=0;i<18;i++){                  //Se introducen 9 pulsos al primer contador
   RA5=~RA5;                //para que vaya a la última fila
   __delay_us(10);
}
RC0=0;
TMR1IE=1;                           //Habilita interrupciones por TMR1
TMR1ON=1;                           //Enciende el Timer 1
TMR2ON=1;                           //Enciende el TMR2
TMR0IE=1;                           //Habilita interrupciones por TMR0
musica();
srand ((int)TMR0);
while(1){                                   //Bucle infinito
   a=5;
   b=0;
   num_pieza=rand() % 7;            //Genera un número aleatorio entre 0 y 6
   for(i=0;i<4;i++){                //Introduce una pieza nueva
             pieza[i]=mat_pieza[i][num_pieza];
   }
   choque_abajo=0;
   while(!choque_abajo){            //Hasta que la pieza no llegue abajo
             if(joy_y==0x03)       cae();  //Si se quiere que caiga más rápido
             if(joy_x==0x03){              //Si se quiere mover a la derecha
               joy_x=0x01;                 //Para que no vuelva a saltar
               choque=0;
               if(a>6){                    //Se comprueba que no esté en el límite derecho
```

```
                        for(i=b;i<b+4;i++){
                                fila=pieza[i-b]>>(a-7);
                                if(fila&0x01)  choque=1;
                        }
                    }
                    if(choque==0){              //Si no está en el límite derecho
                        for(i=b;i<b+4;i++){     //Se comprueba que pueda moverse a la derecha
                                if(a>6){
                                        for(i=b;i<b+4;i++){
                                                fila=pieza[i-b]>>(a-6);
                                        }
                                }
                                else    fila=pieza[i-b]<<(6-a);
                                if(base[i]&fila)      choque=1;
                        }
                        if(choque==0)  a++;    //Si no hay choque se mueve
                    }
                }
                if(joy_x==0){           //Si se quiere mover a la izquierda
                    joy_x=0x01;                 //Para que no vuelva a saltar
                    choque=0;
                    if(a<5){                    //Se comprueba que no esté en el límite izquierdo
                        for(i=b;i<b+4;i++){
                                fila=pieza[i-b]<<(7-a);
                                if(fila&0x80)  choque=1;      //Se comprueba que no haya nada
                                                              //a la izquierda
                        }
                    }
                    if(choque==0){              //Si no está en el límite derecho
                        for(i=b;i<b+4;i++){     //Se comprueba que pueda moverse a la derecha
                                if(a>7){
                                        for(i=b;i<b+4;i++){
                                                fila=pieza[i-b]>>(a-7);
                                        }
                                }
                                else    fila=pieza[i-b]<<(8-a);
                                if(base[i]&fila)      choque=1;
                        }
                        if(choque==0)  a--;    //Puede moverse a la izquierda
                    }
                }
                if(girar==1)   giro();
                __delay_us(30);
            }
        }
}

void pantalla(void){
        if(cont_fila<10){                   //Para las 10 primeras filas
            RC0=1;                          //Enciende el primer contador
            RC3=0;                          //Apaga el segundo contador
```

```
          if(fila_pant&0x01)              RC4=0;
          else                            RC4=1;
          if(fila_pant&0x02)              RC6=0;
          else                            RC6=1;
          if(fila_pant&0x04)              RB4=0;
          else                            RB4=1;
          if(fila_pant&0x08)              RC5=0;
          else                            RC5=1;
          if(fila_pant&0x10)              RB0=0;
          else                            RB0=1;
          if(fila_pant&0x20)              RB2=0;
          else                            RB2=1;
          if(fila_pant&0x40)              RC7=0;
          else                            RC7=1;
          if(fila_pant&0x80)              RB1=0;
          else                            RB1=1;

       }
       else if(cont_fila>=10){            //Para las filas últimas 6 filas
          RC3=1;                          //Enciende el segundo contador
          RC0=0;                          //Apaga el primer contador
          if(fila_pant&0x01)              RC4=0;
          else                            RC4=1;
          if(fila_pant&0x02)              RC6=0;
          else                            RC6=1;
          if(fila_pant&0x04)              RB4=0;
          else                            RB4=1;
          if(fila_pant&0x08)              RC5=0;
          else                            RC5=1;
          if(fila_pant&0x10)              RB0=0;
          else                            RB0=1;
          if(fila_pant&0x20)              RB2=0;
          else                            RB2=1;
          if(fila_pant&0x40)              RC7=0;
          else                            RC7=1;
          if(fila_pant&0x80)              RB1=0;
          else                            RB1=1;
       }
       cont_fila++;
}
//////////////////////SUBPROGRAMA PARA LA CAÍDA DE LA PIEZA//////////////////////////////////////
void cae(void){
       choque_abajo=0;
       joy_y=0x01;                        //Para que no vuelva a saltar
       for(i=b;i<b+4;i++){
          if(a>6)    fila=pieza[i-b]>>(a-7);
          else       fila=pieza[i-b]<<(7-a);
          if(fila&base[i+1])       choque_abajo=1;       //Se comprueba que la pieza no tenga
                                                         //nada debajo
          if(i==18){
                    if(fila!=0)    choque_abajo=1;
```

```
         }
      }
      if(choque_abajo==0)  b++;           //Si no hay nada debajo se avanza una fila
      else if(choque_abajo==1){
         for(i=b;i<b+4;i++){
                  if(a>6) fila=pieza[i-b]>>(a-7);
                  else    fila=pieza[i-b]<<(7-a);
                  base[i]|=fila;
                  pieza[i-b]=0x00;
                  if(base[i]==0xFF)     base[i]=0x00;  //Si  una  línea  está  completa  se  borra

         }
         for(i=3;i<19;i++){
                  if(base[i-1]!=0x00){          //Si en la fila superior hay un bloque
                    if(base[i]==0x00){          //y esta fila no tiene ninguno
                          temp[i]=base[i-1];    //Se copia la fila superior
                          base[i]=temp[i];      //y se guarda
                          base[i-1]=0x00;               //Se borra la fila superior
                          i=3;                          //vuelve a empezar
                    }
                  }
         }
      }
      if(base[3]!=0){                                   //Cuando se haya llegado hasta arriba
         for(i=3;i<19;i++){
                  base[i]=0xFF;                         //Se enciende toda la pantalla
         }
         TMR1ON=0;                          //Se apaga el timer 1
         TMR2ON=0;                          //Se apaga el TMR2 para que deje de sonar la música
         while(1){                          //Se queda en bucle, se ha acabado el juego
                  __delay_us(10);
         }
      }
}


void giro(void){                            ////////////////////////////////SUBPROGRAMA PARA GIRAR
LA PIEZA//////////////////////////////////////////////
unsigned char trans[4];
      choque=0;
      girar=0;
                                          //Introduce en una matriz auxiliar la pieza girada
      trans[3]=pieza[0]&0x08|(pieza[1]&0x08)>>1|(pieza[2]&0x08)>>2|(pieza[3]&0x08)>>3;
      trans[2]=(pieza[0]&0x04)<<1|pieza[1]&0x04|(pieza[2]&0x04)>>1|(pieza[3]&0x04)>>2;
      trans[1]=(pieza[0]&0x02)<<2|(pieza[1]&0x02)<<1|pieza[2]&0x02|(pieza[3]&0x02)>>1;
      trans[0]=(pieza[0]&0x01)<<3|(pieza[1]&0x01)<<2|(pieza[2]&0x01)<<1|pieza[3]&0x01;

      for(i=b;i<b+4;i++){               //Comprueba si choca con la base o está en algún límite
         if(a>6)    fila=trans[i-b]>>(a-7);
         else       fila=trans[i-b]<<(7-a);
         if(fila&base[i])  choque=1;      //La pieza girada choca con la base
```

```
            if(fila&0x01)     choque=1;             //La pieza girada está en el límite derecho
            if(fila&0x80)     choque=1;             //La pieza girada está en el límite izquierdo
        }
        if(choque==0){
            for(i=0;i<4;i++)  pieza[i]=trans[i];    //Si no choca gira la pieza
        }
        while(!RA2);          //Se queda esperando hasta que se deje de pulsar el botón

}


void musica(void){
        if(Notas[nota]==0xFF)       nota=0;               //Cuando llegue al final vuelve a empezar
        PR2=Notas[nota];
        CCPR2L=PR2/2;  //Ciclo de trabajo del 50%
}
```

2.4. Montaje

2.4.1. Diseño de la PCB

En la Figura se muestra el diseño de la placa de circuito impreso desarrollada para el Tetris. El diseño se ha realizado utilizando el programa DesignSpark PCB. Dado que los pines de las matrices LEDs estaban desordenados, no se ha podido realizar las conexiones en una sola capa; por lo que se han utilizado tanto la capa superior como la inferior. La mayoría de los componentes que se han utilizado son de empaquetado SMD, ya que estos ocupan mucho menos espacio que los de orificio pasante; aun así, se han tenido que emplear varios componentes de orificio pasante con el fin de disminuir la cantidad de vías. Se ha tenido en cuenta en el diseño el trazado de las pistas de alimentación utilizando un mayor grosor, así como la utilización de un plano de masa.

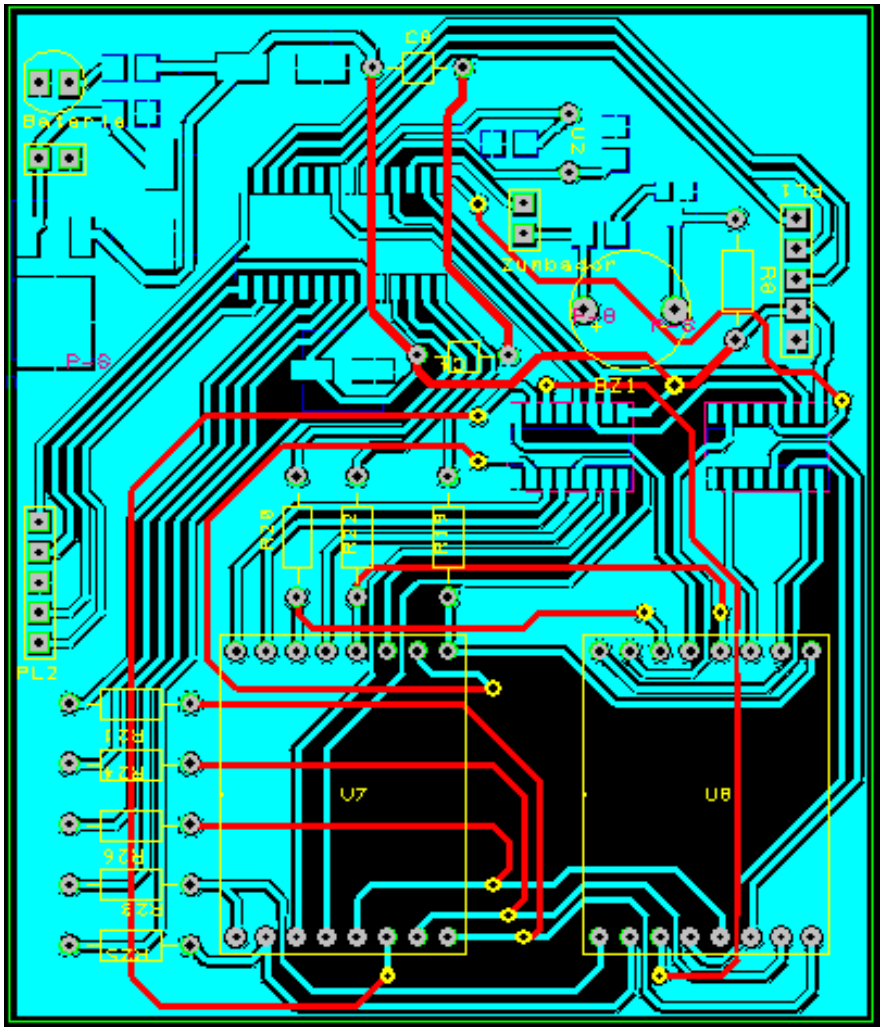


Figura 19. Diseño de la PCB del Tetris

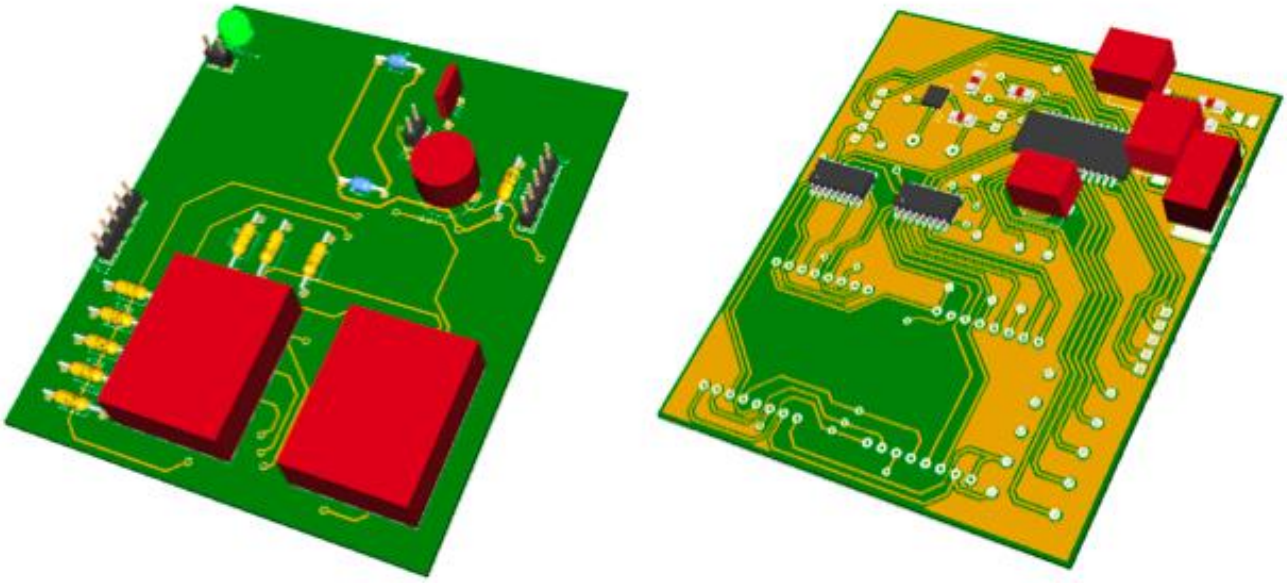


Figura 20. Diseño 3D de la PCB del Tetris

2.4.2. Diseño mecánico

La carcasa del Tetris se ha dividido en dos partes: la base y la tapa. La tapa deberá tener unos agujeros para poder utilizar el joystick y ver la pantalla.

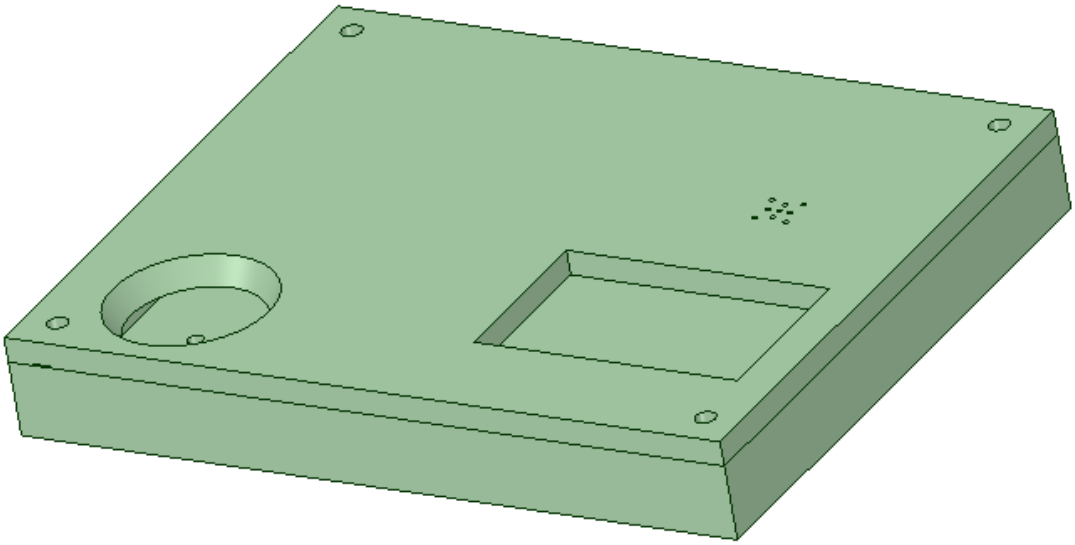


Figura 21. Diseño 3D de la carcasa del Tetris

3.- Medidor de distancia

3.1 Introducción

El medidor de distancia digital mide la distancia que hay hasta el objeto que tenga delante (normalmente una pared). Para la implementación se utilizará un sensor de ultrasonidos, una pantalla LCD (donde el usuario podrá ver la distancia que hay en cm) y un botón (el cual se deberá pulsar para que se mida la distancia).

3.2. Diseño esquemático

El circuito de control de la alarma se basa en el microcontrolador **PIC16F876A**, que trabaja a 5V, además se introducirá un oscilador externo de 20MHz, dado que este microcontrolador no tiene ningún oscilador interno.

Se utilizará una pila de 8.4V para alimentar la PCB, por lo que será necesario el uso de un regulador que lo convierta a 5V.

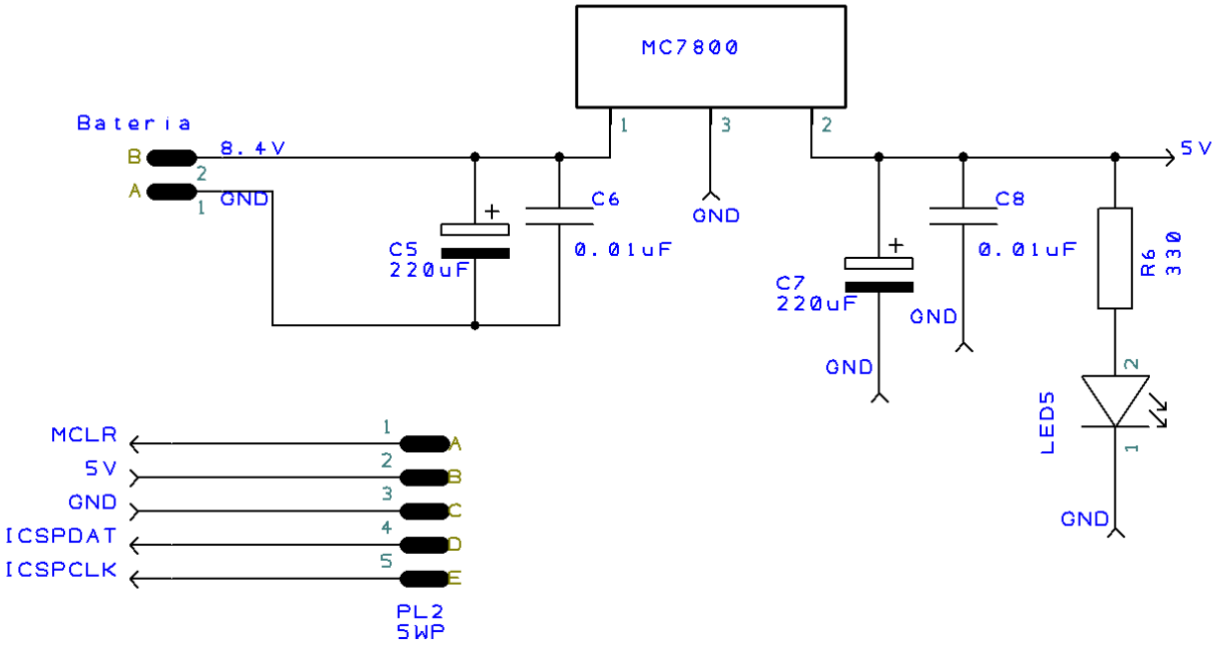


Figura 23. Diseño esquemático de la alimentación del medidor de distancia

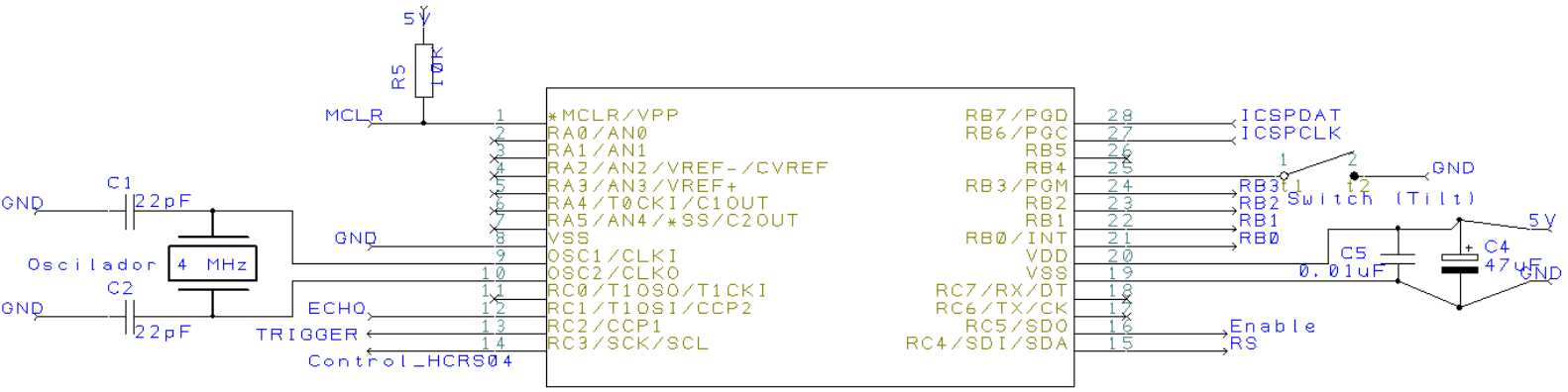


Figura 22. Diseño esquemático del microcontrolador del medidor de distancia

El pulsador irá conectado a RB3, que está configurado para que use la resistencia pull-up, de esta forma cuando el botón esté sin pulsar RB3 estará a 5V; mientras que cuando se pulse el circuito se cerrará y RB3 se quedará a 0V.

El sensor de ultrasonidos está compuesto por 4 pines: 2 de alimentación (5V y GND); el pin *trigger*, que se le mandará un pulso de 10us; y el pin *echo* que será el que mande un pulso al microcontrolador, la anchura de este pulso dependerá de la distancia a la que esté el objeto del sensor.

Figura 24. Sensor de ultrasonidos HC-SR04

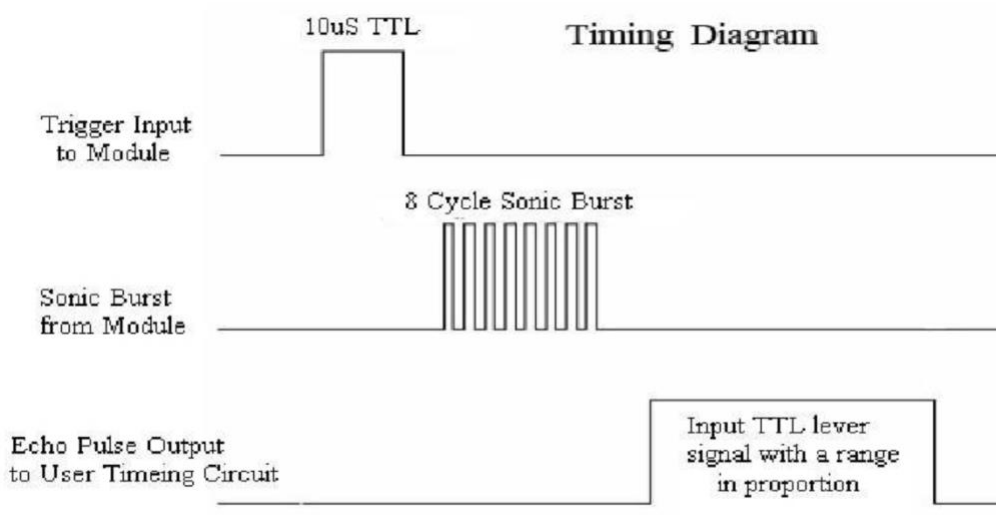


Figura 25. Funcionamiento del sensor de ultrasonidos

Para el control de la pantalla LCD se utilizarán pines: *RS* y *Enable*, ya que el pin R/W estará cortocircuitado a tierra, de esta forma estará siempre en modo *write*. Por otro lado, el bus de datos será de 4 bits (DB4:DB7) e irán conectados a los pines RB0:RB3 del microcontrolador.

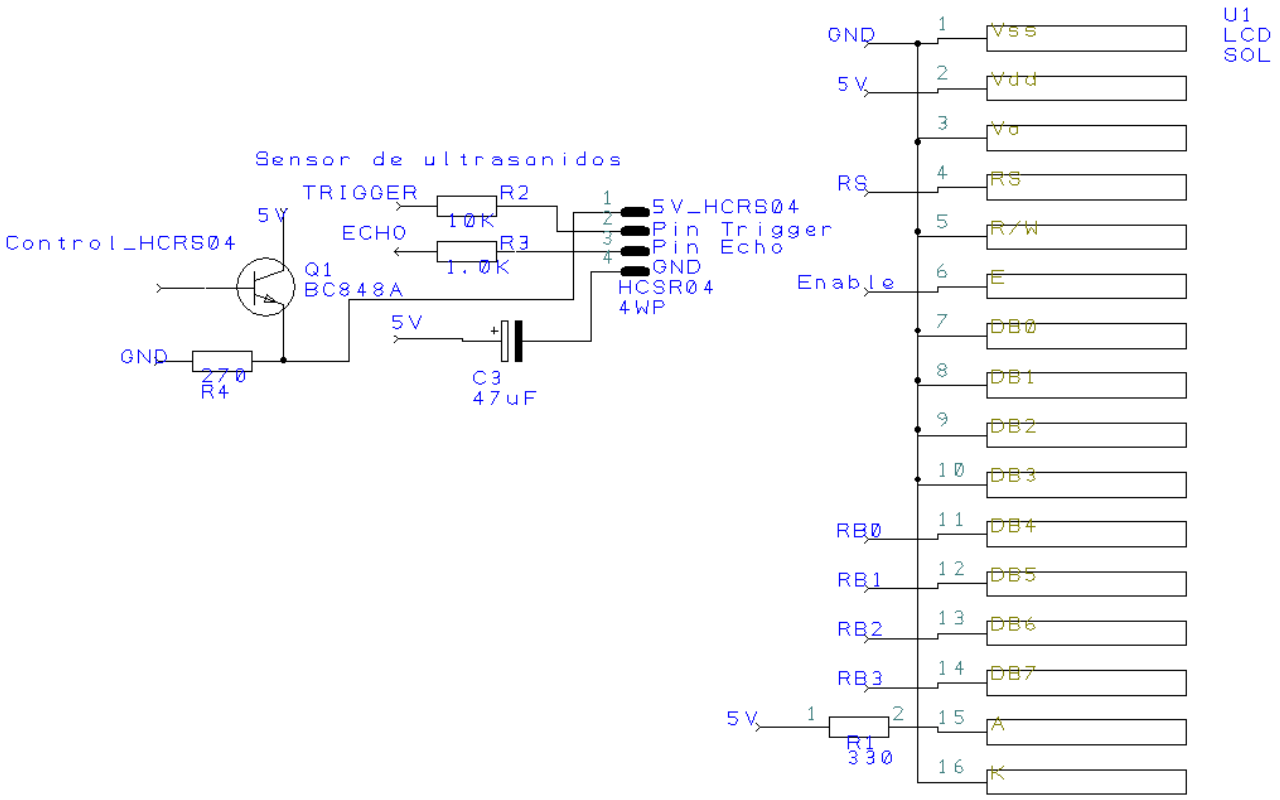


Figura 26. Diseño esquemático del control de los periféricos del medidor de distancia

Para el circuito de control del zumbador se utilizará un transistor NPN, de esta forma cuando se mande un “1” desde el micro el transistor se activará y el zumbador estará entre 5V y 0V. Además, se ha introducido un jumper entre el micro y el zumbador, así se podrá desconectar el zumbador sin necesidad de alterar el programa.

3.2.1 Listado de componentes

REFERENCIA	COMPONENTE	VALOR	EMPAQUETADO
R1,6	Resistencia	330Ω	SMD
R2,5	Resistencia	10kΩ	SMD
R3	Resistencia	1kΩ	SMD
R4	Resistencia	270Ω	SMD
C1-2	Condensador cerámico	22pF	SMD
C3-4	Condensador electrolítico	47uF	SMD
C5,7,9	Condensador cerámico	10nF	SMD
C6,8	Condensador electrolítico	220uF	SMD
XTAL	Oscilador de cristal	20MHz	DIL
U1	LCD	-	SOL
U3	Microcontrolador	PIC16F876A	SOIC
U5	Switch	-	DIL
U6	Regulador	MC7800	SOL
Q1	Transistor NPN	BC848	SMD
PL1-2	Conector 2 pines	-	DSC
PL3	Sensor ultrasonidos	HC-SR04	DSC
PL4	Conector 5 pines	-	DSC
LED1	LED rojo	-	DIL

Tabla 6. Lista de componentes del medidor de distancia

3.3. Configuración del microcontrolador

Se configurará el microcontrolador de tal forma que esté en modo *sleep* hasta que se pulse el botón, que medirá la distancia a la que está el objeto. Una vez se deje de pulsar el botón el PIC volverá a entrar en modo *sleep*.

3.3.1. Módulo CCP2

Se ha conectado el pin *Echo* del sensor de ultrasonidos al pin RC1 del microcontrolador, por lo que se utilizará el módulo CCP2 en modo captura (se irá alternando entre captura cada flanco de bajada y subida) para medir la distancia.

```
CCP2CON=0x05;          //Configura CCP2 en modo captura cada flanco de subida
CCP2CON=0x04;          //Configura CCP2 en modo captura cada flanco de bajada
```

La anchura del pulso que mandará en sensor de ultrasonidos por el pin *Echo* será el tiempo que tardan los pulsos que ha enviado en llegar al objeto que tenga delante y volver. Por lo que para obtener la distancia a la que está el objeto se deberá medir la anchura del pulso que llega desde el pin *Echo*. Primero se configurará CCP2 para que salte el flag cuando llegue un flanco de subida; cuando ocurra la interrupción se configurará CCP2 para que salte el flag cuando llegue un flanco de bajada y se pondrá a 0 el Timer 1, que irá contando el tiempo hasta que que ocurra una nueva interrupción. El tiempo ha pasado se obtendrá de la siguiente ecuación:

$$TMR1 = (65536 - N_{TMR1}) * P * T_i$$

Donde el pre-divisor será igual a 8. Una vez sabido el tiempo que ha transcurrido, para obtener la distancia en centímetros:

$$D = \frac{T * 340}{2} * 100$$

3.3.2. Timer 2

Se empleará el Timer 2 para controlar la frecuencia a la que se manda un pulso al trigger del sensor de ultrasonidos. La frecuencia deseada es de 2 pulsos por segundo, es decir, que mande un pulso cada 500ms; para ello se configurará el TMR2 para que salte el flag cada 10ms. Será necesario utilizar un contador auxiliar que cuando llegue a 50 (500ms) mande el pulso.

$$TMR2 = P1 * (N_{PR2} + 1) * P2 * T_i$$

Con P1 y P2 igual a 16, el valor de PR2 necesadio para que salte el flag cada 10ms es de 194.

```
T2CON=0x7B;                 //TMR2 apagado, factores pre- y post-divisores=16
TMR2=194;                   //Salta cada 10ms
```

3.3.3. Resistencias pull-up

Ya se ha comentado con anterioridad que será necesario activar las resistencias pull-up de PORTB para el correcto funcionamiento del *switch*. Estas resistencias se activan modificando el registro *OPTION_REG*, poniendo a 0 el bit 7 se activarán.

```
OPTION_REG=0b01111111;      //Activar PORTB Pull-up
```

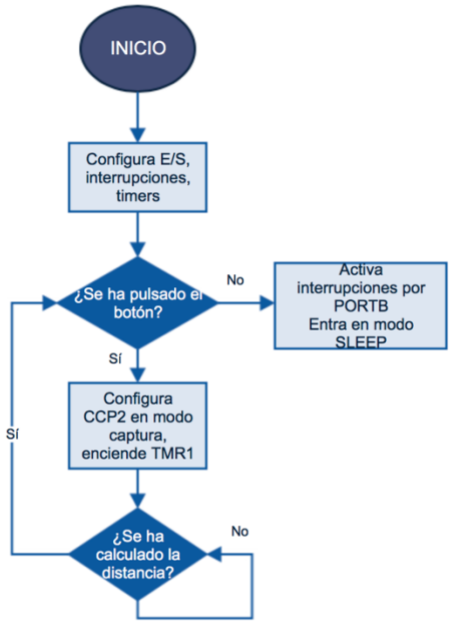
3.3.4. Interrupciones

Para el funcionamiento del medidor se utilizarán las interrupciones por CCP2 (para medir el tiempo transcurrido desde que se ha enviado el pulso al objeto hasta que ha llegado), por TMR2 (para controlar la frecuencia a la que se envía un pulso al *trigger*) y por cambio de estado en RB4:RB7 (para que el PIC deje de estar en modo *sleep* cuando se pulse el botón.

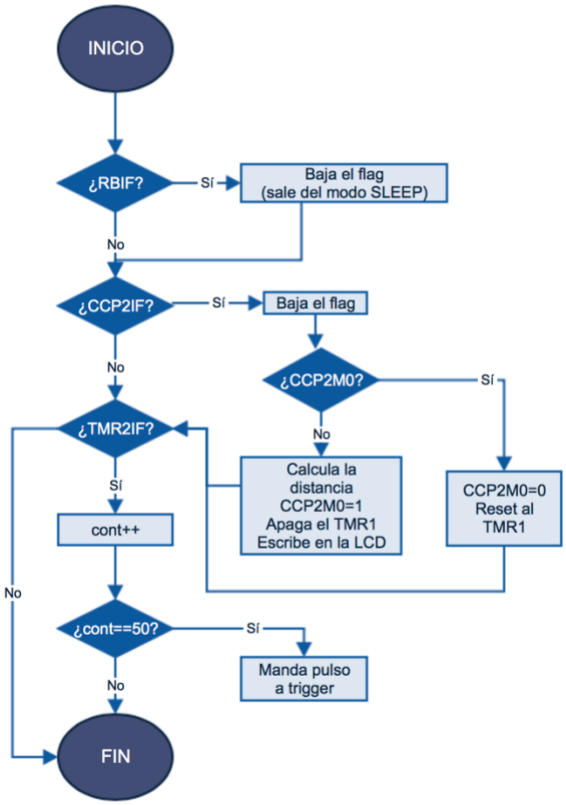
```
GIE=1;                      //Habilitar interrupciones globales
PEIE=1;                     //Habilitar interrupciones por periféricos
TMR1IE=0;                   //Deshabilitar interrupciones por Timer1
TMR2IE=1;                   //Habilitar interrupciones por Timer2
CCP2IE=1;                   //Habilitar interrupciones por módulo CCP2
```

3.3.5. Funcionamiento

PROGRAMA PRINCIPAL



RUTINA DE ATENCIÓN A LAS INTERRUPCIONES



3.3.6. Programa fuente

CÓDIGO FUENTE

```
#include <htc.h>
#include "lcd.h"                              //Incluimos librería de la LCD
#include "stdio.h"
__CONFIG(WRT_OFF & WDTE_OFF & PWRTE_OFF & FOSC_HS & LVP_OFF);
#define _XTAL_FREQ 20000000          //Oscilador Interno de 20MHZ

float   dist, aux;
unsigned char         espera=1,i=0,cont=0;

void pantalla(void);
void putch(unsigned char c){
        lcd_putch(c);
}

static void interrupt ISR(void){
        if(RBIF)      RBIF=0;
```

```
        if(CCP2IF){                         //Comprueba si se ha generado interrupción por CCP2
           CCP2IF=0;                //Baja le flag de interrupción
           if(CCP2M0){                      //Comprueba si es el flanco de subida
                     CCP2M0=0;              //Configura CCP2 en modo captura cada flanco de bajada
                     TMR1H=0;                       //Se ponen a 0 los contadores del
                     TMR1L=0;                       //Timer 1
           }
           else{                                    //Es flanco de bajada
                     dist=(CCPR2H*256+CCPR2L)*0.0272;     //Se calcula la distancia
                     TMR1H=0;
                     TMR1L=0;
                     CCP2M0=1;                      //Se configura para flanco de subida
                     espera=0;                      //Ha acabado de calcula la distancia
                     TMR1ON=0;                      //Apaga el TMR1
                     pantalla();                    //Escribe la distancia en la pantalla
           }
        }
        if(TMR2IF){                         //Salta el flag de interrupción del TMR2
           TMR2IF=0;
           cont++;

           if(cont==50){                    //Han pasado 500ms
                     RC3=1;                         //Enciende el sensor
                     __delay_us(20);                //Se le da un pequeño margen
                     RC2=1;
                     __delay_us(10);                //Manda un pulso de 10us al trigger
                     RC2=0;
                     cont=0;                        //Resetea el contador
                     TMR1ON=1;                      //Enciende el TMR1 para poder tomar la distancia
                     TMR2ON=0;                      //Apaga el timer 2 hasta que tome la distancia
           }
        }
}


void main(void){
        RC5=1;
        lcd_init();
        lcd_clear();
        OPTION_REG=0b01111111;      //Activar PORTB Pull-up
        T1CON=0x31;                 //TMR1 con pre-divisor=8;
        T2CON=0x7B;                 //TMR2 apagado, factores pre- y post-divisores=16
        TMR2=194;                   //Salta cada 10ms
        TRISB=0xF0;                 //RB4, RB6 y RB7 entradas
        TRISC=0x02;                 //RC1 entrada
        GIE=1;                      //Habilitar interrupciones globales
        PEIE=1;                     //Habilitar interrupciones por periféricos
        TMR1IE=0;                   //Deshabilitar interrupciones por Timer1
        TMR1IF=0;
        TMR2IE=1;
        CCP2IE=1;                   //Habilitar interrupciones por módulo CCP2
```

```
        while(1){
           if(RB4){          //Si no se está pulsando el botón
                     RBIE=1;        //Habilita las interrupciones por cambio de estado en RB4:RB7
                     RBIF=0;                    //Baja el flag de interrupción
                     RC3=0;                     //Se apaga el sensor de ultrasonidos
                     CCP2CON=0;
                     TMR1ON=0;
                     TMR2ON=0;
                     SLEEP();                   //Pasa a modo de bajo consumo
                     __delay_us(20);
           }
           while(!RB4){                  //Mientras el botón esté pulsado
                     RBIE=0;             //Se desactivan las interrupciones por cambio
                                         //de estado en RB4:RB7
                     CCP2CON=0x05;       //Configura CCP2 en modo captura cada flanco de subida
                     TMR2ON=1;           //Enciende el TMR2
                     TMR1H=0;            //Pone a 0 los contadores del TMR1
                     TMR1L=0;
                     TMR1ON=1;           //Enciende TMR1
                     espera=1;           //Se utiliza una variable auxiliar que indicar·
                     while(espera);      //cuándo ha acabado de calcular la distancia
                     TMR2ON=1;
           }
        }
}



void pantalla(void){
        lcd_init();
        lcd_clear();
        lcd_goto(0x040);                 //Pasa a la segunda fila
        printf("%d cm", dist);           //Escribe la distancia
}
```

3.4. Montaje

En la Figura se muestra el diseño de la placa de circuito impreso desarrollada para el medidor de distancia digital. El diseño se ha realizado utilizando el programa DesignSpark PCB. Con el fin de disminuir el tamaño de la placa casi todos los componentes son de empaquetado SMD, ya que estos ocupan un espacio mucho menor que los de orificio pasante. Además, gracias a la simplicidad del circuito, ha sido posible implementar las pistas en usa sola cara de la placa, por lo que el coste tanto

económico como de tiempo diminuye. Se ha tenido en cuenta en el diseño el trazado de las pistas de alimentación utilizando un mayor grosor, así como la utilización de un plano de masa.

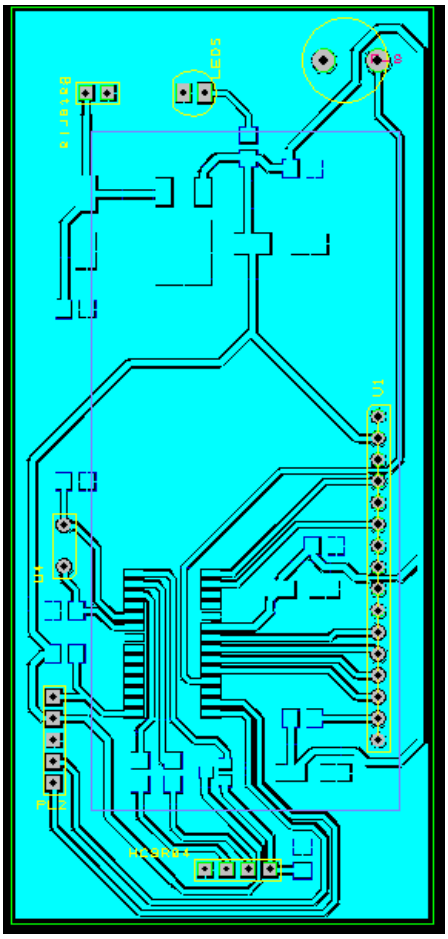


Figura 27. Diseño de la PCB del medidor de distancia

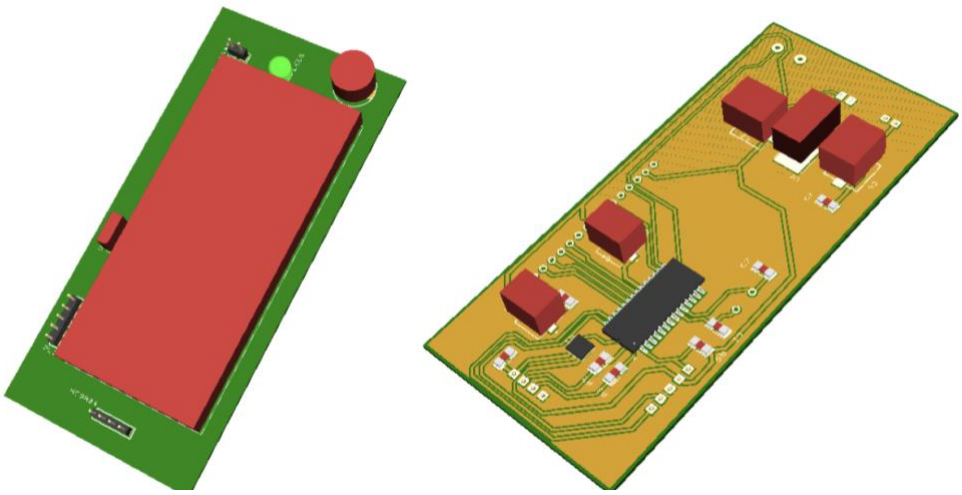


Figura 28. Diseño 3D de la PCB del medidor de distancia

Conclusiones

El primer proyecto era en teoría el que más tiempo iba a llevar, ya que era el que más sensores y actuadores tenía; además, al ser el primer proyecto hubo algunos fallos por falta de experiencia. El segundo ha sido el más complicado de los tres debido a la complejidad de la programación y ciertos fallos con los contadores Johnson. Al ser más complejo de lo esperado se tardó demasiado en conseguir que funcionase y eso causó una falta de tiempo en el último proyecto, por lo que no fue posible terminarlo.

En cierta medida el resultado obtenido ha sido el deseado: la idea principal del proyecto era realizar una central de alarmas, una máquina de juegos y un monedero electrónico. El primero se ha cumplido; el segundo, en cierta forma también, dada la complejidad de la programación del Tetris no ha sido posible implementar más juegos, de todas formas, se podría decir que con el Tetris es suficiente. Por último, debido a que el segundo proyecto tomó más tiempo de lo que se esperaba, se cambió el monedero electrónico por uno más simple, el medidor de distancia; aun así, no ha habido tiempo suficiente y no se ha conseguido que el medidor funcione correctamente, ya que sí que mide bien la distancia, pero no la imprime en la pantalla LCD.

Líneas futuras

Dado que todos los proyectos realizados son prototipos, todos necesitarían alguna mejora para poder tener una aplicación real. En la central de alarmas de podría implementar un modo *sleep*, ya que la mayoría del tiempo estará esperando, de esta forma el consumo de energía bajaría drásticamente. Además, sería necesario introducir más sensores, ya que este proyecto se ha realizado para una maqueta de dos habitaciones.

En el Tetris se podría arreglar algún pequeño fallo que aparece de vez en cuando, para mejorar de esta forma la experiencia del usuario. Además, sería interesante introducir algún juego más como el *Snake* o el *Pong* (para este último se necesitaría un joystick más, puesto que es de dos jugadores); de esta forma se tendría una máquina de juegos portátil (tamaño reducido).

Por último, una posible mejora para el medidor de distancia sería su rango de trabajo; dado que el sensor HC-SR04 no es capaz de detectar un objeto a más de 4m, el rango de trabajo del medidor se reduce bastante. Con un sensor de mayor rango el medidor se podría utilizar en muchas más aplicaciones.

A1.- Registros de Funciones Especiales (SFRs)

A1.1.- STATUS

Registro STATUS (direcciones 03h, 83h, 103h, 183h)

R/W-0	R/W-0	R/W-0	R-1	R-1	R/W-x	R/W-x	R/W-x
IRP	RP1	RP0	TO#	PD#	Z	DC	C
bit 7							bit 0

Valores de los bits después de un reset
R = el bit se puede leer
W = el bit se puede escribir
x = Valor desconocido

Nota:
TO# = $\overline{TO}$

bit 7 IRP: Selecciona el Banco de Memoria de datos en el direccionamiento indirecto.
0 = Bancos 0 y 1 (00h - FFh) (256 bytes)
1 = Bancos 2 y 3 (100h - 1FFh) (256 bytes)

bit 6-5 RP1:RP0: Seleccionan el Banco de Memoria de datos en el direccionamiento directo.
00 = Banco 0 (00h - 7Fh) (128 bytes)
01 = Banco 1 (80h - FFh) (128 bytes)
10 = Banco 2 (100h - 17Fh) (128 bytes)
11 = Banco 3 (180h - 1FFh) (128 bytes)

bit 4 TO# *(Time-out)*: Indicador de desbordamiento del perro guardián WTD *(**W**atch**d**og **T**imer)*
0 = Cuando se desborda el WTD
1 = Después de un reset por encendido (*power-up*) y con las instrucciones CLRWDT y SLEEP

bit 3 PD# (*Power-down*): Indicador de modo de bajo consumo.
0 = Cuando entra en bajo consumo con la instrucción SLEEP
1 = Después del encendido o con la instrucción CLRWDT

bit 2 Z *(Zero)*: Indicador de cero
1 = Si el resultado de una operación aritmética o lógica es cero
0 = Si el resultado de una operación aritmética o lógica es no es cero

bit 1 DC *(Digit carry/borrow)*: Indicador de acarreo o préstamo auxiliar en las operaciones aritméticas de suma o resta (instrucciones ADDWF, ADDLW, SUBLW, SUBWF)
1 = Si hay acarreo del bit 3 al 4 en el resultado de una operación aritmética de suma binaria o si no hay préstamo en una operación de resta
0 = Si no hay acarreo en la suma o si hay préstamo del bit 4 al bit 3 en una operación de resta

bit 0 C *(Carry/borrow)*: Indicador de acarreo o préstamo en las operaciones aritméticas de suma o resta (instrucciones ADDWF, ADDLW, SUBLW, SUBWF)
1 = Si hay acarreo en el resultado de una operación aritmética de suma binaria o si no hay préstamo en una operación de resta
0 = Si no hay acarreo en la suma o si hay préstamo en una operación de resta

A1.2.- Registro OPTION_REG

Registro OPTION_REG (direcciónes 0x81h, 0x181h de la memoria de datos RAM)

R/W-1	RW-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
RBPU#	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

bit 7 RBPU#: Habilita/deshabilita las resistencias internas de Pull-up del puerto B.

1 = PORTB pull-ups deshabilitadas
0 = PORTB pull-ups habilitadas

bit 6 INTEDG: Selección del planco para la generación de la interrupción externa por RB0.

1 = Interrupción externa RB0/INT por flanco de subida
0 = Interrupción externa RB0/INT por flanco de bajada

bit 5 T0CS: configura al Timer0 como temporizador (T0CS = 0) o como contador (T0CS = 1).
bit 4 T0SE: configura el flanco de la señal externa con el que se incrementa el Timer0 si ha sido programado como contador (T0SE = 0 para flanco de subida y TS0E = 1 para flanco de bajada).

bit 3 PSA: asigna el pre-divisor al Timer0 (PSA = 0) o al Perro Guardián WDT (PSA =1)

Bits 2, 1 y 0 PS2:PS1:PS0: programan el factor de división del pre-divisor.

PS2 PS1 PS0	Factor división para TMR0	Factor división para WDT
000	2	1
001	4	2
010	8	4
011	16	8
100	32	16
101	64	32
110	128	64
111	256	128

A1.3.- T1CON

Registro T1CON (dirección 0x10 de la memoria de datos RAM)

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
-	-	T1CKPS1	T1CKPS0	T1OSCEN	T1SYNC#	TMR1CS	TMR1ON
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

bit 7-6 No implementados: se leerían como '0'.

bit 5-4 T1CKPS1:T1CKPS0: Bits de selección del factor de división del pre-divisor para la entrada de reloj del Timer1

T1CKPS1-T1CKPS0	Factor de división
00	1
01	2
10	4
11	8

bit 3 T1OSCEN: Bit de habilitación (enable) del oscilador externo (T1OSC) del TMR1
1 = Oscilador habilitado
0 = Oscillador apagado

bit 2 T1SYNC: Bit de control de la sincronización del reloj externo del Timer1
Cuando TMR1CS=1:
1 = No sincroniza la entrada de reloj externo
0 = Sincroniza la entrada de reloj externo
Cuando TMR1CS = 0:
Este bit se ignora, pues TIMER1 usa el reloj interno que ya está sincronizado.

bit 1 TMR1CS: Bit de selección de reloj para el TIMER 1
1 = Reloj externo del pin RC0/T1OSO/T1CKI (flancos de subida)
0 = Reloj interno (FOSC/4)

bit 0 TMR1ON: Bit para arranque/paro del Timer1.
1 = Timer1 cuenta pulsos
0 = Timer1 parado

A1.4.- T2CON

Registro T2CON (dirección 0x12 de la memoria de datos RAM)

U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
-	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0
bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0

bit 7 No implementado: Se lee como 0

bit 6:3 TOUTPS3:TOUTPS0: Bits de selección del factor de división del post-divisor del Timer2

TOUTPS3-TOUTPS0	Factor de división
0000	1
0001	2
0010	3
0011	4
0100	5
0101	6
0110	7
0111	8
1000	9
1001	10
1010	11
1011	12
1100	13
1101	14
1110	15
1111	16

bit 2 TMR2ON: Bit de paro/arranque del TMR2
1 = Timer2 on (habilitado)
0 = Timer2 off (no habilitado)

bit 1:0 T2CKPS1:T2CKPS0: Bits de selección del factor de división del pre-divisor del Timer2

T2CKPS1-T2CKPS0	Factor de división
00	1
01	4
1x	16

A1.5.- Registro INTCON

Registro INTCON (direcciones 0Bh, 8Bh, 10Bh, 18Bh)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE	PEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF
bit 7							bit 0

bit 7 GIE *(Global Interrupt Enable):* Habilitación del sistema de interrupciones del microcontrolador
1 = Se habilitan todas las interrupciones no enmascaradas
0 = Se inhabilitan todas las interrupciones.

bit 6 PEIE *(Peripheral Interrupt Enable):* Habilitación de las interrupciones de los periféricos (en registros PIE1,PIR,PIE2, PIR2)
1 = Se habilitan todas las interrupciones de los periféricos no enmascaradas
0 = Se inhabilitan todas las interrupciones de periféricos.

bit 5 T0IE *(TMR0 Overflow Interrupt Enable):* Habilitación de la interrupción por desbordamiento del Timer0
1 = Se habilita la interrupción
0 = Se inhabilita la interrupción

bit 4 INTE *(RB0/INT External Interrupt Enable)*: Habilitación de la interrupción externa
1 = Habilita la interrupción externa por RB0/INT
0 = Inhabilita la interrupción externa por RB0/INT

bit 3 RBIE *(RB Port Change Interrupt Enable)*: Habilitación de la interrupción por cambio en cualquier terminal RB4-RB7
1 = Se habilita la interrupción
0 = Se inhabilita la interrupción

bit 2 T0IF *(TMR0 Overflow Interrupt Flag)*: Señalizador del desbordamiento del timer0
1 = El registro TMR0 se ha desbordado (se debe limpiar, desactivar por software)
0 = el registro TMR0 no se ha desbordado

bit 1 INTF *(RB0/INT External Interrupt Flag)*: Señalizador de petición de interrupción externa
1 = Se produce petición de interrupción (se debe desactivar por software)
0 = No se produce petición de interrupción externa

bit 0 RBIF (RB Port Change Interrupt Flag): Señalizador de cambio en algún terminal RB4-RB7
1 = Al menos un terminal RB7-RB4 ha cambiado de estado (se debe desactivar por software).
0 = Ningún terminal RB7-RB4 ha cambiado de estado

A1.6.- Registro PIE1

Registro PIE1 (dirección 8Ch)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
PSPIE(1)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE
bit 7							bit 0

bit 7 PSPIE(1) : Permiso de interrupción para el puerto paralelo Esclavo (PSP) al realizar una operación de lectura /escritura
1 = Habilita la interrupción
0 = Inhabilita la interrupción

bit 6 ADIE: Permiso de interrupción para el conversor A/D al finalizar la conversión
1 = Habilita la interrupción
0 = Inhabilita la interrupción

bit 5 RCIE: Permiso de interrupción para el receptor del USART cuando el buffer se llena
1 = Habilita la interrupción
0 = Inhabilita la interrupción

bit 4 TXIE: Permiso de interrupción para el transmisor del USART cuando el buffer se vacía
1 = Habilita la interrupción
0 = Inhabilita la interrupción

bit 3 SSPIE: Permiso de interrupción para el puerto serie síncrono (SSP)
1 = Habilita la interrupción
0 = Inhabilita la interrupción

bit 2 CCP1IE: Permiso de interrupción para el módulo CCP1 cuando se produce una captura o comparación
1 = Habilita la interrupción
0 = Inhabilita la interrupción

bit 1 TMR2IE: Permiso de interrupción para el TMR2 con su desbordamiento.
1 = Habilita la interrupción
0 = Inhabilita la interrupción

bit 0 TMR1IE: Permiso de interrupción para el TMR1 con su desbordamiento.
1 = Habilita la interrupción
0 = Inhabilita la interrupción

Nota: PSPIE no existe en los PIC16F873/873A/876/876A (modelos de 28 terminales que no tienen el PSP)

A1.7.- Registro PIR1

Registro PIR1 (dirección 0Ch)

R/W-0	R/W-0	R-0	R-0	R/W-0	R/W-0	R/W-0	R/W-0
PSPIF(1)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF
bit 7							bit 0

bit 7 PSPIF(1) : Señalizador (de interrupción) del puerto paralelo Esclavo (PSP) al realizar una operación de lectura /escritura
1 = Se ha producido una operación de R/W (desactivar por software)
0 = No se ha producido operación de R/W.

bit 6 ADIF: Señalizador (de interrupción) del conversor A/D al finalizar la conversión
1 = La conversión A/D está completada
0 = La conversión A/D no está completada

bit 5 RCIF: Señalizador (de interrupción) del receptor del USART cuando el buffer se llena
1 = Buffer lleno
0 = Buffer no lleno

bit 4 TXIF: Señalizador (de interrupción) del transmisor del USART cuando el buffer se vacía
1 = Buffer vacío
0 = Buffer no vacío

bit 3 SSPIF: Señalizador (de interrupción) del puerto serie síncrono (SSP)
1 = Se ha producido la condición de interrupción del SSP, es decir, se ha producido una transmisión/recepción (desactivar por software antes de salir de la RAI)
0 = No se produce condición de in6terrupción de SSP

bit 2 CCP1IF: Señalizador (de interrupción) del módulo CCP1
MODO CAPTURA:
1 = El registro del CCP captura el valor del registro TMR1 (valor del Timer1) (desactivar por software)
0 = No se produce la captura del registro TMR1
MODO COMPARACIÓN:
1 = Coinciden el valor del CCP con el valor del TMR1
0 = No coinciden
MODO PWM (modulación de pulsos en anchura): no se usa

bit 1 TMR2IF: Señalizador (de interrupción) por coincidencia de TMR2 con PR2
1 = Se produce la coincidencia
0 = No se produce la coincidencia

bit 0 TMR1IF: Señalizador (de interrupción) por desbordamiento (overflow) del TMR1
1 = Desbordamiento del TMR1 (desactivar por software)
0 = No desbordamiento del registro TMR1

Note 1: PSPIE no existe en los PIC16F873/873A/876/876A (modelos de 28 terminales que no tienen el PSP)

A1.8.- Registro PIE2 y PIR2

Registro PIE2 (dirección 8Dh)

U-0	R/W-0	U-0	R/W-0	R/W-0	U-0	U-0	R/W-0
-	CMIE	-	EEIE	BCLIE	-	-	CCP2IE
bit 7							bit 0

bit 7,5,2,1 Unimplemented: No implementado. Se lee '0'
bit 6 CMIE: Permiso de interrupción para el Comparador
1 = Habilita la interrupción
0 = Inhabilita la interrupción
bit 4 EEIE: Permiso de interrupción por fin de escritura en la EEPROM de datos
1 = Habilita la interrupción
0 = Inhabilita la interrupción
bit 3 BCLIE: Permiso de interrupción por colisión de Bus en el el puerto serie síncrono (SSP) cuando dos o más maestros tratan de transferir al mismo tiempo.
1 = Habilita la interrupción
0 = Inhabilita la interrupción
bit 0 CCP2IE: Permiso de interrupción en el módulo CCP2
1 = Habilita la interrupción
0 = Inhabilita la interrupción

Registro PIR2 (dirección 0Dh)

U-0	R/W-0	U-0	R/W-0	R/W-0	U-0	U-0	R/W-0
-	CMIF	-	EEIF	BCLIF	-	-	CCP2IF
bit 7							bit 0

bit 7,5,2,1 Unimplemented: No implementado. Se lee '0'
bit 6 CMIF: Flag(de interrupción) del comparador
1 = Ha cambiado la entrada del comparador
0 = No ha cambiado
bit 4 EEIF: Flag (de interrupción) del fin de escritura en la EEPROM
1 = Se ha completado la operación de escritura (desactivar por software)
0 = No se ha completado o iniciado la operación de escritura

bit 3 BCLIF: Flag (de interrupción) por colisión de BUS.
1 = Ha ocurrido una colisión de bus en el SSP.
0 = No ha ocurrido la colisión
bit 0 CCP2IF: Flag (de interrupción) del módulo CCP2
MODO CAPTURA:
1 = El registro del CCP captura el valor del registro TMR1 (valor del Timer1) (desactivar por software)
0 = No se produce la captura del registro TMR1
MODO COMPARACIÓN:
1 = Coinciden el valor del CCP con el valor del TMR1
0 = No coinciden
MODO PWM (modulación de pulsos en anchura): no se usa

A1.9.- CCPxCON

Bits 3-0	CCPxM3: CCPxM0		Bits de selección del modo de funcionamiento del módulo CCPx	
	00	00	Módulo CCPx desactivado (resetea el módulo CCPx)	
	01	00	Modo Captura	Captura cada flanco de bajada
		01	Modo Captura	Captura cada flanco de subida
		10	Modo Captura	Captura cada 4 flancos de subida
		11	Modo Captura	Captura cada 16 flancos de subida
	10	00	Modo Comparación	El terminal CCPx es iniciado en bajo '0' y se pone en alto '1' cuando el resultado de la comparación es positivo, es decir, cuando TMR1 alcanza el valor del registro de16 bits (CCPRxH:CCPRxL). El bit CCPxIF se activa y se pone a 1.
		01	Modo Comparación	El terminal CCPx es iniciado en alto '1' y se pone en bajo '0' cuando el resultado de la comparación es positivo, es decir, cuando TMR1 alcanza el valor del registro de16 bits (CCPRxH:CCPRxL). El bit CCPxIF se activa y se pone a 1.
		10	Modo Comparación	El bit CCPIF es puesto a 1 cuando el resultado de la comparación es positivo, es decir, generación de interrupción software cuando se produce la igualdad. El terminal CCPx no se ve afectado.
		11	Modo Comparación	Generación de disparo de evento especial (special event trigger) cuando el resultado de la comparación es positivo. El bit CCPxIF se pone a 1. El terminal CCPx no se ve afectado. CCP1 resetea el TMR1; CCP2 resetea el TMR1 y lanza una conversión A/D nueva (si el módulo del conversor A/D está habilitado.
	11	xx	PWM	
Bits 5-4	DCxB1:DCxB0 (CCPxX:CCPxY)		Modo PWM	Bits 0 y 1 del valor que fija el ciclo de trabajo en el modo PWM En los modos CAPTURA y COMPARACIÓN NO SE UTILIZAN
Bits 7-6	No implementados			Se leerían como '0'

A1.10.- ADCON0

REGISTRO ADCON0 (dirección 1Fh)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0
ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/DONE#	-	ADON
bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0

bit 7-6 ADCS1:ADCS0: Selección del reloj para la conversión A/D (en los PIC16F87x y versiones "antiguas") y junto con ADCS2 que está en ADCON1 (en los PIC16F87x A)

ADCS1:ADCS0	Fuente de reloj	Frecuencia del reloj del convertidor A/D	
		ADCS2 = 0	ADCS2 = 1
00	Oscilador principal	Fosc/2	Fosc/4
01	Oscilador principal	Fosc/8	Fosc/16
10	Oscilador principal	Fosc/32	Fosc/64
11	Oscilador RC interno	167 KHz a 500 KHz	167 KHz a 500 KHz

bit 5-3 CHS2:CHS0:
Selección del canal de conversión

000 = Canal 0 — 100 = Canal 4
001 = Canal 1 — 101 = Canal 5
010 = Canal 2 — 110 = Canal 6
011 = Canal 3 — 111 = Canal 7

bit 2 GO/DONE#:
Estado de la conversión
Si ADON=1:
1 = Conversión en progreso
0 = Conversión finalizada

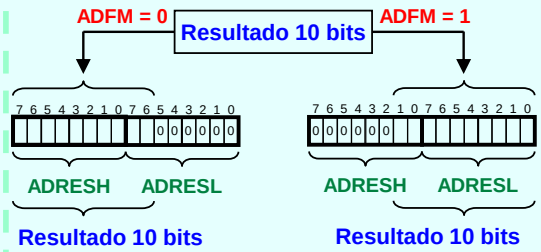
bit 0 ADON:
Bit de encendido del convertidor A/D
1 = Módulo A/D encendido
0 = Módulo A/D apagado

A1.11.- ADCON1

REGISTRO ADCON1 (dirección 9Fh)

R/W-0	R/W-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
ADFM	ADCS2	-	-	PCFG3	PCFG2	PCFG1	PCFG0
bit 7	bit 6	bit 5	bit 4	bit3	bit 2	bit 1	bit 0

bit 7 ADFM: Selección de formato del resultado
1 = Ajuste a la derecha
0 = Ajuste a la izquierda



bit 6 ADCS2: Selección de reloj para conversión A/D junto con ADCS1 y ADCS0 de ADCON0
(No existe en los PIC16F87x y versiones "antiguas")

bit 3-0 PCFG3:PCFG0: Configuración de las entradas al módulo A/D

PCFG3:PCFG0	AN7 RE2	AN6 RE1	AN5 RE0	AN4 RA5	AN3 RA3	AN2 RA2	AN1 RA1	AN0 RA0	V_{REF+}	V_{REF-}
0000	A	A	A	A	A	A	A	A	V_{DD}	V_{SS}
0001	A	A	A	A	V_{REF+}	A	A	A	RA3	V_{SS}
0010	D	D	D	A	A	A	A	A	V_{DD}	V_{SS}
0011	D	D	D	A	V_{REF+}	A	A	A	RA3	V_{SS}
0100	D	D	D	D	A	D	A	A	V_{DD}	V_{SS}
0101	D	D	D	D	V_{REF+}	D	A	A	RA3	V_{SS}
011x	D	D	D	D	D	D	D	D	-	-
1000	A	A	A	A	V_{REF+}	V_{REF-}	A	A	RA3	RA2
1001	D	D	A	A	A	A	A	A	V_{DD}	V_{SS}
1010	D	D	A	A	V_{REF+}	A	A	A	RA3	V_{SS}
1011	D	D	A	A	V_{REF+}	V_{REF-}	A	A	RA3	RA2
1100	D	D	D	A	V_{REF+}	V_{REF-}	A	A	RA3	RA2
1101	D	D	D	D	V_{REF+}	V_{REF-}	A	A	RA3	RA2
1110	D	D	D	D	D	D	D	A	V_{DD}	V_{SS}
1111	D	D	D	D	V_{REF+}	V_{REF-}	D	A	RA3	RA2

D: digital y A; analógica

Bibliografía

- Hoja de Especificaciones del PIC16F877A. PIC16F87xA DataSheet. Microchip DS39582b (www.microchip.com).
- Guía de usuario HI-TECH C® for PIC10/12/16. DS51865A (www.microchip.com).
- Guía de usuario de los microcontroladores PIC de gama media. PICmicro Mid-Range MCU Family Reference Manual. DS33023a (www.microchip.com).
- Compiled Tips 'N Tricks Guide. DS01146B (www.microchip.com).
- Hoja de especificaciones de la pantalla LCD Hitachi HD44780 https://www.sparkfun.com/datasheets/LCD/HD44780.pdf.
- Hoja de especificaciones del sensor PIR sensor https://www.mpja.com/download/31227sc.pdf.
- Hoja de especificaciones del MQ2 https://www.pololu.com/file/0J309/MQ2.pdf
- Hoja de especificaciones del MQ7 https://www.sparkfun.com/datasheets/Sensors/Biometric/MQ-7.pdf
- Hoja de especificaciones de la matriz de LEDs 1088AS http://megtestesules.info/hobbielektronika/adatlapok/LED8x8_1088AS.pdf
- Hoja de especificaciones del contador Johnson https://datasheet.octopart.com/74HC4017D-Philips-datasheet-9723805.pdf
- Hoja de especificaciones del teclado 4x4 https://www.parallax.com/sites/default/files/downloads/27899-4x4-Matrix-Membrane-Keypad-v1.2.pdf
- "Ejercicios de programación con microcontroladores PIC" de Jesús M. Corres, Carlos Ruiz y Cándido Bariáin. Ed. Marcombo.