Utilización del PICkit™ 2 Debug Express de MICROCHIP para construir un teremín

Por Colin Greaves, Microchip Technology

Para empezar a diseñar su teremín, o cualquier otro proyecto interesante, consulte la oferta especial de Microchip en la página 41.

Es posible que el debugger altamente compacto de Microchip pueda recordarle a un llavero electrónico, pero contiene todo lo necesario para ayudarle a desarrollar un proyecto basado en un microcontrolador empezando desde cero.
El autor ha decidido revisar algunas de estas funciones, creando para ello un sencillo proyecto que emplea poco más de lo suministrado en el kit. Con la incorporación de tres simples componentes, que la mayoría de ingenieros tendrá a su alcance, se puede crear un sencillo controlador de sonido, que ofrezca algo parecido a un teremín (theremin en inglés), algo que aparece publicado en las revistas de vez en cuando como un proyecto complicado. Esta sencilla técnica logra que el micro haga todo el trabajo.

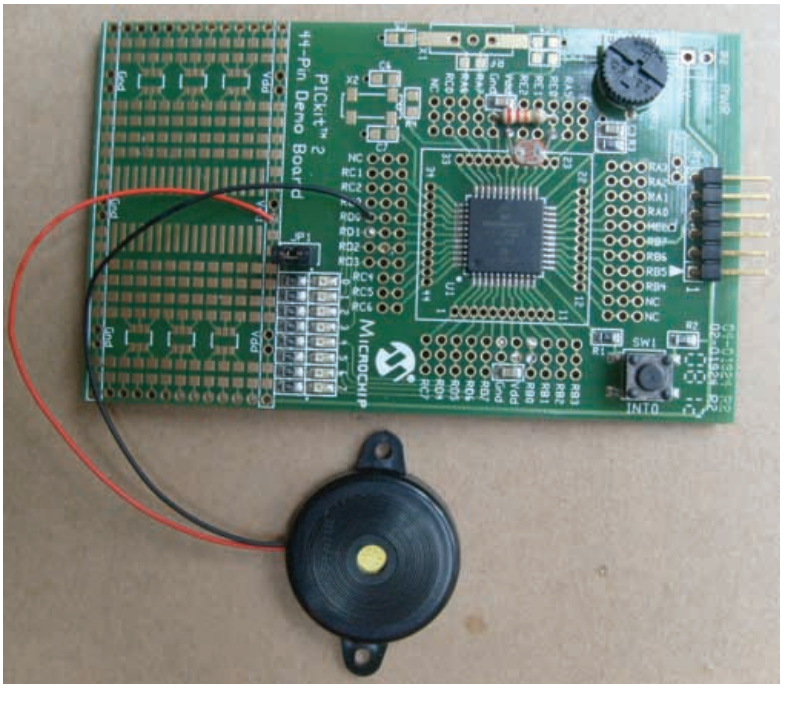

Figura 1. Tarjeta de demostración de PICkit 2. Sólo se han añadido LDR, R1 y un resonador piezoeléctrico

Una buena forma de desarrollar un proyecto es desglosarlo en pequeños pasos. Muchos ingenieros empezarán por iluminar un LED para probar el funcionamiento del kit, el código se compila y el microcontrolador realiza su trabajo. Una vez llevado esto cabo, ya se dispone de un punto de inicio estable. La tarjeta de demostración se suministra con un práctico conjunto de LED conectados al PORT D del microcontrolador. Empezaremos por ahí.

El entorno de desarrollo que proporciona Microchip es MPLAB®. Éste es el interface de usuario para todas las funciones, para todos sus micros. Se suministra con el kit en un CD, para un fácil acceso, pero siempre es una buena idea visitar el sitio web para obtener la última versión. La ejecución de Project Wizard, incluido en MPLAB, es una forma rápida de configurar un nuevo proyecto. El procesador de la placa de demostración era un PIC16F887, y por tanto fue el seleccionado desde el programa asistente. A continuación se seleccionó el conjunto de herramientas, en este caso MPASM™, dado que el proyecto se escribirá en lenguaje ensamblador. Si hay instalado un compilador C también podría existir esta opción.

Luego se escogió la carpeta de proyecto. Una vez creada en el directorio raíz, denominado Theremin, se asignó el nombre del proyecto, también Theremin. Esto creará el archivo del proyecto (Theremin.MCP) y el archivo del espacio de trabajo, que es donde se guarda la configuración personal de las ventanas en MPLAB.

Por el momento no existen archivos de proyecto, y por tanto se puede obviar la parte de 'añadir archivos' y finalizar el programa asistente. Estos archivos de proyecto los debe crear el usuario, pero no necesariamente desde cero. Necesitamos un archivo fuente (*.ASM) y un fichero incluido para definir todos los nombres de registros y de bit que nos sean de utilidad y a los cuales se corresponda la hoja de datos. El archivo ASM se puede crear consultando en el disco duro en C:\Program Files\Microchip\MPASM Suite\Template\Code. Esta carpeta tiene una plantilla básica para cada procesador, y copiando el archivo 16F887TEMP.ASM en la nueva carpeta C:\Theremin, se puede ahorrar mucho tecleo. Una vez copiado, se asigna su nuevo nombre Theremin.ASM para que sea más reconocible.

La función 'View Project' de MPLAB abrirá la ventana de visualización general del proyecto, y se mostrará el proyecto Theremin.MCP. Pulse el botón derecho del ratón en los archivos fuente y seleccione añadir archivos. Navegando por la carpeta del proyecto, podemos seleccionar Theremin.ASM como nuestro código fuente y luego añadir un archivo apropiado para el PIC16F887. Pulsando el botón derecho en los archivos de cabecera, se añadimos archivos, exploramos en C:\Program Files\Microchip\MPASM Suite y seleccionamos el archivo P16F887.INC. Nuestro proyecto ya está completo y podemos intentar realizarlo. 'Project Build All' intentará ensamblar el archivo de plantilla y debería generar una ventana de salida con el mensaje 'Build Succeeded' en la parte inferior. Si fallara, podría ser que se estén utilizando archivos equivocados; compruebe que se han seleccionados los archivos del 887 y no del 877, un error habitual.

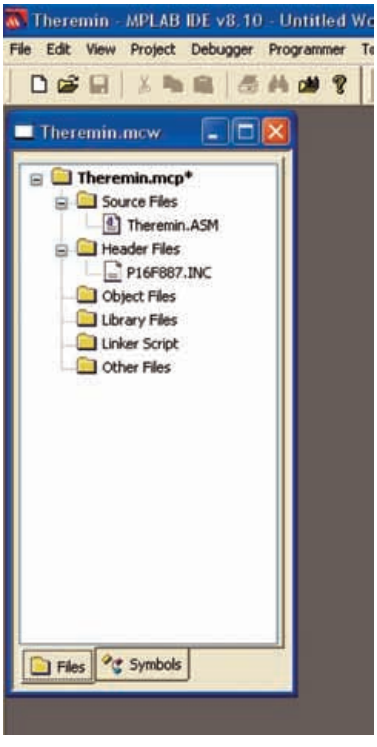

Figura 2. Información general sobre el proyecto en MPLAB

Al abrir el archivo fuente, Theremin.ASM, podemos ver lo que estaba ensamblado en realidad. El primer punto de interés es el ajuste de CONFIG. Está inicializado en la plantilla, pero compruebe siempre que su estado es el requerido. Son importantes el tipo de oscilador, el control MCLR (patilla de puesta a cero o 'reset') y la configuración del supervisor (watchdog). Necesitamos que el oscilador sea interno, que el MLCR esté habilitado y que el supervisor esté deshabilitado. Estos estados están configurados por defecto, así que no es necesario cambiarlos.

El siguiente paso es el de nuestras definiciones variables; vamos a crear más variables después, por lo que hay que tener en cuenta este bloque. El vector de puesta a cero y la rutina de interrupción ya vienen configurados. La puesta a cero se configura para saltar a una etiqueta denominada 'main' así se puede utilizar. Las interrupciones no se utilizarán, por lo que se puede ignorar este código. Esto es en suma la plantilla completa: un simple entorno básico para empezar a añadir el código propio.

El primer paso es configurar los puertos de entrada/salida. PORTD está conectado a los LED, por lo que es nuestro puerto de salida. El registro que controla si el puerto es una entrada o salida se denomina TRISD. Al escribir un 0 en sus 8 bits podemos configurar todos los bits del puerto como salida.

Nota: los registros de control del micro se organizan en bancos, por lo que es importante seleccionar el banco correcto de registros al leer o escribir desde un registro. Esto se ha visto simplificado con el uso de la instrucción 'banksel'. Antes de escribir a TRISD, por ejemplo, introduzca antes del instrucción 'banksel TRISD' para asegurar que se ha configurado correctamente.

A continuación se crea un bucle simple, escribiendo 0 y 1 en los bits de salida por turnos. Para que podamos ver el cambio de los LED necesitamos ralentizar todo el proceso, por lo que podemos añadir también una función de retardo en el bucle, después de cada cambio. La función de retardo es una subrutina, creada fuera del flujo de programa principal, y denominada 'delay'. Este código puede ser accesible desde el micro cuando se utiliza una instrucción 'call', y simplemente irá descontando un número de 255 a 0 hasta volver al lugar desde donde se emitió 'call'. Como este retardo es un corto período de tiempo (el PIC® trabaja internamente a 8 MHz) se puede utilizar un segundo bucle, logrando así que el retardo completo sea de 255x255 instrucciones. Mucho mejor.

Reconstruyamos el proyecto y veamos cómo funciona. Esto puede hacerse como comprobación para ver hasta dónde llega y qué bits del código no pudieran ser correctos. Seguramente veremos un par de errores – no se han reconocido las variables utilizadas en la subrutina de retardo – que no fueron detectados. Estas variables se utilizan para conservar los números que la función de retardo utiliza para contar hacia atrás y deben definirse en el bloque de definición de Variable que vimos en la parte superior de la fuente de código. Añádalas ahora y seleccione un lugar en la memoria para ellos. Una mirada rápida a la hoja de datos (por la página 25) muestra la tabla de registros de funciones especiales, y en la parte inferior del primer banco los registros posteriores a 20h están libres. Éste es un buen lugar para colocar nuestras variables. La hoja de datos puede encontrarse en el CD del kit, junto con una completa serie de notas de aplicación útiles, consejos y trucos y mucho más. Por supuesto, siempre es una buena idea obtener la copia más reciente a través del sitio web de Microchip. En este punto, también es una muy buena idea comprobar la hoja de erratas del microcontrolador con el que se está trabajando; algunos cambios hacen referencia a las funciones, a dispositivos o procesos de fabricación en desarrollo, y esa es la forma de ayudarle a conocer los cambios que se producen. Compruebe siempre las erratas.

Ahora tenemos las variables definidas y podemos intentar construir otra vez. Corregir y construir es una forma algo farragosa de escribir el código, pero parece llevarnos rápidamente a un proyecto operativo. Seguramente nuestro proyecto se construye con un mensaje de éxito de bienvenida. Tenemos una aplicación funcionando y podemos intentar programar la tarjeta de la aplicación con él.

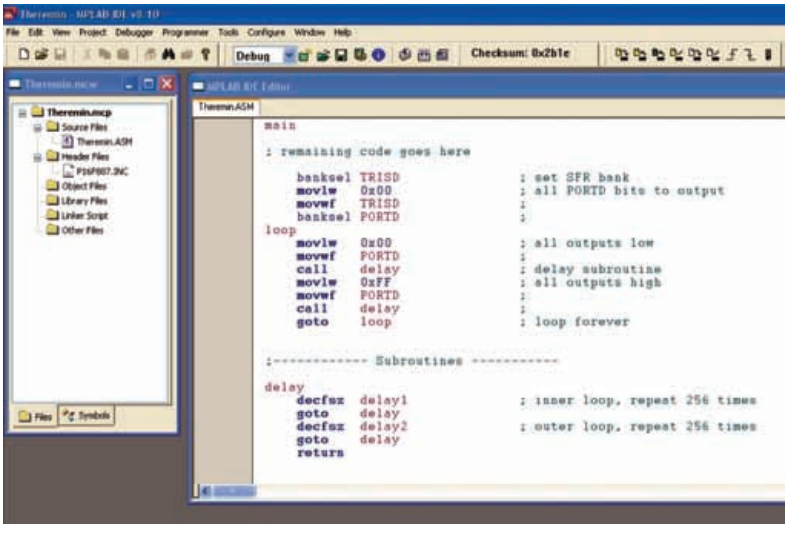

Figura 3. Añadir código de usuario a la plantilla para controlar los LED

Conecte el depurador PICkit 2 a la tarjeta de demostración de la aplicación y al PC con un cable USB. El puerto USB alimentará la tarjeta de demostración de forma bastante sencilla. Cuando conecte un kit de desarrollo USB a su PC por primera vez, Windows probablemente le pedirá que proporcione un controlador (driver), para saber qué hacer con él. Microchip suministra los controladores para su kit de desarrollo, así que no utilice los controladores estándar de Windows. Éstos se encuentran en la carpeta Microchip\MPLAB IDE y cada elemento del kit de desarrollo tiene su propia carpeta de controlador. Sin embargo, el PICkit 2 es un dispositivo HID estándar, por lo que funcionará con tan sólo conectarlo.

Una vez conectado, el PICkit 2 puede seleccionarse como herramienta de programación. Vaya a 'Programmer', seleccione el programador, PICkit 2. La ventana de salida informará del proceso, como la inicialización del PICkit 2 conectado, así como cualquier actualización del SO. Al seleccionar 'Programmer', 'Program' descargará su código en la tarjeta de demostración. De nuevo la ventana de salida se utiliza para monitorizar este proceso. Cualquier fallo en este punto podría deberse con mucha seguridad a una mala conexión; compruebe la alineación de las patillas del PICkit 2 conectadas a la tarjeta de la aplicación.

Ahora seleccione 'Programmer', salga de la puesta a cero y debería ver inmediatamente los LED parpadear a unos 2-3 ciclos por segundo. El entorno de desarrollo está funcionando. Todas las selecciones del menú que hemos estado haciendo son accesibles desde los iconos incluidos en MPLAB, así que valdrá la pena comprobar su funcionamiento para acelerar la selección.

Sería útil en este punto echar una mirada a cómo el depurador puede darnos alguna información sobre lo que está haciendo el programa. Al poner a cero el microcontrolador PIC pulsando el icono del flanco de reloj negativo (o seleccionando 'Programmer', 'Hold in Reset') se detiene el parpadeo de los LED. El PICkit 2 puede configurarse en modo depuración seleccionando 'Debugger', 'Select Tool', 'PICkit 2'. Puede surgir un aviso que informa que la herramienta no puede estar en modo de depuración y programación simultáneamente. El modo depuración utiliza un pequeño bit de código para comunicar el estado de los registros y las E/S al MPLAB. Esto debe programarse en el microcontrolador PIC mediante reprogramación mientras está en modo depuración. Una vez programado, tenemos un nuevo conjunto de iconos para jugar el PICkit 2, incluyendo 'Run', 'Step' y 'Reset'. Utilice el icono 'RUN' para asegurarse de que el código está funcionando, entonces pulse el icono 'HALT' para detener el microcontrolador PIC. Dado que el programa utiliza el 99% de su tiempo ejecutando la subrutina de retardo, es casi seguro que se parará completamente con la flecha verde en la ventana de código fuente apuntando a una instrucción en esta subrutina. Ésta es la siguiente instrucción que se ejecutará, y pulsando el icono 'Step Into' pasará una a una las instrucciones en el microcontrolador PIC. Mueva el puntero del ratón sobre las variables de nombres 'delay1' y 'delay2', y podrá visualizar sus valores actuales. Se requerirán muchos pasos para volver al código de programa principal, y la ejecución y parada casi siempre acabará donde estamos ahora, por lo que una forma elegante de detener la ejecución en el bit principal del código es utilizar un punto de ruptura.

Pulse dos veces sobre la línea situada por debajo de la etiqueta 'loop', la instrucción movlw 0x00. Aparecerá un círculo rojo sobre B junto a la línea para mostrar que se ha configurado un punto de ruptura. Pulsando el icono 'RUN' otra vez y el microcontrolador PIC se ejecutará hasta llegar al punto de ruptura, y entonces se parará. Situando el ratón sobre el nombre de registro 'PORTD' en el listado de programa se mostrará un valor de 0xFF (durante 1s) y los LED estarán todos encendidos. Ahora, pulsando el paso único se apagarán los LED mientras se ejecuta la siguiente instrucción y el valor de 'PORTD' indicará 0x00.

Mientras disfrutamos de este control de nuestro hardware, vayamos un paso más allá: echemos una mirada a los Registros de Funciones Especiales o 'Special Function Registers' dentro del microcontrolador PIC seleccionando 'View', 'Special Function Registers'. Ahí se encuentran todos los registros de control y sus estados, mostrados para nuestra información. Seleccione el registro de 'PORTD', que probablemente todavía mostrará nuestro valor 0x00 (o 00000000 en la columna Binaria) y pulse dos veces. Cambiando uno de los bits binarios de 0 a 1, debería ser capaz de editarlo directamente. Ahora compruebe la tarjeta: ¡el LED se ilumina! Este acceso al programa, los bits de E/S y todo lo demás es justo lo que necesita un desarrollador para comprender qué está pasando y depurar su proyecto.

Para crear nuestro teremín necesitaremos algún tipo de interface analógico. Una solución sencilla sería un potenciómetro conectado a una de las patillas de entrada analógica del PIC16F887, pero para tener la sensación de teremín una solución mejor sería un sensor óptico, utilizado para detectar la proximidad de las manos del usuario. Una sencilla resistencia dependiente de la luz (LDR) configurada como divisor de potencial podría hacerlo bastante bien. Este dispositivo se encontró (literalmente) en la mesa del autor y los valores se comprobaron para los niveles de luz de la habitación. Con una mano cubriendo el LDR, su valor fue de unos 200-300 kΩ, y expuesta a la luz de la habitación, el valor fue de unos 2 kΩ, Para dar una buena variación de tensión a la entrada del convertidor A/D se eligió un valor de unos 20 kΩ como resistencia fija para el divisor de potencial. Esto debería ser 10 veces más pequeño que la mayor lectura del LDR, y 10 veces mayor que la menor lectura de la LDR.

El circuito necesario es muy simple (ver figura 4) y se encontró una posición de la tarjeta para que el ensamblaje fuera más sencillo. La conexión RE0 estaba bastante cerca como para montar las resistencias a Vdd y Gnd sin necesidad de puentes (ver figura 5). También se escogió RE0 porque una de sus funciones es la entrada analógica, AN5. El convertidor A/D (ADC) en este dispositivo tiene catorce canales, seleccionables mediante un multiplexor. Utilizaremos el canal 5.

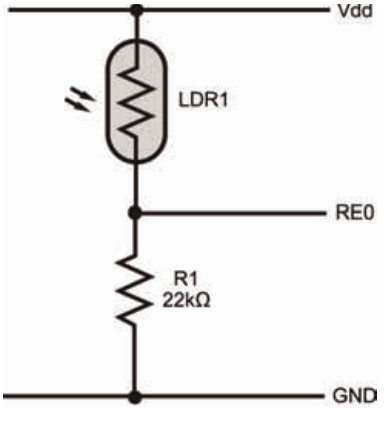

Figura 4. Sensor de luz/mano utilizando sólo 2 componentes

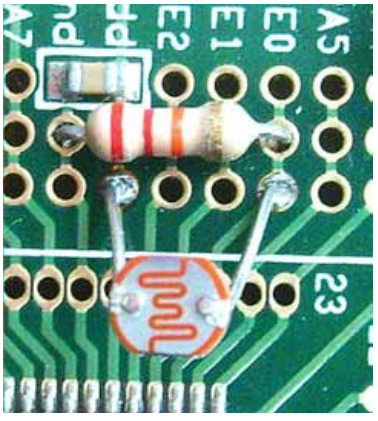

Figura 5. Lugares de montaje apropiados para LDR y R1

La sección 9 de la hoja de datos del PIC16F887 cubre todos los detalles relativos al ADC. Entre las opciones se encuentran la velocidad del reloj, el formato de los resultados, la referencia utilizada para realizar una medida, etc. El uso de las funciones del ADC es común a muchos de los dispositivos microcontroladores PIC, por lo que el tiempo ocupado aquí en la lectura es una buena inversión. Como rápido resumen, se decidió que sólo necesitamos realmente un valor de 8 bit, por lo que la salida se formateó para estar justificada a la izquierda (todos los bits altos en un registro, de forma que los bits bajos pueden ignorarse), la velocidad de reloj en Fosc/2 (4MHz) y Vdd como referencia; no hace falta que sea especialmente preciso o estable. Esto nos proporcionará un valor de 8 bit dado que el nivel analógico en AN5 varía entre Vdd (+5v) y Gnd.

El código para configurar el ADC (ver figura 6) se basa en tres registros de control, ADCON0, ADCON1 y ADSEL. Asegúrese de que las instrucciones 'banksel' se utilizan cuando sea necesario. Esto fue seguido por un rápido 'build all' para comprobar la sintaxis.

Figura 6. Configuración y uso del ADC

```
loop
    bsf     ADCON0,ADON         ; enable ADC
    bsf     ADCON0,GO_DONE      ; start conversion
wait1
    btfsc   ADCON0,GO_DONE      ; did conversion complete?
    goto    wait1               ; no, wait
    movf    ADRESH,w            ; read top 8 bits of ADC
    movwf   ADCval
    movlw   0x00                ; set all outputs low
    movwf   PORTD               ;
    call    delay               ; delay subroutine
    movlw   0xFF                ; all outputs high
    movwf   PORTD               ;
    call    delay               ;
    goto    loop                ; loop forever

;------------ Subroutines -----------

delay
    movf    ADCval,w            ; use ADC value for delay loop
    movwf   delay2
deloop
    decfsz  delay1,f            ; inner loop, repeat 256 times
    goto    deloop
    decfsz  delay2,f            ; outer loop, repeat ADCval times
    goto    deloop
    return
```

El ADC está ahora configurado para trabajar y podemos cambiar nuestro bucle de iluminación de LED para utilizar la entrada del ADC. Una conversión de ADC empieza escribiendo en ADCON0. Primero está habilitado y entonces empieza la conversión. Estos dos bits de control para habilitar y después para activar el ADC están en el mismo registro, pero deben proporcionarse dos instrucciones separadas, primero de activación y luego de inicio. Al escribir un 1 en el bit de GO_DONE se inicia la conversión. El ADC hará la medida y presentará el resultado en los registros de ADRESH y ADRESL (es un convertidor de 10 bit). Una vez completado, el bit GO_DONE pasará a 0 y se puede leer el resultado. Sólo necesitamos leer ADRESH dado que sólo nos interesan los 8 bits superiores del resultado. Lea este valor y guárdelo en el registro 'pitchval'. Éste es el valor que vamos a utilizar para controlar el tono de nuestro teremín.

Podemos dejar la última parte del bucle principal: las salidas van a bascular, con un valor de retardo determinado por el resultado de nuestro ADC. Un rápido 'build all' revela un error; retrocedamos y definamos nuestra nueva variable 'pitchval' en la parte superior del código. Tras las dos definiciones de variable de retardo, 0x22 es una buena elección.

Figura 7. Cambios en el código acabado mediante un bucle de retardo controlado

```
    movlw   0xFF                ; all outputs high
    movwf   PORTD               ;
    call    delay               ;
    goto    loop                ; loop forever

;------------ subroutines -----------

delay
    delay
    movf    pitchval,w          ; use ADC value for delay loop
    movwf   delay2
deloop
    decfsz  delay2,f            ; outer loop, repeat 'pitchval' times
    goto    deloop
    return
```

Finalmente, para lograr que la velocidad de conmutación de salida varíe con el valor de 'pitchval', debemos añadirlo al bucle de retardo. La rutina de retardo se cambia para que sea un solo bucle, no un bucle incluido en un bucle. Necesitamos que la salida conmute a frecuencias de audio, no sólo a unos pocos hercios para que la conmutación sea visible. Entonces la subrutina de retardo se simplifica; sólo un par de instrucciones a la carga del valor de 'pitchval' en la variable 'delay2', un simple bucle, descontando este valor hasta cero. Un rápido 'build all' desvela que no hay problemas, y... ¡el código ya está acabado! La programación del dispositivo tarda un segundo y el proyecto está funcionando. Una simple sonda piezoeléctrica conectada a una de las salidas de PORTD (se utilizó RD0) y se oyó un tono, que varía a medida que se movía una mano cerca de la LDR. El bucle de retardo se modeló situando unas instrucciones de no operación (nop) tras la etiqueta 'deloop'. Esto permite modelar el tono para que se ajuste mejor al rango audible del autor.

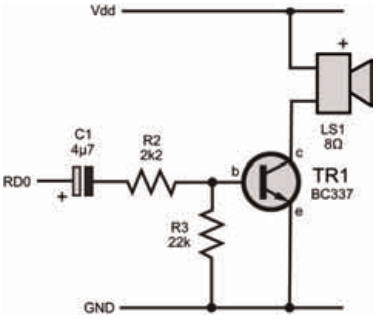

Figura 8. Circuito de salida de sonido mejorado

Antes de compartir esta magistral pieza musical con los amigos, sería buena idea añadir un controlador de altavoz a la salida. La calidad de sonido piezoeléctrico es bastante mala. Se realizó un sencillo controlador a partir de un BC337 (ver fig. 8) y un par de resistencias. (Además, sería conveniente alimentarlo desde una fuente independiente puesto que el puerto USB no soportaría demasiada corriente). Se añadió un condensador para prevenir que se queme, justo en el caso en que el microcontrolador PIC se parara con todas las salidas en alta. Un altavoz barato de 8Ω proporcionó unos resultados agradables. No olvide que para programar el microcontrolador PIC para que trabaje en modo autónomo (sin conexión a PICkit 2) debería programarse en modo programador, no en modo depurador. Una vez que el programa ha sido depurado y en funcionamiento, ya no se necesita el modo depurador. El autor se ha convertido ahora en un músico de teremín muy competente, la melodía de la canción de Dr Who es su especialidad...