COMENTARIO TECNICO

Generando "Ring Tones" con un PIC16F87x..

Por el Departamento de Ingeniería de Electrónica Elemon.

Los teléfonos celulares han "invadido" nuestra vida cotidiana a tal punto que solo en la república Argentina hay más de 53 millones de unidades!!!, según las últimas estadísticas, lo que representa algo así como un 1 y ½ unidad por habitante .... Una verdadera locura!!! ....

Bien, entonces por qué no sumarnos a esta fiebre y hacer con un "PIC" reproduzca "Ring Tones" tal como lo haría cualquiera de nuestros "imprescindibles" teléfonos móviles?

Entonces, manos a la obra!!!

Usando solamente un pequeño parlante y un capacitor de desacople es posible generar tonos o melodías desde un MCU **PIC16F87x** de **Microchip**. Uno de los Timers puede ser utilizado para generar cada una de las 11 notas musicales y otro de los Timers puede ser empleado para controlar el tiempo de la duración de la nota. También puede implementarse el soporte de varias octavas musicales si estamos preparados para trabajar un poco más en nuestro propio programa.

El código puede servir de base para un rango muy amplio de aplicaciones tales como juegos de navidad, timbres de puerta personalizados, o simples campanas. Se puede agregar un DIP Switch para que nuestro "experimento" soporte múltiples melodías.
Sin embargo, la parte más dura de nuestro proyecto será sin dudas el crear y codificar nuestras propias melodías a ejecutar.

Hay decenas de miles de "Ring Tones" de numerosas marcas de teléfonos móviles sonando en nuestra mente.... ¿Pero cuál elegir? ...

Uno de los más populares "standard" en codificación de Ring Tones es la **RTTTL** (**R**inging **T**ones **T**ext **T**ransfer **L**anguage", esta especificación es utilizada por **NOKIA** en sus teléfonos celulares.
Estos tonos pueden almacenarse y transferirse usando la especificación **RTX** de ring tones.
Esta especificación no es más que un simple archivo de texto ASCII el cual incluye el nombre del ring tone, una sección de control que especifica los atributos por defecto y una coma (,) delimitando el "string" de notas las cuales pueden codificarse con "octava y duración" en forma opcional.

Entendiendo la especificación RTTTL.

Por ejemplo, una melodía simple RTTTL puede ser el ring tone de ***"Tomy y Daly"*** (itchy and scratchy) de la popular serie de ***"Los Simpsons"*** que se muestra a continuación:

```
itchy:d=8,o=6,b=160:c,a5,4p,c,a,4p,c,a5,c,a5,c,a,4p,p,c,d,e,p,e,f,g,4p,d,c,4d,f,
      4a#,4a,2c7
```

Este ring tone puede descomponerse en **3 secciones:**

- **Título:** Es el título del ring tone que comienza el string seguido por 2 puntos (:). Existen variaciones en las especificaciones de la máxima longitud del título, pero se sugiere no exceder los 10 caracteres en ello.

- **Control:** La sección de control establece los parámetros por defecto que transportan la melodía. La idea aquí es reducir el tamaño del "string" por medio de la omisión de parámetros comunes. En lugar de cada coma delimitadora de nota(,) conteniendo la información de nota, octava y duración, se pueden omitir la octava y la duración si son las mismas a las especificadas por defecto.

- **Comandos de Nota:** El cuerpo de un ring tone está conformado por notas delimitadas por comas precedidas en forma opcional por la "duración" y seguidas por la "octava". La nota "punto" (.) puede especificarse después de la octava, la cual indica la duración de la nota extendida un 50%, lo que significa una duración de 1.5x veces de la original.

Los parámetros que pueden especificarse en la sección de Control son:

- **d** (duración por defecto). La duración por defecto puede ser 1 de 1, 2, 4, 8, 16, 32, o 64.
  1 especifica una "Semibreve" o nota redonda, 2 indica que es una "Blanca" (Media Nota),
  4 es una "negra" o cuarto de nota, etc. hasta el 64 que es un 64 avo. de nota.

- **o** (Octava por defecto). La octava por defecto (escala) puede ser 4, 5, 6, o 7.

- **b** (Golpes por minutos). El BPM o "Tempo" puede tomar cualquiera de los siguientes valores: 25, 28, 31, 35, 40, 45, 50, 56, 63, 70, 80, 90, 100, 112, 125, 140, 160, 180, 200, 25, 250, 285, 320, 355, 400, 450, 500, 565, 635, 715, 800, 900.

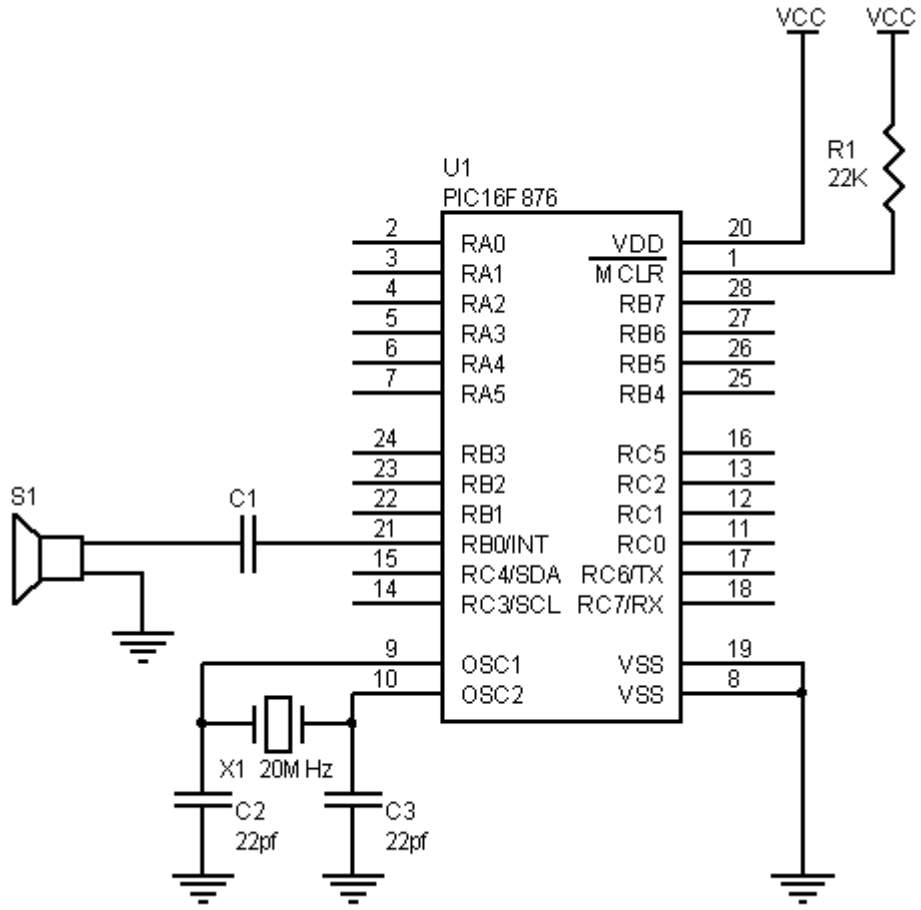
- **s** (Estilo). El estilo puede ser S = Staccato, N= Natural, C = Continuo.

- **l** (lazo o looping). El valor del lazo o loop puede ser desde 0 a 15. El "0" deshabilita el lazo.
  El 15 habilita en lazo infinito. Los valores entre 1 y 14 especifican cuantos lazos se harán antes de detenerse.

Si se omite alguno de los parámetros desde la sección de control, se asumirán los siguientes valores por defecto: 4 = duración, 6 = Escala, 63 = BPM (Golpes por Minuto).

El circuito.

Como se puede observar en la siguiente figura, el circuito a implementar para generar los tonos es muy simple. El cristal de 20 MHz (FBus = 5 MHz) controla los tiempos y si se lo quisiera reemplazar por otro valor, el diseñador debería recalcular nuevamente todos los divisores para generar cada tono.



Los cálculos con un cristal de 20 MHz.

Las siguientes hojas de cálculo muestran las frecuencias deseadas y las realmente conseguidas para cada nota a la frecuencia de Cristal de 20 MHz, o sea Fbus = 5 MHz. El código soporta 4 octavas.

Clock	20000000
Fosc/4	5000000
Divider	1
Freq	5000000

Octave	4			5			6			7		
Note	Desired	Divider	Actual	Desired	Divider	Actual	Desired	Divider	Actual	Desired	Divider	Actual
A	440	11364.00	440.0	880	5682.00	880.0	1760	2841.00	1759.9	3520	1420.00	3521.1
A#	466	10726.00	466.2	932	5363.00	932.3	1865	2681.00	1865.0	3729	1341.00	3728.6
B	494	10123.00	493.9	988	5062.00	987.8	1976	2531.00	1975.5	3951	1265.00	3952.6
C	523	9555.00	523.3	1047	4778.00	1046.5	2093	2389.00	2092.9	4186	1194.00	4187.6
C#	554	9019.00	554.4	1109	4509.00	1108.9	2218	2255.00	2217.3	4435	1127.00	4436.6
D	587	8513.00	587.3	1175	4256.00	1174.8	2349	2128.00	2349.6	4699	1064.00	4699.2
D#	622	8035.00	622.3	1244	4018.00	1244.4	2489	2009.00	2488.8	4978	1004.00	4980.1
E	659	7584.00	659.3	1319	3792.00	1318.6	2637	1896.00	2637.1	5274	948.00	5274.3
F	698	7158.00	698.5	1397	3579.00	1397.0	2794	1790.00	2793.3	5588	895.00	5586.6
F#	740	6757.00	740.0	1480	3378.00	1480.2	2960	1689.00	2960.3	5920	845.00	5917.2
G	784	6378.00	783.9	1568	3189.00	1567.9	3136	1594.00	3136.8	6272	797.00	6273.5
G#	830	6024.00	830.0	1660	3012.00	1660.0	3320	1506.00	3320.1	6640	753.00	6640.1

El código Fuente.

El código del programa fue escrito en lenguaje “C” y compilado con el **“Hi – Tech PICC Compiler”** (**www.htsoft.com**). Hitech software dispone de una versión demo del PICC que puede descargarse y sirve por 30 días (**www.htsoft.com/software/index.html**).
Junto al presente artículo se adjunta el archivo comprimido **“16f87xringtone.zip”** que contiene además del archivo fuente en lenguaje C, el archivo compilado en formato “HEX” el cuál ha sido preparado para trabajar con o sin el ICD de Microchip.
Para agregar nuevos tonos, simplemente es un tarea de “Cortar y Pegar”. Se pueden utilizar cualquiera de los cientos y cientos de sitios de Internet para conseguir nuevos “Ring tones”.
Cuando se ha elegido uno en especial, simplemente corte los comandos de notas en el “array” de la melodía y solo hay que ajustar la duración y la octava por defecto junto con la velocidad de Golpes (batidos).

```
/************************************************************************/
/*                                                                      */
/*       RTTTL Ring Tone Player for Microchip PIC16F87x Microcontrollers     */
/*                    Copyright Craig.Peacock@beyondlogic.org               */
/*                        Version 1.0 17th August 2003                      */
/*                                                                      */
/************************************************************************/

#include <pic.h>

#define TONE    RB0

void InitTimer(void);
void delayms(unsigned char cnt);
void PlayNote(unsigned short note, unsigned char octave, unsigned int duration);

unsigned char beep;
unsigned char preloadTMR1L;
unsigned char preloadTMR1H;
unsigned short TMR0Count;
unsigned char beat_speed;

#define MissionImpossible
```

```
void main(void)
{
    unsigned int pointer = 0;
    unsigned int octave = 0;
    unsigned int duration = 0;
    unsigned short note = 0;
    unsigned int defaultoctave = 0;
    unsigned int defaultduration = 0;

#ifdef AxelF
    /* AxelF */
    const unsigned char static Melody[] = {"32p,8g,8p,16a#.,8p,16g,16p,16g,8c6,8g,8f,8g,8p,16d.6,8p,16g,16p,
                                           16g,8d#6,8d6,8a#,8g,8d6,8g6,16g,16f,16p,16f,8d,8a#,2g,4p,16f6,8d6,
                                           8c6,8a#,4g,8a#.,16g,16p,16g,8c6,8g,8f,4g,8d.6,16g,16p,16g,8d#6,86,
                                           8a#,8g,8d6,8g6,16g,16f,16p,16f,8d,8a#,2g"};
    defaultoctave = 5;
    defaultduration = 4;
    beat_speed = 125;
#endif

#ifdef HappyBirthday
    /* HappyBirthday */
    const unsigned char static Melody[] = {"8g.,16g,a,g,c6,2b,8g.,16g,a,g,d6,2c6,8g.,16g,g6,e6,c6,b,a,8f6.,
                                           16f6,e6,c6,d6,2c6,8g.,16g,a,g,c6,2b,8g.,16g,a,g,d6,2c6,8g.,
                                           16g,g6,e6,c6,b,a,8f6.,16f6,e6,c6,d6,2c6"};
    defaultoctave = 5;
    defaultduration = 4;
    beat_speed = 125;
#endif

#ifdef Itchy
    /* Itchy & Scratcy */
    const unsigned char static Melody[] = {"8c,8a5,4p,8c,8a,4p,8c,a5,8c,a5,8c,8a,4p,8p,8c,8d,8e,8p,8e,
                                           8f,8g,4p,8d,8c,4d,8f,4a#,4a,2c7"};
    defaultoctave = 6;
    defaultduration = 8;
    beat_speed = 198;
#endif

#ifdef MissionImpossible
    /* Mission Impossible */
    const unsigned char static Melody[] = {"16d5,16d#5,16d5,16d#5,16d5,16d#5,16d5,16d5,16d#5,16e5,
                                           16f5,16f#5,16g5,8g5,4p,8g5,4p,8a#5,8p,8c6,8p,8g5,4p,
                                           8g5,4p,8f5,8p,8p,8g5,4p,4p,8a#5,8p,
                                           8c6,8p,8g5,4p,4p,8f5,8p,8f#5,8p,8a#5,8g5,1d5"};
    defaultoctave = 6;
    defaultduration = 4;
    beat_speed = 150;
#endif

    TRISB0 = 0;     /* Make TONE an output */

    beep = 0;

    InitTimer();
    PEIE = 1;
    GIE = 1;      /* Enable General Purpose Interrupts */

    do {

        octave = defaultoctave;         /* Set Default Octave */

        if ((Melody[pointer] == '3') && (Melody[pointer+1] == '2')) {
            duration = 32;
            pointer += 2;
        }
        else if ((Melody[pointer] == '1') && (Melody[pointer+1] == '6')) {
            duration = 16;
            pointer += 2;
        }
        else if (Melody[pointer] == '8') {
            duration = 8;
            pointer++;
        }
        else if (Melody[pointer] == '4') {
            duration = 4;
            pointer++;
        }
```

```
else if (Melody[pointer] == '2') {
    duration = 2;
    pointer++;
}
else if (Melody[pointer] == '1') {
    duration = 1;
    pointer++;
} else duration = defaultduration;

if (Melody[pointer + 1] == '#') {

    /* Process Sharps */

    switch (Melody[pointer]) {
        case 'a' : note = 10726;
                   break;
        case 'c' : note = 9019;
                   break;
        case 'd' : note = 8035;
                   break;
        case 'f' : note = 6757;
                   break;
        case 'g' : note = 6024;
                   break;
    }
    pointer +=2;

} else {

    switch (Melody[pointer]) {
        case 'a' : note = 11364;
                   break;
        case 'b' : note = 10123;
                   break;
        case 'c' : note = 9555;
                   break;
        case 'd' : note = 8513;
                   break;
        case 'e' : note = 7584;
                   break;
        case 'f' : note = 7158;
                   break;
        case 'g' : note = 6378;
                   break;
        case 'p' : note = 0;
                   break;
    }
    pointer++;
}

if (Melody[pointer] == '.') {
    /* Duration 1.5x */
    duration = duration + 128;
    pointer++;
}

if (Melody[pointer] == '4') {
    octave = 4;
    pointer++;
} else if (Melody[pointer] == '5') {
    octave = 5;
    pointer++;
} else     if (Melody[pointer] == '6') {
    octave = 6;
    pointer++;
} else     if (Melody[pointer] == '7') {
    octave = 7;
    pointer++;
}

if (Melody[pointer] == '.') {
    /* Duration 1.5x */
    duration = duration + 128;
    pointer++;
}

PlayNote(note, octave, duration);
```

```
     } while (Melody[pointer++] == ',');

    /* Wait until last note has played */
    while(TMR0Count) { };
    beep = 0;

    /* Loop */
    while(1) {};
}

void PlayNote(unsigned short note, unsigned char octave, unsigned int duration)
{

    /* Process octave */
    switch (octave) {
        case 4 : /* Do noting */
             break;
        case 5 : /* %2 */
             note = note >> 1;
             break;
        case 6 : /* %4 */
             note = note >> 2;
             break;
        case 7 : /* %8 */
             note = note >> 4;
             break;
    }

    /* Wait until last note has played */
    while(TMR0Count) { };
    beep = 0;

    /* Process New Note Frequency */
    if (note) {
        note = ~note;
        preloadTMR1L = (note & 0xFF);
        preloadTMR1H = ((note & 0xFF00) >> 8);
    }

    /* Process Note Duration */
    TMR0Count = 255/(duration & 0x7F);

    /* If duration is 1.5x add .5 to duration */
    if (duration & 0x80) TMR0Count = (TMR0Count + (TMR0Count >> 1));

    if (note) beep = 1;
}

void InitTimer(void)
{
    /* Initialise Timer 0 */
    OPTION = 0b11010111;  /* Set TMR0 to Internal CLk, 1:256 */
    T0IF = 0;             /* Clear TMR0 Flag, ready for use */
    T0IE = 1;             /* Enable Timer Overflow Interrupt */

    /* Initialise Timer 1 */
    T1CON = 0b00000001;   /* Counter Enabled, Using Ext Pin 1:1 Prescaler */
    TMR1IF = 0;           /* Clear Flag */
    TMR1IE = 1;           /* Enable Interrupt */
}
```

```
void interrupt interr(void)
{
    if (T0IF) {
        TMR0 = beat_speed;
        if (TMR0Count) TMR0Count--;
        T0IF = 0;
    }
    if (TMR1IF) {
        if (beep) TONE = !TONE;
        else      TONE = 0;
        TMR1H = preloadTMR1H;
        TMR1L = preloadTMR1L;
        TMR1IF = 0; /* Clear Flag */
    }
}
The above example compiled with the Mission Impossible theme takes a modest 1K of memory. .
Memory Usage Map:

Program ROM   $0000 - $004D  $004E (    78) words
Program ROM   $006F - $01BA  $014C (   332) words
Program ROM   $05B9 - $07FF  $0247 (   583) words
                             $03E1 (   993) words total Program ROM

Bank 0 RAM    $0020 - $0038  $0019 (    25) bytes
Bank 0 RAM    $0071 - $0078  $0008 (     8) bytes
                             $0021 (    33) bytes total Bank 0 RAM


Program statistics:

Total ROM used      993 words (12.1%)
Total RAM used       33 bytes (9.0%)
```

Material extractado y corregido de “Beyond logic”

Solicite mayor información en:

Electrónica Elemon
www.elemon.com.ar
ventas@elemon.com.ar