Generate Ring Tones on your PIC16F87x Microcontroller

Using only a speaker and decoupling capacitor, it is possible to generate tunes or melodies from your Microchip PIC16F87x processor. A timer can be used to generate each of the eleven musical notes and another timer can be used to time the note duration. You can even choose to support several octaves if you want a challenge.

The code can form the foundation for a range of applications such as christmas toys to customised doorbells or chimes. Add a DIP switch to support multiple melodies.

However the hard parts comes from making and coding your own melodies to play. Wouldn't it make sense to use some of the tens of thousands of Mobile Phone Ring Tones floating around the place. This is what we have done here.

One of the more popular standards is the RTTTL (Ringing Tones Text Transfer Language) specification which is used by Nokia mobile phones. These tunes can be save and transported using the .RTX ringtone specification. This specification is no more than a ASCII text file which includes the ringtone name, a control section specifying default attributes and a comma delimited string of notes that can be optionally encoded with the octave and duration.

Understanding RTTTL (Ringing Tones Text Transfer Language)

A simple RTTTL ring tone is the itchy and scratchy theme song which is displayed below :

```
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
```

This ring tone can be split into three sections :

- **Title :** The title of the ring tone starts the string followed by a semicolon. There are varying specifications on it's maximum length but it is suggested it shouldn't be any more than 10 characters long.

- **Control :** The control section sets up default parameters which are carried throughout the melody. The idea is to reduce the size of the string, by omitting common parameters. Instead of each comma delimited note containing the note, octave and duration information, the octave and duration can be omitted if it is the same than the specified default.

- **Note Commands :**The body of the ring tone is made up of comma delimited notes prefixed by the optional duration and postfixed by the octave. A dotted note (.) can be specified after the octave which indicates the duration of the note is extended by 50%, making it 1.5x the original note duration.

The parameters which can be specified in the control section are :

- **d** (default duration). The default duration can be one of 1, 2, 4, 8, 16, 32 or 64. The default duration can be one of 1, 2, 4, 8, 16, 32 or 64. 1 specifies a Semibreve (Whole Note), 2 indicates it a Minim (Half Note), 4 is a Crotchet (Quarter Note) etc up to 64 which is a Hemidemisemiquaver (64th note). .

- **o** (default octave). The default octave (scale) can be 4, 5, 6, or 7.

- **b** (beats per minute). The BPM or tempo can be any one of the following values 25, 28, 31, 35, 40, 45, 50, 56, 63, 70, 80, 90, 100, 112, 125, 140, 160, 180, 200, 225, 250, 285, 320, 355, 400, 450, 500, 565, 635, 715, 800, 900.

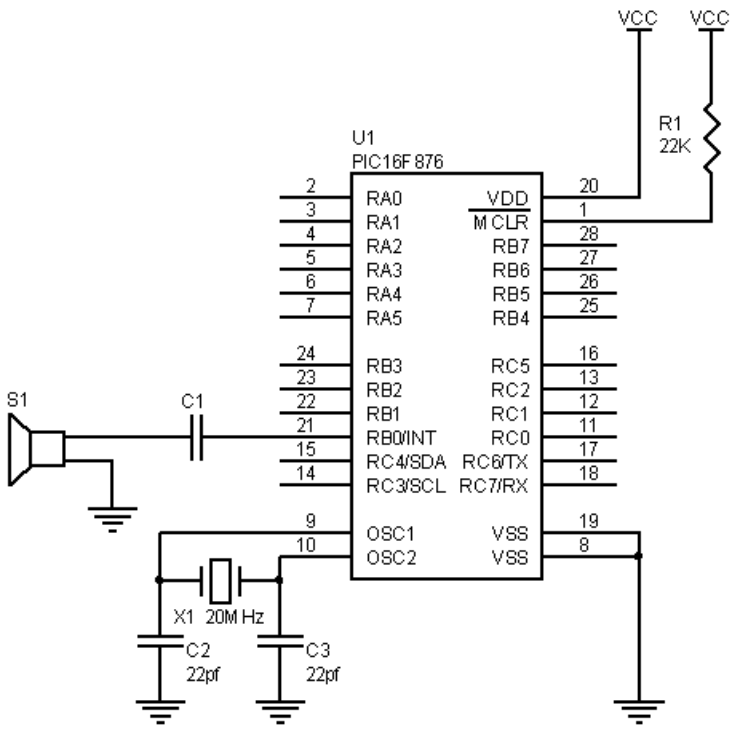
- **s** (style). Styles can be S=Staccato, N=Natural, C=Continuous.

- **l** (looping). The loop value can be 0 to 15. 0 disables looping. 15 enables infinite looping. Values between 1 and 14 specify how many loops to make before stopping.

If any of the parameters is missing from the control section, the following defaults are assumed : 4=duration, 6=scale, 63=beats-per-minute.

The circuit

As you can see the circuit required to generate tones is very simple. The 20MHz crystal controls the timing and can not be substituted with another value without recalculating the divisors for each tone.



Calculations for 20MHz

The following spreedsheet shows the desired and actual frequencies for each note @ 20MHz. The code supports 4 octaves.

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

The source code

The code has been written in C and compiled with the Hi-Tech PICC Compiler. HiTech Software have a demo version of the PICC for download which works for 30 days. A pre-compiled .HEX file has been included in the archive which has been compiled for use with (or without) the ICD.

To add new tones is simply a matter of cut and paste. You may choose to search Overtonez.co.uk or any number of internet sites for new ring tones. Once you have one, simply cut the note commands into the `Melody[ ]` array and adjust the `defaultduration`, `defaultoctave` and `beat_speed` to suit.

If you have a Nokia F-Bus cable, you can download your favourite tunes from your phone using software such as Logomanager or the Oxygen Phone Manager. You can then save them as .rtl files and paste it into your source code.

```
/************************************************************************/
/*                                                                      */
/*       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%)
```

Downloading the Source Code

- Version 1.0, 14k bytes

  Revision History

  - 17th August 2003 - Version 1.0.

Glossary

- **Semibreve -** Semibreve is the British term for Whole Note. A semibreve is worth 4 beats.
- **Minim -** Minim is the British term for Half Note. A minim is worth 2 beats.
- **Crotchet -** Crotchet is the British term for Quarter Note. A crotchet is worth 1 beat.
- **Quaver -** Quaver is the British term for 8th Note. A quaver is worth 1/2 a beat.
- **Semiquaver -** Semiquaver is the British term for 16th Note. A semiquaver is worth 1/4 of a beat.
- **Demisemiquaver -** Demisemiquaver is the British term for 32nd Note. A demisemiquaver is worth 1/8 of a beat.
- **Hemidemisemiquaver -** Hemidemisemiquaver is the British term for 64th Note. A hemidemisemiquaver is worth 1/16 of a beat.
- **Octave -** With the 12 musical notes, let's call note number one C. If we start on C and work our way up the 12 notes in pitch, we will eventually hit C again but of a higher pitch (exactly one octave higher). At this point the C we are playing is in the next Octave on from the C we started on. For example if we started on C4 (4th Octave) we would end up on C5 (5th Octave) and this can keep going endlessly until the frequency of the pitch reaches beyond our aural hearing frequency range. The same can apply going down in pitch / octaves. So Octave is specifiying what Octave or Pitch/Frequency Range to play your specified note from.
- **Staccato -** Staccato is a direction to perform a note quickly, lightly, and seperated from the notes before and after it. Staccato performance in practice reduces the time value of a note by 50%, thus a staccato'd crotchet (quarter note) lasts only as long as a quaver (8th note).

Links

- http://www.ringtone-converters.com/christmas - Many Christmas Ring Tone Tunes in RTTTL Format.

- http://overtonez.co.uk/frame_me/index.pl - OvertoneZ - Search and download ASCII RTTTL Ring Tones to import into your source code.

- http://www.htsoft.com/ HiTech Software - Make the PICC C Cross Compiler for Microchip PIC16x Family of Microcontrollers.
- http://www.microchip.com - Microchip PIC Series of Microcontrollers
- http://www.microchipc.com - Program Microchip PIC micros with C

Musical terms and insight kindy provided by Andrew Toner of http://www.andrewtoner.com