# Chapter 3 Ring Tone Music

## 3.1 Simple beep by timer 2

The hardware is shown in Figure 3.1.1. Pin P1.0 drives the BUZZER U5 via Q2 for amplification. Formally U5 is a piezo sounder. It requires an a.c. signal to drive it instead of a D.C. voltage because there is no internal oscillator. R5 is a current limiting resistor. If a louder noise is required, a smaller R5 value can be chosen.

Driving a sounder is not much different from driving an LED in chapter 2. We could use Timer 0 or even a software delay for timing, toggle P1.0 to output high/low for a given square wave period. There is a whole chapter in the reference "Microcontroller Projects in C for the 8051 by Dogan Ibrahim" so it is not repeated here.

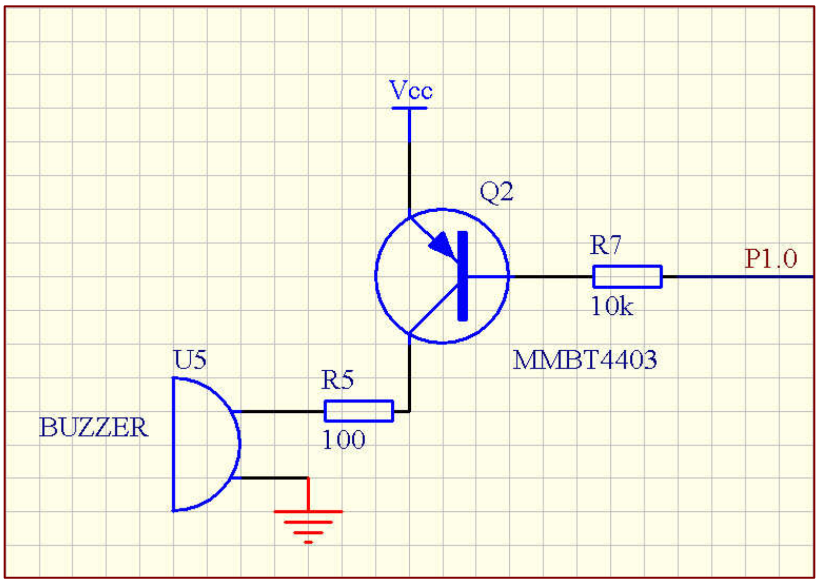


Figure 3.1.1

**Timer 2 is used in this section to generate a programmable clock out** signal to P1.0. It is one of the unique hardware features of AT89S52. A 50% duty cycle clock is programmed to come out on P1.0. The Clock-out frequency is give by the following equation. For further details, please refer to page 16 of ATMEL AT89S52 data sheet.

$$\text{Clock-Out Frequency} = \frac{\text{Oscillator Frequency}}{4 \times [65536-(\text{RCAP2H},\text{RCAP2L})]}$$

Rearranging the equation, we can obtain the RCAP2H,RCAP2L values.

**RCAP2H,RCAP2L = 65536 – Oscillator Freq/(4*Clock-out Freq.)**

Suppose we want to generate a tone "Do" of frequency 262Hz, the value of RCAP2H,RCAP2L can be calculated as:

RCAP2H,RCAP2L = 65536 – 11.0592x1,000,000/4*262
= 65536 – 10552
= 54984.

Thus RCAP2H = 0xD6
RCAP2L = 0xC8.

The source code of this chapter is located at cd:\src\chp3\src3_1\. Project name is buzzer.uv2.

```
#include <REGX52.h>                                              (1)
…

sbit LED       = P2^0;                                           (2)
sbit BUZZER    = P1^0;                                           (3)
…

void ToneGen(unsigned int tone, unsigned int duration)           (4)
{
        unsigned long tmp;
        if(tone!=0)
        {
        T2CON = 0x00;                                            (5)

        T2MOD = T2MOD&0xFE;                                      (6)
        T2MOD = T2MOD|0x02;
        tmp = (11059200UL>>2)/tone;                              (7)
        tmp = 65536UL - tmp;
        RCAP2H = (unsigned char) (tmp>>8);                       (8)
        RCAP2L = (unsigned char) (tmp&0x000000FF);
        TH2 = RCAP2H;
        TL2 = RCAP2L;

        TR2 = 1;                                                 (9)
        DelayMs(duration);                                       (10)
        TR2 = 0;                                                 (11)
        BUZZER = OFF;                                            (12)
        }
}

void main(void)
{
        Tmr0Init();
        EA = 1;                                                  (13)

        for(;;)
        {
                ToneGen(262, 1000);       //Do                   (14)
                ToneGen(294, 1000);       //Re
                ToneGen(330, 1000);       //Mi
                ToneGen(349, 1000);       //Fa
                ToneGen(392, 1000);       //So
                ToneGen(440, 1000);       //La
                ToneGen(494, 1000);       //Si
        }
}
```

**Listing 3.1**

**Line (1)** is the include file different from previous examples. This time, the register T2MOD is needed for Timer 2 mode control which is not there inside the generic REG52.h register file. This header file REGX52.h can be found under the directory C:\Keil\C51\INC\Atmel.

**Line (2)** is the hardware definition for the LED D3. Exactly the same as the example under section 2.2. This is just to show a real-time concept here. Even though this program is making tones in its main loop, the LED is still flashing out of the Timer 0 interrupt.

**Line (3)** is the hardware definition for the Buzzer.

**Line (4)** is the entry point of the tone generation function ToneGen( ).

**Line (5)** resets T2CON in which TR2 (T2CON.2) bit is located. TR2 starts and stops Timer 2. It also clears T2CON.1 (bit C/T2) for clock generation. There are other functions of Timer 2 besides continuous clock out, such as capture and 16-bit timer (pls refer to page 8 of AT89S52 data sheet). They are not used in this example so T2CON is reset to zero.

**Line (6)** configures the Timer 2 as a clock generator, by setting bit T2OE (T2MOD.1).

**Line (7)** begins computation of the capture registers RCAP2H and RCAP2L. A temporary variable 'tmp' is used for computation on (11059200/(4*tone)). Bitwise >>2 is equivalent to an operation of division by 4.

**Line (8)** substitutes the capture registers and then preloads TH2 and TL2. Up to here, Timer 2 is ready for clock out a square wave of frequency specified by the variable 'tone', i.e. making a single note of frequency given by the variable 'tone'.

**Line (9)** Starts Timer 2 thus starts making noise!

**Line (10)** delays the tone for a duration of xxx millisecond, specified by the variable 'duration'. This is a potential problem if we need a real-time system. The program is hanged there for hundreds of millisecond! We will see how to solve this problem in the next section.

**Line (11)** stops the note.

**Line (12)** makes sure there is no current flow through the BUZZER upon exit.

**Line (13)** is actually not related to making tone; instead, it is just for flashing LED. Try amending EA = 0, tone still there. In the clock-out mode, Timer 2 roll-overs will NOT generate an interrupt.

**Line (14)** starts Do, Re, Mi, Fa, So…..There is a long delay loop by DelayMs( ) inside ToneGen( ), but the LED still flashes out of Timer 0 interrupt.

Again, compile the program by **Rebuild Target** and program the results HEX file buzzer.hex into the mcu. You will hear some single tones with a blinking LED.

## 3.2 Play music by Timer Interrupt

In section 3.1, we play music "Do", "Re", "Mi" by the following steps:

1. Preload (RCAP2H,RCAP2L) and load TH2 and TL2 by RCAP2H,L
2. Enable Timer 2 programmable clockout (TR2 = 1)
3. Delay for the required note duration by DelayMs(duration)
4. Stops Timer 2 clockout (TR2=0) and then set BUZZER pin to high impedance

```
void ToneGen(unsigned int tone, unsigned int duration)
{
        …..
        RCAP2H = (unsigned char) (tmp>>8);
        RCAP2L = (unsigned char) (tmp&0x000000FF);          1
        TH2 = RCAP2H;
        TL2 = RCAP2L;

        TR2 = 1;                                             2

        DelayMs(duration);   //potential problem             3

        TR2 = 0;                                             4
        BUZZER = OFF;
}
```

This is alright if we want the mcu to do nothing else except playing notes. However, most often we prefer leaving the main() loop for other tasks as well. For example, we may want the following features from the main() loop.

```
void main(void)
{
        for(;;)
        {
        //gets input from key for user interaction
        //if key = UP, play music "happy birthday" Do, Re, Mi…..
        //if key = DOWN, stop music….
        //blink an LED for 1 sec as a confirmation
        //update LCD to show the name of the song
        …. etc
        }
}
```

**Figure 3.2.1**

The logic in Figure 3.2.1 is reasonable in the sense that, a music player should be able to respond to user interaction and/or give visual indication any time after music started. There is a problem with the code in section 2.1. Software delay function "DelayMs(x)" holds the mcu by an infinite for-loop until expiry of xxx millisecond. The period can be as long as 500msec or even 1 sec depending on what music to play. To solve this, we may use hardware timer (Timer 0 or Timer 1) to count the note duration and still use Timer 2 for frequency out.

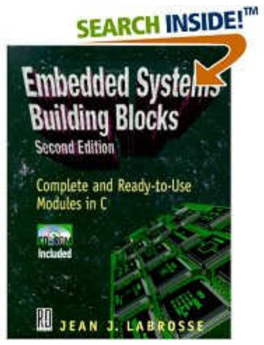
The source code of this section is located at cd:\src\chp3\src3_2\. Project name is ringtone1.uv2.

This program plays a short ring tone music composed of different notes with individual note duration defined in arrays noteDuration[] and noteFreq[], respectively. Instead of using software delay, hardware Timer 0 is used here. The noteDuration[] is defined in the unit of millisecond.

Therefore, noteDuration[0]=15 => the first note sounds for 150ms;
noteFreq[0]=2093 => the note frequency is 2093 Hz.

Timer 0 is employed to count expiry of the noteDuration[noteIdx], with noteIdx being the array index. Since hardware timer Timer 0 has been loaded with (TH0,TL0) of 0xDC00, it gives 10ms interrupt when external crystal is 11.0592MHz. A variable notePlayCtr is used to make a countdown on the variable 'noteDuration[noteIdx]' until the end of the ring tone music, indicated by a noteDuration of 0.

Reference is made to the world-famous RTOS (Real-Time Operating System) author Jean J. Labrosse's book, Embedded Systems Building Blocks: Complete and Ready-To-Use Modules in C, 2nd edition. There is a chapter inside about Timer Manager. I have extracted some of the code there for Timer 0.

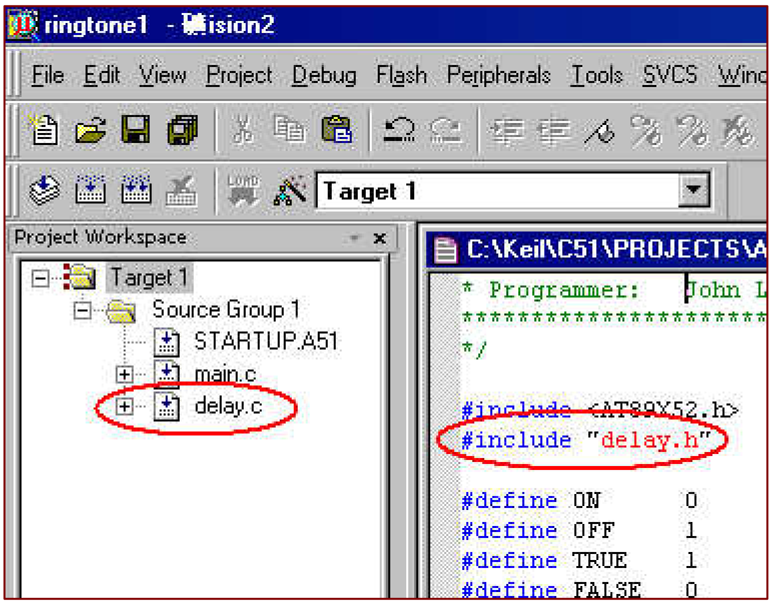

SEARCH INSIDE!™
Embedded Systems Building Blocks
Second Edition
Complete and Ready-to-Use Modules in C
Included
JEAN J. LABROSSE

```
#include <REGX52.h>
#include "delay.h"                                           (1)

#define ON          0
#define OFF         1
#define TRUE        1
#define FALSE       0
#define CLOSE       0
#define OPEN        1

sbit LED       = P2^0;
sbit KEYUP     = P1^6;                                       (2)
sbit KEYDN     = P1^4;                                       (3)
sbit BUZZER    = P1^0;

bit notePlayEn = FALSE;                                      (4)
unsigned int notePlayCtr=0;                                  (5)
unsigned char noteIdx=0;                                     (6)
code unsigned int noteDuration[]                             (7)
={15,15,30,15,15,30,15,15,15,15,15,15,30,15,15,15,…,0};
code unsigned int noteFreq[]                                 (8)
={2093,880,0,2093,1760,0,2093,880,2093,880,…1760,4186,0};
```

**Listing 3.2.1**

**Listing 3.2.1** shows the first part of the source code.

**Line (1)** is the include statement for software delay module. Because software delay function is very common and frequently called, it has been made a re-usable software module. We just need to make an **include "delay.h"** statement at the beginning of our code and "Add files" under the source group as shown in Figure 3.2.2. Then our main( ) can make use of DelayMs() and DelayUs() defined under delay.h. Please consult delay.h for details of these functions.



```
* Programmer:   John L
***********************
*/

#include <AT89X52.h>
#include "delay.h"

#define ON      0
#define OFF     1
#define TRUE    1
#define FALSE   0
```

**Figure 3.2.2**

**Line (2) & (3)** define two keys of the 5-way navigator switch. Referring to schematic we know KEYUP is wired to pin 1.6, and KEYDN wired to pin 1.4.

**Line (4)** defines the enable/disable flag for music playback. It is a gate for the countdown process for the notePlayCtr in Line (5).

**Line (5)** defines the notePlayCtr. Timer 0 has been configured for a 10ms interrupt. When notePlayEn is enabled, notePlayCtr decrements towards 0.

Therefore, total time suspended = notePlayCtr * 10 ms.

If we want to play a certain note of frequency 2093Hz for 150ms (say), we have to load notePlayCtr with 15.

**Line (6)** is the array index for noteDuration[] and noteFreq[] arrays.

**Line (7)** is the note duration array defined in code space.
**Line (8)** is the note frequency array defined in code space.

Altogether, Line (6), (7), & (8) define the ring tone music.

The heart of this program is shown in Listing 3.2.2.

```
Tmr0() interrupt 1
{
        if(notePlayEn==TRUE){                                   (1)
                if(notePlayCtr>0){
                        notePlayCtr--;                          (2)
                        if(notePlayCtr==0){                     (3)
                                notePlayEn = FALSE;
                                if(noteDuration[noteIdx]!=0)    (4)
                                {
                                        ToneGen(noteFreq[noteIdx]);             (5)
                                        notePlayCtr = noteDuration[noteIdx];    (6)
                                        noteIdx++;                              (7)
                                        notePlayEn = TRUE;
                                } else {
                                        TR2     = 0;                            (8)
                                        BUZZER       = OFF;                     (9)
                                }
                        }
                }
        }
        TH0 = 0xDC;                                             (10)
        TL0 =  0x00;
}
```

**Listing 3.2.2**

**Line (1)** is used to enable and disable the countdown process. Because Tmr0() interrupt executes every 10ms, the statement if(notePlayEn==TRUE) is checked every 10ms.

**Line (2)** decrements notePlayCtr until 0 in every 10ms if notePlayEn==TRUE.

**Line (3)** checks if notePlayCtr expired.

**Line (4)** checks if it is the end of music.

**Line (5)** loads the noteFreq[noteIdx] to the ToneGen() function, thus making a single tone of frequency 'noteFreq[noteIdx]'.

**Line (6)** loads the corresponding note duration noteDuration[noteIdx] to notePlayCtr for a countdown process. This is the key to count the note duration.

**Line (7)** prepares for the next note.

**Line (8)** turn off TIMER 2 if it is the last note of the song.

**Line (9)** is to make sure the BUZZER pin in high impendence state upon finished playback.

**Line (10)** reloads Timer 0 for 10ms interrupt.

Listing 3.2.3 shows the start and stop functions for playing the music.

```
void notePlayStart(void)
{
        noteIdx = 0;                                                (1)
        notePlayEn = TRUE;                                          (2)
        notePlayCtr = 1;                                            (3)
}

void notePlayStop(void)
{
        notePlayEn = FALSE;                                         (4)
        TR2     = 0;                                                (5)
        BUZZER = OFF;                                               (6)
}
```

**Listing 3.2.3**

**Line (1)** sets the noteIdx 0; therefore, we instruct to play from the beginning of the music. We could have set the noteIdx to a random number as long as it is within the array size of the music. In this case, the music is played from the middle.

**Line (2)** starts counting the notePlayCtr.

**Line (3)** loads the first notePlayCtr value to 1 for an immediate play action. Upon entry of the Tmr0 interrupt service routine, if the gate of notePlayEn enabled, value of notePlayCtr would be checked for a positive value >0. Preloading a value 1 to notePlayCtr makes notePlayCtr equal 0 after the first entry to Line (2) of Listing3.2.2, thus making the code execute ToneGen(noteFreq[0]) for the first note. A square pulse of period defined by noteFreq[0] is generated so we hear the tone from Buzzer.

**Line (4)** stops playing the music by disabling the notePlayEn gate.

**Line (5)** is to make sure Timer 2 in clock-out mode turned off.

**Line (6)** is to make sure BUZZER pin P1.0 in high impedance state upon exit.

Here we come to the main loop of our program. We want to give the program some user interaction. There is a 5-way navigator switch below the graphical LCD. This switch is like the navigator switch found on most mobile phones or digital cameras. Inside, it is not much different from a combination of 5 push buttons, as shown in Figure 3.2.3.

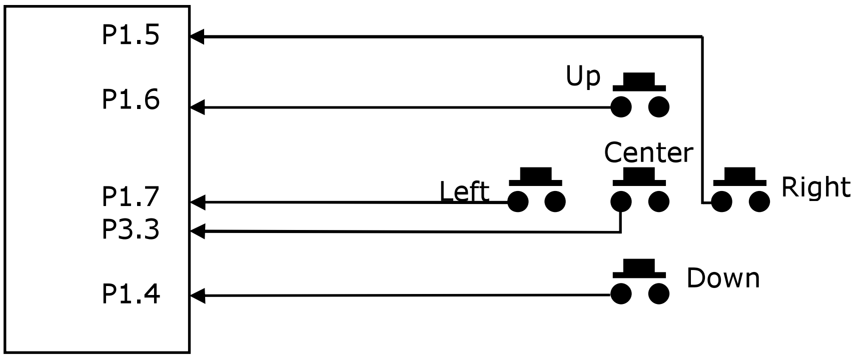


**Figure 3.2.3** Remarks: keys' pins short to ground to read 0 when pressed

Recall from listing 3.2.1 that the button UP and button DOWN have been assigned name of KEYUP and KEYDN by the following code:

```
sbit KEYUP = P1^6;
sbit KEYDN = P1^4;
```

Therefore we may use the keyword KEYUP and KEYDN in our program without referring to schematic again. We want the following logic (Listing 3.2.4) for demonstration purpose.

```
void main(void)
{
        Tmr0Init();
        EA = 1;
        for(;;)
        {
                if(KEYUP==0)
                {
                        //key debounce for 20 ms
                        //if that is a true KEYUP press, wait until key release
                        //play the ringtone and turn on LED D3
                }

                if(KEYDN==0)
                {
                        //key debounce for 20 ms
                        //if that is a true KEYDN press, stop playing the ringtone
                        //turn off LED D3
                }
        }
}
```

**Listing 3.2.4**

Key press detect is done within the infinite loop inside main(void). Therefore KEYUP and KEYDN pins are checked in a round-robin manner here. Pin I/O for 8051 is different from other microcontrollers like Microchip's PIC, or Texas Instrument's MSP430, in which there are registers controlling I/O direction. Honestly, this gave me a little hesitation when I began with 8051 because I started with PIC micro. Writing a value of 1 to any pin make that pin as input. Upon system reset, all ports would be reset to a value 0xFF, so all pins default to input. Therefore we just check for the result if KEYUP/KEYDN is shorted to ground by the following statements:

```
If(KEYUP==0)….

If(KEYDN==0)….
```

Listing 3.2.5 shows the actual program void main(void).

```
void main(void)
{
        Tmr0Init();                                                 (1)
        EA = 1;

        for(;;)
        {
                if(KEYUP==CLOSE)                                    (2)
                {
                        DelayMs(20);                                (3)
                        if(KEYUP==CLOSE)                            (4)
                        {       while(KEYUP==CLOSE);                (5)
                                notePlayStart();                    (6)
                                LED = ON;                           (7)
                        }
                }

                if(KEYDN==CLOSE)                                    (8)
                {
                        DelayMs(20);
                        if(KEYDN==CLOSE) notePlayStop();            (9)
                        LED = OFF;                                  (10)
                }

        }
}
```

**Listing 3.2.5**

**Line (1)** configures Timer 0 for a 10ms interrupt, important for timing note duration.

**Line (2)** checks for key press for KEYUP. CLOSE is equivalent to 0 as defined earlier.

**Line (3)** debounces the key for 20 ms. There is a better way to debounce. We would leave it later.

**Line (4)** checks if the key is still pressed after the debounce period. If it is a LOW (0), that key press is a real one.

**Line (5)** waits until key release. What happens if we skip this? The music would be played repeating from the first note, depending on how fast the infinite loop for(;;) runs! Try putting a comment in front of this line, recompile the program and load the corresponding hex code to the mcu and see. It can be funny.

```
//while(KEYUP==CLOSE);
```

This code is again a potential problem! The mcu hangs up at the while-loop as long as the user is holding the key. Imagine the situation with our mobile phone. It would be disastrous if it hanged up and couldn't received phone call on a number key hold! We would solve this problem in chapter 4.

However, this while-loop statement gives us the sufficient information to prove music is played in a proper manner by hardware Timer 0 interrupt. Upon releasing the key, you may try pushing it up again to hold the program at the while-loop statement. You will still hear the ring tone music, until the key is released then the music would be started from the beginning again. This shows we are playing the music by a hardware manner.

**Line (6)** starts playing ring tone upon release of the KEYUP.

**Line (7)** turns on the LED D3 as a visual indicator.

**Line (8) – (10)** stop playing the ring tone and turn off LED D3. The logic is the same as starting playback.

⇒ **Chapter 4**