

DTMF Generator by Using PWM

Using PWM to Generate DTMF Signals

DESCRIPTION

Many telecom applications, such as auto dialers, telephone keypads and security systems, require DTMF transmission for dialing and data transmission. This application note describes how DTMF(Dual Tone Multiple Frequencies) signaling can be implemented using any HMS800 series microcontroller with 10bit high resolution PWM. Applications such as phones are using DTMF signals for transmitting dialing information. DTMF signals consisted of two different signal which are high frequency and low frequency. These two different Sine waves are synthesized to make DTMF signals. Table 1 shows how the different frequencies are mixed to generator DTMF tones.

Table 1: DTMF Tone Matrix

fa / fb	697Hz	770Hz	852Hz	941Hz
1209Hz	1	4	7	*
1336Hz	2	5	8	0
1477Hz	3	6	9	#

The rows of the matrix shown in Table 1 represent the low frequencies while the columns represent the high frequency values. For example, this matrix shows that digit 1 is represented by a low frequency of fb=697Hz and a high frequency of fa = 1336Hz. These two frequencies are synthesized to generate a DTMF signal using equation 1.

$$f(t) = (A_a \sin(2\pi f_a t) + B_b \sin(2\pi f_b t)) \quad (1)$$

$$(A_a/B_b) = K \quad 0.7 < K < 0.9$$

Theory of Operation

Starting from a general overview about the usage of the PWM, it will be shown how the PWM allows to generate sine waves. In the next step, an introduction is given in low frequencies that are different from the ground frequency of the PWM can be generated.

Generating Sine Waves

According to the relation between high and low level at the output pin of the PWM, the average voltage at this pin varies. Keeping the relation between both levels constant generates a constant voltage level[AN010].

A sine wave can be generated if the average voltage generated by the PWM is changed every PWM period. The relation between high and low level has to be adjusted

according to the voltage level of sine wave at the respective time. Figure 1 describes that the solid part is voltage level which is adjusted every PWM period from lookup table.

Figure 1 also shows the dependency between frequency of the ground sine wave and the amount of samples. The more samples are used, the more accurate the output signal gets. Equation 2 shows this correlation:

$$f = \frac{f_{PWM}}{N_s} = \frac{f_s/255}{N_s} \quad (2)$$

f = Sine Wave frequency

f_{PWM} = PWM frequency

f_s = Timer Clock

N_s = Number of Samples

The PWM frequency is dependent on the PWM resolution. For an 8-bit resolution, the Timer maximum value is 0xff. So f_{PWM} is 31.372kHz at 8MHz of f_s. If N_s is 12, f is 2.6kHz.

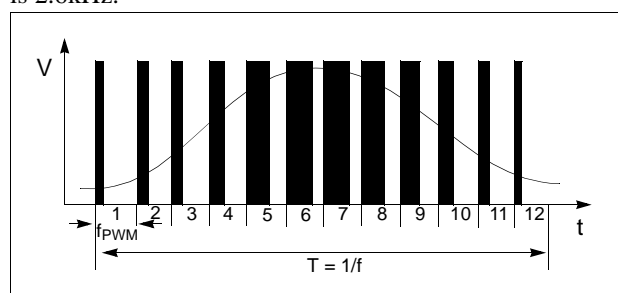


Figure 1. Generation a Sine Wave with PWM

Let's assume that the sinusoid samples for adjusting the PWM are not read in a sequentially manner from the lookup table but just every second value. At the same sample frequency an output signal with twice the frequency is generated.

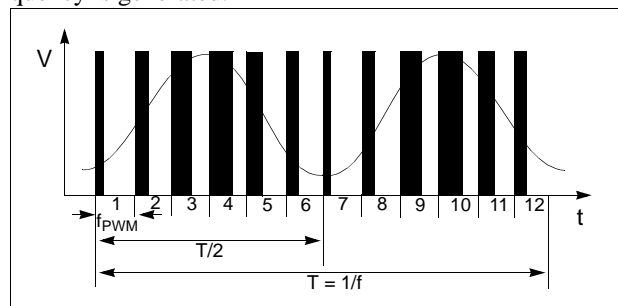


Figure 2. Doubling the Output Frequency

In using not every second sample but every third, fourth, fifth and so on. it is possible to generate N_S different frequencies in the range from $[1/T \text{ Hz} \dots 0 \text{ Hz}]$. The step width between samples is specified by X_{SW} . Equ.3 describes this relation

$$X_{SW} = f \cdot \frac{N_S}{f_{PWM}} = \frac{f \cdot N_S \cdot 255}{f_S} \quad (3)$$

How to calculate the actual value with which the PWM has to be adjusted every PWM period is shown equ.4. Based on the value of the previous period X'_{LUT} the new value X_{LUT} is calculated in adding the step-width(X_{SW}).

$$X_{LUT} = X'_{LUT} + X_{SW} \quad (4)$$

X'_{LUT} = old position in lookup table
 X_{LUT} = new position in lookup table

Synthesis of the two different frequencies sine waves

A DTMF signal has to be generated according to equ.1. Since this is easy to obtain with simple shift register operations a K-Factor of $K=3/4$ has been chosen. By using equ.4 the lookup table position of the next value for adjusting the PWM can be calculated as follows. A DTMF tone generator is built using one of the 8bit PWM outputs and a sinusoid table with $N_S=128$ samples each with $n=7$ bit.

$$f(X_{LUT}) = f(X_{LUTa}) + \frac{3}{4} \cdot f(X_{LUTb}) \quad (5)$$

$$X_{LUTa} = X'_{LUTa} + X_{SWa}$$

$$X_{LUTb} = X'_{LUTb} + X_{SWb}$$

$$f(x) = 63 + 63 \times \sin\left(\frac{2\Pi x}{128}\right) \quad (x \in 0 \dots 127) \quad (6)$$

The advantage in using 7 bits is that the sum of the high and low frequency signals fits in one byte. To support the whole DTMF tone set, we have to calculate eight X_{SW} values, one for each DTMF frequency, and place them in a table.

To achieve a higher accuracy, the following solution has been implemented. X_{SW} values calculated after equation 3 need only five bits. To use all eight bits to have a lower round off error. this value is multiplied by eight. The pointer to the lookup table is saved in the same manner. But here two bytes are needed to store the actual value times eight. This means that three right shifts and a module operation with N_S have to be executed before using them as pointers to the sine values in the lookup table.

$$X_{LUTa,b} = \text{Round}\left(\frac{1}{8}\left(X'_{LUTa,b} + \frac{8N_S f_{PWM} 255}{f_S}\right)\right) \quad (7)$$

Figure 3 shows simple schematics of HMS87C1816B for DTMF generator. And Figure 4 is DTMF 0 key simulation result of using MATLAB.

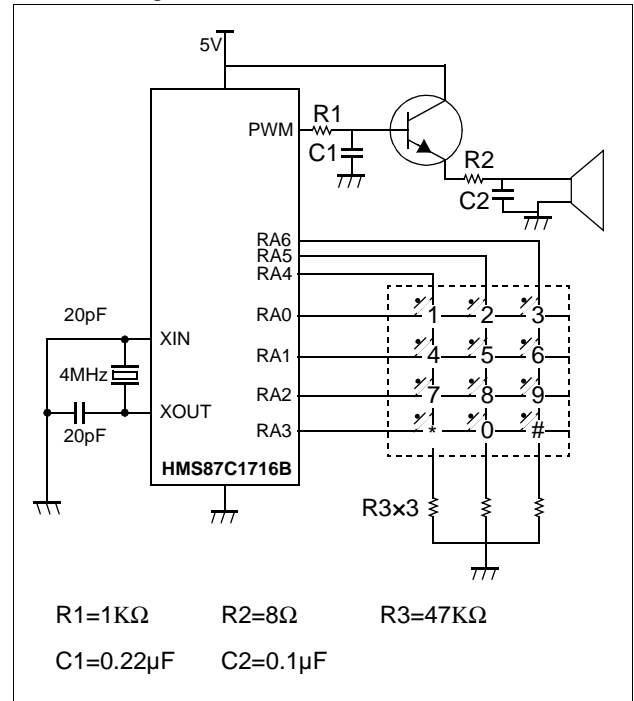


Figure 3. DTMF Generator

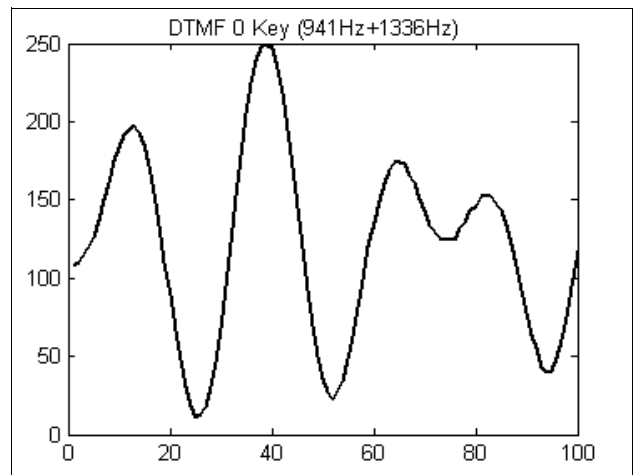


Figure 4. DTMF 0 key simulation result

Author: June Choi
 MCU application team
 e-mail: jungmin1.choi@hynix.com

Appendix A:

```

/*****
Programmer : Jungmin Choi
Data      : 2004.06.01
Subject   : DTMF generator using PWM
Device    : HMS87C1816B
File name : DTMF.c
*****/

#include "HMS800.h"
#include "HMS87C1816B.h"
#define PWM_ON      (0x10)
#define EI          asm("EI")

typedef union UNI_BYTE
{
    unsigned char nBYTE;
    struct nBIT
    {
        char f0:1;      // 0   slection flag
        char f1:1;      // 1   continue cook flag
        char f2:1;      // 2   manual cook flag
        char f3:1;      // 3   AC 100msec flag
        char f4:1;      // 4   AC 500msec flag
        char f5:1;      // 5   AC 1sec flag
        char f6:1;      // 6   AC 8sec flag
        char f7:1;      // 7   4sec flag for continue key check
    } nBIT;
} UNI_BYTE;

unsigned char SIN[] CODE = {
    63,  66,  69,  72,  75,  78,  81,  84,  87,  90,  93,  95,  98,
    101, 103, 105, 108, 110, 112, 114, 115, 117, 119, 120, 121, 122,
    123, 124, 125, 125, 126, 126, 126, 126, 126, 125, 125, 124, 123,
    122, 121, 120, 119, 117, 115, 114, 112, 110, 108, 105, 103, 101,
    98,  95,  93,  90,  87,  84,  81,  78,  75,  72,  69,  66,  63,
    60,  57,  54,  51,  48,  45,  42,  39,  36,  33,  31,  28,  25,
    23,  21,  18,  16,  14,  12,  11,  9,   7,   6,   5,   4,   3,
    2,   1,   1,   0,   0,   0,   0,   0,   1,   1,   2,   3,   4,
    5,   6,   7,   9,  11,  12,  14,  16,  18,  21,  23,  25,  28,
    31,  33,  36,  39,  42,  45,  48,  51,  54,  57,  60 };

unsigned char row_1[] CODE = {
    4,   7,  10,  13,  15,  18,  21,  24,  27,  30,  33,  36,  38,  41,  44,
    47,  50,  53,  56,  59,  61,  64,  67,  70,  73,  76,  79,  82,  84,  87,
    90,  93,  96,  99, 102, 105, 107, 110, 113, 116, 119, 122, 125, 128,  2, 0 };

unsigned char row_2[] CODE = {
    4,   7,  10,  14,  17,  20,  23,  26,  29,  32,  35,  39,  42,  45,  48,
    51,  54,  57,  60,  64,  67,  70,  73,  76,  79,  82,  85,  89,  92,  95,
    98, 101, 104, 107, 110, 114, 117, 120, 123, 126,  1,  0 };

unsigned char row_3[] CODE = {
    5,   8,  12,  15,  19,  22,  26,  29,  33,  36,  40,  43,  47,  50,  54,
    57,  61,  64,  68,  71,  75,  78,  82,  85,  89,  92,  96,  99, 103, 106,
    110, 113, 117, 120, 124, 127,  0 };

unsigned char row_4[] CODE = {
    5,   9,  13,  17,  20,  24,  28,  32,  36,  40,  44,  48,  51,  55,  59,
    63,  67,  71,  75,  79,  82,  86,  90,  94,  98, 102, 106, 110, 113, 117,
    121, 125,  0 };

unsigned char col_1[] CODE = {
    6,  11,  16,  21,  26,  31,  36,  41,  46,  51,  56,  61,  66,  71,  76,
    81,  86,  91,  96, 101, 106, 111, 116, 121, 126,  0 };

unsigned char col_2[] CODE = {

```

```

    7, 12, 18, 23, 29, 34, 40, 45, 51, 56, 62, 67, 73, 78, 84,
    89, 95, 100, 106, 111, 117, 122, 128, 0 };
unsigned char col_3[] CODE = {
    7, 13, 19, 25, 31, 37, 43, 49, 55, 61, 67, 73, 79, 85, 91,
    97, 103, 109, 115, 121, 127, 0 };
unsigned char col_4[] CODE = {
    8, 15, 21, 28, 35, 42, 48, 55, 62, 69, 75, 82, 89, 96, 102,
    109, 116, 123, 0 };
unsigned char key_strobe_data[] CODE = {
    0x01, 0x02, 0x08, 0x04, 0x10};
unsigned char key_number_data[] CODE = {
    0, 4, 8, 12, 16};

UNI_BYTE Normal_Flag;
#define flmsec          Normal_Flag.nBIT.f0

UNI_BYTE Timer3_Flag;
#define flusec          Timer3_Flag.nBIT.f0

UNI_BYTE Key_Flag;
#define fkeyon          Key_Flag.nBIT.f0
#define fchatt          Key_Flag.nBIT.f1
#define fnokey          Key_Flag.nBIT.f2

unsigned char temp,temp1,temp2;
unsigned char *Xa, *Xb;
unsigned char Xa_Cnt,Xb_Cnt;
unsigned char newkey;
unsigned char oldkey;
unsigned char totkey;
unsigned char prtdat;
unsigned char chtcnt;
unsigned char strobe_cnt;

void Key_Check_fun(void)
{
    if ( !fkeyon )
    {
        if ( strobe_cnt == 0 )
        {
            newkey = 0;
            totkey = 0;
            prtdat = 0;
        }
        RA = key_strobe_data[strobe_cnt];
        asm("nop");
        asm("nop");
        asm("nop");
        prtdat = RA & 0xf0;

        if ( prtdat )
        {
            temp = key_number_data[strobe_cnt];
            asm("nop");
            if ( prtdat & 0x10 ) { totkey ++; newkey = temp; }
            if ( prtdat & 0x20 ) { totkey ++; newkey = temp+1; }
            if ( prtdat & 0x40 ) { totkey ++; newkey = temp+2; }
            if ( prtdat & 0x80 ) { totkey ++; newkey = temp+3; }
        }
    }
}

```

```

    if ( strobe_cnt >= 3 )
    {
        if ( totkey == 1 )
        {
            if ( fnokey )
            {
                if( newkey != oldkey )
                {
                    oldkey = newkey;
                    fchatt = 0;
                    chtcnt = 0;
                }
            }
            else
            {
                if ( !fchatt )
                {
                    if ( chtcnt > 5 )
                    {
                        oldkey = newkey;
                        fkeyon = 1;
                        fchatt = 1;
                    }
                    else
                    {
                        chtcnt ++;
                    }
                }
            }
        }
        else
        {
            oldkey = newkey;
            chtcnt = 0;
            fchatt = 0;
        }
    }
    else
    {
        if ( totkey == 0 )
            fnokey = 1;
        else
            fnokey = 0;

        oldkey = newkey;
        chtcnt = 0;
        fchatt = 0;
    }
}
strobe_cnt++;
if ( strobe_cnt > 3 )
    strobe_cnt = 0;
}

void NUM1(void)
{
    Xa = row_1;
    Xb = col_1;
}
void NUM2(void)
{

```

```
    Xa = row_1;
    Xb = col_2;
}
void NUM3(void)
{
    Xa = row_1;
    Xb = col_3;
}
void NUM4(void)
{
    Xa = row_2;
    Xb = col_1;
}
void NUM5(void)
{
    Xa = row_2;
    Xb = col_2;
}
void NUM6(void)
{
    Xa = row_2;
    Xb = col_3;
}
void NUM7(void)
{
    Xa = row_3;
    Xb = col_1;
}
void NUM8(void)
{
    Xa = row_3;
    Xb = col_2;
}
void NUM9(void)
{
    Xa = row_3;
    Xb = col_3;
}
void STAR(void)
{
    Xa = row_4;
    Xb = col_1;
}
void NUM0(void)
{
    Xa = row_4;
    Xb = col_2;
}

void JUNG(void)
{
    Xa = row_4;
    Xb = col_3;
}
void No_fun(void){}

void (* KeyFunction[])( void ) CODE = {
NUM0,  NUM1,  NUM2,  NUM3,
NUM4,  NUM5,  NUM6,  NUM7,
NUM8,  NUM9,  STAR,  JUNG,
No_fun, No_fun, No_fun, No_fun,
No_fun, No_fun, No_fun, No_fun};
```

```

void Key_fun(void)
{
    (* KeyFunction[newkey])();
    Xa_Cnt=0;
    Xb_Cnt=0;
    TM1 = 0x23;
    TM3 = 0x03;
    RBFUNC = PWM_ON;
    T3E = 1;
    T0E = 0;
}

void Initial_SFR(void)
{
    RAIO = 0x0f;          // low nibble : key scan, high nibble : key in
    RBIO = 0xfe;          // Buzzer(1)
    RCIO = 0xff;
    RDIO = 0xff;
    REIO = 0xff;

    /* Timer0 interrupt : 1msec */
    TM0 = 0x0f;
    TDR0 = 249;

    /* Timer3 interrupt : */
    TM2 = 0x0;
    TM3 = 0x03;
    TDR3 = 0xff;          // 255/8Mhz

    /* PWM setting */
    TM1 = 0x20;
    PWMOHR = 0;
    T1PPR = 0xff;          // 255/8Mhz Period setting

    CKCTLR = 0;
    RBFUNC = 0;
    RBIO = 0xff;
}

main()
{
    Initial_SFR();
    T0E = 1;
    EI;
    while(1)
    {
        if ( f1msec )
        {
            f1msec = 0;
            Key_Check_fun();
        }
        if( fkeyon )
        {
            fkeyon = 0;
            Key_fun();
        }
    }
}

```

```

/*****
Programmer : Jungmin Choi
Data      : 2004.06.01
Subject   : DTMF generator using PWM
Device    : HMS87C1816B
File name : Int.asm
*****/

RA      EQU      0C0h   ; /* [R/W] RA[7:0] Port Data Reg.      */
RD      EQU      0C6h   ; /* [R/W] RD[7:0] Port Data Reg.      */
RC      EQU      0C4h   ; /* [R/W] RC[7:0] Port Data Reg.      */
TM1     EQU      0D2h   ; /* [R/W] Timer1 Mode Control Reg.    */
RBFUNC  EQU      0CBh   ; /* [W]   RB Fuction Selection Reg.    */
T1PDR   EQU      0D4h   ; /* [R/W] PWM0 Duty Reg.(PWM Mode)    */
TM3     EQU      0D8h   ; /* [R/W] Timer3 Mode Control Reg.    */
IENH    EQU      0E2h   ; /* [R/W] INT Enable Reg. High        */
INT0E   EQU      7,IENH ; /* Ext. INT0 Enable                  */
INT1E   EQU      6,IENH ; /* Ext. INT1 Enable                  */
T0E     EQU      5,IENH ; /* Timer0 INT. Enable                */
T1E     EQU      4,IENH ; /* Timer1 INT. Enable                */
INT2E   EQU      3,IENH ; /* Ext. INT2 Enable                  */
INT3E   EQU      2,IENH ; /* Ext. INT3 Enable                  */
T2E     EQU      1,IENH ; /* Timer2 INT. Enable                */
T3E     EQU      0,IENH ; /* Timer3 INT. Enable                */

GLOBAL __.INT6

.sect .text
.globl __TM3_Int
.type __TM3_Int,@function
.interrupt __TM3_Int

__.INT6:

__TM3_Int:
    push    A
    push    Y
    push    X

Xb_S:
    ldy     __Xb_Cnt
    lda     [016h]+y
    cmp     #0
    bne     No_lotate_b
    ldm     __Xb_Cnt,#0
    bra     Xb_S

No_lotate_b:
    tay
    lda     !_SIN+y
    sta     __temp2
    inc     __Xb_Cnt

Xa_S:
    ldy     __Xa_Cnt
    lda     [02bh]+y
    cmp     #0
    bne     No_lotate_a
    ldm     __Xa_Cnt,#0
    bra     Xa_S

No_lotate_a:
    tay
    lda     !_SIN+y
    inc     __Xa_Cnt
    clrc

```

```

        adc     _temp2
        sta     T1PDR

        ldm     RA,0fh
        nop
        nop
        lda     RA
        and     #0f0h
        cmp     #0
        bne     no_key
        ldm     RBFUNC,#0
        ldm     TM3,#0
        clr1    T3E
        set1    T0E

no_key:
        pop     X
        pop     Y
        pop     A
        reti

```

```

/*****
Programmer : Jungmin Choi
Data       : 2004.06.01
Subject    : Batch file for compiling Int.asm and DTMF.C
Device     : HMS87C1816B
File name  : make.bat
*****/

```

```

hms800-elf-as Int.asm -o Int.out
hms800-cc -Os -g -Wa,"-ahls=DTMF.l" --option-file=HMS87C1816B.opt -c -o DTMF.out DTMF.c
hms800-cc -ramclr -Wl,"-Map=PWM.map" --option-file=HMS87C1816B.opt Int.out DTMF.out -o PWM.out
hms800-elf-objcopy -O srec PWM.out PWM.hex

```