To our customers,

Old Company Name in Catalogs and Other Documents

On April 1st, 2010, NEC Electronics Corporation merged with Renesas Technology Corporation, and Renesas Electronics Corporation took over all the business of both companies. Therefore, although the old company name remains in this document, it is a valid Renesas Electronics document. We appreciate your understanding.

Renesas Electronics website: http://www.renesas.com

April 1st, 2010
Renesas Electronics Corporation

Issued by: Renesas Electronics Corporation (http://www.renesas.com)

Send any inquiries to http://www.renesas.com/inquiry.

H8/300L

PWM Producing DTMF Signal (DTMF)

Introduction

Dual Tone Multi-Frequency (DTMF) is generally used for telephony application. In the case when the MCU does not carry a DTMF generator, a software implementation can be achieved by using the Pulse Width Modulation (PWM) module of the MCU.

This application note demonstrates ways to generate 16 distinct tones, in which each tone is the summation of two frequencies: one from a low and one from a high frequency group.

This technique can also be used for generating other composite signals.

Target Device

SLP – H8/38024

Contents

1. Overview

There are 16 distinct tones in DTMF. Each tone is the sum of two frequencies: one from a low and one from a high frequency group. There are four different frequencies in each group:

Table 1 DTMF Tones Group

Group	Frequency (Hz)
Low Tones	• 697Hz • 770Hz • 852Hz • 941Hz
High Tones	• 1209Hz • 1335Hz • 1477Hz • 1633Hz

The frequencies and the keypad layout of DTMF tone dialing have been internationally standardized, but the tolerances on individual frequencies may vary in different countries. The North American standard is 1.5% for the generator and 2% for the receiver. Since the DTMF tone is generated by PWM module and timer interrupt, the accuracy of the frequencies generated is directly affected by the accuracy of the frequency of the crystal oscillator of the system. If the system uses a different clock rate, the source code has to be adjusted accordingly.

Moreover, the tone has to be held for a specific minimum time before it is accepted as a valid dialing digit. In North America, the minimum time for a digit is about 50ms and the inter-digit interval is also about 50ms.

Digit 1 697Hz 1209Hz	Digit 2 697Hz 1335Hz	Digit 3 697Hz 1477Hz	Digit A 697Hz 1633Hz
Digit 4 770Hz 1209Hz	Digit 5 770Hz 1335Hz	Digit 6 770Hz 1477Hz	Digit B 770Hz 1633Hz
Digit 7 852Hz 1209Hz	Digit 8 852Hz 1335Hz	Digit 9 852Hz 1477Hz	Digit C 852Hz 1633Hz
Digit * 941Hz 1633Hz	Digit 0 941Hz 1335Hz	Digit # 941Hz 1477Hz	Digit D 941Hz 1633Hz

Figure 1 Keypad Digits and Frequencies of Tone Pairs

There are three processes to produce DTMF signal from PWM module:

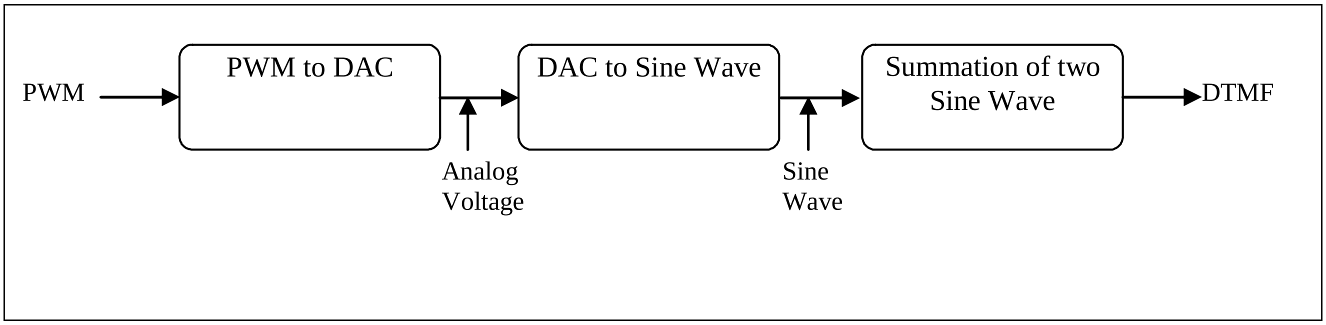

Figure 2 Overview of DTMF Signal Generation Process

1.1 PWM to DAC

Analog voltage can be generated by connecting PWM module to a low-pass filter (a simple RC circuit). The PWM waveform will be charging through a capacitor when output at HI and discharging when output at LO. Figure 2 illustrates the operation of PWM to DAC process. For the detailed description of DAC, please refer to AN: 03/03/004 – "PWM as a DAC".

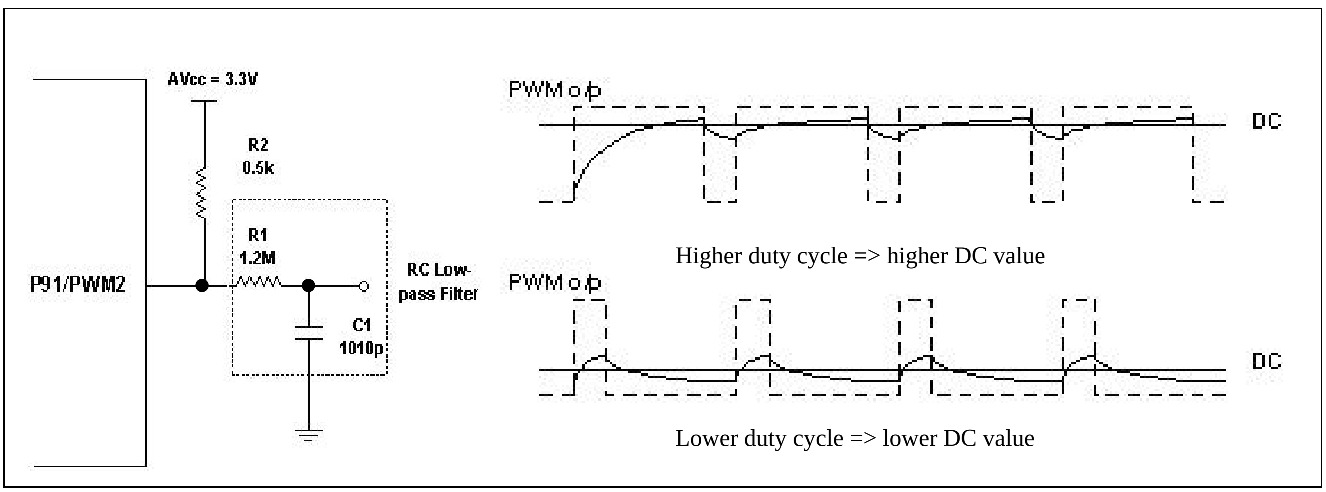

Figure 3 PWM to DAC Example

1.2 DAC to Sine Wave

If the generated DC voltage level is in a sinusoidal manner, a sine wave is generated. These two parameters need to be calculated:

- Signal Period
- Sample Period

Note: please refer to AN: "PWM Sine Wave Generation"

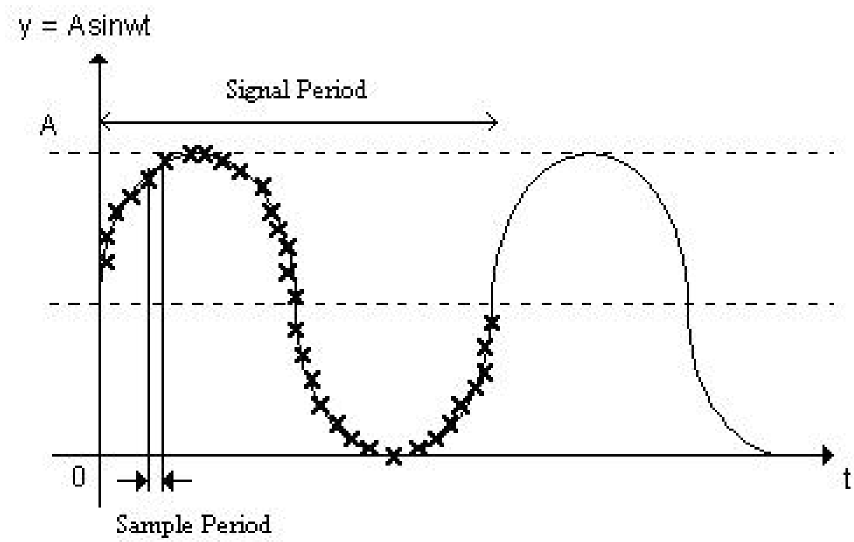

Figure 4 Typical Sine Wave Diagram

For this example, signal period is the desired frequency that user want to generate. The Asynchronous Event Counter (AEC) is selected to generate the sample period. AEC is configured as an 8-bit event counter and input clock source is set to ∅/2 (∅ = crystal frequency / 2). AEC interrupt is generated every 256 counts of input clock source.

For example crystal used is 9.8304 MHz,

Time for one AEC interrupt occur, $T_{interrupt}$

$T_{interrupt}$ = ((1 / (∅/2)) x 256 count
= (1 / (crystal / 4)) x 256 count
= (1 / (9.8304MHz / 4) x 256 count
= 10.42µs

The sample period is equal one AEC interrupt occur, every time AEC interrupt generated the Interrupt Service Routine (ISR) will put the calculated pulse width into the PWM width register.

Sample frequency = 1 / $T_{interrupt}$
= 9600Hz

The calculation of the pulse width requires increment counter value. The increment counter value is calculated as follows.

Assumption:

- 256 sample for the complete sine wave table
- sample frequency = 9600Hz
- signal frequency = 852Hz

Increment counter value = 256 / number of increments

Number of increments depend on sample frequency and signal frequency and it's equal to how many time the given signal increments through the sine wave table in one complete cycle.

Number of increments = sample frequency / signal frequency
∴ Increment counter value = 256 / (sample frequency / signal frequency)
= 256 * signal frequency / sample frequency
= 256 * (852Hz) / (9600Hz)
= 22.72

All these calculations are done by compiler, therefore user must change the default value in order to use with other parameter.

Table 2 Calculation of Increment Counter Value

Frequency (Hz)	Increment Counter Value
• 697Hz	• 18.587
• 770Hz	• 20.533
• 852Hz	• 22.720
• 941Hz	• 25.093
• 1209Hz	• 32.240
• 1335Hz	• 35.600
• 1477Hz	• 39.387
• 1633Hz	• 43.547

1.3 Summation of Two Sine Waves

The DTMF tone dialing requires the generation of two sine waves. The basic concept of modulation is to use two counters to increment through a sine wave table at different speed. To generate lower frequencies steps through the table in smaller steps. Similarly, higher frequencies require larger steps. The two sine waves are superimposed to give the desired DTMF waveform. The process is depicted in Figure 5 below:

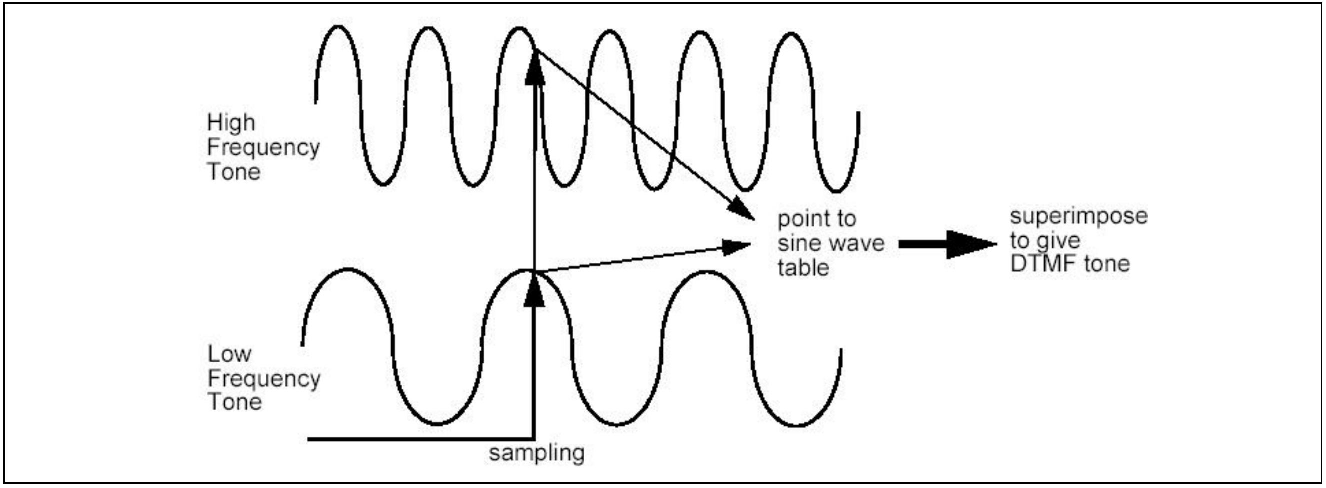

Figure 5 Summation of Two Sine Waves

2. Hardware Implementation

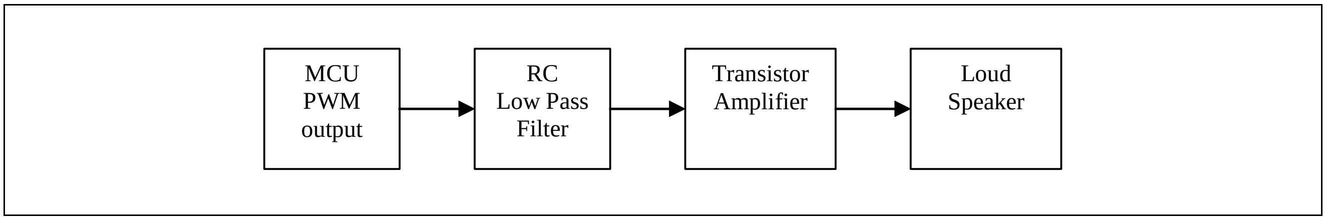

Figure 6 Block Diagram of DTMF Generation

The DTMF tone is generated by the Pulse Width Modulation (PWM) of SLP series Microcontroller. The software will modulate the sinusoidal signal into a pulse train of fixed periods but changing width. The changing width of the pulses corresponds to the voltage level of the sine wave. With an external Low Pass Filter (LPF) at the PWM output pin, the PWM signal will be demodulated. The LPF acts as an integrator which transforms the pulse train into analog sinusoidal signal. The DTMF waveform is then sent to the audio amplifier for sound output.

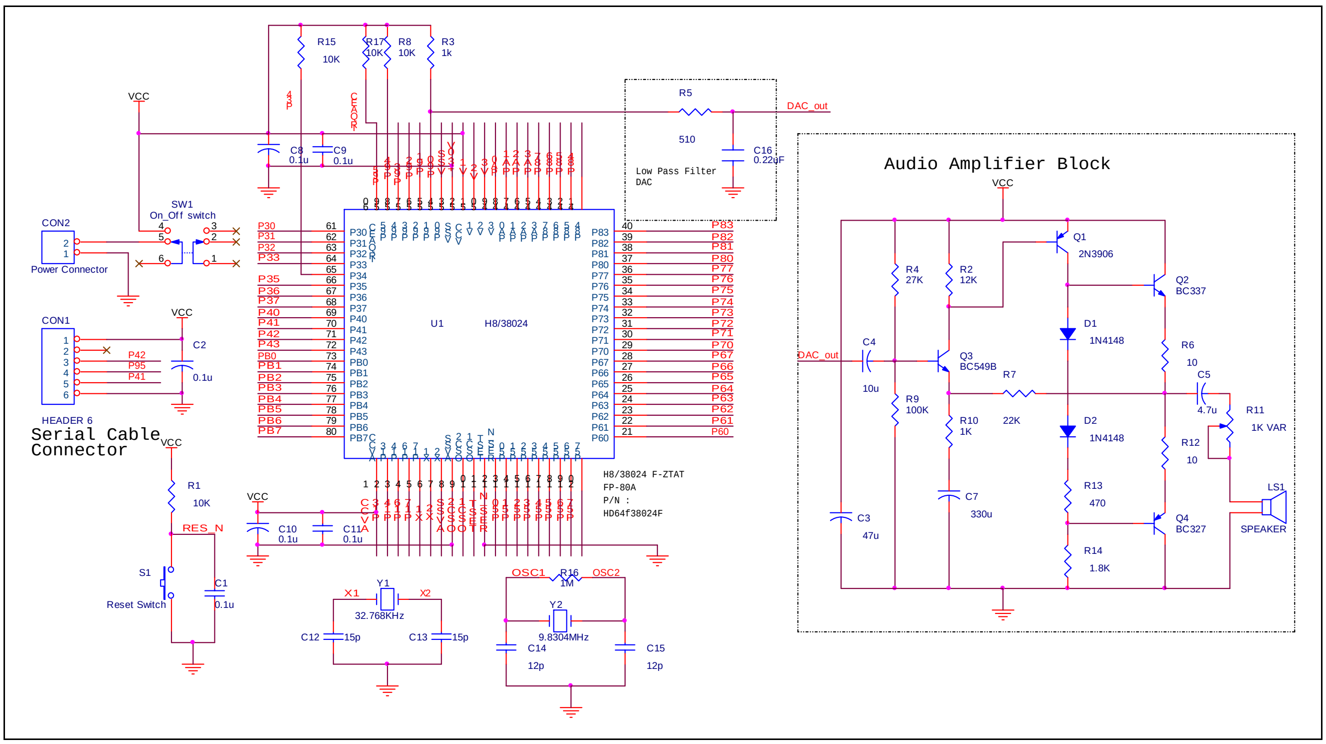

Figure 7 Schematic of DTMF Generation

In audio signal generations, the waveform has an average value at ground level, which is 0V. The voltage varies between positive and negative values. However, in our application using PWM, the voltage level can only be positive. It means that the waveform is generated with a DC offset. Warm-up function is used to eliminate this problem, software code is written to warm-up the audio output so the reference pulse width can be gradually increased from 0 to 512 (mid value for 10-bit PWM). The longer the warm-up time, the smoother the waveform, and vice versa. In between any two digits, instead of disabling and enabling the PWM module, user can set PWM value at 512 and keep the DAC output voltage at 1.65volt.

3. Operation and Observation

The hardware circuitry provides Flash programming capability. User can download DTMF dialing demo program via PC serial port. The PC application software used to download user program is the freeware - Flash Development Toolkit (FDT) which is available from www.eu.renesas.com .

After the DTMF dialing program has been successfully downloaded, reset the MCU and execute the program. During the execution, user should be able to listen to the DTMF tone coming out from the speaker. The demo program will end after all the digits have been dialed. User can repeat this process by resetting the MCU.

The DTMF dialing demo program demonstrates the dialing of phone number through on-chip PWM module. The phone number to be dialed is stored in the variable array of PHONE_NO, which is set to “123456789” in this example. Telephone number of any length may be dialed, Digit 0 to 9 are represented in hexadecimal as H’00, H’01, …., H’08, H’09, while H’0A to H’0F represents the digit A, B, C, D, * and # respectively. The phone number should be terminated by the byte H’10.

The DTMF dialing demo program also can be used with other crystal oscillator value by changing the XTAL value in #define statement.

e.g. :

if crystal = 9.8304MHz	➔	#define	XTAL	9830400L	(default)
if crystal = 4MHz	➔	#define	XTAL	4000000L	

There are two PWM channel in H8/38024F MCU, user has to define which PWM channel to use before the source code compilation. e.g.:

if PWM1 is used	➔	#define	PWM_use	1	(default)
if PWM2 is used	➔	#define	PWM_use	2	

Figures 8 and 9 show the DTMF waveform and frequency spectrum for phone digit ‘8’ captured from Tektronix TDS 3054 digital oscilloscope. From Figure 8, the signal to noise ratio is about 30.4dB. Figure 9 shows the two peak frequencies of phone digit ‘8’ – 852Hz and 1335Hz.

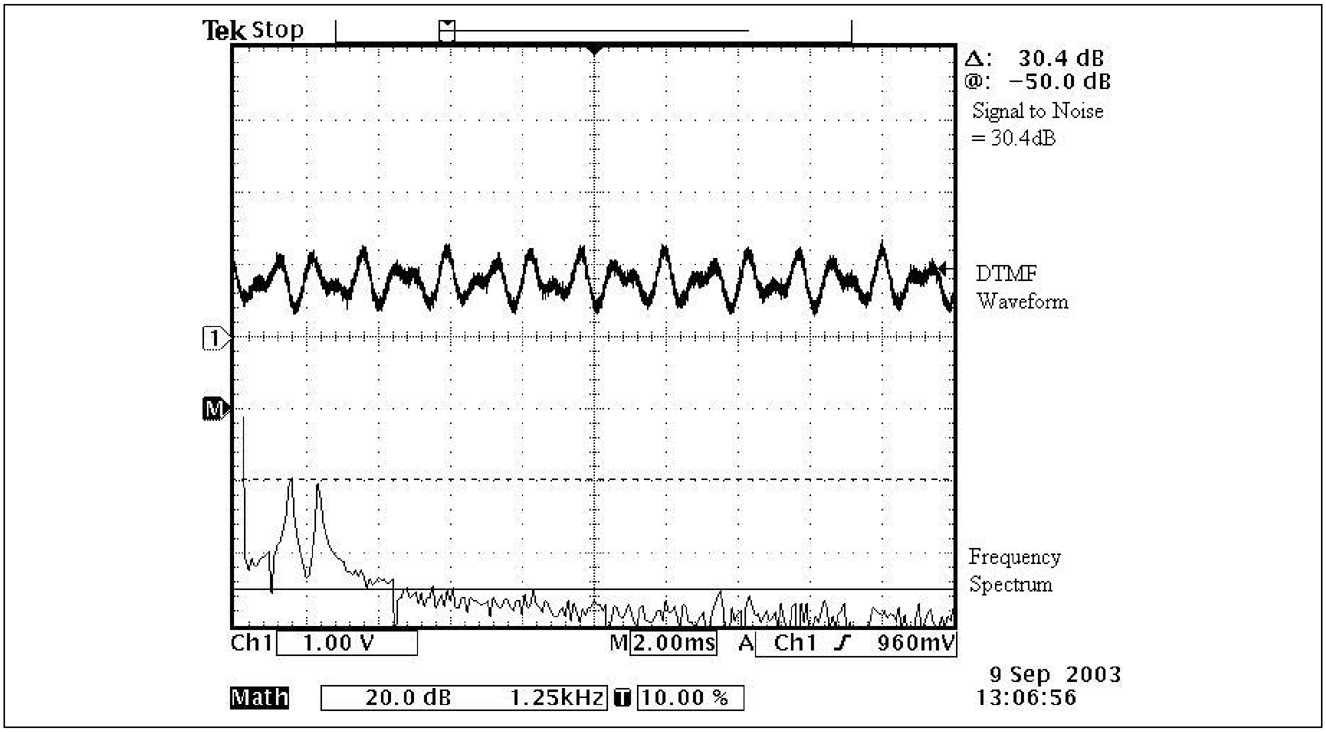

Figure 8 DTMF Waveform and Frequency Spectrum for Digit ‘8’ (SNR)

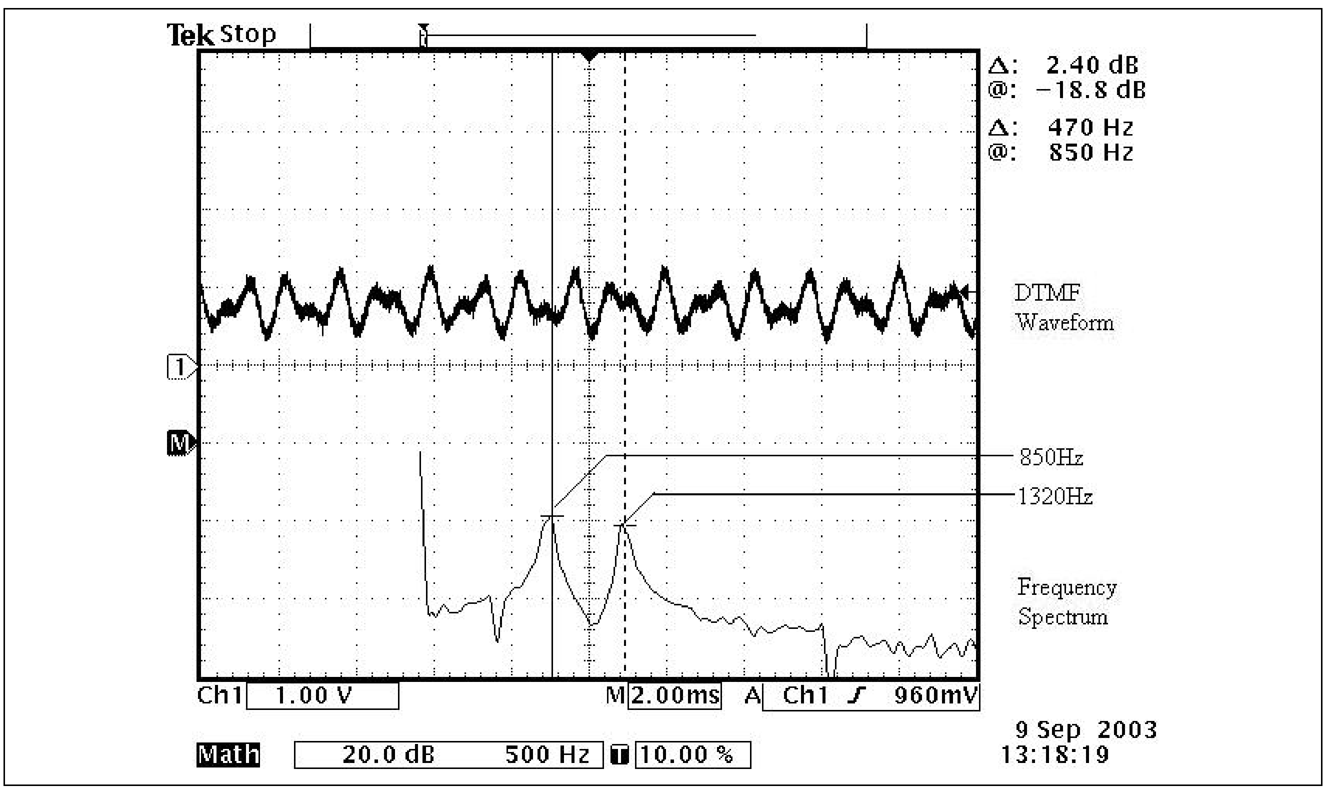

Figure 9 DTMF Waveform and Frequency Spectrum for Digit ‘8’ (Peak Frequencies)

4. Code Listing

The attached code is generated using HEW project generator at H8/38024F SLP MCU. The toolchain used is the free SLP/Tiny toolchain.

The following code listing is the main functionality of "DTMF_dialing.c"

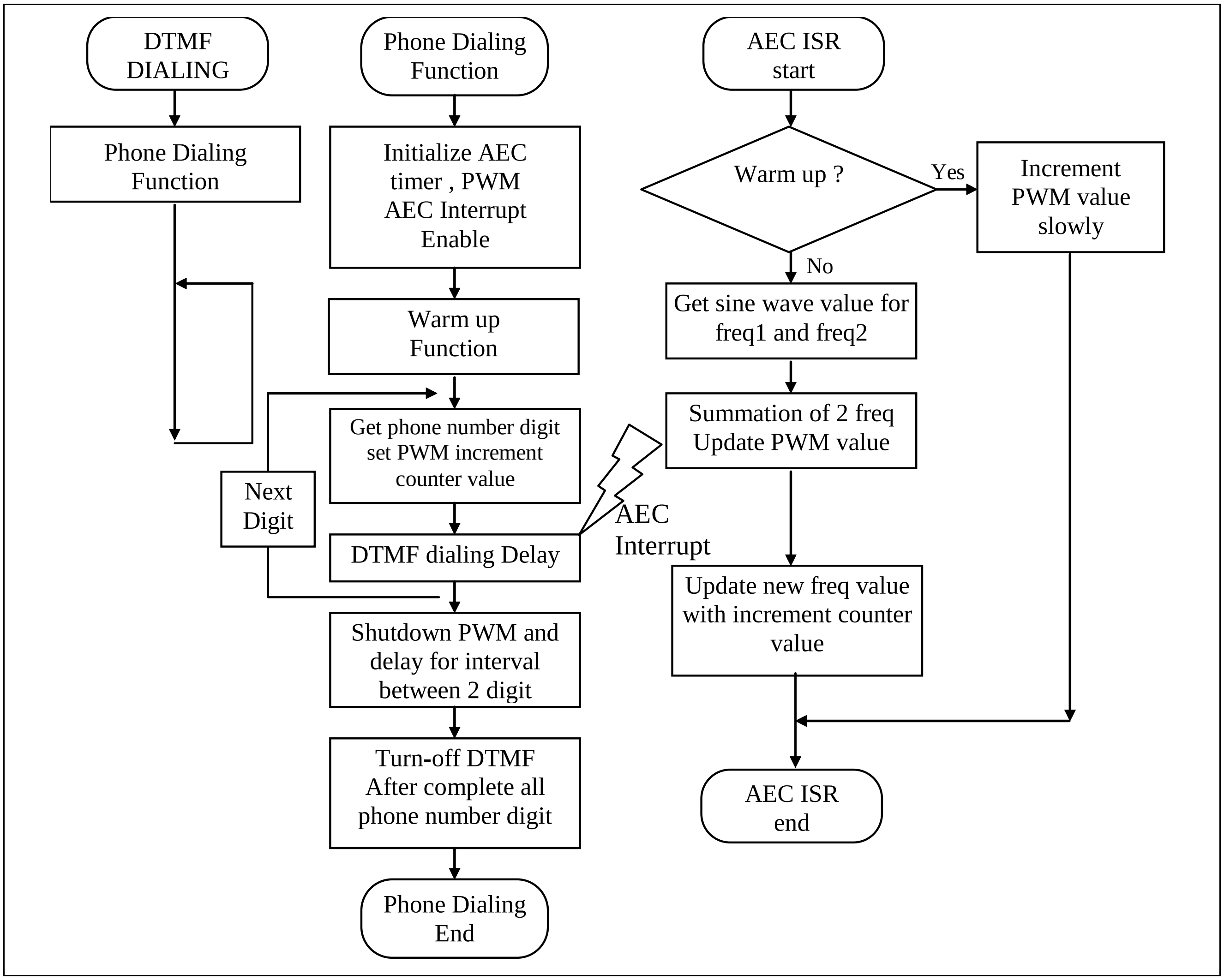

Figure 10 Flow Chart for DTMF_dialing.c

```
/*******************************************************************/
/*                                                                 */
/*  FILE        :DTMF_Dialing.c                                    */
/*  DATE        :Tue, Mar 04, 2003                                 */
/*  DESCRIPTION :Main Program                                      */
/*  CPU TYPE    :H8/38024F                                         */
/*                                                                 */
/*  This file is generated by Hitachi Project Generator (Ver.2.1). */
/*                                                                 */
/*******************************************************************/

/*************************************************************/
/*  File Include                                             */
/*************************************************************/
#include    <machine.h>
#include    "iodefine.h"
/*************************************************************/
/* define                                                    */
/*************************************************************/
#define  XTAL     9830400L
#define  sample_freq (XTAL/4L) / 256L //256 clock cycles per interrupt
#define  tone_low_a  ((256L * 697L)/100)/(sample_freq/100)
#define  tone_low_b  ((256L * 770L)/100)/(sample_freq/100)
#define  tone_low_c  ((256L * 852L)/100)/(sample_freq/100)
#define  tone_low_d  ((256L * 941L)/100)/(sample_freq/100)
#define  tone_high_a ((256L * 1209L)/100)/(sample_freq/100)
#define  tone_high_b ((256L * 1335L)/100)/(sample_freq/100)
#define  tone_high_c ((256L * 1477L)/100)/(sample_freq/100)
#define  tone_high_d ((256L * 1633L)/100)/(sample_freq/100)
#define  PWM_use     2       //select "1" for PWM channel 2
                             //select "0" for PWM channel 1
/*************************************************************/
/*  Function define                                          */
/*************************************************************/

void init_PWM(unsigned char);
void storeCount(unsigned short);
void aecint( void );
void init_AEC(void);
void init_DTMF(void);void off_DTMF(void);
void init_PWM1(unsigned char selClk1);
void init_PWM2(unsigned char selClk2);
void warm_up(void);
void dialing(void);

/*************************************************************/
/*Constant Look up Table for Sine Wave value
/*************************************************************/
const unsigned int   sample =sample_freq;
const unsigned int   Sine_Table[256]=
{
512,518,525,531,537,543,550,556,
562,568,574,580,586,592,598,604,
610,616,621,627,633,638,644,649,
```

```
654,659,664,669,674,679,684,688,
693,697,702,706,710,714,717,721,
725,728,731,734,737,740,743,746,
748,750,753,755,756,758,760,761,
762,763,764,765,766,766,766,767,
767,767,766,766,766,765,764,763,
762,760,759,757,755,754,751,749,
747,744,742,739,736,733,730,726,
723,719,715,712,708,704,699,695,
691,686,681,677,672,667,662,657,
652,646,641,635,630,624,619,613,
607,601,595,589,583,577,571,565,
559,553,546,540,534,528,521,515,
509,503,496,490,484,478,471,465,
459,453,447,441,435,429,423,417,
411,405,400,394,389,383,378,372,
367,362,357,352,347,343,338,333,
329,325,320,316,312,309,305,301,
298,294,291,288,285,282,280,277,
275,273,270,269,267,265,264,262,
261,260,259,258,258,257,257,257,
257,257,258,258,259,260,261,262,
263,264,266,268,269,271,274,276,
278,281,284,287,290,293,296,299,
303,307,310,314,318,322,327,331,
336,340,345,350,355,360,365,370,
375,380,386,391,397,403,408,414,
420,426,432,438,444,450,456,462,
468,474,481,487,493,499,506,512
};

const unsigned char tone[16][2] =
{                              //Keypad digit  Tone Pair (Hz)
{tone_low_d,tone_high_b},      //  '0'               941Hz, 1335Hz
{tone_low_a,tone_high_a},      //  '1'               697Hz, 1209Hz
{tone_low_a,tone_high_b},      //  '2'               697Hz, 1335Hz
{tone_low_a,tone_high_c},      //  '3'               697Hz, 1477Hz
{tone_low_b,tone_high_a},      //  '4'               770Hz, 1209Hz
{tone_low_b,tone_high_b},      //  '5'               770Hz, 1335Hz
{tone_low_b,tone_high_c},      //  '6'               770Hz, 1477Hz
{tone_low_c,tone_high_a},      //  '7'               852Hz, 1209Hz
{tone_low_c,tone_high_b},      //  '8'               852Hz, 1335Hz
{tone_low_c,tone_high_c},      //  '9'               852Hz, 1477Hz
{tone_low_a,tone_high_d},      //  'A'               697Hz, 1633Hz
{tone_low_b,tone_high_d},      //  'B'               770Hz, 1633Hz
{tone_low_c,tone_high_d},      //  'C'               852Hz, 1633Hz
{tone_low_d,tone_high_d},      //  'D'               941Hz, 1633Hz
{tone_low_d,tone_high_a},      //  '*'               941Hz, 1209Hz
{tone_low_d,tone_high_c},      //  '#'               941Hz, 1477Hz
};
```

```
/*************************************************************/
/*Global variable
/*************************************************************/
unsigned char PWDR_L2, PWDR_U2;
unsigned int i=0,j=0, count=0, inc1=0, inc2=0, final=0;
unsigned int lowcnt=0, hicnt=0;
unsigned char Ready = 0, DIGIT = 0;
unsigned char PHONE_NO[]={0x01,0x02,0x03,0x04,0x05,0x06,0x07,0x08,0x09,0x10};

/*************************************************************/
/*  Main Program                                             */
/*************************************************************/
void main ( void )
{
   //Dialing
   dialing();          //DTMF dialing
    while (1)
   {
      //Write user program here
    }
}

/*************************************************************/
/*  Initialize Program                                       */
/*************************************************************/
//Initialize DTMF function
void init_DTMF(void)
{
   set_imask_ccr(1);                // Interrupt Disable
   init_AEC();
   #if (PWM_use==1)
   init_PWM1(0); //Select conversion period = 512/(PWM input clock)
   #else
   init_PWM2(0); //Select conversion period = 512/(PWM input clock)
   #endif
}

void init_PWM1(unsigned char selClk1)
{
   if (selClk1 <= 3)              // Check if valid, otherwise PWM2 is off
   {
      P_IO.PMR9.BIT.PWM1 = 1;      // Configure P91 as PWM2 output pin
      P_PWM1.PWCR1.BYTE = selClk1; // Clock select for PWM2,write only
   }
}

void init_PWM2(unsigned char selClk2)
{
   if (selClk2 <= 3)              // Check if valid, otherwise PWM2 is off
   {
      P_IO.PMR9.BIT.PWM2 = 1;      // Configure P91 as PWM2 output pin
      P_PWM2.PWCR2.BYTE = selClk2; // Clock select for PWM2,write only
   }
}
```

```
void off_DTMF(void)
{
   P_SYSCR.IENR2.BIT.IENEC = 0;
   //compiler directive to select which code to be compile
   #if (PWM_use==1)
   P_IO.PMR9.BIT.PWM1 = 0;        // Turn off PWM1
   #else
   P_IO.PMR9.BIT.PWM2 = 0;        // Turn off PWM2
   #endif
}

/************************************************************/
/*  Initialize Program                                      */
/************************************************************/
void warm_up(void)
{
   set_imask_ccr(0);              // Interrupts, 0-Enable, 1-Disable
   while(count<0x3000) ;
   set_imask_ccr(1);              // Interrupts, 0-Enable, 1-Disable
   Ready = 1;
}

/************************************************************/
/*  DTMF dialing Program                                    */
/************************************************************/
void dialing(void)
{
   init_DTMF();

   warm_up();

   i = 0;
   while (PHONE_NO[i]!=0x10)
   {
      inc1 = tone[PHONE_NO[i]][0];
      inc2 = tone[PHONE_NO[i]][1];

      //Dialing
      set_imask_ccr(0);                 // Interrupts, 0-Enable, 1-Disable
      for (j=0; j<15000; j++) ;         // short delay Tone dialing
      set_imask_ccr(1);                 // Interrupts, 0-Enable, 1-Disable

      storeCount(0);
      for (j=0; j<15000; j++) ;// short delay interval between 2 phone digit

      i++;                              //next digit
   }

   set_imask_ccr(1);                    // Interrupts, 0-Enable, 1-Disable

   off_DTMF();

}
```

```
/************************************************************/
/*  Write each digital code into PWDR registers             */
/************************************************************/
void storeCount(unsigned short PWDRval_2)
{
   //compiler directive to select which code to be compile
   #if (PWM_use==1)
   P_PWM1.PWDRL1.BYTE = (unsigned char)(PWDRval_2 & 0x00FF);
                               // Write lower 8bits of 10bits data
   P_PWM1.PWDRU1.BYTE = (unsigned char) ((PWDRval_2 & 0x0300) >> 8);
                               // Write upper 8bits of 10bits data
   #else
   P_PWM2.PWDRL2.BYTE = (unsigned char)(PWDRval_2 & 0x00FF);
                               // Write lower 8bits of 10bits data
   P_PWM2.PWDRU2.BYTE = (unsigned char) ((PWDRval_2 & 0x0300) >> 8);
                               // Write upper 8bits of 10bits data
   #endif
}

/************************************************************/
/*  AEC Interrupt Service Routine                           */
/************************************************************/
void aecint (void)
{
   P_SYSCR.IRR2.BIT.IRREC = 0;     // Clear IRREC flag

   if(P_AEC.ECCSR.BIT.OVL == 1)    // Check for ECL overflow flag
   {  P_AEC.ECCSR.BIT.OVL = 0;     // Clears flag

      if(Ready == 0)
      {
         storeCount(count++/128);
      }
      else
      {
         final = (Sine_Table[lowcnt])/2+(Sine_Table[hicnt])/2;
         storeCount(final);
         lowcnt = lowcnt + inc1;
         if(lowcnt>255) lowcnt = lowcnt-255;
                               // If reached end of 1 period, then reset
         hicnt = hicnt + inc2;
           if(hicnt>255) hicnt = hicnt-255;
                               // If reached end of 1 period, then reset
      }
   }
}

void init_AEC(void)
{
   P_AEC.ECCSR.BYTE = 0x15;
   P_AEC.ECCR.BYTE = 0x10;
   P_SYSCR.IRR2.BIT.IRREC = 0;     // Clear IRREC flag
   P_SYSCR.IENR2.BIT.IENEC = 1;    // AEC Interrupt Request
}
```

The following code listing is the Interrupt service program of "intprg.c", please insert the below code.

```
extern void aecint (void);              //insert AEC ISR function
.
.
.
.
.
.
__interrupt(vect=12) void INT_Counter(void)
{
aecint();                               //insert AEC ISR function
}
```

5. References

1. AN: 03/03/003 - “PWM Sine Wave Generation”
2. AN: 03/03/004 – “Use PWM as A DAC”

Revision Record

Rev.	Date	Description	
		Page	Summary
1.00	Sep.19.03	—	First edition issued