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

SLP Tone Generator (ToneGen)

Introduction

Two methods of generating tones using the H8/38024 SLP MCU are:

1. Pulse width modulation (PWM) implementation
2. Timer toggle output implementation

Target Device

SLP H8/38024

Contents

1. Overview

Tone generator is a methodology whereby tone signals are defined in a musical sequence to produce a song. Two types of implementation are described here. Both implementations use the same musical tone data and rhythm between two musical tones (rhythm is fixed to reduce the size of musical tone data).

1.1 Musical Tone (Notes)

If a long hollow tube is hit, a fairly constant sound (pitch) is heard due to a shock-wave oscillating along the tube at a certain speed (frequency). A "note" is described a musical frequency i.e., the pitch of a piano key or guitar string. By convention, notes are named as:-

A, A#, B, C, C#, D, D#, E, F, F#, G, G#

The suffix "#" denotes sharp and "b" denotes flat. Also note that A# = Bb, C# = Db, D# = Eb, F# = Gb and G# = Ab. The names chosen are the de facto standard for nearly all music.

"Octaves" of a note are just multiples of the original frequency. Let's say that a length of hollow tube has a frequency of 264Hz and normally call it "C". If the length is half of the original length, the frequency will be double. This creates another "C" but at one octave higher than the first (264 x 2 = 528Hz).

Table 1 Notes, Octave and Frequency

Hertz	Octave = 0	Octave = 1	Octave = 2	Octave = 3	Octave = 4	Octave = 5
A	55.000	110.000	220.000	440.000	880.000	1760.000
A#	58.270	116.541	233.082	466.164	932.328	1864.655
B	61.735	123.471	246.942	493.883	987.6\767	1975.533
C	65.406	130.813	261.626	523.251	1046.502	2093005
C#	69.296	138.591	277.183	554.365	1108.731	2217.461
D	73.416	146.832	293.655	587.330	1174.659	2349.318
D#	77.782	155.563	311.127	622.254	1244.508	2489.016
E	82.407	164.814	329.628	659.255	1318.510	2637.020
F	87.307	174.614	349.228	698.456	1396.913	2793.826
F#	92.499	184.997	369.994	739.989	1479.978	2959.955
G	97.999	195.998	391.995	783.991	1567.982	3135.963
G#	103.826	207.652	415.305	830.609	1661.219	3322.438
A	110.000	220.000	440.000	880.000	1760.000	3520.000

1.2 PWM Implementation

The built-in 10-bit PWM module can be used to generate PWM pulse stream with desired duty cycle. It can also be used as a D/A convert by connecting a low pass filter. There are four clock sources available as input clock. With 10-bit resolution, we can get 4 pulse trains in each conversion period. Depending upon the register bit settings, we can get four conversion periods as described above. This module can be placed independently in standby mode when not in use to conserve the power.

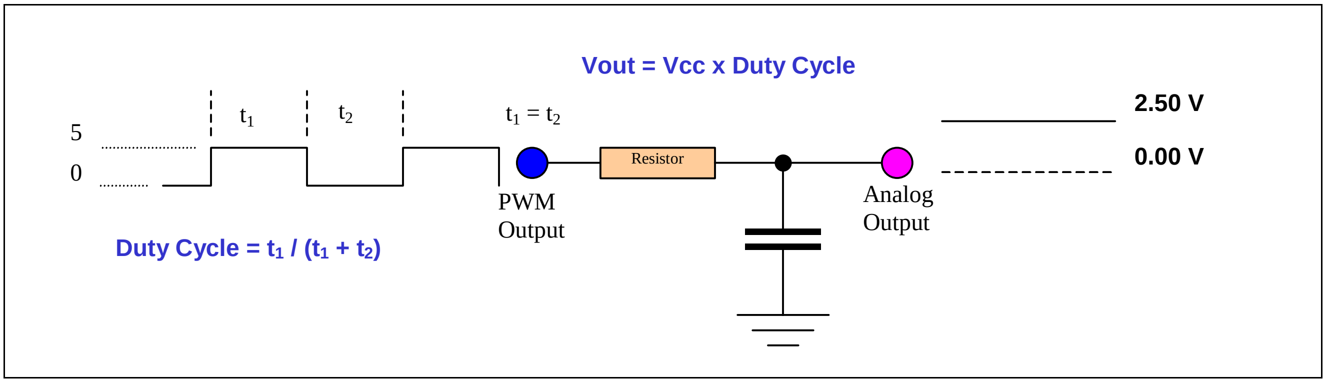


Figure 1 Usage of PWM as D/A Converter

The primary purpose of 10-bit PWM is to provide a high resolution D/A using an external low pass filter. The basic task of any D/A converter is to take a binary number and convert it to voltage or current with an analog form. Other than a traditional D/A converter, which is difficult to implement under CMOS fabrication technology for precision, the alternative solution is to make a counter whose output duty cycle can be varied under software control – that is a Pulse Width Modulation.

Using a simple Low-Pass (or band pass if no DC component is desired), the Analog output of the filter is basically Vcc x Duty Cycle (in a ideal case, notice that the output is a function of duty cycle rather than frequency

For example: Vout = 5.00V x 50% Duty Cycle = 2.5V

If the generated DC voltage level is in a sinusoidal manner, a sine wave is generated.

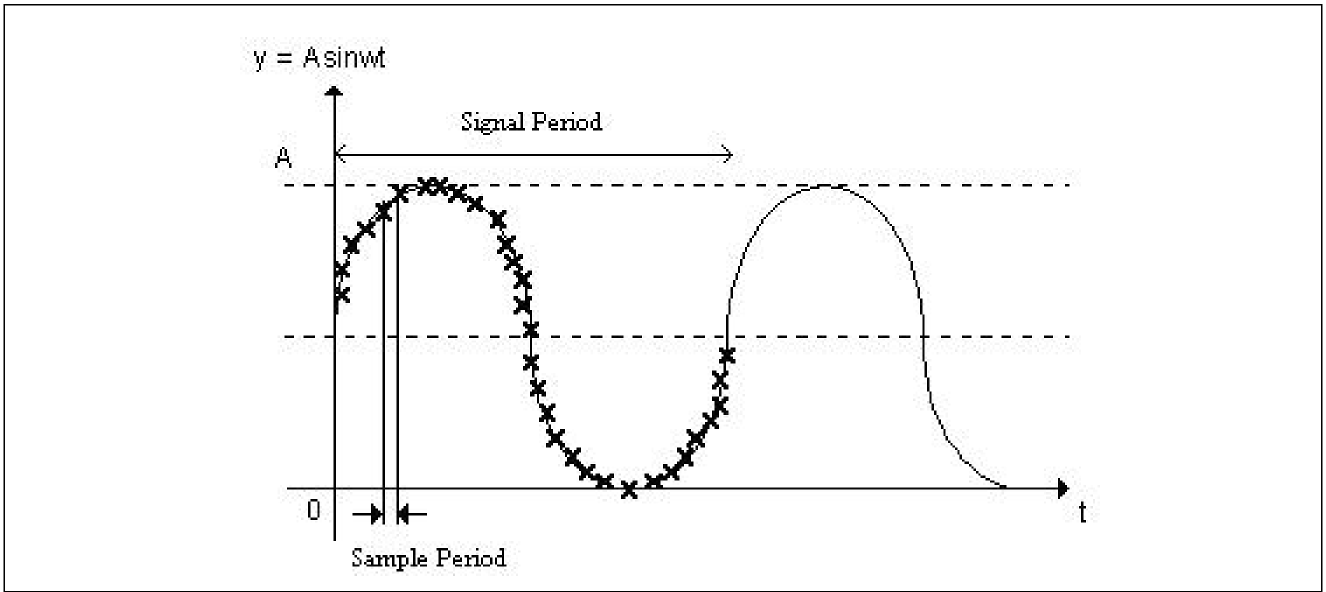


Figure 2 Typical Sine Wave Diagram

The sample period is time duration between two PWM values. Normally, timer is used to reload the sine wave value into the PWM module. Therefore AEC (asynchronous event counter) timer is used for this purpose.

For example, the frequency of the crystal used is 9.8304 MHz,

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

$$
\begin{aligned}
T_{interrupt} &= ((1 / (\varnothing/2)) \times 256 \text{ count} \qquad \text{Note: } \varnothing = \varnothing osc/2 \\
&= (1 / [(\varnothing osc/2)/ 2]) \times 256 \text{ count} \\
&= (1 / (9.8304\text{MHz} / 4) \times 256 \text{ count} \\
&= \underline{104.16\mu s}
\end{aligned}
$$

The sample period is equal to one AEC interrupt occurrence. The Interrupt Service Routine (ISR) will put the calculated pulse width into the PWM width register.

$$
\begin{aligned}
\text{Sample frequency} &= 1 / T_{interrupt} \\
&= 9600\text{Hz}
\end{aligned}
$$

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

- 256 sample for the complete sine wave table
- sample frequency = 9600Hz
- signal frequency = 440Hz (e.g. note “A” at the third octave)

$$\text{Increment counter value} = 256 / \text{number of increments}$$

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.

$$
\begin{aligned}
&\text{Number of increments} &&= \text{sample frequency / signal frequency} \\
\therefore\ &\text{Increment counter value} &&= 256 / (\text{sample frequency / signal frequency}) \\
& &&= 256 * \text{signal frequency / sample frequency} \\
& &&= 256 * (440\text{Hz}) / (9600\text{Hz}) \\
& &&= 11.73
\end{aligned}
$$

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

1.3 Timer Toggle Output Implementation

There are several methods to implement tone generator by software means. For example, timer F is chosen because it is equipped with toggle output and output compare functions. The initial value of the toggle output can be set. Timer F counter value will increment on each input clock pulse. The timer F counter value is constantly compared with the value set in output compare register F, and the counter can be cleared, an interrupt request, or output toggled, when the two values match. Timer F can also function as two independent 8-bit timers.

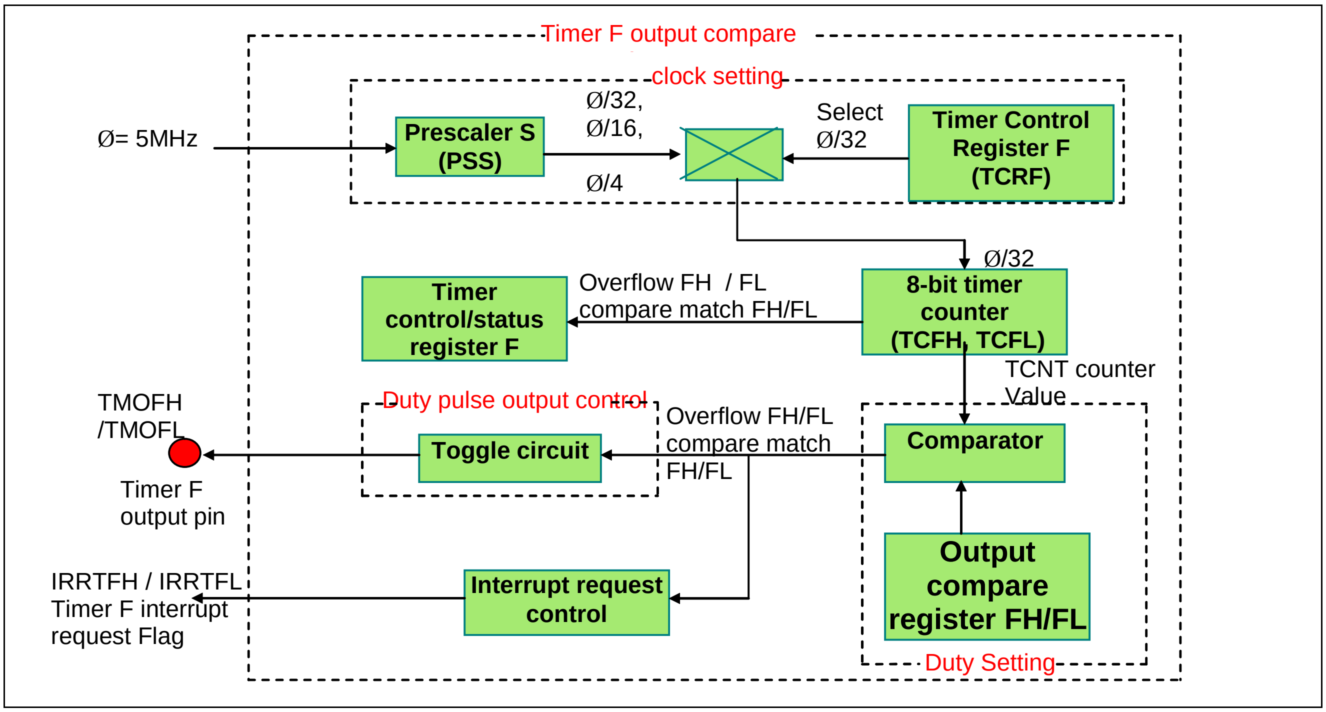


Figure 3 Block Diagram of Timer F Output Compare Operation

Figure 3 describes how a PWM is output through TMOFH/ TMOFL pin using the Timer F output compare function.

- The 5MHz system clock is input to the Prescaler S that divides the clock by 32, 16 and 4.
- TCRF is an 8-bit write-only register, which selects an input clock and sets the output level of TMOFL pin.
- Timer counters FL and FH (TCFL / TCFH) are 8-bit read/write up-counters. In this example, the input clock is Ø/32.
- Timer control/status register F (TCSRF) disables the clearing TCFL by compare match and enables the counter FL overflow interrupts.
- The data of output compare register FL (OCRFL) is always compared with that of TCFL.
- When the values of both registers match, the compare match is generated and TMOFL pin is toggled. At the same time, a compare match flag L (CMFL) is set to 1 and an interrupt is requested to CPU.

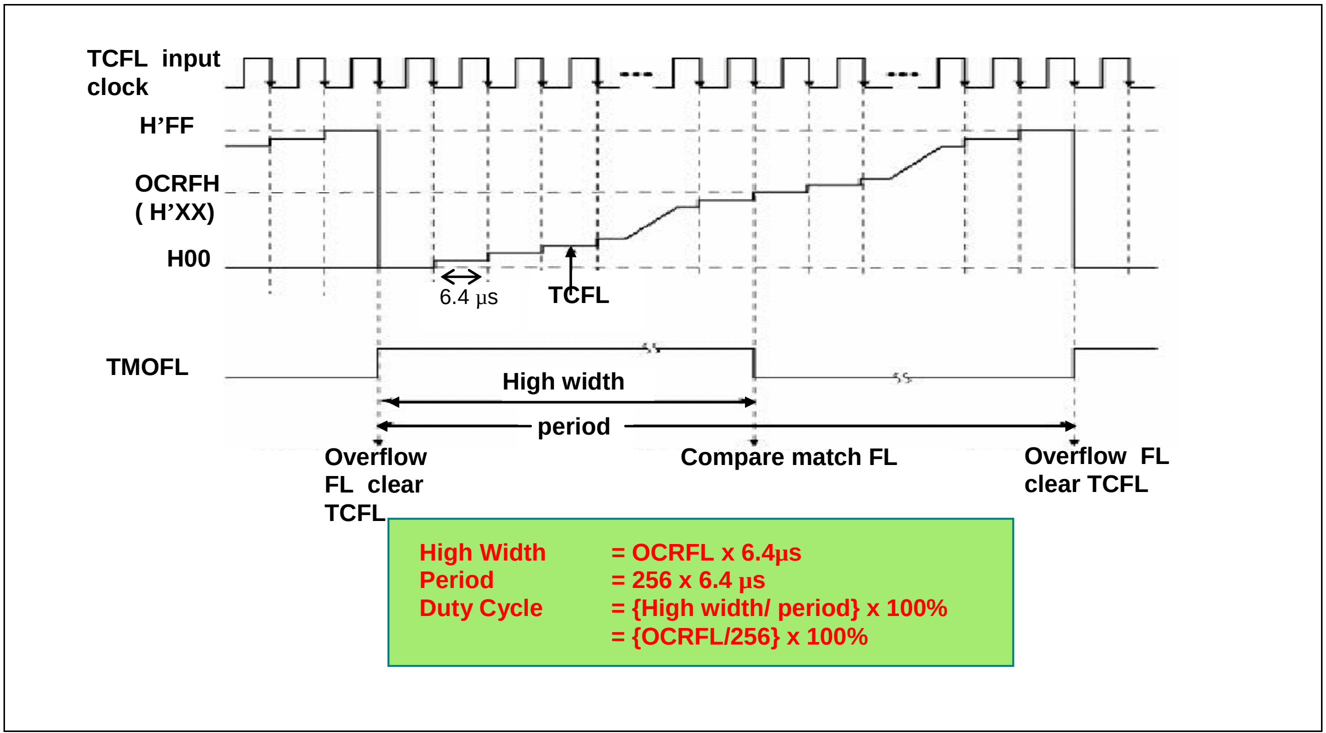


Figure 4 Timer F Output Compare Operation

Figure 4 shows how the Timer F compare-match function can be used to generate a pulse with an arbitrary duty cycle i.e., a digital tone signal. The Timer counter Register FL (TCFL) determines the tone signal clock cycle, or period, of the output waveform, while the value stored in Output compare Register (OCRFL) determines the duty cycle. The calculation of desired duty cycle can be done as shown in the above formula. It is only necessary to program Timer F once. There is no need to reload OCRFL unless you want to change the duty cycle of the output.

User can generate two digital tones by combining the two Timer F toggle outputs (TMOFL and TMOFH), e.g. one for treble (high frequency) and one for bass (low frequency). Figure 5 below shows the block diagram of Timer toggle output tone generator.

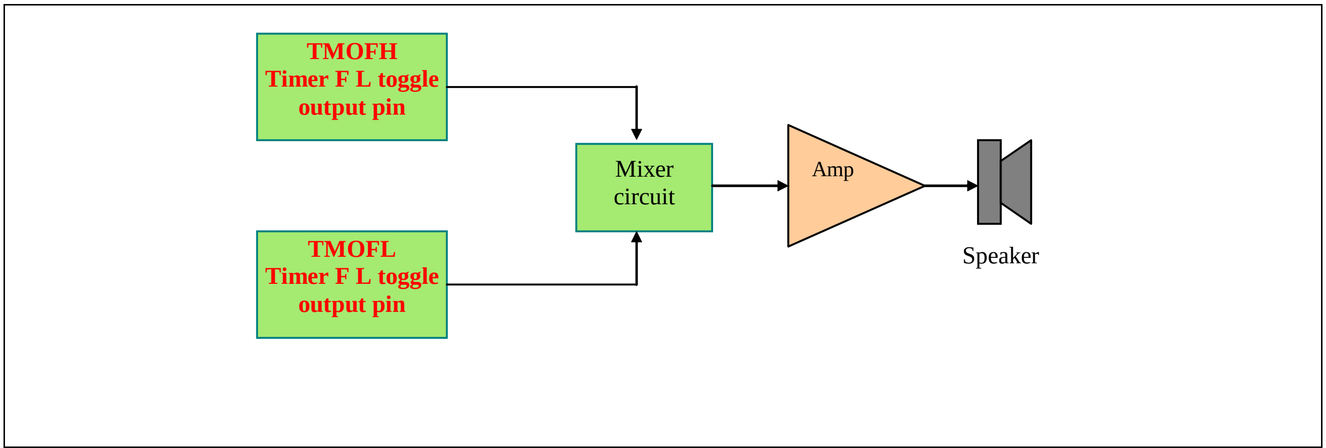


Figure 5 Block Diagram of Timer Toggle Output Tone Generator

2. Hardware Implementation

2.1 PWM Implementation

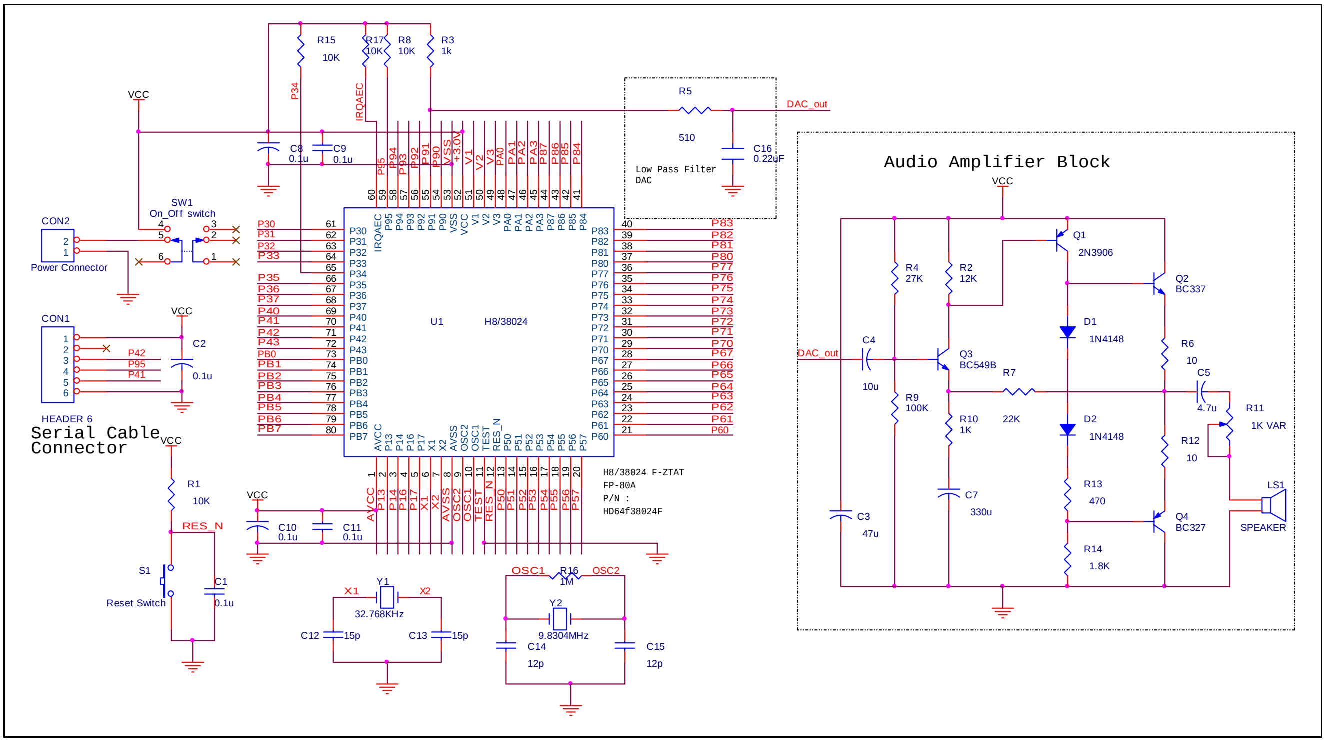
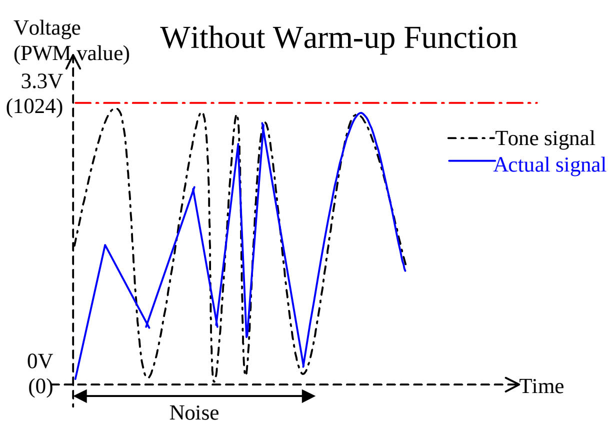


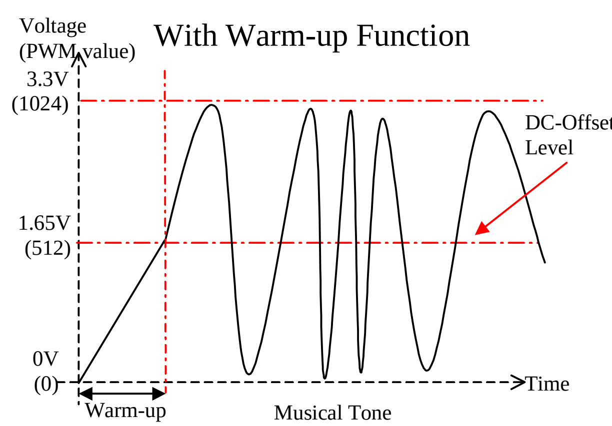
Figure 6 Schematic Diagram for PWM Tone Generator

The musical tone is generated by the Pulse Width Modulation (PWM) of SLP MCU. 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 musical tone is then sent to the audio amplifier for sound output.

Warm-up Function:

Generally audio signal has an average value at ground level (It will fluctuated between positive and negative regions). However there is no negative supply in this implementation, thus a DC offset to 1/2 Vcc level is required. This is known as the “warming up” of the audio amplifier. This is required only at the power up stage (to charge up the capacitor), to avoid unnecessary noise output at the early stage.





2.2 Timer Toggle Output Implementation

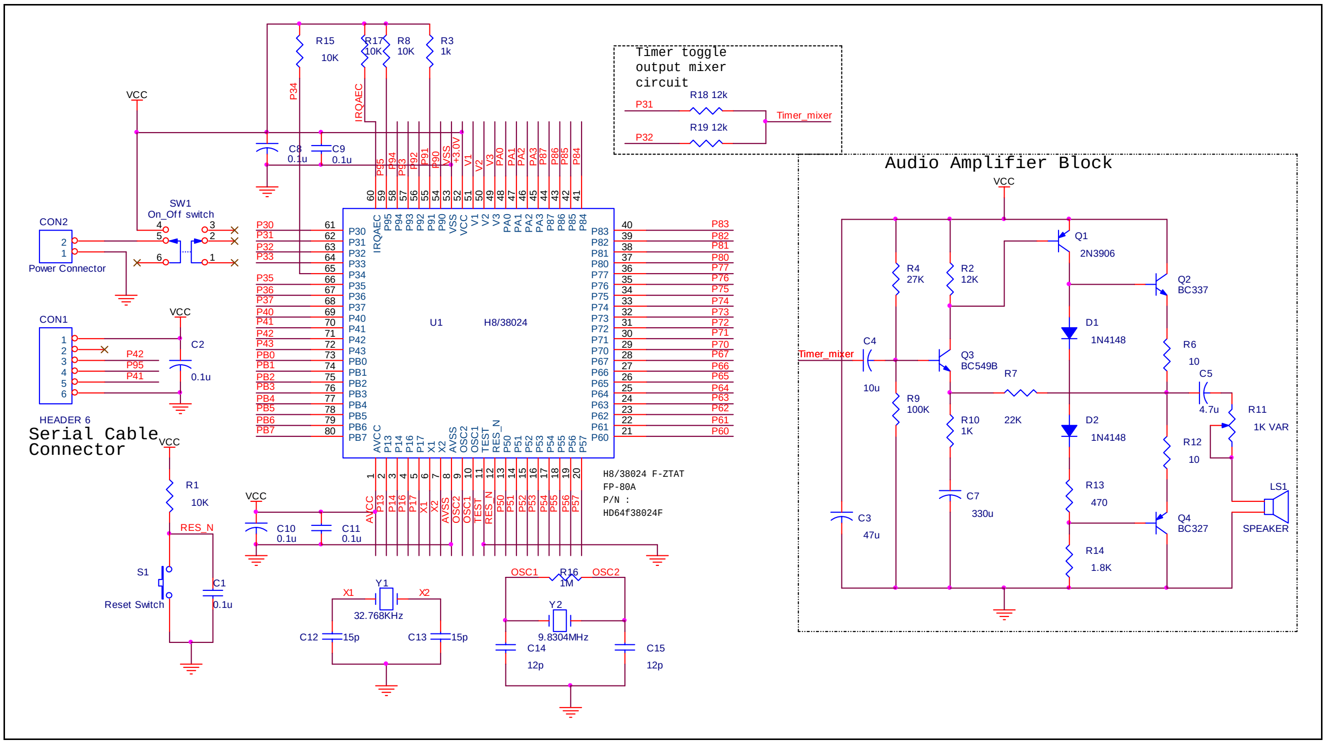


Figure 7 Schematic Diagram for Timer Toggle Output Tone Generator

The digital tone is generated by the Timer F toggle output of SLP MCU. The software will generate signal with different pulse width when the timer F output compare value is reloaded with new value. The two Timer F toggle outputs (Low counter and High counter) are combined, resulting in the generation of two digital tones simultaneously. The two digital tones are fed to the audio amplifier via the resistor mixer. User will be able to hear the tones from the loud speaker.

3. Operation and Observation

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

After the program has been successfully downloaded, reset the MCU and execute the program. During the execution, user should be able to listen to the musical tones coming out from the speaker. The demo program will play the same song repeatedly.

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

For example,

If crystal = 9.8304MHz	→	#define	XTAL	9830400L	(default)
If crystal = 4MHz	→	#define	XTAL	4000000L	

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

If PWM1 is used	→	#define	PWM_use	1	(default)
If PWM2 is used	→	#define	PWM_use	2	

4. Code Listing

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

4.1 PWM Implementation

Figure 8 shows the flowchart for the PWM implementation. The source codes for "PWM_tone.c" are listed.

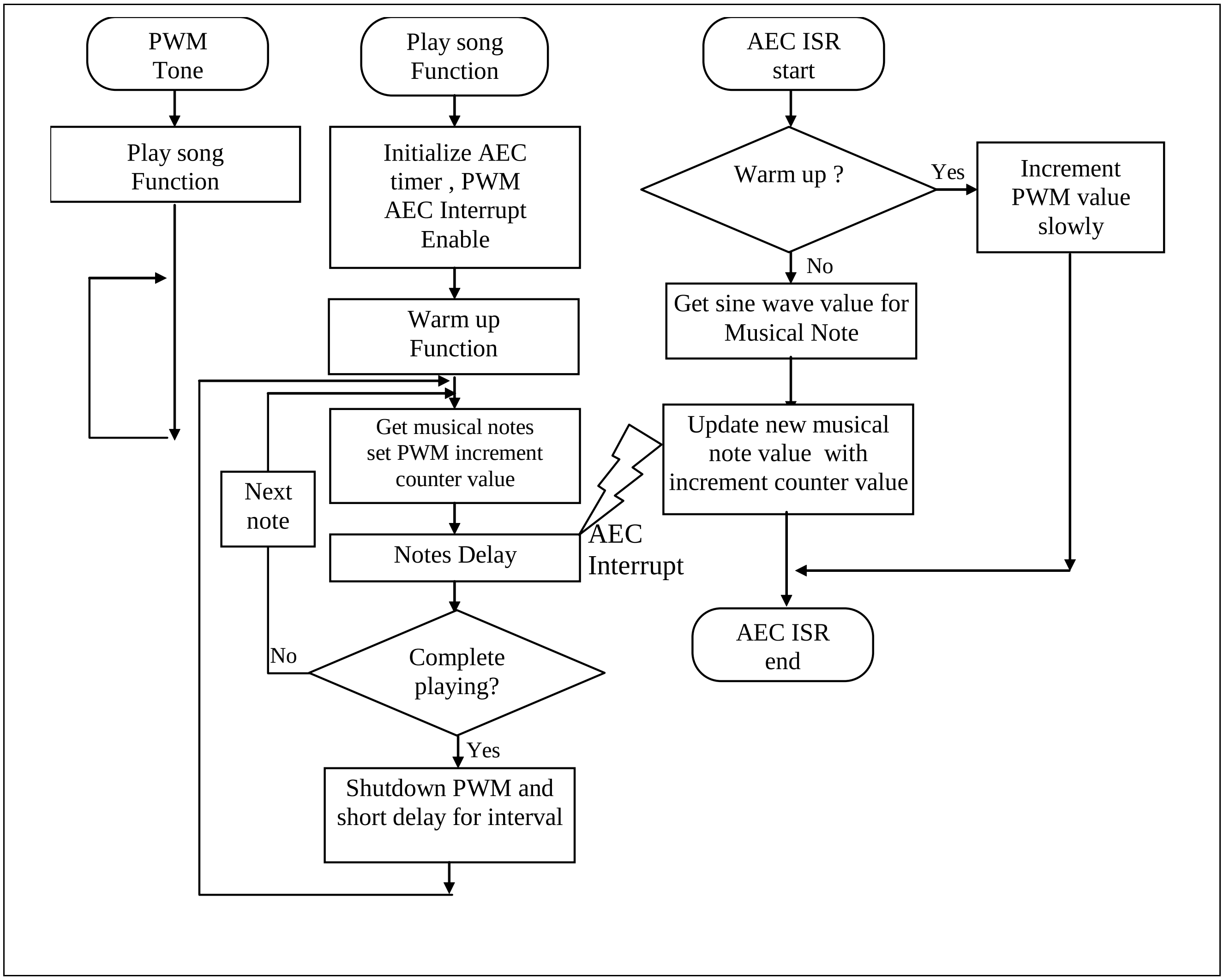


Figure 8 Flow Chart for PWM_Tone.c

```
/************************************************************************/
/*                                                                      */
/*  FILE        :PWM_Tone.c                                             */
/*  DATE        :Tue, Sep 09, 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"
#include    <math.h>
/**************************************************************/
/* define                                                     */
/**************************************************************/
#define  XTAL        9830400L
#define  sample_freq (XTAL/4L) / 256L //256 clock cycles per interrupt

#define  C1          ((256L * 523L)/100)/(sample_freq/100)
#define  C1S         ((256L * 554L)/100)/(sample_freq/100)
#define  D1          ((256L * 587L)/100)/(sample_freq/100)
#define  D1S         ((256L * 622L)/100)/(sample_freq/100)
#define  E1          ((256L * 659L)/100)/(sample_freq/100)
#define  F1          ((256L * 698L)/100)/(sample_freq/100)
#define  F1S         ((256L * 740L)/100)/(sample_freq/100)
#define  G1          ((256L * 784L)/100)/(sample_freq/100)
#define  G1S         ((256L * 830L)/100)/(sample_freq/100)
#define  A1          ((256L * 880L)/100)/(sample_freq/100)
#define  A1S         ((256L * 932L)/100)/(sample_freq/100)
#define  B1          ((256L * 987L)/100)/(sample_freq/100)

#define  C2          ((256L * 1046L)/100)/(sample_freq/100)
#define  C2S         ((256L * 1109L)/100)/(sample_freq/100)
#define  D2          ((256L * 1174L)/100)/(sample_freq/100)
#define  D2S         ((256L * 1244L)/100)/(sample_freq/100)
#define  E2          ((256L * 1318L)/100)/(sample_freq/100)
#define  F2          ((256L * 1396L)/100)/(sample_freq/100)
#define  F2S         ((256L * 1480L)/100)/(sample_freq/100)
#define  G2          ((256L * 1568L)/100)/(sample_freq/100)
#define  G2S         ((256L * 1661L)/100)/(sample_freq/100)
#define  A2          ((256L * 1760L)/100)/(sample_freq/100)
#define  A2S         ((256L * 1864L)/100)/(sample_freq/100)
#define  B2          ((256L * 1864L)/100)/(sample_freq/100)

#define  C3          ((256L * 2093L)/100)/(sample_freq/100)
#define  C3S         ((256L * 2217L)/100)/(sample_freq/100)
#define  D3          ((256L * 2349L)/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_Tone(void);void off_DTMF(void);
void init_PWM1(unsigned char selClk1);
void init_PWM2(unsigned char selClk2);
void warm_up(void);
void play_song(void);

/************************************************************/
/*Constant Look up Table for Sine Wave value
/************************************************************/
const unsigned int song1[]=
{
B2, B2, B2, A2S, G2S, A2S,
F2S, C2S, C2, F2S, F2, F2S,
A2S, G2S, B2, B2, A2S, G2S,
A2S, F2S, A1S, A1S, D2S, D2,
D2S, F2S, F2, F2, F2, F2S,
F2,  C2S, F2, D2S, B1, C2S,
D2S, C2S, D2S, F2, F2S, F2,
F2S, F2S, G2S, A2S, A2S, G2S,
G2S, G2S, 0xFF
};


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
};

/*************************************************************/
/*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 int hold=0;

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

/*************************************************************/
/*  Initialize Program                                       */
/*************************************************************/
//Initialize tone generation function
void init_Tone(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;
                       // AEC Interrupt Request, 1-Enable, 0-Disable
   //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;
}

/*************************************************************/
/*  play_song Program                                        */
/*************************************************************/
void play_song(void)
{
   i=0;

   init_Tone();

   warm_up();
   while(1)
   {
      while (song1[i]!=0xFFFF)
      {  i++;
         inc1 = song1[i++];
         set_imask_ccr(0);         // Interrupts, 0-Enable, 1-Disable
         for (j=0; j<0x35000; j++) ;
```

```
      }

      storeCount(512);
      for (j=0; j<10000; j++) ;          // short delay Tone
      set_imask_ccr(1);                  // Interrupts, 0-Enable, 1-Disable
      i = 0;
   }

   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]);
         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, 1-Enable, 0-
Disable
}
```

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
}
```

4.2 Timer Toggle Output Implementation

Figure 9 shows the flowchart for the Timer Toggle Output Implementation. The source codes for "timer_tone.c" are given.

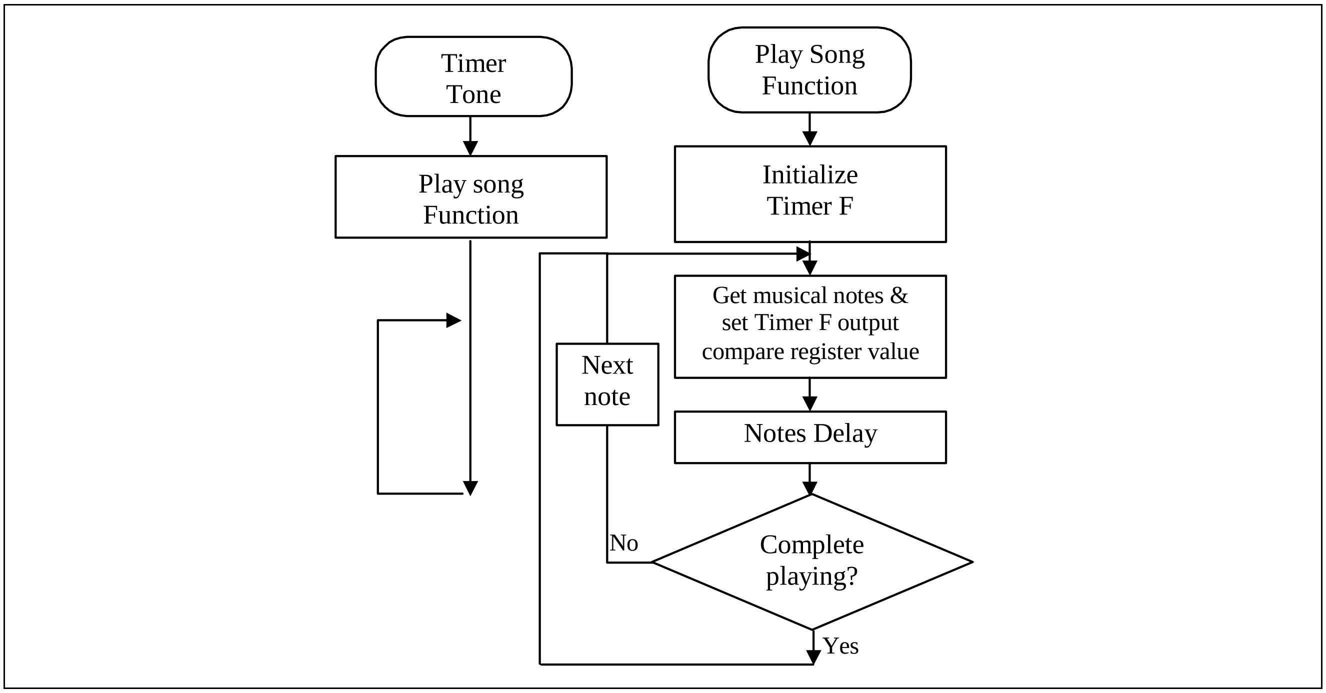


Figure 9 Flow Chart for timer_tone.c

```
/**********************************************************************/
/*                                                                    */
/*  FILE        :Timer_tone.c                                         */
/*  DATE        :Fri, Sep 12, 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  Timer_clk   32L // main clock / 32

#define  C1          (XTAL / (Timer_clk*4L*523L))
#define  C1S         (XTAL / (Timer_clk*4L*554L))
#define  D1          (XTAL / (Timer_clk*4L*587L))
#define  D1S         (XTAL / (Timer_clk*4L*622L))
#define  E1          (XTAL / (Timer_clk*4L*659L))
#define  F1          (XTAL / (Timer_clk*4L*698L))
#define  F1S         (XTAL / (Timer_clk*4L*740L))
#define  G1          (XTAL / (Timer_clk*4L*784L))
#define  G1S         (XTAL / (Timer_clk*4L*830L))
#define  A1          (XTAL / (Timer_clk*4L*880L))
#define  A1S         (XTAL / (Timer_clk*4L*932L))
#define  B1          (XTAL / (Timer_clk*4L*987L))

#define  C2          (XTAL / (Timer_clk*4L*1046L))
#define  C2S         (XTAL / (Timer_clk*4L*1109L))
#define  D2          (XTAL / (Timer_clk*4L*1174L))
#define  D2S         (XTAL / (Timer_clk*4L*1244L))
#define  E2          (XTAL / (Timer_clk*4L*1318L))
#define  F2          (XTAL / (Timer_clk*4L*1396L))
#define  F2S         (XTAL / (Timer_clk*4L*1480L))
#define  G2          (XTAL / (Timer_clk*4L*1568L))
#define  G2S         (XTAL / (Timer_clk*4L*1661L))
#define  A2          (XTAL / (Timer_clk*4L*1760L))
#define  A2S         (XTAL / (Timer_clk*4L*1864L))
#define  B2          (XTAL / (Timer_clk*4L*1975L))

#define  C3          (XTAL / Timer_clk*4L)/(2093L)
#define  C3S         (XTAL / Timer_clk*4L)/(2217L)
#define  D3          (XTAL / Timer_clk*4L)/(2349L)
```

```
/*************************************************************/
/*  Function define                                          */
/*************************************************************/

void init_Tone(void);
void play_song(void);

/*************************************************************/
/*Constant Look up Table for Sine Wave value
/*************************************************************/
const unsigned char song1[]=
{
B2,  B2,  B2, A2S, G2S, A2S,
F2S, C2S, C2, F2S, F2, F2S,
A2S, G2S, B2, B2, A2S, G2S,
A2S, F2S, A1S, A1S, D2S, D2,
D2S, F2S, F2, F2, F2, F2S,
F2,  C2S, F2, D2S, B1, C2S,
D2S, C2S, D2S, F2, F2S, F2,
F2S, F2S, G2S, A2S, A2S, G2S,
G2S, G2S, 0xFF
};

/*************************************************************/
/*Global variable
/*************************************************************/
unsigned int i=0,j=0, count=0;

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

/*************************************************************/
/*  Initialize Program                                       */
/*************************************************************/
//Initialize tone generation function
void init_Tone(void)
{
    set_imask_ccr(1);                      // Interrupt Disable

   //Init Timer F start

   // 8 bit timer F counter, Sub clock / 4 selected toggle output enable
   P_IO.PMR3.BYTE = 0x06;
   P_TMRF.TCRF.BYTE = 0xCE;
   P_TMRF.TCSRF.BYTE = 0x11;
   //TCF cleared when TCF and OCRF match
```

```
   if (P_TMRF.TCSRF.BIT.CMFH == 1) P_TMRF.TCSRF.BIT.CMFH = 0;
   if (P_TMRF.TCSRF.BIT.CMFL == 1) P_TMRF.TCSRF.BIT.CMFL = 0;

    set_imask_ccr(0);                     // Interrupt Enable

   //Init Timer F end
}


/**************************************************************/
/*  play_song Program                                          */
/**************************************************************/
void play_song(void)
{
   unsigned int i=0, j=0;

   init_Tone();
   while(1)
   {
      while (song1[i]!=0xFF)
      {
         P_TMRF.OCRF.BYTE.H = song1[i];
         P_TMRF.OCRF.BYTE.L = song1[i];
         i++;
         for (j=0; j<35000; j++) ;
      }
      for (j=0; j<35000; j++) ;
      i=0;
   }
   P_TMRF.TCRF.BYTE = 0x00;
}
```

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.23.03	—	First edition issued