AN10875

IEC 60601-1-8 audible alert generator using the LPC1700

Rev. 01 — 1 October 2009 Application note

Document information

Info	Content
Keywords	IEC60601-1-8, LPC1700, Cortex-M3, MCB1700, Goertzel, Medical Alerts
Abstract	This application note describes an algorithmic method of generating audible medical alarms that comply with IEC60601-1-8. An overview of medical alarms is presented and the derivation of the algorithm used is provided. The algorithm code implementation is then detailed and discussed including a detailed performance analysis.

Revision history

Rev	Date	Description
01	20091001	Initial version.

Contact information

For additional information, please visit: http://www.nxp.com

For sales office addresses, please send an email to: salesaddresses@nxp.com

1. Introduction

1.1 Background on audible medical alarms

The ACCE Healthcare Technology Foundation recently did a study and survey to determine the impact of clinical alarms on patient healthcare. Task force chairman Toby Clark reported a number of limitations of clinical alarms and the impact they have on patient health. The study searched a database on fatality incidents where the word “alarm” was included in the product problem description. The search results showed an average of about 80 deaths a year during the period of 2002 to 2004 may be attributed to issues with medical alarms. Some of the alarm limitations the study identified by surveys are outlined below:

- Difficulty in Learning more than 6 alarm signals – ICU and Surgery have >> 6 Alarms
- Difficulty in discerning between high and low priority alarms
- Perceived urgency of alarms may not be consistent with criticality of situation
- False alarms

1.2 Alarms and human behavior

A report on this subject appeared in the British Journal of Anaesthesia titled, “Alarms and Human Behaviour: Implications for Medical Alarms”.

One of the subjects of the report was identifying the characteristics of an ideal alarm sound and the suggested knowledge on how those characteristics can be achieved. Table 1 lists some of the findings.

Table 1. Ideal alarm characteristics and how to achieve an ideal alarm sound

Characteristics of an ideal alarm	Relevant finding
Easy to localize	The ear uses two mechanisms for localizing sound, one at high frequencies and one at low frequencies. Neither functions well in the mid-to-high normal band of frequencies of normal hearing
Resistant to masking by other sounds	Sounds that are acoustically ‘rich’, that is, contain a number of harmonics, are more resistant to masking
Allow communication	Continuous sounds are more likely to be irritating and interfere with communication
Easy to learn and retain	People are poor at retaining the absolute pitch of a tone and find it difficult to distinguish sounds that vary only in pitch unless they are heard in close temporal proximity. In addition, abstract sounds are harder to learn and retain than environmental sounds or auditory icons.

1.3 IEC60601-1-8 audible and visual alarm standard

To address some of the limitations of medical alarms and to utilize modern research available on ideal alarm characteristics, the IEC (International Electrotechnical Commission) has provided the first focused standard on audible and visual alarms for medical equipment, the IEC60601-1-8. Focusing on the audible alert portion of the document, the IEC60601-1-8 standard requires a specific melody correspond with a specific physiological function. This ensures that the number of alarms is contained, instead of varying across different manufacturers. It also limits the number of alarms to eight and uses a cautionary and an emergency version of each. The emergency (high priority) version uses a five note melody that is repeated. The cautionary (medium priority) alarm uses the first three notes of the high priority version and does not repeat. Some of the principles of designing perceived urgency into sound have been applied to these signals such as a slower rise and fall times on medium priority tones compared to high priority tones and a faster tempo for the high priority alarms. An optional low priority alarm tone is also provided in the standard that sounds only two notes. The high priority melody corresponding to the physiological function it represents is given in the table below. (Note: high priority melodies repeat once.)

Table 2. Audible alert types and associated melodies

Alarm	High priority melody	Mnemonic notes
General	C4-C4-C4-C4-C4	Fixed pitch
Cardiac	C4-E4-G4-G4-C5	Trumpet call; Call to arms; Major chord
Artificial Perfusion	C4-F#4-C4-C4-F#4	Artificial sound; Tri-tone
Ventilation	C4-A4-F4-A4-F4	Inverted major chord; Rise and fall of the lungs
Oxygen	C5-B4-A4-G4-F4	Slowly falling pitches; Top of a major scale; Falling pitch of an oximeter
Temperature	C4-E4-D4-F4-G4	Slowly rising pitches; Bottom of a major scale; Related to slow increase in energy or (usually) temperature
Drug delivery	C5-D4-G4-C5-D4	Jazz chord (inverted 9th); Drops of an infusion falling and "splashing"
Power failure	C5-C4-C4-C5-C4	Falling or dropping down

The melody note C4 in the above table refers to middle C, with C5 being one octave above middle C. The IEC states that you do not have to use those specific notes. As long as the fundamental note is within the specified frequency range, the alarm melody could be transposed to different keys and still be compliant with the specification. The IEC does state that the note must consist of the fundamental tone and at least 4 harmonics. The fundamental and 4 harmonics must not differ by more than 15 db in amplitude.

The IEC60601-1-8 Audible alarm standard provides tones that are rich in harmonics to make them easy to localize and resistant to masking. There are a limited amount of categories and corresponding melodies to make the different alarm sequences easier to learn. And, the priority of an alarm determines the number of notes in the alarm and the dynamic characteristics of the note sequence. This makes it easy to determine the

criticality of the alarm. Thus, the IEC60601-1-8 standard addresses many of the limitations of previous alarms and will hopefully contribute to improved patient safety.

2. Generating the IEC60601-1-8 alarms algorithmically

Many of the present IEC60601-1-8 implementations playback a recorded version of the alarm that is stored in memory. The drawback to this is it takes up a lot of memory space to store the files as well as the program space to control the playback. Generating the alarms algorithmically is a much more efficient method in terms of memory usage and allows a lot of flexibility in being able to customize the tones while still meeting IEC60601-1-8 specifications.

2.1 Functional resources required on chip

To synthesize the alarm tones on chip requires the following functions to be implemented in firmware:

- Timing Generator – This provides the timing reference to digitally construct the alarm tones effectively setting the internal sample rate and the DAC output rate. This also provides the time reference for the note sequencer and envelope generator.
- Envelope Generator – This controls the rise time, fall time, and amplitude of the note pulses. The rise and fall time of a note is also a function of the alarm priority.
- Note Sequencer – This sequences through the correct notes at the correct tempo based on the type of alarm and the priority level.
- Note generator – This generates multiple sine wave tones that are combined to form the fundamental and harmonics that make up the alert note.

For the demonstration code, a menu driven user interface is provided via UART1 and a terminal program. This includes the different menus, a serial port driver, and a simple command handler.

2.1.1 Timing generator

This utilizes the on chip timer to set the sample rate / DAC output rate for the algorithmic tone generator. The timer is set up to generate an interrupt every 40 μs for a 25 ksps DAC output rate. This was chosen to be above the audible range and much higher than the Nyquist frequency to allow low cost filters on the DAC. A software timer in the interrupt service routine also provides a 1 ms timeout that is used by the envelope control functions and note sequencer as described below.

2.1.2 Envelope generator

The envelope generator controls the dynamic volume of the tones being generated. Since the IEC specification includes rise and fall times for the tones, a variable is generated that is time dependent. When a note is turned on, the value of the envelope variable increases from 0 to the maximum set level at a controlled rate. The same happens when the note is turned off; the level will decrease at a controlled rate until it reaches 0. The rise and fall times are programmable and the medium and low priority tones have a slower rise/fall time than the high priority tones. The envelope generator uses the 1 ms timeout as its timing reference.

2.1.3 Note sequencer

The IEC 60601-1-8 standard specifies the relative note sequences and temporal characteristics for the tones as a function of the classification and priority of the alarm. The note sequencer outputs the tones with the right duration and spacing to meet the

temporal characteristics for a given priority of alarm. For high priority alarms, the sequence is specified to be a 5 note sequence that is repeated once for a total of 10 notes. The medium priority alarm is a 3 note sequence and is not repeated. The three note sequence for a given alarm type is the same as the first three notes of the high priority sequence to make learning the alarms easier. The tempo of the high priority alarm is faster than the medium priority alarms. The different number of notes and tempo differences make it easy to discern the priority of the alarm.

2.1.4 Note generator

In order to make it easy to comply with the standard, the fundamental and 4 harmonics will be generated as separate sine waves and combined digitally. There are many ways to generate sine waves on chip including sine table look up, math library algorithms, Taylor series expansions, and recursive oscillators. The recursive oscillator is an IIR filter structure with the proper coefficients to oscillate given the proper initialization. Since this is one of the more efficient methods, and is also very low distortion, the recursive oscillator will be used to generate the fundamental and 4 harmonics that are required for each note.

2.1.4.1 The Goertzel algorithm

One of the most useful recursive methods is the Goertzel Algorithm, a simple two tap IIR filter shown in Fig 1. This is a very useful algorithm as it can also be used as a narrow band tone detector in addition to a sine wave generator. Analyzing the algorithm as an oscillator we want to be able to calculate the sine of an angle as we increment the angle in fixed steps. Assuming we can calculate the sine of an angle from the two previous incremental values in the sine series, equation (1) can be written for Fig 1:

$$\sin(a + n * b) = x * \sin(a + (n - 2) * b) + y * \sin(a + (n - 1) * b) \quad (1)$$

In equation (1), x and y are the coefficients of the IIR filter, ‘a’ is the starting angle, and ‘b’ is the incremental angle. To find the values of the two coefficients, we will first re-arrange and simplify equation 1 as shown below in equation 2.

$$\sin(a + n * b) = x * \sin(a + n * b - 2 * b) + y * \sin(a + n * b - 1 * b) \quad (2)$$

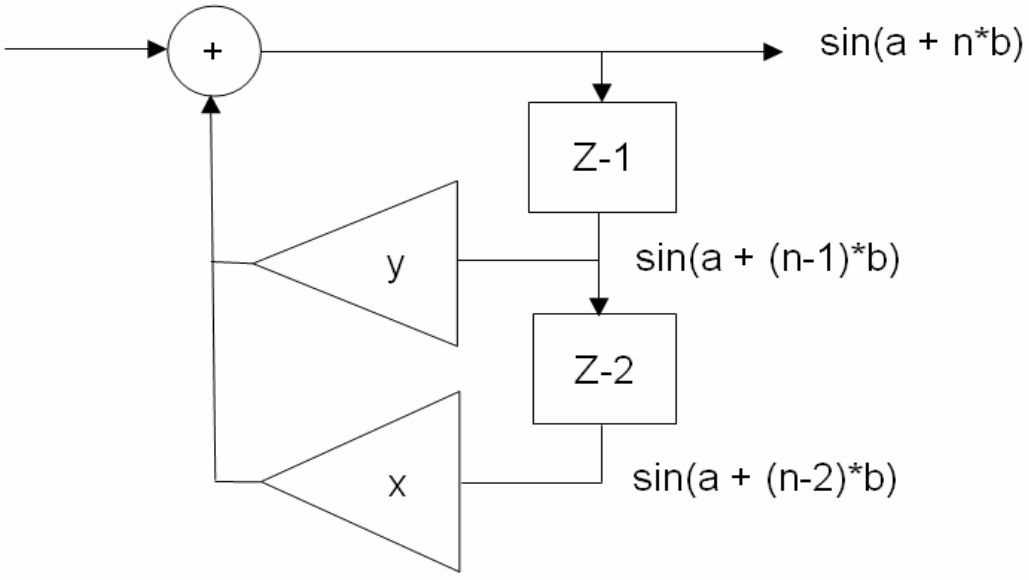

Fig 1. IIR filter

Continuing to expand equation 2 by substituting the following trigonometric identity:

$$\sin(a+b) = \sin(a) * \cos(b) + \cos(a) * \sin(b) \quad (3)$$

We get:

$$\sin(a + n * b) = x * [\sin(a + n * b) * \cos(2 * b) - \cos(a + n * b) * \sin(2 * b)] + y * [\sin(a + n * b) * \cos(b) - \cos(a + n * b) * \sin(b)] \quad (4)$$

Re-arranging:

$$\sin(a + n * b) = [x * \cos(2 * b) + y * \cos(b)] * \sin(a + n * b) - [x * \sin(2 * b) + y * \sin(b)] * \cos(a + n * b) \quad (5)$$

For this to be true for all n, we must have the two expressions in brackets satisfy:

$$[x * \cos(2 * b) + y * \cos(b)] = 1$$
$$[x * \sin(2 * b) + y * \sin(b)] = 0 \quad (6)$$

That, when solved, yields the coefficients for the recursive IIR filter:

$$x = -1$$

$$y = 2 * \cos(b) \quad \text{(where b is the step angle } 2\pi \frac{f}{f_{sample}}) \quad (7)$$

Substituting this back into the original equation yields:

$$\sin(a + n * b) = -\sin(a + (n-2) * b) + 2 * \cos(b) * \sin(a + (n-1) * b) \quad (8)$$

Re-arranging:

$$\sin(a + n * b) = 2 * \cos(b) * \sin(a + (n-1) * b) - \sin(a + (n-2) * b) \quad (9)$$

Substituting our sample value Y[n] = sin(a+nb) yields:

$$y[n] = 2 * \cos b * y[n-1] - y[n-2] \quad (10)$$

So, as a result of one of the coefficients being equal to -1, the calculations at each step angle increment requires only one multiplication and one subtraction involving the results of the two previous calculations. (This assumes the coefficient is calculated ahead of time.) After executing the equation above, the y[-1] value is moved into the y[-2] variable and the calculated y[n] value is moved into the y[-1] variable to prepare for the calculations at the next step angle. This makes for very efficient operation for the Cortex-M3 as we will see later in the code implementation.

2.1.4.2 Analysis of the Goertzel algorithm

If we want to analyze the Goertzel algorithm, we can assume it has an input x(n) and write the transfer function as follows:

$$y[n] = 2 \cos b * y[n1] - y[n2] + x[n] \quad (11)$$

If we take the Z transform of this we first write the equation as follows:

$$Y[Z](1 - (2 \cos b * Z1) + Z2) = X[Z] \quad (12)$$

The transfer function can then be written as:

$$H(Z) = \frac{Y[Z]}{X[Z]} = \frac{1}{1 - (2\cos b)Z^{-1} - Z^{-2}} \quad (13)$$

Factoring out the roots:

$$H(Z) = \frac{Y[Z]}{X[Z]} = \frac{1}{(1 - e^{+bi}Z^{-1})(1 - e^{-bi}Z^{-1})} \quad (14)$$

Thus, the Goertzel algorithm's transfer function has poles where:

$$(1 - e^{+bi}Z^{-1}) = 0$$
$$(1 - e^{-bi}Z^{-1}) = 0 \quad (15)$$

Since the magnitude of Z is '1', the magnitude of the transfer functions poles is 1, placing them on the unit circle of the Argand diagram. The location of the two poles is then:

$$e^{+bi} = \cos b + i \sin b$$
$$e^{-bi} = \cos b - i \sin b \quad (16)$$

We know that the impulse response of a feedback system such as an IIR filter has the following characteristics as a function of the pole locations relative to the unit circle in the Z domain:

- if the poles are inside the unit circle, the transient terms will die away
- if the poles are on the unit circle, oscillations will be in a steady state
- if the poles are outside the unit circle, the transient terms will increase

So, with the poles on the unit circle, this meets the criteria for steady state oscillation.

2.1.4.3 Goertzel initialization

In order for the Goertzel to function as an oscillator, the y[-1] and y[-2] values must be initialized. If we set y[-1] = 0, then y[-2] would have the value of one incremental sine value before the zero crossing, or:

$$y[-1] = 0$$

$$y[-2] = A * \sin 2\pi \frac{F_{output}}{F_{sample}} \quad (17)$$

The coefficient must also be calculated as follows in equation 18.

$$coef = 2 * \cos 2\pi \frac{F_{output}}{F_{sample}} \quad (18)$$

To use the Goertzel Algorithm as a tone detector, both y[-1] and y[-2] would be initialized to '0'. The input would then be summed into the calculation. After a certain number of samples, the input's amplitude at the detect frequency can be calculated from the following:

$$real \quad y[\ 1] \quad y[\ 2] \quad \cos 2\pi \frac{F_{detect}}{F_{sample}}$$

$$imaginary \quad y[\ 2] \quad \sin 2\pi \frac{F_{detect}}{F_{sample}} \quad (17)$$

$$magnitude^2 \quad real^2 \quad imaginary^2$$

There are optimized versions of this to simplify the math, but this explains the principle. To use this as a continuous tone detector, after making this calculation, y[-1] and y[-2] would again be set to zero and the next acquisition and detection sequence would proceed.

3. Code implementation - audible alarm synthesis

3.1 Timing generator code

The timing generator code consists of the initialization for Timer 0 and the interrupt service routine to handle Timer 0 interrupts. This provides the 40 μs (25 kHz) time base for generating the alarm tones as well as a software counter to generate a 1 ms time base that is used by the envelope generator and the note sequencer.

3.1.1 Timer 0 initialization

Timer 0 is initialized to generate a match interrupt every 40 μs to provide the 25 kHz sample rate. The initialization code is shown below.

```
1    void init_timer (void)
2    {
3      TIM0->MR0 = 959;         // 40 uSec = 960-1 counts (25ksps @ 24 mhz)
4      TIM0->MCR = 3;           // Interrupt and Reset on MR0
5      TIM0->TCR = 1;           // Timer0 Enable
6      timeval=0;                    // variable initializations
7      mscount=0;
```

```
8       sequence = 0;
9     }
```

3.1.2 Timer 0 interrupt service routine

The Timer 0 interrupt service routine is the heart of this application since it provides all the timing for tone generation as well as tone sequencing. The code tests to see if the envelope is on, and if so, will output tones. In addition to clearing the interrupt, a software counter is incremented until a 1 millisecond timeout is reached. Every time the 1 ms timeout occurs, the state of the sequencer is incremented and any required actions will be taken by the envelope generator (to be discussed later). The code listing for the Timer 0 Interrupt service routine is shown below.

```
10    void TIMER0_IRQHandler (void)
11    {
12          if (envelope_on )
13          {
14                OutputTones(active_note,note_level); // params are set in sequencer
15          }
16          TIM0->IR = 1;                           // Clear interrupt flag
17
18          timeval++;
19          if (timeval == 25)         // 1 millisecond interval @ 25 khz sample rate
20          {
21              if (sequence)
22                {
23                      switch (priority)
24                      {
25                            case 1:
26                                  HighPriSequence(alarm);
27                                  break;
28                            case 2:
29                                  MedPriSequence(alarm);
30                                  break;
31                            case 3:
32                                  LowPriSequence(alarm);
33                                  break;
34                            case 4:
35                                  TestSequence(alarm);
36                                  break;
37                      }
38                }
39                mscount++;                // increment ms counter
40                timeval = 0;              // clear interval counter
41                EnvelopeControl();        // envelope actions required?
42          }
43
44    }
```

3.2 Envelope control function

The envelope generator controls the dynamic level of the tones as the IEC60601-1-8 requires rise times and fall times to be within a specified range. Also, the first note in an alarm sequence is supposed to be lower in amplitude than the others. The envelope generator output is the variable 'envelope'. When a tone is off, envelope = 0. When a note is turned on in a sequence, the envelope variable will increase at a set rate every millisecond until it reaches the maximum level set. When a note is turned off, the tone continues, but the envelope variable begins to decrease at its set rate until it reaches '0'. The envelope variable will be used as the volume control for the dynamic characteristics of the note. The listing for the envelope control function is given below:

```
45   void EnvelopeControl(void)
46   {
47      if (note_on)
48      {
49          if (envelope >= note_level)
50          {
51              envelope = note_level;
52           }
53           else
54           {
55              if (priority == 1)
56              {
57                  envelope += HP_RISE;   // high priority risetime control
58              }
59              else
60              {
61                  envelope += MP_RISE;    // Medium priority risetime control
62              }
63         }
64     }
65     else
66     {
67         if (envelope >0)
68         {
69           if (priority == 1)
70              {
71                   envelope -= HP_FALL;    // high priority falltime control
72              }
73              else
74              {
75                   envelope -= MP_FALL;    // Medium priority falltime control
76              }
77         }
78     }
79     if ((envelope <= 0)&& (note_on == 0)&& (envelope_on == 1) )
80     {
81          envelope = 0;
82          envelope_off = 1;    // synchronize with zero cross
83
84     }
```

```
85    }
```

3.3 Note sequencer functions

The note sequencer provides the timing and note sequencing for the for the different alarm melodies specified in IEC60601-1-8. The high priority alarms consist of a 5 note sequence that is repeated. The medium priority version of the same alarm category uses the first three notes of the high priority alarm and is not repeated. The note spacing and duration is shorter for the high priority alarms as this gives a higher sense of urgency. The code to implement these is shown below for the high priority example. The 1 ms time base, discussed in section 3.1.2, increments the sequence counter. As a result, the numbers associated with each case statement are in milliseconds to facilitate making adjustments in the timing. The medium and low priority versions use the same basic switch statement format and similar function calls, but different timing states are used in the associated case statements since the tempo is different.

```
86    void HighPriSequence (unsigned char alarm_type)
87    {
88       switch (mscount)
89       {
90            case 1:
91               active_note = tune_sequence [alarm_type][0];  // 1rst note of sequence
92               note_level = 200;
93               TurnOnNote();
94               break;
95            case 145:                        //145 ms (trise +tduration)
96               note_on = 0;                  // begin decay as note turns "off"
97               break;
98            case 224:
99               active_note = tune_sequence [alarm_type][1];  // 2nd note of sequence
100              note_level = 255;
101              TurnOnNote();
102              break;
103           case 368:
104              note_on = 0;                  // begin decay as note turns "off"
105              break;
106           case 447:
107              active_note = tune_sequence [alarm_type][2];  // 3rd note of sequence
108              note_level = 255;
109              TurnOnNote();
110              break;
111           case 591:
112             note_on = 0;              // begin decay as note turns "off"
113              break;
114           case 835:
115              active_note = tune_sequence [alarm_type][3];  // 4th note of sequence
116              note_level = 255;
117              TurnOnNote();
118              break;
119           case 929:
120              note_on = 0;                  // begin decay as note turns "off"
121              break;
```

```
122          case 1008:
123             active_note = tune_sequence [alarm_type][4];  // 5th note of sequence
124             note_level = 255;
125             TurnOnNote();
126             break;
127          case 1152:
128             note_on = 0;                // begin decay as note turns "off"
129             break;
130
131          case 1200:                     // allows for fall time of envelope
132             if (sequence == 2)
133             {
134                   sequence = 0;
135                   mscount = 0;
136             }
137             break;
138          case 1671:
139             if (sequence == 1)       // If this is the first time through, repeat
140             {
141                   sequence++;
142             }
143             mscount = 0;
144             break;
145       }
146   }
147   void TurnOnNote(void)
148   {
149         envelope = 0;
150         note_on = 1;
151         envelope_on = 1;
152   }
```

3.4 Note generator functions

For the note generator, there is a lot of data to be processed dealing with multiple frequencies and multiple sine wave generators. If we organize the data variables and constants that are associated with each frequency in arrays that have the same dimensions, we can then just use a simple recursive function to “walk” through the array data.

3.4.1 Note generator definitions

In the definitions we define several fixed and floating point constants, as well as defining a structure tag and two arrays with one of these being an array of structures.

```
153   #define Amp 200                   // Output Amplitude
154   #define PI  3.1415926
155   #define Fsample  25000            // Timer  Reload Frequency
156
157   struct wave  {                    // structure tag for Sine Generator
158     short coef;                     // IIR filter coefficient
159     long y1;                        // y[-1] value
160     long y2;                        // y[-2] value
```

```
};

struct wave Waves[9][5];    // 'Waves' is an array of structures whose values are
                            // calculated during initialization
long output;
long output_old;

enum {C4,D4,E4,F4,Fsharp4,G4,A4,B4,C5};  // Can address array rows with notes

float const FreqArray[][5]= {{261.626,523.252,784.878,1046.50,1308.13},  // C4
                             {293.67,587.34,881.01,1174.7,1468.3},       // D4
                             {329.63,659.26,988.89,1318.52,1648.15},     // E4
                             (349.23,698.46,1047.69,1396.9,1746.15},     // F4
                             {369.99,739.98,1109.97,1479.96,1849.95},  // FSharp4
                             {392.00,784.00,1176.0,1568.0,1960.0},       // G4
                             {440.000,880.00,1320.0,1760.00,2200.00},    // A4
                             {493.88,987.76,1481.64,1975.5,2469.4},      // B4
                             {523.251,1046.50,1569.756,2093.00,2616.25}}; // C5

unsigned char ToneWeights[] = {255,255,255,255,255};// used for test and
                                                    // adjusting harmonic levels
```

The advantage of using a structure of the variables and coefficients used in the algorithm is that it allows us to have a similar array organization in the structure array and the frequency constant array. The one for one relationship between the constant frequency array and the algorithm structure array makes it easy to use similar indexes for both arrays when initializing each frequency.

3.4.2 Note generator initialization

As mentioned in section 2.1.4.3, for the Goertzel algorithm to oscillate, the y[-1] and y[-2] values must be initialized in addition to the coefficient. This must be done for the structure variables that correspond to each frequency. The code for the initializations is shown below. Each of the 5 structures in a row is initialized then each additional row is initialized until the entire array of structures is initialized. In this demo application, these calculations are done during reset initialization. However, if you were optimizing this, code could be saved by making these calculations ahead of time and storing the results as constants in flash memory. This is because if the floating point and sine/cosine algorithms that are needed from the math library that would not be required with pre-computed initialization value. These library routines use about half of the code space used by this application. The coefficient and initialization values are scaled by 32768 (signed short). Also, the coefficient calculation does not include the 2X factor shown in the equation (18). This is to keep the size of the coefficient to a signed short to minimize storage requirements. The 2X is included in the final Goertzel calculation where the output is scaled by >>14 instead of >>15, effectively multiplying by 2.

```
void InitToneCoefArray(void)     // initialize the structure for each frequency
{
  unsigned char n;
  unsigned char j;
  for (j=0;j<9;j++)              // Initialize all nine scale tones (C4-C5)
  {
     for (n=0;n<5;n++)           // fundamental and 4 harmonics for IEC60601-1-8
```

```
189       {
190       Waves[j][n].coef = ((cos (2*PI*(float)(FreqArray[j][n]/Fsample)))* 32768);
191            Waves[j][n].y1  = 0;
192            Waves[j][n].y2  = ((sin (2*PI*(float)((FreqArray[j][n]/Fsample))) *
                                    Amp * 32768));
193       }
194    }
195  }
```

3.4.3 Multiple sine wave generation, summing, and DAC output

Once the algorithm variables and coefficients have been initialized it is easy to then make the Goertzel calculations to generate the fundamental and 4 harmonics by simply incrementing through a row in the array of structures and summing the five values. As mentioned in section 3.4.2, line 204 includes scaling by >>14 instead of >>15 to factor in the 2X that was left out of the coefficient initialization.

```
196  void GenerateMultiTone (struct wave *t)
197  {
198    long y;
199    unsigned char i;
200    unsigned char env_weights;
201    output = 0;                    // clear output accumulator
202    for (i=0;i<5;i++)              // cycle through the 5 structures in the array
203    {
204       y = ((t->coef *(long long)(t->y1)>>14)) - t->y2;  // Goertzel Calculation
205       t->y2 = t->y1;                                    // store for next time
206       t->y1 = y;                                        // store for next time
207       env_weights = envelope * ToneWeights[i]>>8;
208       output += ((t->y1* env_weights)>>8);  // sum fundamental and harmonics
209       t++;                                  // increment structure pointer
210    }
211    DAC->DACR = ((output >> 10) & 0xFFC0) + 0x8000; //make unsigned & output to DAC
212    if ((output >= 0)&& (output_old <= 0))    // zero crossing detect
213    {
214       if (envelope_off && (note_on==0))
215        {
216            envelope_on = 0;        // synchronizes turn off with zero cross
217            envelope_off = 0;       // reset envelope flag
218        }
219    }
220    output_old = output;
221  }
222
223  void OutputTones(unsigned char note, unsigned char level)
224  {
225    note_level = level;
226    GenerateMultiTone (&Waves[note][0]);
227  }
```

Once the function has done the calculations for all 5 structures in the array row, the summed value is scaled, formatted, and converted from a signed to unsigned value and before being sent to the DAC. Since these calculations are performed at each 25 kHz timer interrupt (when a note is active), the DAC output rate is 25 kHz. This allows

inexpensive output filters as this is approximately a 9X over-sampling relative to the highest sine wave frequency being generated.

3.5 User / command interface

For the demonstration firmware that was written for this application, the Keil MCB1700 was targeted and UART1 is used to provide a menu driven terminal interface to activate the different alarms. To keep the application note brief, the UART code is not shown. A code example of one of the command functions is shown as an illustration of how the alarm sequences are initiated using this firmware.

```
228  void cmd_test(void)
229   {
230        if (proc_cmd)
231        {
232           switch (priority)
233           {
234               case 1:
235                    HPCommnds();
236                    break;
237               case 2:
238                    MPCommnds();
239                    break;
240               case 3:
241                    LPCommnds();
242                    break;
243               case 4:
244                    TestCommnds();
245                    break;
246           }
247        }
248   }
249  void HPCommnds(void)
250  {
251        switch (rcv_buf)
252        {
253           case '1':
254               putstr ("High Priority General Alarm\n\n");
255               alarm = GENERAL;
256               sequence = 1;
257               break;
258           case '2':
259               putstr ("High Priority Cardiac Alarm\n\n");
260               alarm = CARDIOVASCULAR;
261               sequence = 1;
262               break;
263           case '3':
264               putstr ("High Priority Artificial Perfusion Alarm\n\n");
265               alarm = PERFUSION;
266               sequence = 1;
267               break;
268           case '4':
```

```
269             putstr ("High Priority Ventilation Alarm\n\n");
270             alarm = VENTILATION;
271             sequence = 1;
272             break;
273         case '5':
274             putstr ("High Priority Temperature Alarm\n\n");
275             alarm = TEMPERATURE;
276             sequence = 1;
277             break;
278         case '6':
279             putstr ("High Priority Oxygen Alarm\n\n");
280             alarm = OXYGEN;
281             sequence = 1;
282             break;
283         case '7':
284             putstr ("High Priority Drug Delivery Alarm\n\n");
285             alarm = DRUG_DELIVERY;
286             sequence = 1;
287             break;
288         case '8':
289             putstr ("High Priority Equipment/Supply Failure Alarm\n\n");
290             alarm = POWER_FAIL;
291             sequence = 1;
292             break;
293         default:
294             putstr ("Command not supported\n\n");
295             break;
296         }
297           proc_cmd=0;      // reset command status
298           mscount = 0;
299 }
```

As you can see, all that is needed to start the alarm sequence is to set the alarm type using the ‘alarm’ variable and enable the sequencer by setting the ‘sequence’ variable to a non-zero value. The menu structure will be shown later in section 4.

The entire code package for this application is available from NXP.

4. IEC60601-1-8 audible alarm demo operation

4.1 Reset menu

When the MCB1700 is loaded with the IEC Audible Alarm firmware, the user is prompted with the following menu after reset when the COM1 interface on the MCB1700 is connected to a PC running HyperTerminal, TeraTerm, or similar terminal program. The settings are 9600, one start bit, one stop bit and no parity or flow control.

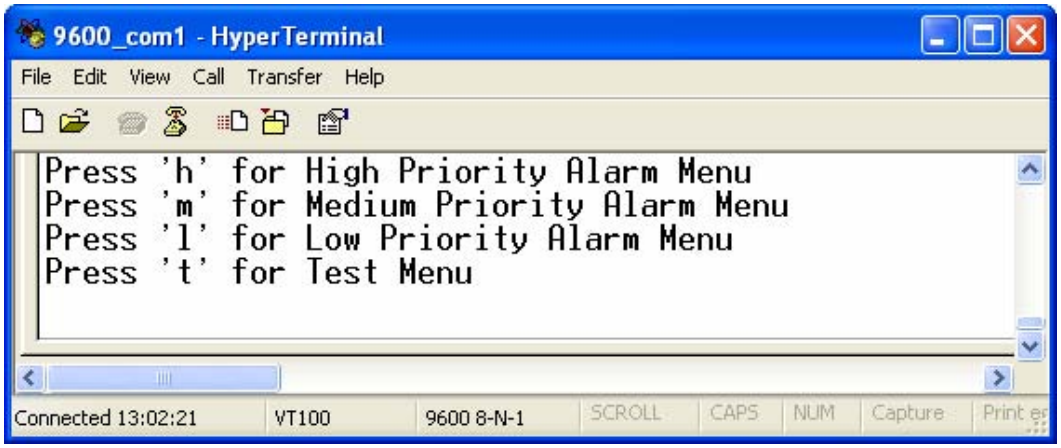

Fig 2. Reset menu

If we press ‘h’, we get the menu shown in Fig 3.

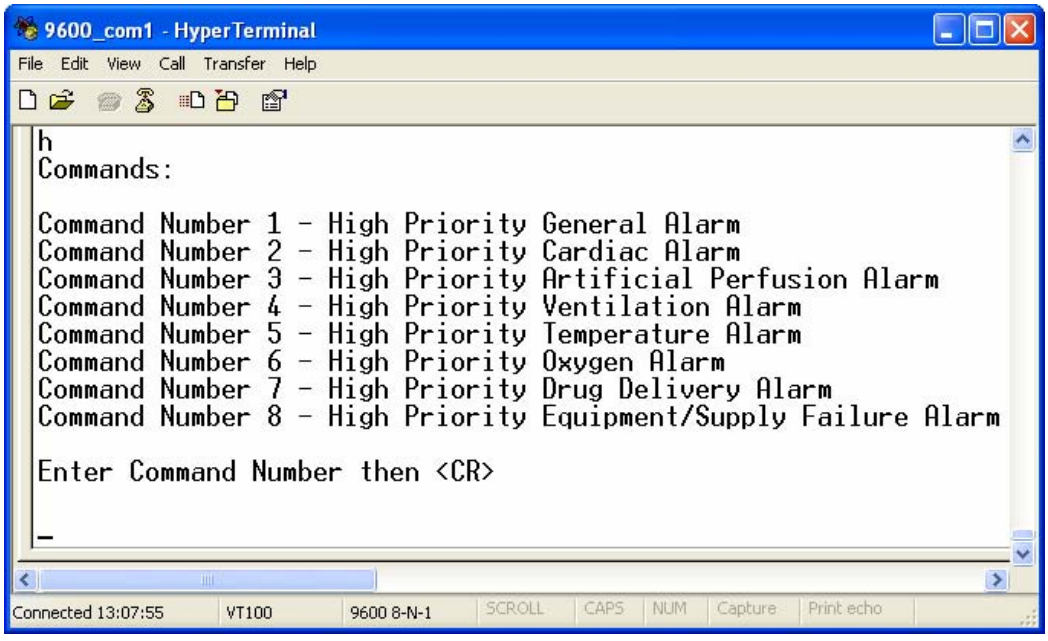

Fig 3. High priority alarm menu

Pressing any number between 1 and 8, and, pressing enter, will start the high priority alarm sequence for that corresponding alarm type. Similar menu actions are taken for the Medium Priority, low Priority, and Test menus as shown in Fig 4, Fig 5, and Fig 6.

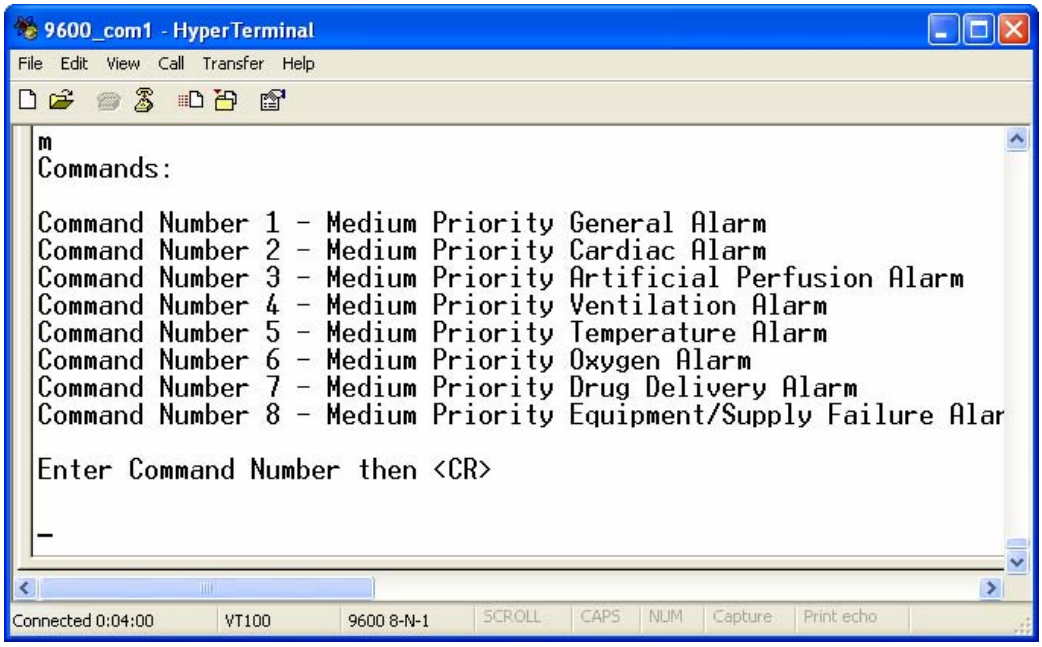

Fig 4. Medium priority menu

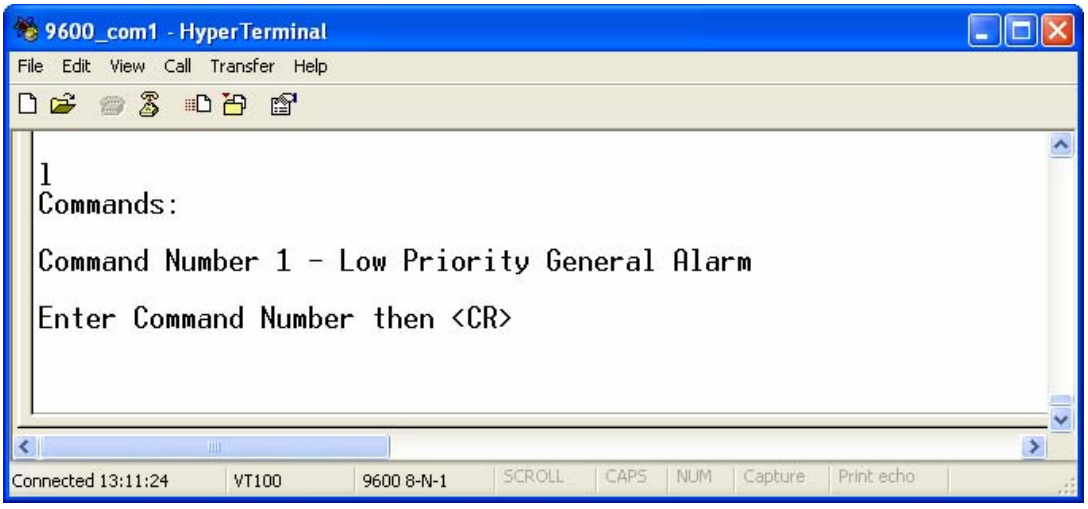

Fig 5. Low priority menu

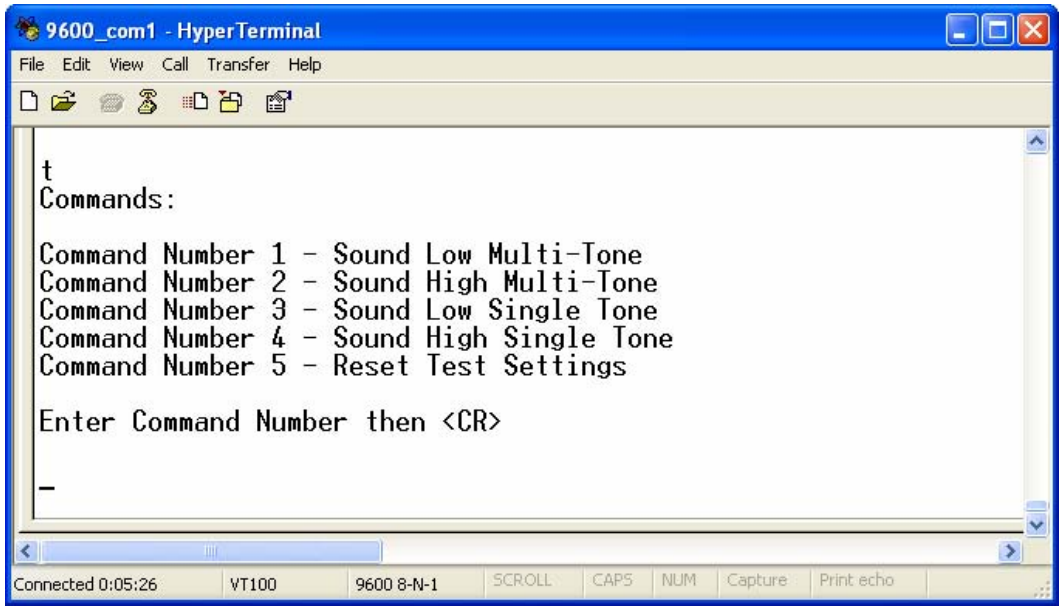

Fig 6. Test menu

The commands in the test menu provide short 1 second bursts of the highest and lowest frequency tones with harmonics as well as the highest and lowest single frequency tones needed for this application. This is to facilitate making the performance tests. The single tones are generated by making all the values in the 'ToneWeights' array = 0, except the isolated tone being generated. Command 5 resets all the ToneWeights array values to their initial values.

5. External hardware requirements

5.1 External DAC filter

A simple 3-pole RC filter was added to filter the 25 kHz sample rate component from the signal for the tests done in this applications note. The filter used is shown in Fig 7 and provides a cutoff frequency in the range of 9 kHz to10 kHz. Since the sample rate is above the normal audio hearing range, cost sensitive applications may be able to get by without a filter. Also, a coupling capacitor is generally required after this network, as the DC level of the DAC is nominally 1.65 V.

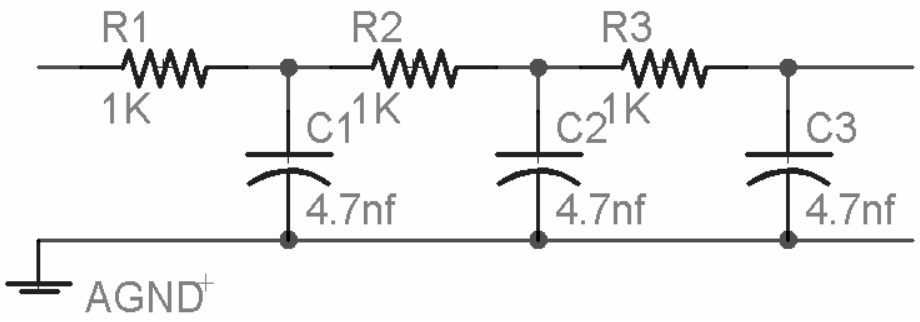

Fig 7. Three pole RC DAC filter

If this is not going to a high impedance input, or if this signal is fed externally, a buffer is recommended.

6. Performance analysis

Using the output filter described in section 5.1, spectrum analysis tests were performed to verify the spectral content of the signal and estimate signal to noise ratios. In addition Oscilloscope captures were made to show the rise fall and other temporal characteristics of the alert tones generated. The output of the MCB1700 DAC was fed to a Mackie CR1604-VLZ Audio Mixer (EQ set for flat frequency response) to provide the correct levels to the internal Sigmatel Codec in a Dell laptop. The laptop was running True Audio's TrueRTA Spectrum Analyzer Software package. The Test Menu, detailed earlier, was used to provide the test tone bursts used in the analysis.

6.1 Spectral analysis of multiple tone generation with DAC filter

The following Spectrum Analyzer captures are made with the board output taken after a three pole, 10 kHz RC filter detailed in section 5.1.

The spectrum of the lowest frequency tone in the application (C4) with harmonics is shown below in Fig 8. As you can see, the harmonics are very close in amplitude to each other. This will easily meet the IEC 60601-1-8 specifications as they only require the harmonics to be within 15dB of each other. The signal to noise ratio looks to be in excess of 55dB providing very good noise performance also. The lack of other harmonics showing up in the spectral analysis also demonstrates the low distortion of the sine waves generated by the Goertzel algorithm.

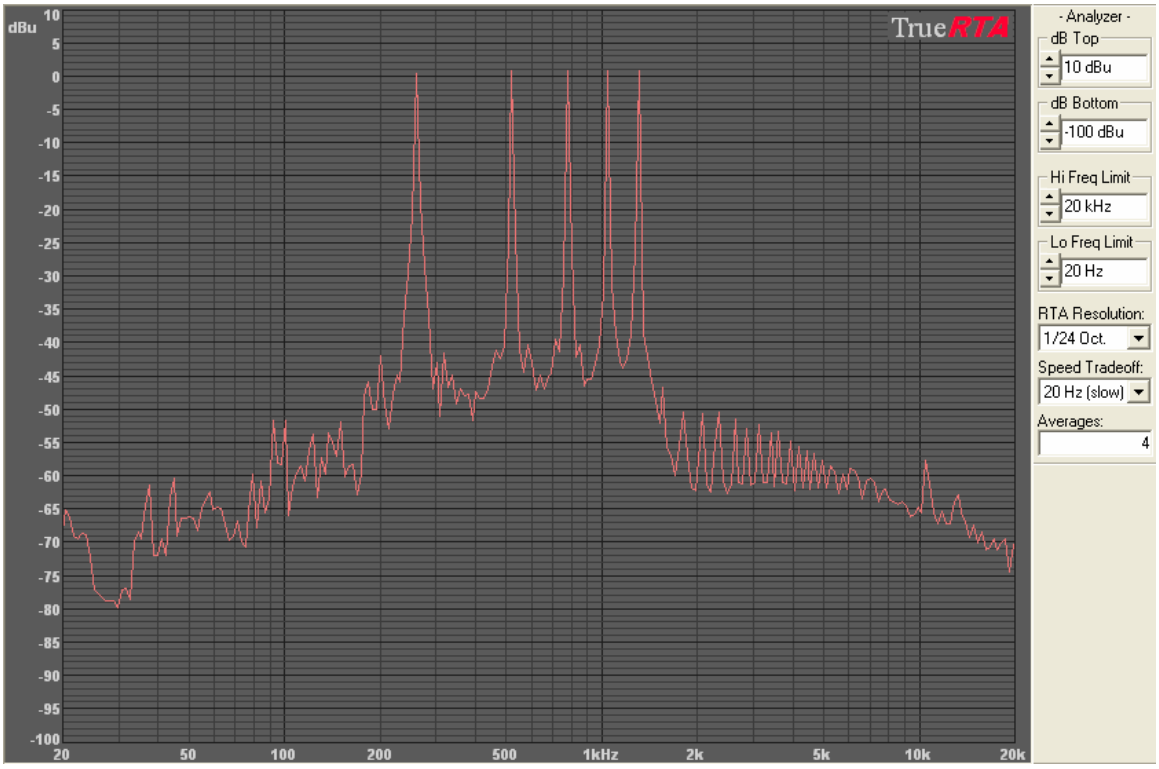

Fig 8. C4 with harmonics

Fig 9 shows the spectral analysis of the highest frequency tone in the application (C5) with harmonics.

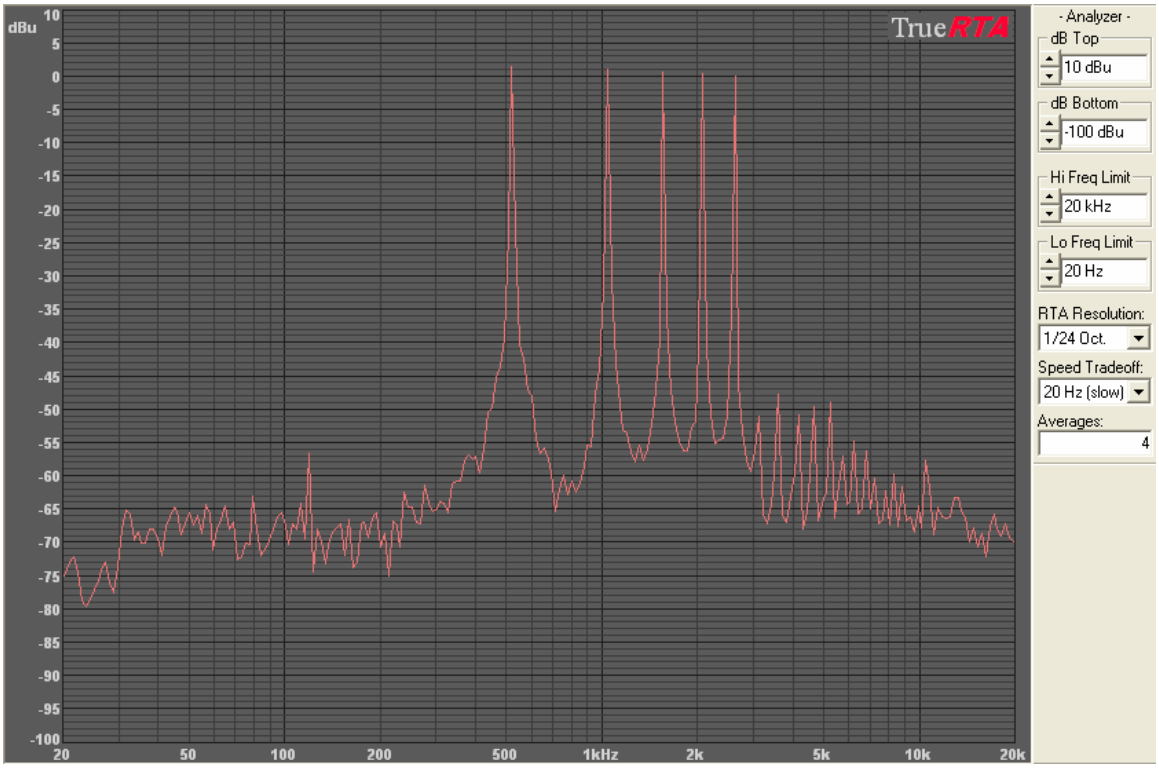

Fig 9. C5 with harmonics

As you can see, the spectrum of the C5 tone with harmonics also has good signal to noise ratios and low distortion while keeping the amplitude of the fundamental and harmonics within a couple of dB.

6.2 Envelope timing

The IEC specifications state the rise time and fall time requirements of the notes generated is a function of the duration of the note (rise time = 10% to 20% of Td). Since the duration of a medium priority note is longer than a high priority note, the rise time should be faster on the high priority notes that are generated. The spec allows more flexibility for fall times, so the fall time for this demo application is chosen to be the same as the rise time. Fig 10 shows an oscilloscope capture of the rising edge of a high priority note. The falling edge is shown in Fig 11. The rise and fall time for the high priority notes in this example is around 20 milliseconds. A capture of the medium priority rise time is shown in Fig 12 and the fall time is shown in Fig 13. The rise and fall time of the medium priority notes is around 30 ms. These can be easily adjusted by changing a code constant if needed.

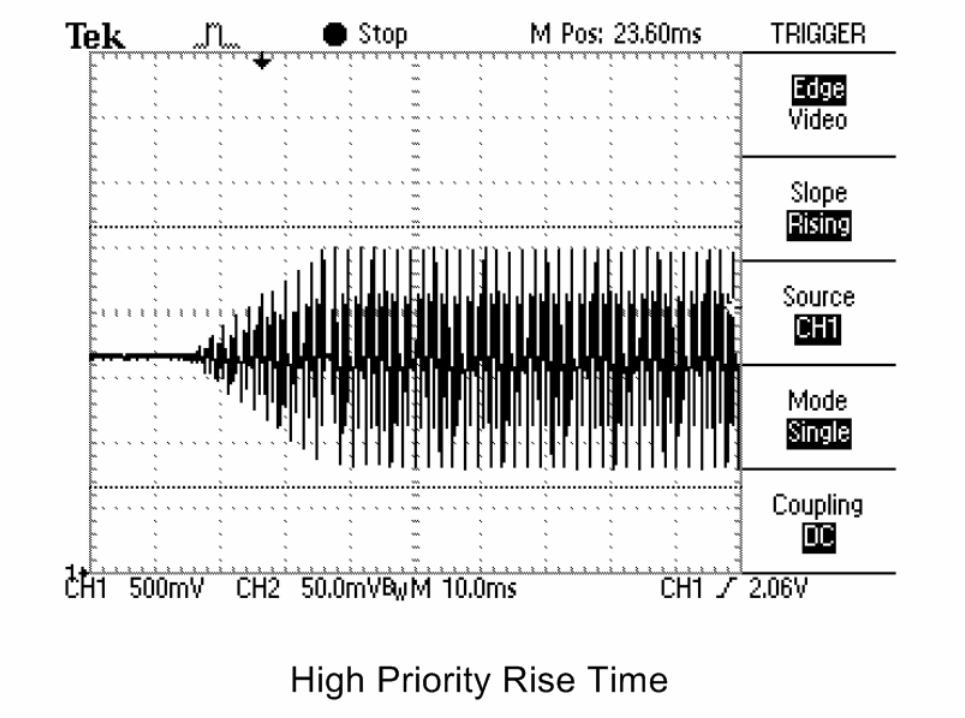

Fig 10. High priority rise time

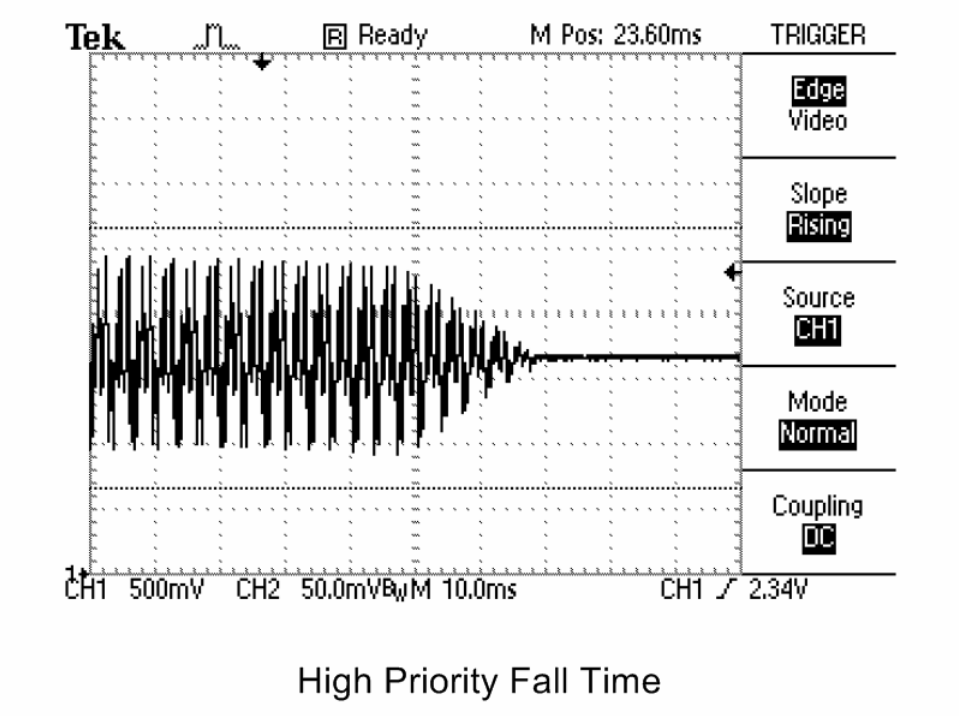

Fig 11. High priority fall time

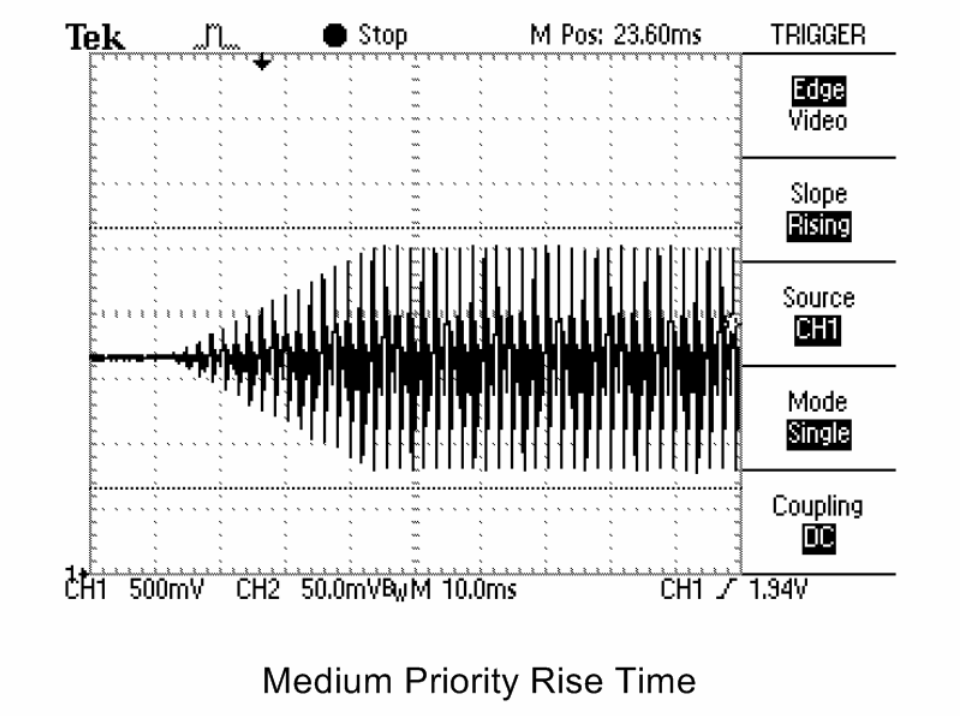

Fig 12. Medium priority rise time

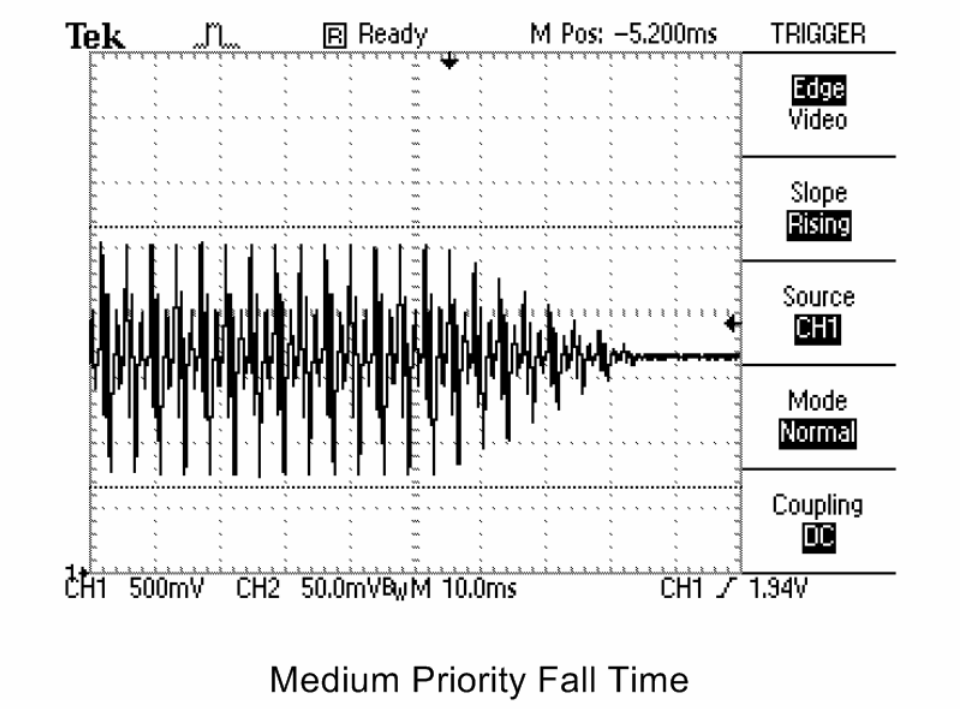

Fig 13. Medium priority fall time

6.3 Sequence timing

Fig 14 shows the timing of a High Priority Sequence.

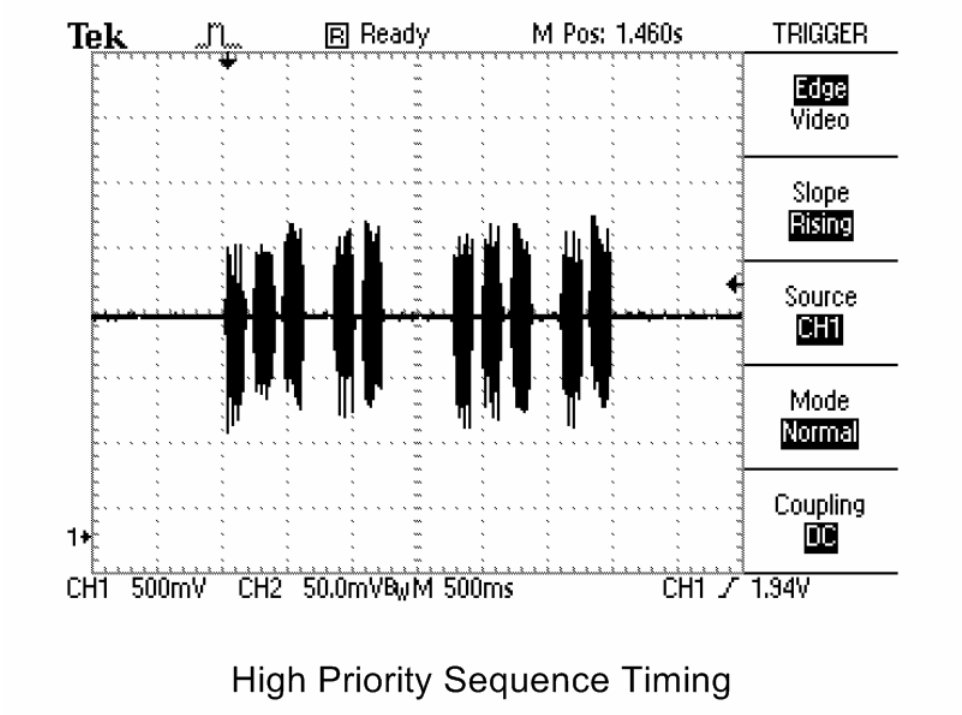

Fig 14. High priority sequence timing

We can see that the high priority sequence has a little more delay between the 3rd and 4th notes of the sequence, and an even longer delay between the repeating 5 note sequence. This is per the IEC60601-1-8 temporal specifications. Fig 15 shows the medium priority sequence timing.

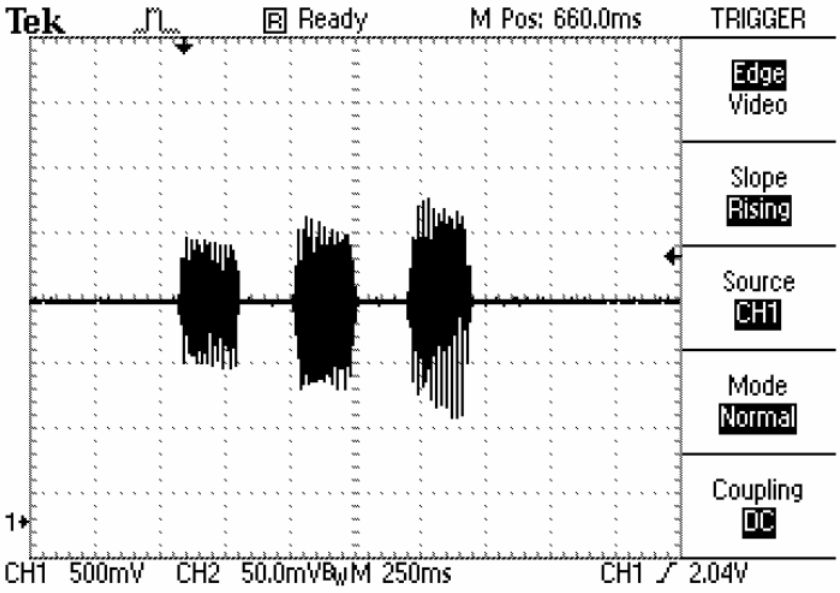

Fig 15. Medium priority sequence timing

We can see that the medium priority sequence only has three notes and the spacing between the notes is greater than the spacing of the first three notes in the high priority sequence. This is also in compliance with the IEC60601-1-8 specification.

7. Conclusions

The method of generating the medical alarms presented in this application note provides an efficient, low cost, high performance method of generating audible medical alerts that comply with the IEC60601-1-8 standard. The specification also states that subtle degrees of equipment differentiation in terms of alarm sounds can be advantageous to the operator. In addition to meeting the requirements of the standard, the firmware implementation provided here allows easy customization of the tones while still staying within the specification parameters. The NXP LPC17xx family of ARM Cortex-M3 microcontrollers provides very high speed performance and deterministic timing that is ideal for implementing algorithms like the one used in this example. With the LPC1768 processor running at 96 MHz, this application uses around 8 % of the available processor bandwidth and less than 10K of code space. This leaves ample code space and processing power for additional applications. As a result, it is now very easy to add support for the IEC60601-1-8 standard to any medical electronic application.

8. Legal information

8.1 Definitions

Draft — The document is a draft version only. The content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included herein and shall have no liability for the consequences of use of such information.

8.2 Disclaimers

General — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in medical, military, aircraft, space or life support equipment, nor in applications where failure or malfunction of a NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors accepts no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is for the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from national authorities.

8.3 Trademarks

Notice: All referenced brands, product names, service names and trademarks are property of their respective owners.

9. Contents

For more information, please visit: http://www.nxp.com
For sales office addresses, email to: salesaddresses@nxp.com

Date of release: 1 October 2009
Document identifier: AN10875_1