Embedded Systems Final Project

Using the C6713 DSK to Generate and Analyze Music
Written, Coded & Researched by Andrew Ippoliti
For ECEG-721, Embedded Systems, Taught by Evi Voudouri at Manhattan College
December 2011

Abstract

This project will explore the ability of the Texas Instruments C6713 DSK to generate and analyze music. It will utilize the C6713 DSK to generate sinusoids. It will also use the C6713 DSK to determine which musical note most accurately represents a monophonic signal. It will also attempt to construct a musical representation of a polyphonic signal in real time.

Requirements

The embedded system must be able to generate different tones. The code must run on the C6713 DSK platform created by Texas Instruments. The system will be able to detect a list of frequencies corresponding to specific music notes. It will only detect pure tones; it will not anticipate the different characteristics of specific instruments. The system will either output the results to the onboard LEDs or to a host computer via a USB cable.

Contents

Generating Tones

The DSK can easily generate sinusoids using the standard *sin* and *cos* functions. Those sinusoids can be outputted to the "LINEOUT" or "HEADPHONE" plugs on the DSK board. There are some files which must be included in the main source file to allow that to happen as shown in Listing - Tone 1 Required "Include" Files.

```
#include <math.h>           /* required for cos function*/
#include "tonecfg.h"        /* auto-generated by CCStudio */
#include "dsk6713.h"        /* board support library */
#include "dsk6713_aic23.h" /* required for using the codec for audio in/output */
#include "notes.h"          /* pre-computed musical note frequencies */
```

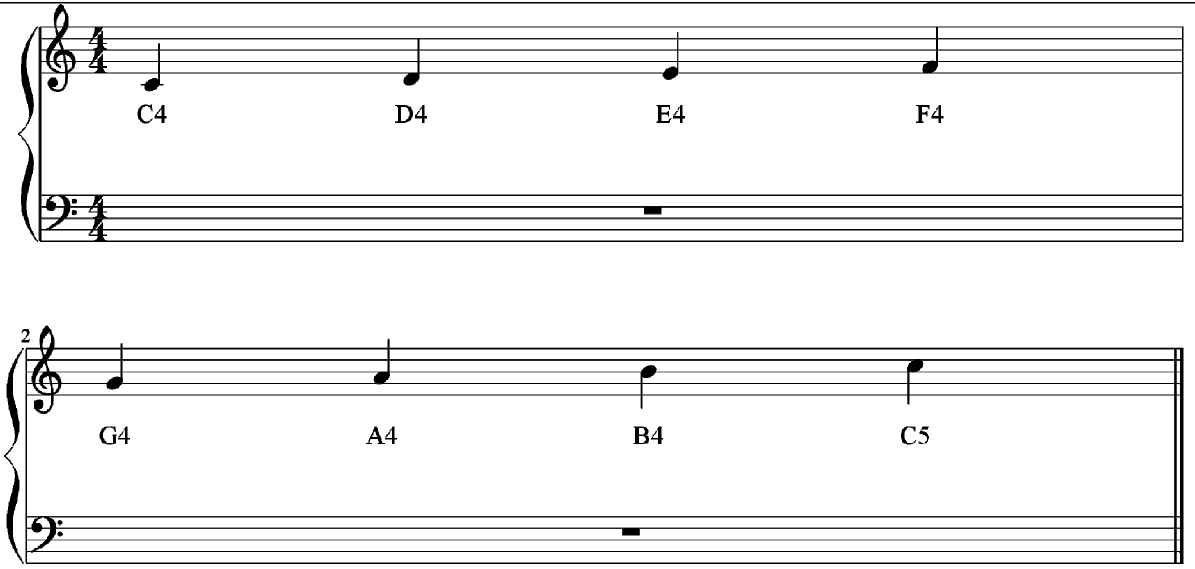
Listing - Tone 1 Required "Include" Files

The first four files are included with the DSK; however, the last file, "notes.h" is not. Music notes are sinusoids of specific frequencies as shown in Equation 1 - Musical Note Frequencies. The equation reads as follows: the frequency f_n of a note n half-steps away from a reference note of frequency f_0 is f_0 times the a to the n. The symbol a is the twelfth root of 2 or $2^{1/12}$. Musicians typically select A4 with frequency 440Hz as the reference note (1). That equation was used to generate all of the frequencies in "notes.h" as *#define noteName noteFrequency*.

$$f_n = f_0 \cdot a^n = f_0 \cdot 2^{n/12}$$

Equation 1 - Musical Note Frequencies

Table 1, below, shows different notes placed on a musical scale and their respective frequencies.



Note	Frequency (Hz)	Note	Frequency (Hz)
C4	261.63	D4	293.66
E4	329.63	F4	349.23
G4	392.00	A4	440.00
B4	493.88	C5	523.25

Table 1 - Note Frequencies near Middle C

To make it easier to output sinusoids of varying frequencies, a function called *cosOut* was created to handle the generation and output of the data. The code for *cosOut* is shown in Listing - Tone 2 Definition of cosOut Function.

```
/* We will be setting the coded to run at 8kHz which gives a
 * sample period of 1/8000 = 0.000125s */
#define SAMPLE_PERIOD 0.000125
#define PI 3.14159265358979323846264338327950

/* The cosOut outputs a cosine of frequency "freq" for "duration" seconds
 * to the codec with handle "hCodec"
 * This is basically a utility function that busy-waits while generating
 * and outputting a cosine wave */
void cosOut(double freq, double duration, DSK6713_AIC23_CodecHandle hCodec){
       double i; int sample;
       for (i = 0; i < duration; i+=SAMPLE_PERIOD){
             sample = (int)(2048.0*cos(2*PI*freq*i));
        // Send a sample to the left channel
        while (!DSK6713_AIC23_write(hCodec, sample ));
        //Send a sample to the right channel
        while (!DSK6713_AIC23_write(hCodec, sample ));
    }
       return;
}
```

Listing - Tone 2 Definition of cosOut Function

The remaining code is used to setup the board and play several notes as shown in Listing - Tone 3 Main Function. The sequence of notes can also be represented as the musical score of Table 1.

```
/* Codec configuration settings */
DSK6713_AIC23_Config config = DSK6713_AIC23_DEFAULTCONFIG;

void main(){
    DSK6713_AIC23_CodecHandle hCodec;

    int i;

    /* Initialize the board support library, must be called first */
    DSK6713_init();

    /* Start the codec, set sample rate to 8kHz */
    hCodec = DSK6713_AIC23_openCodec(0, &config);
    DSK6713_AIC23_setFreq(hCodec, DSK6713_AIC23_FREQ_8KHZ);

       /* Output a bunch of notes 2 times */
    for(i=0;i<2;i++){
       cosOut(C4,0.5,hCodec);//DO
       cosOut(D4,0.5,hCodec);//RAY
       cosOut(E4,0.5,hCodec);//MI
       cosOut(F4,0.5,hCodec);//FA
       cosOut(G4,0.5,hCodec);//SO
       cosOut(A4,0.5,hCodec);//LA
       cosOut(B4,0.5,hCodec);//TI
       cosOut(C5,0.5,hCodec);//DOH!
    }
    /* Close the codec */
    DSK6713_AIC23_closeCodec(hCodec);
}
```

Listing - Tone 3 Main Function

Analyzing Tones via the Fourier Transform

The obvious way to analyze a sound for musical notes is to take the Fourier transform of the signal. Then compare the magnitude of the frequency bins and determine which note or notes correspond to the frequency bin with the highest magnitude. The discrete Fourier transform $X(k)$ of a signal $x(t)$ is given by Equation 2 - Discrete Fourier Transform.Equation 2 - Discrete Fourier Transform Note that N is the total number of samples and k is the index of the frequency bin (2).

$$X(k)=\sum_{n=0}^{N-1} x[n]e^{-j\frac{2\pi}{N}nk}$$

Equation 2 - Discrete Fourier Transform

A direct, un-optimized implementation of the Discrete Fourier Transform is given in Listing DFT 1 - Code for Computing Discrete Fourier Transform. However, there are several problems with the direct discrete Fourier transform (DDFT) for this particular application. Because this will be running in an embedded system: speed, scheduling and memory are important. The DDFT requires a rather large number of samples for good results. Additionally these samples should be "windowed" to get better results. This requires a large amount of processing to occur at once which can make scheduling difficult. Additionally it generates data that is not needed in this particular application. All of the samples must be processed at once which means that a relatively large amount of memory is required.

```c
#define PI 3.14159265358979323846264338327950
void fourier(unsigned short *x, double *Xcos, double *Xsin, int N, double sign){
        int k, n;
        double PIN2 = 2*PI/((double)N);
        double arg;
        for(k = 0; k<N; k+=1){
                Xcos[k]=0; Xsin[k]=0; arg=0;
                for(n = 0; n<N; n+=1){
                        //arg = 2*PI*n/N; //for speed:
                        arg += PIN2; //
                        Xcos[k] += ((double) x[n])*cos(arg);
                        Xsin[k] += ((double) x[n])*sign*sin(arg);
                }
        }
}
```

Listing DFT 1 - Code for Computing Discrete Fourier Transform

When the code shown in Listing DFT 1 - Code for Computing Discrete Fourier Transform was used to analyze sinusoids generated on the C6713 DSK, it failed to meet the required deadlines. A better implementation, for example the Fast Fourier Transform, could alleviate these problems; however, a different method of frequency analysis will be employed.

Frequency Analysis via the Goertzel Algorithm

Another method for frequency domain analysis is the Goertzel algorithm. This algorithm interprets the discrete Fourier transform as a convolution which can be written as a recursive difference equation (3). The realization of the difference equation is shown in Figure 1 - Goertzel Algorithm Realization.

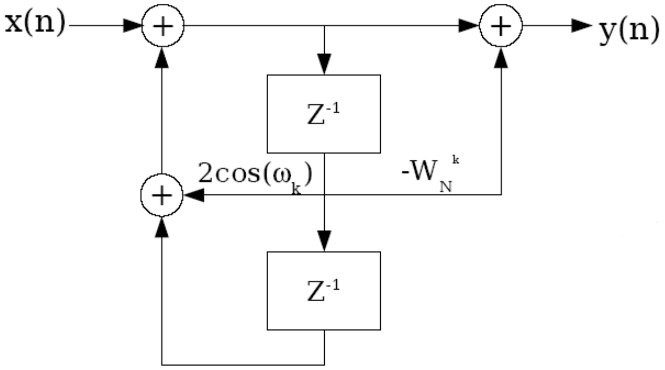


Figure 1 - Goertzel Algorithm Realization

As shown in Figure 1 - Goertzel Algorithm Realization, there only are 2 delay elements in the Goertzel algorithm realization which means that it does not require a lot of memory and can process data as it is ready. In embedded systems this offers a huge advantage over the fast Fourier transform (FFT) which requires a fair amount of memory and can only process data in blocks. When computing the values for many frequencies, the FFT is more computationally efficient than Goertzel's algorithm. However, we only need information about a few particular frequencies, those corresponding to specific musical notes. These conditions make the Goertzel algorithm ideal for this application.

There are two parameters which characterize Goertzel's algorithm: the sampling rate f_S and the block size N. The sampling rate was chosen to be 8 kHz because that should allow plenty of time for the board to perform necessary tasks and not inundate the processor with too much data to analyze. The ratio of the sampling rate to the block size determines the frequency resolution of the filter (4). A sampling rate of 8000 Hz and a block size of 2000 samples yields a resolution of 4 Hz. This means that the filter is more precise when given more samples to analyze; however, this requires more time; using the previous example, 2000 samples at 8000 samples per second requires 0.25 seconds of data.

To help determine the ideal parameters for this application a simulation was built and run using HTML and JavaScript. Listing Goertzel 1 shows my JavaScript implementation of the Goertzel algorithm for processing a block of data.

```
/* GoertzelBlock - Applies Goertzel Algorithm to a Block of Data
 * data is an array of [time,sample] pairs
 * testFreq is the frequency that is being tested for
 * sampleFreq is the rate at which the data was sampled
 * procLen is the number of samples that should be processed */
function GoertzelBlock(data, testFreq, sampleFreq, procLen) {
      /* Z1 is the previous value, Z2 is the previous previous value
       * S is the current value, i is the iteration */
      var Z1=0.0, Z2=0.0, S = 0.0, i = 0;
      /* Compute the normalized frequency that we are looking for */
      var nomFreq = testFreq / sampleFreq;
      /* realW is the real part of the complex exponential */
      var realW = 2*Math.cos(2*Math.PI*nomFreq);
      /* imagW is the imaginary part of the complex exponential,
       * Note: imagW is not needed for the magnitude computation */
      var imagW = 1*Math.sin(2*Math.PI*nomFreq);
      /* iterate over all samples */
      for (i=0; i<procLen; i++) {
            S = data[i][1] + realW * Z1 - Z2; //calculate current
            Z2 = Z1; //update previous previous
            Z1 = S;  //update previous
      }
      /* Return the power, this isn't normalized so it can be rather high */
      return Math.sqrt(Z2*Z2+Z1*Z1-realW*Z1*Z2);
}
```

Listing Goertzel 1 - JavaScript Implementation of the Goertzel Algorithm for Processing a Block of Data

The code was also modified to output an array of data to show how the output progresses as the number of samples increases. It also shows the algorithm’s progression for the note that is a half-step above the desired note as well as for the note that is a half-step below. Several simulations were run as shown on the following pages.

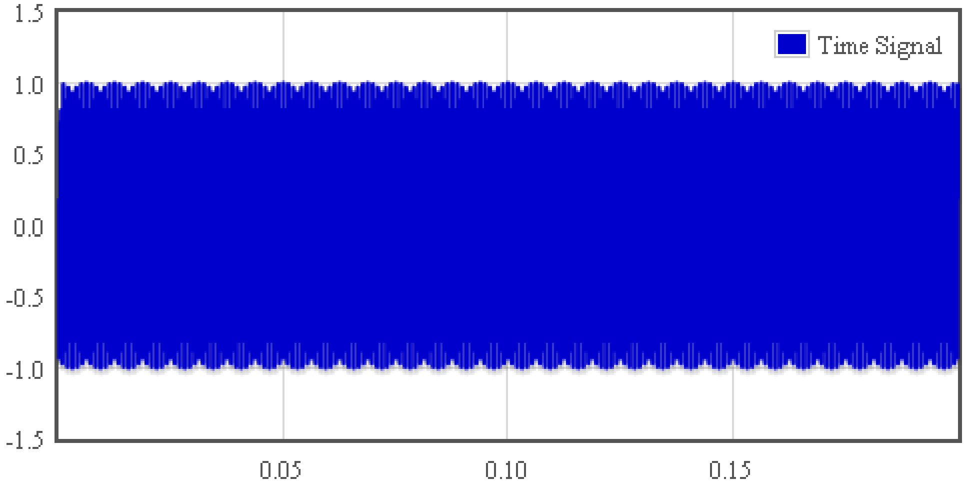
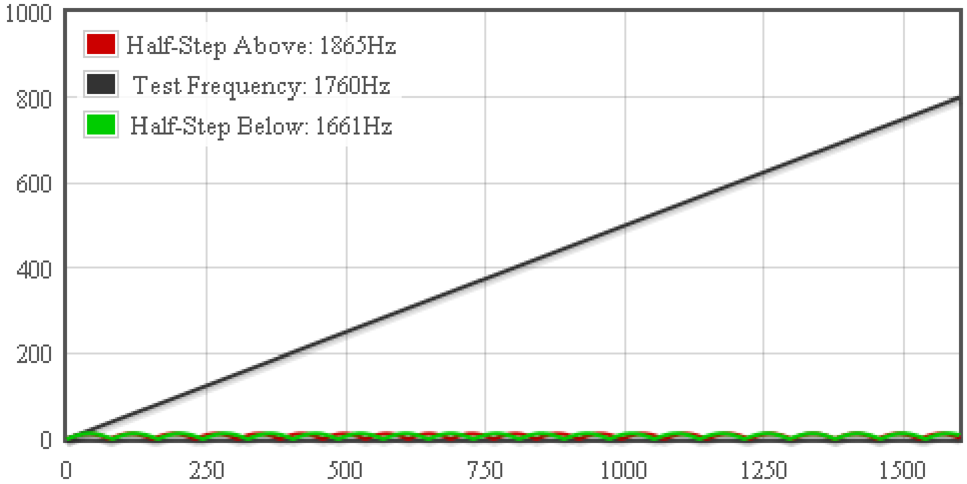
Simulation #1 – Detecting A6 in a Pure A6 Tone	
Sampling Frequency (Hz)	8000
Test Frequency (Hz)	1760
Number of Samples	1600
Input Signal: cos(2*PI*1760*t)	
	
Progression of the Algorithm	
	
Algorithm Output	799.9999999999943

Table 2 - Simulation of Detecting a 1760 Hz Component in a Pure 1760 Hz Cosine Wave

As shown in Table 2 the Goertzel algorithm is easily able to find a single 1760 Hz tone. The graph showing the progression of the algorithm suggests that even with as few as 250 samples it is possible to confidently say if the 1760 Hz tone is or is not present. A 1760 Hz sinusoid represents the note A6, the closest note to A6 is G#6 which is a half step below at 1661 Hz. The difference in frequency is roughly 100 Hz. Recall that the frequency resolution of the algorithm is ratio of the sampling rate to the block size: in this example, the sampling rate is 8000 Hz and the block size is 1600 yielding a resolution of 5 Hz. Since the nearest note to A6 is 100 Hz away the block size could be reduced to further speed up the processing.

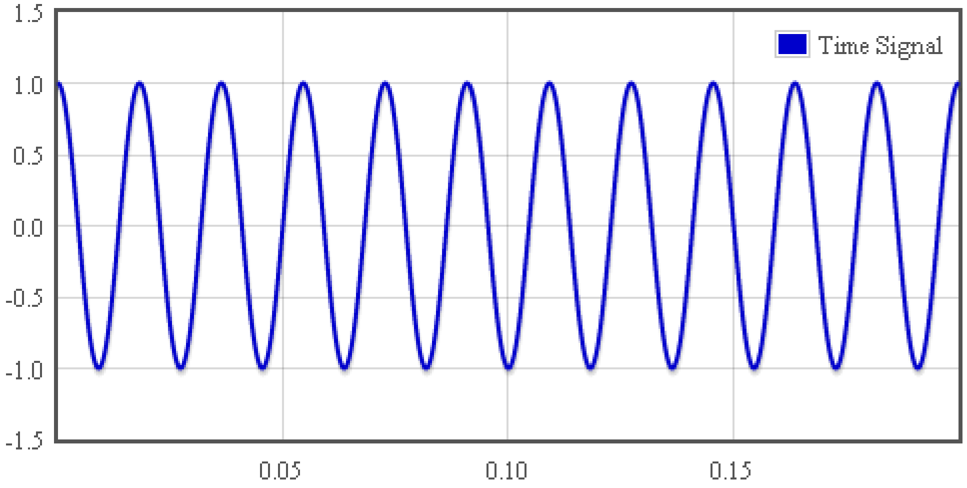
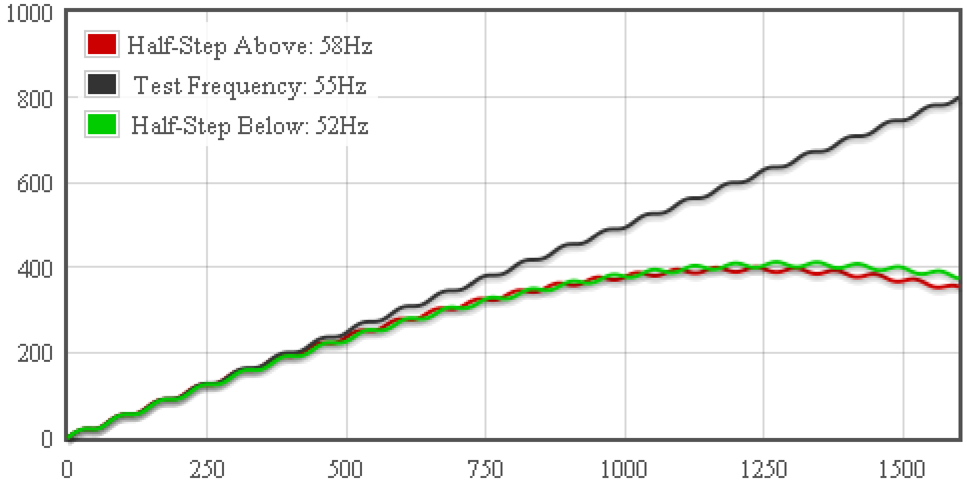
<table>
<tr><th colspan="2">Simulation #2 – Detecting A1 in a Pure A1 Tone</th></tr>
<tr><td>Sampling Frequency (Hz)</td><td>8000</td></tr>
<tr><td>Test Frequency (Hz)</td><td>55</td></tr>
<tr><td>Number of Samples</td><td>1600</td></tr>
<tr><td colspan="2">Input Signal: cos(2*PI*55*t)</td></tr>
<tr><td colspan="2">
</td></tr>
<tr><td colspan="2">Progression of the Algorithm</td></tr>
<tr><td colspan="2">
</td></tr>
<tr><td>Algorithm Output</td><td>799.9999999999801</td></tr>
</table>

Table 3 - Simulation of Detecting a 55 Hz Component in a Pure 55 Hz Cosine Wave

A6 is on the high frequency side of the musical scale; A1 is on the low frequency side of the musical scale. Table 3 shows that more time is required to determine that the signal is a 55 Hz signal and not a 58 Hz or 52 Hz signal. 55 Hz corresponds to A1, 52 Hz corresponds to G#1 and 58 Hz corresponds to A#1. Since the frequency resolution of the algorithm is ratio of the sampling rate to the block size, as the algorithm progresses the block size increases allowing smaller differences between frequencies to be noticed. Because lower frequency notes are closer together they require more processing time than higher frequency notes as shown in Table 2 and Table 3. An additional item of interest: it appears that when the particular frequency is present the output of the algorithm is equal to one half of the block size.

Detecting Notes with the C6713 DSK

As shown in the previous section, the Goertzel Algorithm will be employed to detect specific frequencies on the C6713 DSK. There are 4 user-controllable LEDs on the board as well which will be used to output which note has been detected. To make the code simpler, only the notes that were generated in the previous project will be considered: C4, D4, E4, F4, G4, A4, B4, and C5. The sound that was generated in that project will also be used as the input which will be analyzed. The following shows the header files that need to be included for the project to work.

```
#include <math.h>          /* required for cos function*/
#include "tonecfg.h"       /* auto-generated by CCStudio */
#include "dsk6713.h"       /* board support library */
#include "dsk6713_aic23.h" /* required for codec audio in/output */
#include "dsk6713_led.h"   /* required for working with the LEDs */
#include "notes.h"         /* pre-computed musical note frequencies */
```

Listing Simple DSK Goertzel 1 - Required Header Files

There are many variables which need to be set-up in order for the program to work as shown in Listing Simple DSK Goertzel 2 - Variable Declarations.

```
/* We will be setting the coded to run at 8kHz which gives a
 * sample period of 1/8000 = 0.000125s */
#define SAMPLE_PERIOD 0.000125
#define PI 3.14159265358979323846264338327950

/* Codec configuration settings */
DSK6713_AIC23_Config config = DSK6713_AIC23_DEFAULTCONFIG;

/* Setup the variables which will be used for Goertzel analysis */
#define NOTE_COUNT 8    //we will work only 8 notes
#define BLOCK_SIZE 1600 //samples to process before output & reset
double Z1[NOTE_COUNT];  //the 1st delay element for each note
double Z2[NOTE_COUNT];  //the 2nd delay element for each note
double  S[NOTE_COUNT];  //the current value for each note
double RF[NOTE_COUNT];  //the actual frequency for each note
double NF[NOTE_COUNT];  //the normalized frequency for each note
double RW[NOTE_COUNT];  //2*cos(2*pi*normalizedFreq) for each note
double PO[NOTE_COUNT];  //power of each note
int i;                  //the iteration count
int f;                  //which note we're on
int maxNote;            //which note has the most power
double maxP;            //the max power measured
Uint32 sampleRead; //place to store value read from codec
double sampleDbl;  //used to convert value read from codec to double
```

Listing Simple DSK Goertzel 2 - Variable Declarations

The main function has two major sections: the initialization and the processing loop. The initialization shown in Listing Simple DSK Goertzel 3 - Board Setup Code prepares the codec for audio input and output and configures the LEDs. The initialization shown in Listing Simple DSK Goertzel 4 - Data Structure Initialization prepares the data structures that will be used in the note identification algorithm.

```
DSK6713_AIC23_CodecHandle hCodec;

/* Initialize the board support library, must be called first */
DSK6713_init();
/* Initialize the LED module of the BSL */
DSK6713_LED_init();

/* Start the codec, set sample rate to 8kHz */
hCodec = DSK6713_AIC23_openCodec(0, &config);
DSK6713_AIC23_setFreq(hCodec, DSK6713_AIC23_FREQ_8KHZ);
```

Listing Simple DSK Goertzel 3 - Board Setup Code

```
/* Initialize everything to the starting parameters */
RF[0]=C4;
RF[1]=D4;
RF[2]=E4;
RF[3]=F4;
RF[4]=G4;
RF[5]=A4;
RF[6]=B4;
RF[7]=C5;
for(f=0; f<NOTE_COUNT; f++){
      Z1[f]=0;
      Z2[f]=0;
      NF[f]=RF[f]*SAMPLE_PERIOD;
      RW[f]=2*cos(2*PI*NF[f]);
}
```

Listing Simple DSK Goertzel 4 - Data Structure Initialization

The processing loop can be described by the following diagram, Figure 2 - Main Processing Loop. The source code for the processing loop can be found on the following page.

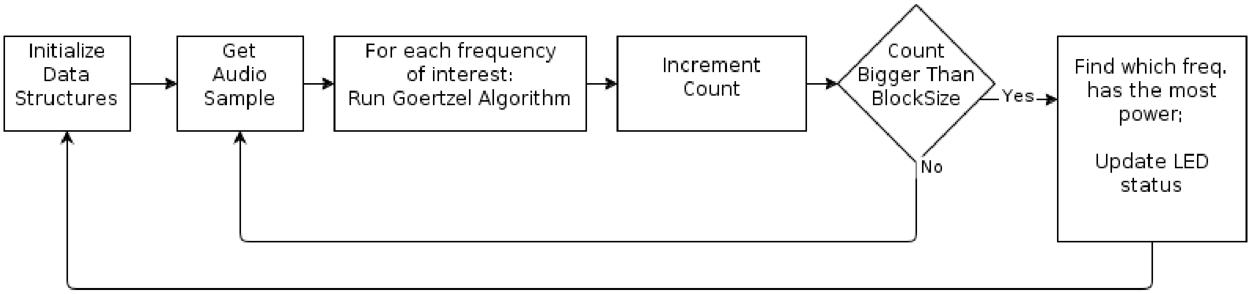

Initialize Data Structures
Get Audio Sample
For each frequency of interest: Run Goertzel Algorithm
Increment Count
Count Bigger Than BlockSize
Yes
No
Find which freq. has the most power;
Update LED status

Figure 2 - Main Processing Loop

```
/* Run Goertzel Algorithm */
for(i=0; i<BLOCK_SIZE; i++){
      //read one channel
      while(!DSK6713_AIC23_read(hCodec, &sampleRead)){}
      sampleDbl = ((double)sampleRead)/16383.0;
      for(f=0; f<NOTE_COUNT; f++){
            S[f]  = sampleDbl + RW[f]*Z1[f] - Z2[f];
            Z2[f] = Z1[f];
            Z1[f] = S[f];
      }
      //read other channel & ignore it
      while(!DSK6713_AIC23_read(hCodec, &sampleRead)){}
}
//Compute the power & which note has the most
//set maxP to 40000.0 because when there is no input signal, the PO[f]
//has values that max around 40000
maxP = 40000.0;
maxNote = -1;
for(f=0; f<NOTE_COUNT; f++){
      //calculate power, dont bother taking the square root
      PO[f] = Z2[f]*Z2[f] + Z1[f]*Z1[f] - RW[f]*Z1[f]*Z2[f];
      if(PO[f] > maxP){
            maxP = PO[f];
            maxNote = f;
      }
      //reset
      Z1[f]=0.0;
      Z2[f]=0.0;
}
//turn off LEDs
DSK6713_LED_off(0);
DSK6713_LED_off(1);
DSK6713_LED_off(2);
DSK6713_LED_off(3);
switch(maxNote){
      case 0:  DSK6713_LED_on(0); break;
      case 1:  DSK6713_LED_on(1); break;
      case 2:  DSK6713_LED_on(0); DSK6713_LED_on(1); break;
      case 3:  DSK6713_LED_on(2); break;
      case 4:  DSK6713_LED_on(0); DSK6713_LED_on(2); break;
      case 5:  DSK6713_LED_on(1); DSK6713_LED_on(2); break;
      case 6:  DSK6713_LED_on(0); DSK6713_LED_on(1);
               DSK6713_LED_on(2); break;
      case 7:  DSK6713_LED_on(3); break;
      default: break;
}
```

Listing Simple DSK Goertzel 5 - Main Processing Loop Code

Table 4 lists the notes that can be detected and the corresponding LED output that will be produced.

Note	LEDs	LED 0	LED 1	LED 2	LED 3
C4	🟩 ░ ░ ░	ON	OFF	OFF	OFF
D4	░ 🟩 ░ ░	OFF	ON	OFF	OFF
E4	🟩 🟩 ░ ░	ON	ON	OFF	OFF
F4	░ ░ 🟩 ░	OFF	OFF	ON	OFF
G4	🟩 ░ 🟩 ░	ON	OFF	ON	OFF
A4	░ 🟩 🟩 ░	OFF	ON	ON	OFF
B4	🟩 🟩 🟩 ░	ON	ON	ON	OFF
C5	░ ░ ░ 🟩	OFF	OFF	OFF	ON
Unknown	░ ░ ░ ░	OFF	OFF	OFF	OFF

Table 4 - LED Status Based on Detected Note

In the "Generating Tones" project, the C6713 DSK was used to generate the following sequence of notes: C4, D4, E4, F4, G4, A4, B4, C5. Each note was played for 0.5 seconds and then the entire sequence was repeated. The output from the DSK was recorded; the recording was played and used as input to test the note detection program. The output is shown in Table 5 and agrees with the input signal.

Observation	LEDs	Detected Note	Note Sequence	Acceptable
Before	░ ░ ░ ░	-	Begin	Yes
1	🟩 ░ ░ ░	C4	C4	Yes
2	░ 🟩 ░ ░	D4	D4	Yes
3	🟩 🟩 ░ ░	E4	E4	Yes
4	░ ░ 🟩 ░	F4	F4	Yes
5	🟩 ░ 🟩 ░	G4	G4	Yes
6	░ 🟩 🟩 ░	A4	A4	Yes
7	🟩 🟩 🟩 ░	B4	B4	Yes
8	░ ░ ░ 🟩	C5	C5	Yes
9	🟩 ░ ░ ░	C4	C4	Yes
10	░ 🟩 ░ ░	D4	D4	Yes
11	🟩 🟩 ░ ░	E4	E4	Yes
12	░ ░ 🟩 ░	F4	F4	Yes
13	🟩 ░ 🟩 ░	G4	G4	Yes
14	░ 🟩 🟩 ░	A4	A4	Yes
15	🟩 🟩 🟩 ░	B4	B4	Yes
16	░ ░ ░ 🟩	C5	C5	Yes
After	░ ░ ░ ░	-	End	Yes
After	░ ░ ░ ░	-	Silence	Yes

Table 5 - Tone Detection Output and Input Comparison

Communicating with the Host Computer via RTDX

It is possible to use Real Time Data eXchange (RTDX) to send information from the target C6713 DSK to the host computer. Configuring and using RTDX is easy to accomplish on the target C6713 DSK. Only two additional header files are required as shown in Listing RTDX 1.

```
#include <rtdx.h>    /* For RTDX communication*/
#include "target.h" /* RTDX setup */
```

Listing RTDX 1 - Additional Header Includes

There is also a small amount of setup that needs to be done before the main function. The code shown in Listing RTDX 2 shows the declaration of an output channel called "ochan." The name "ochan" will be used in other RTDX functions in the target program and the host program when working with this particular channel.

```
/* Declare and initialize an output channel called "ochan" */
RTDX_CreateOutputChannel(ochan);
```

Listing RTDX 2 - Output Channel Declaration

Some initialization also needs to occur within the main function as shown in Listing RTDX 3. Note how "ochan" is used in the enable function.

```
/* RTDX Setup */
TARGET_INITIALIZE();
RTDX_enableOutput(&ochan);
```

Listing RTDX 3 - Output Channel Initialization within the Main Function

Once the RTDX has been enabled and the output channel has been setup it is possible to output data from the target to the host computer.

```
//write to RTDX
if ( !RTDX_write( &ochan, &data, sizeof(data) ) ) {
      //ERROR
}
```

Listing RTDX 4 - Sending Data from Target to Host

When RTDX is no longer needed Listing RTDX 5 shows how to clean it up.

```
/* Stop RTDX */
RTDX_disableOutput(&ochan);
```

Listing RTDX 5 - Disabling Output Channel

The RTDX code is useless if Code Composer Studio is not configured to allow RTDX. As shown in Figure 3 - Accessing the RTDX Configuration Control, to enable RTDX, click on “Tools” in the Code Composer Studio menu. Then in the “RTDX” submenu click “Configuration Control.” This will display the current RTDX setting panel, similar to Figure 4.

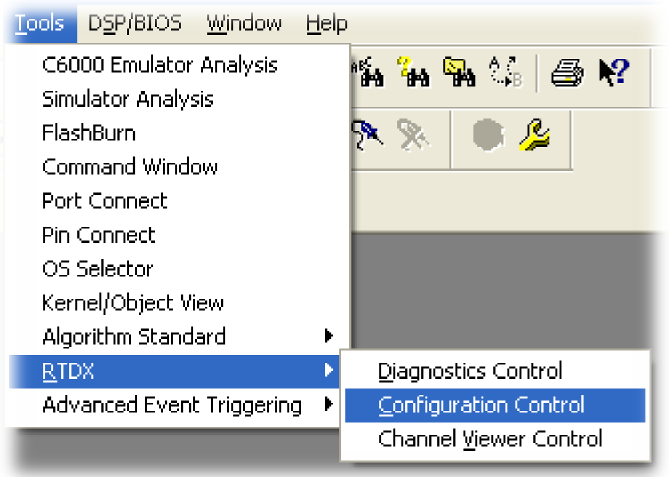
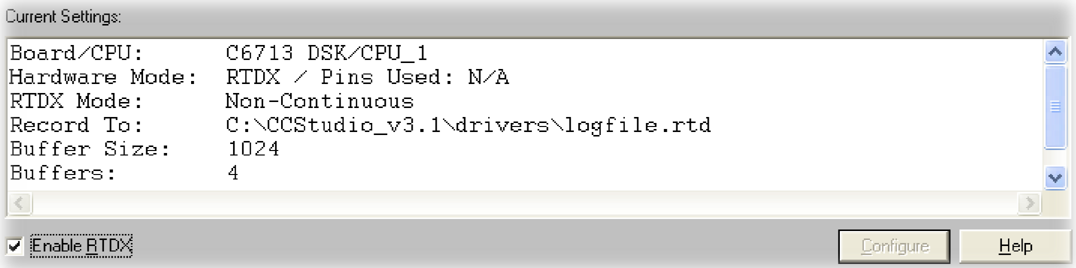


Figure 3 - Accessing the RTDX Configuration Control



Current Settings:

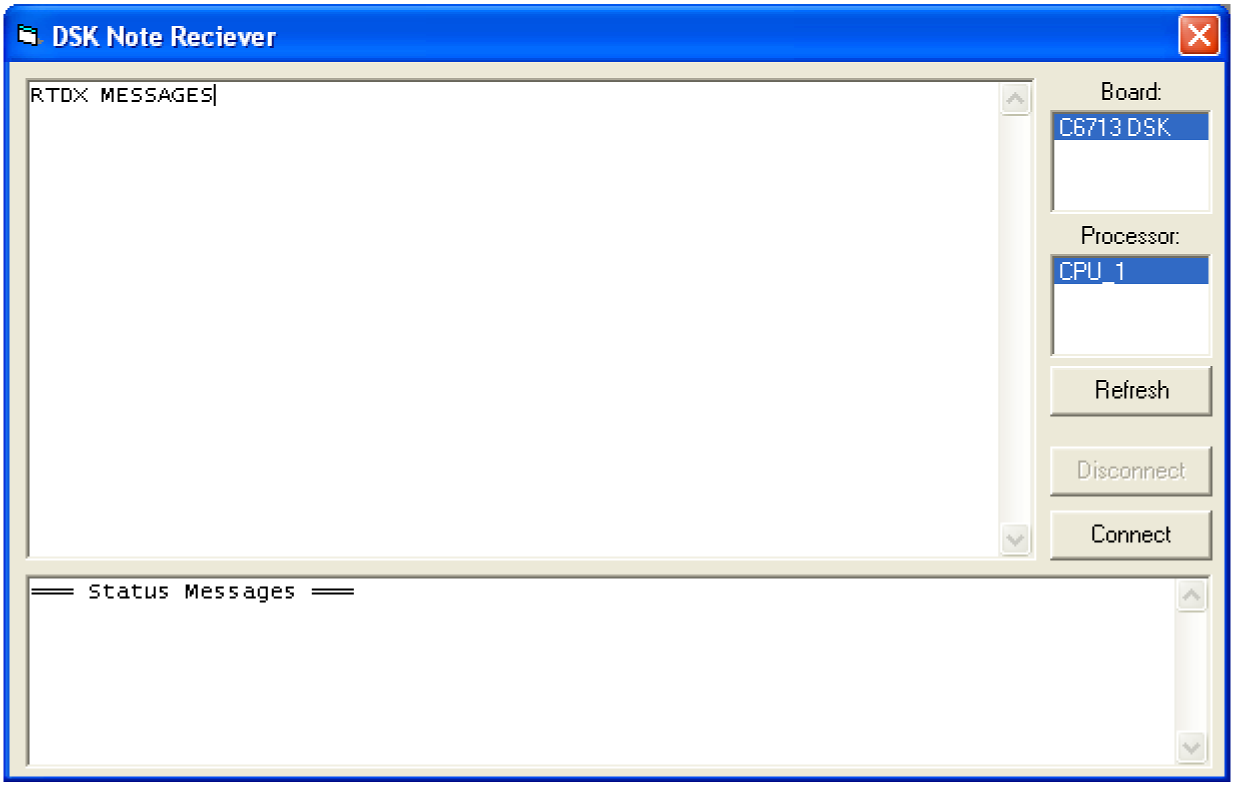
```
Board/CPU:        C6713 DSK/CPU_1
Hardware Mode:    RTDX / Pins Used: N/A
RTDX Mode:        Non-Continuous
Record To:        C:\CCStudio_v3.1\drivers\logfile.rtd
Buffer Size:      1024
Buffers:          4
```

☑ Enable RTDX

Figure 4 - RTDX Current Settings Panel

As shown in Figure 4, the “Enable RTDX” check-box must be checked for RTDX to work. Once RTDX has been enabled and the target setup it is possible to receive information with a host program. I decided to use Visual Basic as to create the host program.

A simple Visual Basic program was written to get data from the target. Figure 5 shows the interface of the host program. There are two list boxes on the right of the graphical user interface (GUI). These list boxes are used to select the board and processor which will be used. The “Refresh” button updates the list of boards and processors. Once a board and processor are selected the “Connect” button can be clicked; it will open an RTDX channel and will attempt to read the transmitted data. The transmitted data will be displayed in the large text area to the left of the “Connect” button. An addition text area, labeled “Status Messages” is used primarily for debugging purposes and will display any pertinent status information or error messages.



DSK Note Reciever

RTDX MESSAGES

Board: C6713 DSK

Processor: CPU_1

== Status Messages ==

Figure 5 - Interface for Program which will Receive RTDX Data

Several variables are used to make the program easier to work with as shown in Listing VB 1.

```
Option Explicit
'''''''''''''''''''''''''''''''''''''''''''''''Utility Variables
Public connected As Integer                  'if RTDX channel has been opened
Public lastMsg As String                     'for debugging purposes

'''''''''''''''''''''''''''''''''''''''''''''''Needed for RTDX/CCStudio setup
Private CCSetup As Object                    'Used To Access CCStudio
functions
Private Boards As Object                     'List of Available boards
Private Board As Object                      'Board
Private Processors As Object                 'List of available processors
Private Processor As Object                  'Processor
Public CurrentSelectedBoard As String        'The name of the board
Public CurrentSelectedProcessor As String    'Which processor on the board
Dim rtdx As Object                           'Acutal RTDX object
```

Listing VB 1 - Some Variables

Some constants are also defined to help decode RTDX status codes, refer to Listing VB 2.

```
''''''''''''''''''''''''''''''''''''''''''''''RTDX OLE API Status Return codes
Const SUCCESS = &H0                         'Method call successful
Const FAIL = &H80004005                     'Method call failure
Const ENoDataAvailable = &H8003001E         'No data is currently available
Const EEndOfLogFile = &H80030002            'End of log file
```

Listing VB 2 - RTDX Status Codes

Two utility functions were created to help present the user with data. One is used to translate integer values received through RTDX into meaning text and display it. The other is used to present status and error messages. The code is shown in Listing VB 3.

```
'Utility function to output debugging messages to lower text area
Private Sub LogMessage(message As String)
    If lastMsg <> message Then
        tbStatus.Text = tbStatus.Text & message & vbNewLine
        lastMsg = message
    End If
End Sub
'Utility function that takes the value read from the board, converts it into
'useful text and displays it to the user
'The program on the board was designed to send 0 through 8 to represent
'different notes and -1 when it could not make a decision
'This converts those integers to a more meaningful string
Private Sub ProcessData(ReadVal As Long)
    Dim NOTES() As Variant
    NOTES() = Array("DO", "RE", "MI", "FA", "SO", "LA", "TI", "DOH")
    If ReadVal > -1 Then
        outputArea.Text = outputArea.Text & NOTES(ReadVal) & ","
    End If
End Sub
```

Listing VB 3 - Utility Functions

The program requires a start-up function and a shut-down function to ensure proper initialization and safe exiting. These functions are shown in Listing VB 4.

```
'When the program starts
Private Sub Form_Load()
    connected = 0 'state that we don't have an RTDX channel open
    If (GetAvailableBoards) Then       ' Get Available Boards and Processors
        list_Boards.Selected(0) = True ' Set the selected board to 0
    End If
End Sub
'When the program ends
Private Sub Form_Unload(Cancel As Integer)
    If connected = 1 Then       'If we have the RTDX channel open
        DisconnectFromBoard     'Close the RTDX channel
    End If
    Set CCSetup = Nothing       'Cleanup
    Set Boards = Nothing        'Cleanup
    Set Board = Nothing         'Cleanup
    Set Processors = Nothing    'Cleanup
    Set Processor = Nothing     'Cleanup
End Sub
```

Listing VB 4 - Startup and Shutdown Functions

When the program runs it needs to be able to determine which boards it can connect to. The function shown in Listing VB 5 gets a list of available board from Code Composer Studio and updated the list of board on the GUI.

```
'Function to update the list of boards that can be connected to
Private Function GetAvailableBoards() As Boolean
    Dim status As Long
    Dim BoardName As String
    ' Initialize Lists
    list_Boards.Clear
    list_Processors.Clear
    ' Instantiate the Code Composer Setup SystemSetup coclass and obtain a
    ' pointer to the ISystemSetup interface
    Set CCSetup = CreateObject("CodeComposerSetup.SystemSetup")
    ' Get a pointer to the IBoards interface
    status = CCSetup.GetBoards(Boards)
    ' Loop through the available boards, get the names of the boards,
    ' and add the board names to the boards list control
    For Each Board In Boards
        ' Get the board name
        status = Board.GetName(BoardName)
        ' Append board name to the board list
        list_Boards.AddItem (BoardName)
    Next
    ' return True
    GetAvailableBoards = True
End Function
```

Listing VB 5 - GetAvailableBoards Function

After a board is selected the processor on the board must be selected. Listing VB 6 shows the function that updates the list of available processors.

```
'Function to update the list of processors which can be used
Private Function GetAvailableProcessors(SelectedBoardName As String) As
Boolean
    Dim status As Long
    Dim ProcessorName As String
    ' Get a pointer to the IBoard interface for the selected
    ' board
    status = CCSetup.GetBoardByName(SelectedBoardName, Board)
    ' Get a pointer to the IProcessors interface
    status = Board.GetProcessors(Processors)
    ' Loop through the available processors, get the names of the
    ' processors, and add the processors to the processors list
    ' control
    For Each Processor In Processors
        ' Get the processor name
        status = Processor.GetName(ProcessorName)
        ' Append processor name to the processor list
        list_Processors.AddItem (ProcessorName)
    Next
    ' Return True
    GetAvailableProcessors = True
End Function
```

Listing VB 6 - GetAvailableProcessors Function

Different events are triggered as the user interacts with the interface. Depending upon which GUI element is clicked, a specific function, shown in Listing VB 7, is run.

```
'When the "connect" button is clicked
Private Sub btConnect_Click()
    If connected = 1 Then       'If we have the RTDX channel open
        DisconnectFromBoard     'Close the RTDX channel
    End If
    ConnectToBoard              'Open the RTDX channel
    btConnect.Enabled = False   'Disable the "connect" button
    btDisconnect.Enabled = True 'Enable the "disconnect" button
End Sub

'When the "disconnect" button is clicked
Private Sub btDisconnect_Click()
    DisconnectFromBoard          'Close the RTDX channel
    btDisconnect.Enabled = False 'Disable the "disconnect" button
    btConnect.Enabled = True     'Enable the "connect" button
End Sub
Private Sub btListRefresh_Click()
    ' Get Available Boards and Processors
    If (GetAvailableBoards) Then
        ' Set the selected board to 0
        list_Boards.Selected(0) = True
    End If
End Sub

'When a board is selected
Private Sub list_Boards_Click()
    ' Clear processor list
    list_Processors.Clear
    ' Get current selected board
    CurrentSelectedBoard = list_Boards.List(list_Boards.ListIndex)
    ' Get available processors for that board
    If (GetAvailableProcessors(CurrentSelectedBoard)) Then
        ' Set the selected processor to 0
        list_Processors.Selected(0) = True
    End If
End Sub

'When a processor is selected
Private Sub list_Processors_Click()
    ' Get current selected processor
    CurrentSelectedProcessor =
list_Processors.List(list_Processors.ListIndex)
End Sub
```

Listing VB 7 - GUI Click Events

Once everything is configured, the RTDX channel can be opened. The heart of connecting to the C6713 DSK is shown in Listing VB 8.

```
Dim status As Long
Set rtdx = CreateObject("RTDX")
status = rtdx.SetProcessor(CurrentSelectedBoard, CurrentSelectedProcessor)
status = rtdx.Open("ochan", "R")
```

Listing VB 8 - RTDX Specific Connection Code

A more robust connection function, as used in my Visual Basic application, is presented in Listing VB 9.

```
'Function which tries to open the RTDX channel on the selected board/processor
Private Sub ConnectToBoard()
   LogMessage "Info  - Attempting to connect to board " & CurrentSelectedBoard
   LogMessage "Info  - Attempting to connect to processor " & CurrentSelectedProcessor
    Dim status As Long
    ' Get application objects
    Set rtdx = CreateObject("RTDX")

    status = rtdx.SetProcessor(CurrentSelectedBoard, CurrentSelectedProcessor)
    If (status <> SUCCESS) Then
        LogMessage "Error - Set Processor failed"
        Exit Sub
    End If

    'open target's input channel
    '"ochan" must agree with RTDX_CreateOutputChannel(ochan);
    'from target source code
    status = rtdx.Open("ochan", "R")
    Select Case status
    Case Is = SUCCESS
        connected = 1
        LogMessage "Info  - Opened RTDX channel for reading"
    Case Is = FAIL
        LogMessage "Error - Unable to open channel for RTDX communications"
        Exit Sub
    Case Else
        LogMessage "Info  - Unknown return value from openning RTDX channel"
        Exit Sub
    End Select
End Sub
```

Listing VB 9 - Function to Open RTDX Channel for Reading Data

A function to close the RTDX connection is shown in Listing VB 10.

```
'Function which tries to close the RTDX channel on the selected
board/processor
Private Sub DisconnectFromBoard()
    Dim status As Long
    ' close target's input channel
    status = rtdx.Close()
    Select Case status
        Case Is = SUCCESS
          LogMessage "Info  - Successfully closed RTDX channel"
        Case Is = FAIL
          LogMessage "Error - Unable to close RTDX channel"
        Case Else
          LogMessage "Info  - Unknown return value from closing RTDX channel"
    End Select
    connected = 0
    Set rtdx = Nothing ' kill RTDX OLE object
End Sub
```

Listing VB 10 - Function to Close RTDX Channel

Once the application has connected to the board and an RTDX channel has been established it is possible to read data that the DSK is sending. To read the data, a function is called every 30 milliseconds; it attempts to process the data. Listing VB 11 shows the source code of this function.

```
'Function to check for new RTDX data every XX milliseconds
Private Sub Timer1_Timer()
    Dim ReadValue As Long
    Dim status As Long
    If connected = 1 Then
        'Do
        status = rtdx.ReadI4(ReadValue)
        Select Case status
            Case Is = SUCCESS
                ProcessData ReadValue
            Case Is = FAIL
                LogMessage "Error - Reading data failed " & ReadValue
            Case Is = ENoDataAvailable
                LogMessage "Error - No data available"
            Case Is = EEndOfLogFile
                LogMessage "Info  - Reached end of log file"
            Case Else
                LogMessage "Error - Unknown error reading RTDX channel"
        End Select
        'Loop Until status = EEndOfLogFile
    End If
End Sub
```

Listing VB 11 - Timer to Check RTDX

The heart of Listing VB 11 is the *rtdx.ReadI4* function. It stores a “Long” data-type in ReadValue. Note that on the board it is called an “int” data-type.

To test then entire setup, the program was run on the target board and the Visual Basic application was run on the host computer. The target program on the board was run first. Then the Visual Basic program was run. In the Visual Basic program’s GUI the connect button was clicked. Then the same sound was sent to the board as was in the previous program. The output is shown in Listing VB 12. It agrees with the notes that were played.

```
DO,DO,DO,RE,RE,MI,MI,MI,FA,FA,SO,SO,SO,LA,LA,TI,TI,TI,DOH,DOH,DO,DO,DO,RE,RE,
MI,MI,MI,FA,FA,SO,SO,SO,LA,LA,TI,TI,TI,DOH,DOH,DOH,
```

Listing VB 12 – Output

Observe that the “quantity” of notes listed does not agree with the “duration” that the note is played. This occurs because the output from the Goertzel algorithm is sent 5 times per second but the each note is “played” 4 times per second. As shown in Listing - Tone 3 Main Function, each note lasts for half of a second or 0.5 seconds but the note detection algorithm identifies a note every 0.2 seconds.

It is interesting to note that CCStudio stores RTDX communications. Suppose, for example, you download the code to the board and run the program. The VB application will show all of the output from the time the program was run on the board not from the time that the VB application was started. When the program is reloaded onto the board, the RTDX communications are reset.

Extending the Possibilities – Detecting More Frequencies

There are more than the eight notes previously shown. The C6713 DSK will be set up to identify 96 different notes from C0 to B7. B7 was selected as the upper limit because B7 is 3951 Hz. Any note above B7 will cause aliasing because the board is set for a sampling rate of 8000 Hz. Because the sampling rate is 8000 Hz no signal greater than 4000 Hz can be successfully passed.

Before the board can identify those notes, it will generate all of those notes. Generating the notes was quite simple by building upon the sinusoidal signal generation of the first project. The output of all of the notes being played was recorded to a file.

The visual basic application which displays the board's output had to be updated to recognize 96 notes. Previously an array was used relate an integer from the board to a note definition; therefore, only the array had to be updated.

The program which detected 8 different notes was modified to detect 96 different notes. The only section that had to be changed was the initialization area. Instead of creating a array of 8 note frequencies, an array of 96 frequencies was created. The algorithm processing code remained the same.

Even though the code remained the same the run-time requirements have greatly increased. Previously, only 8 filters needed to be used; now, 96 are required. Besides the additional memory requirements there are additional computational requirements. Fortunately, the board is able to meet the deadline.

When all of the recorded noted were played for the board, it created the output shown in Table 6.

Read from top down then left to right							
C0	C1	C2	C3	C4	C5	C6	C7
C_0	C_1	C_2	C_3	C_4	C_5	C_6	C_7
D0	D1	D2	D3	D4	D5	D6	D7
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
E0	E1	E2	E3	E4	E5	E6	E7
F0	F1	F2	F3	F4	F5	F6	F7
F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7
G0	G1	G2	G3	G4	G5	G6	G7
G_0	G_1	G_2	G_3	G_4	G_5	G_6	G_7
A0	A1	A2	A3	A4	A5	A6	A7
A_0	A_1	A_2	A_3	A_4	A_5	A_6	A_7
B0	B1	B2	B3	B4	B5	B6	B7

Table 6 - Output Trying to Detect All Notes

The output shown in Table 6 agrees exactly with the generated input; therefore, the C6713 DSK can detect all of the notes correctly.

Extending the Possibilities – Detecting Different Instruments

The code was shown to successfully detect 96 different frequencies, perhaps it can detect specific notes played by specific instruments. The song shown in Figure 6 - Simple Melody was rendered using several different synthesized instruments. The recordings were then played for the C6713 DSK and the output observed.

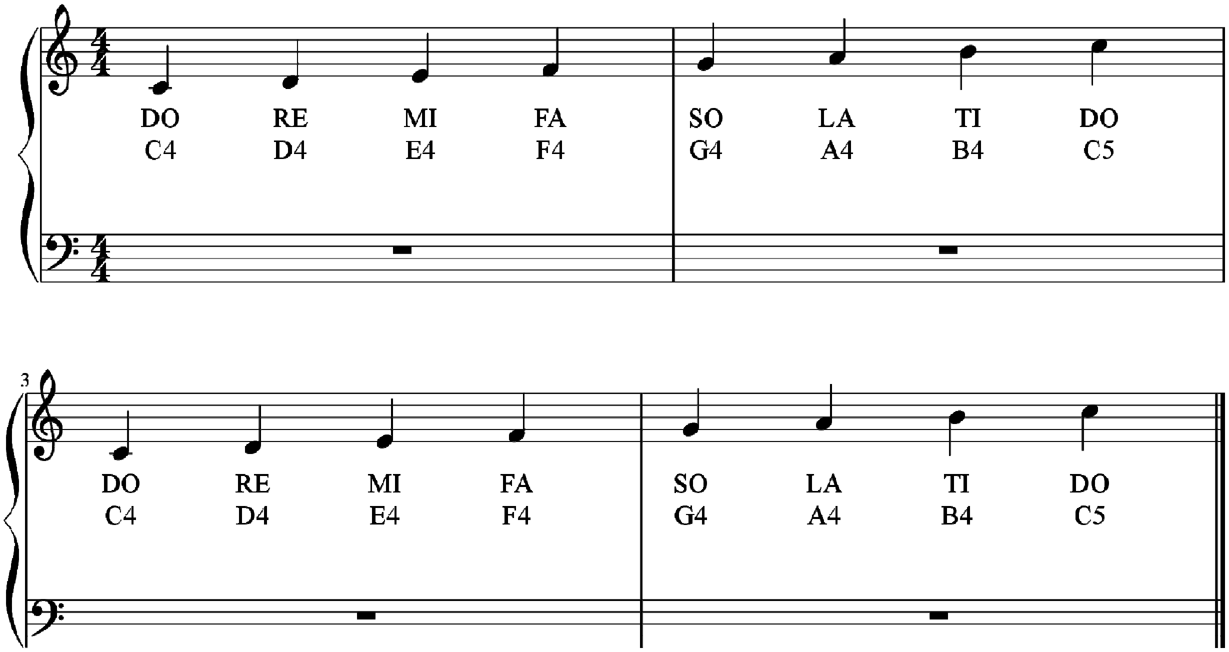


Figure 6 - Simple Melody

Pure Sinusoid				Piano				Guitar		
Actual	Measured	Diff		Actual	Measured	Diff		Actual	Measured	Diff
C4	C4	0		C4	C5	12		C4	F4	5
C4	C4	0		C4	C5	12		C4	F4	5
D4	D4	0		D4	D5	12		D4	G3	-7
D4	D4	0		D4	D5	12		D4	G3	-7
E4	E4	0		E4	E5	12		E4	A3	-7
E4	E4	0		E4	E5	12		E4	A3	-7
F4	F4	0		F4	F4	0		F4	A_3	-7
F4	F4	0		F4	F4	0		F4	A_3	-7
G4	G4	0		G4	G4	0		G4	C4	-7
G4	G4	0		G4	G4	0		G4	C4	-7
A4	A4	0		A4	A4	0		A4	D4	-7
A4	A4	0		A4	A4	0		A4	D4	-7
B4	B4	0		B4	B4	0		B4	E4	-7
B4	B4	0		B4	B4	0		B4	E4	-7
C5	C5	0		C5	C5	0		C5	F4	-7
C5	C5	0		C5	C5	0		C5	F4	-7
Percent Correct		100%		Percent Correct		63%		Percent Correct		0%

Table 7 - Input/Output for Various Instruments

Horn				Viola				Voice		
Actual	Measured	Diff		Actual	Measured	Diff		Actual	Measured	Diff
C4	C5	12		C4	F3	-7		C4	-	#N/A
C4	C5	12		C4	-	#N/A		C4	-	#N/A
D4	D5	12		D4	G3	-7		D4	-	#N/A
D4	-	#N/A		D4	-	#N/A		D4	-	#N/A
E4	-	#N/A		E4	A3	-7		E4	A4	5
E4	-	#N/A		E4	A_3	-6		E4	-	#N/A
F4	-	#N/A		F4	A_3	-7		F4	A_4	5
F4	F5	12		F4	-	#N/A		F4	-	#N/A
G4	G4	0		G4	C4	-7		G4	-	#N/A
G4	G5	12		G4	-	#N/A		G4	-	#N/A
A4	A4	0		A4	D4	-7		A4	-	#N/A
A4	A4	0		A4	-	#N/A		A4	-	#N/A
B4	B4	0		B4	E5	5		B4	-	#N/A
B4	B4	0		B4	-	#N/A		B4	-	#N/A
C5	C5	0		C5	F5	5		C5	-	#N/A
C5	C5	0		C5	-	#N/A		C5	-	#N/A
Percent Correct		44%		Percent Correct		0%		Percent Correct		0%

Table 8 - Input/Output for More Instruments

From Table 7 and Table 8 it is clear that code can correctly the musical note of a pure sinusoidal signal but has trouble when trying to identify certain notes played by certain instruments. The number listed in the “Diff” column shows the number of half-steps the measured note is from the actual note. From this information it might be possible to develop an algorithm to correct the results for a specific instrument or at least develop a correction look-up table.

Looking at the piano results of Table 7 shows that some notes were placed 12 half-steps above their actual frequency. This would imply that some notes (D5, E5) need to be shifted down 12-half steps. Unfortunately, the piano results cannot be corrected because both C4 and C5 are identified as C5.

The guitar results of Table 7 can almost be corrected. All of the notes are identified 7 half-steps below their actual value. Unfortunately, like the piano results, these cannot be corrected because both C4 and C5 are identified as F4.

The horn, viola, and synthesized voice seemingly have no rhyme or reason to their results; therefore, no obvious correction can be seen.

Extending the Possibilities – More Complicated Songs

To test the system in a more realistic situation, the Christmas carol: “Deck the Halls” was rendered using a synthesized piano and played for the DSK. Figure 7 shows the musical representation of “Deck the Halls.” All of the musical notes listed in the text are shown in the same terms that the board’s code agrees with. The musical notes displayed on the lines do not necessarily agree with that listing, for example, the first note is written as a “B flat” but the text and the board’s code describe it as an “A sharp.” In terms of sound, they are the same; the note just has several, valid yet different written representations.

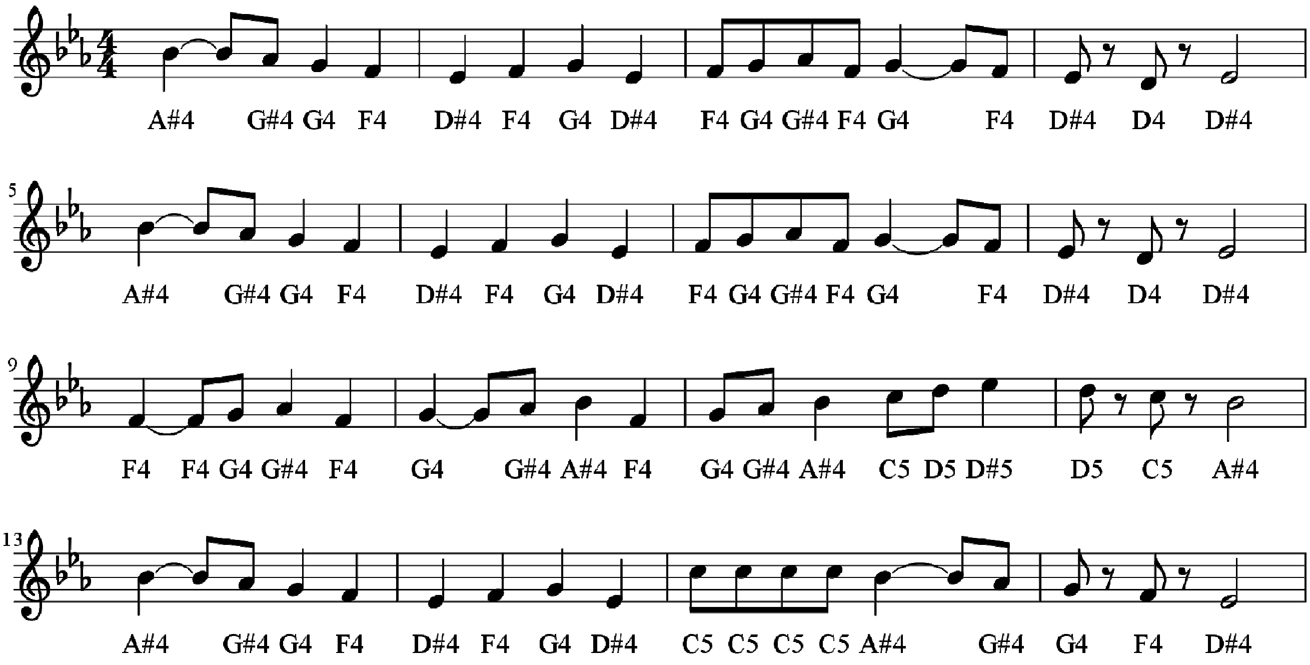

A#4 G#4 G4 F4 D#4 F4 G4 D#4 F4 G4 G#4 F4 G4 F4 D#4 D4 D#4
5
A#4 G#4 G4 F4 D#4 F4 G4 D#4 F4 G4 G#4 F4 G4 F4 D#4 D4 D#4
9
F4 F4 G4 G#4 F4 G4 G#4 A#4 F4 G4 G#4 A#4 C5 D5 D#5 D5 C5 A#4
13
A#4 G#4 G4 F4 D#4 F4 G4 D#4 C5 C5 C5 C5 A#4 G#4 G4 F4 D#4

Figure 7 - Musical Score for "Deck the Halls"

Verse 1 – 21 Correct of 32 Notes – 67%									
Measure 1	Correct	A#4	A#4	A#4	G#4	G4	G4	F4	F4
	DSK	A#4	A#4	A#4	G#4	G4	G4	F4	F4
	Diff	0	0	0	0	0	0	0	0
Measure 2	Correct	D#4	D#4	F4	F4	G4	G4	D#4	D#4
	DSK	D#5	D#5	F4	F4	G4	G4	D#5	D#5
	Diff	12	12	0	0	0	0	12	12
Measure 3	Correct	F4	G4	G#4	F4	G4	G4	G4	F4
	DSK	F4	G4	G#4	F4	G4	G4	G4	F4
	Diff	0	0	0	0	0	0	0	0
Measure 4	Correct	D#4	-	D4	-	D#4	D#4	D#4	D#4
	DSK	D#5	D#5	D5	D5	D#5	D#5	D#5	D#4
	Diff	12	#N/A	12	#N/A	12	12	12	0

Table 9 - Deck the Halls Verse 1 Results

Verse 2 – 21 Correct of 32 Notes – 67%									
Measure 5	Correct	A#4	A#4	A#4	G#4	G4	G4	F4	F4
	DSK	A#4	A#4	A#4	G#4	G4	G4	F4	F4
	Diff	0	0	0	0	0	0	0	0
Measure 6	Correct	D#4	D#4	F4	F4	G4	G4	D#4	D#4
	DSK	D#5	D#5	F4	F4	G4	G4	D#5	D#5
	Diff	12	12	0	0	0	0	12	12
Measure 7	Correct	F4	G4	G#4	F4	G4	G4	G4	F4
	DSK	F4	G4	G#4	F4	G4	G4	G4	F4
	Diff	0	0	0	0	0	0	0	0
Measure 8	Correct	D#4	-	D4	-	D#4	D#4	D#4	D#4
	DSK	D#5	D#5	D5	D5	D#5	D#5	D#5	D#4
	Diff	12	#N/A	12	#N/A	12	12	12	0

Table 10 - Deck the Halls Verse 2 Results

Verse 3 – 29 Correct of 32 Notes – 91%									
Measure 9	Correct	F4	F4	F4	G4	G#4	G#4	F4	F4
	DSK	F4	F4	F4	G4	G#4	G#4	F4	F4
	Diff	0	0	0	0	0	0	0	0
Measure 10	Correct	G4	G4	G4	G#4	A#4	A#4	F4	F4
	DSK	G4	G4	G4	G#4	A#4	A#4	F4	F4
	Diff	0	0	0	0	0	0	0	0
Measure 11	Correct	G4	G#4	A#4	A#4	C5	D5	D#5	D#5
	DSK	G4	G#4	A#4	A#4	C5	D5	D#5	D#5
	Diff	0	0	0	0	0	0	0	0
Measure 12	Correct	D#5	-	C5	-	A#4	A#4	A#4	A#4
	DSK	D5	D5	C5	C5	A#4	A#4	A#4	A#4
	Diff	-1	#N/A	0	#N/A	0	0	0	0

Table 11 - Deck the Halls Verse 3 Results

Verse 4 – 22 Correct of 32 Notes – 69%									
Measure 13	Correct	A#4	A#4	A#4	G#4	G4	G4	F4	F4
	DSK	A#4	A#4	A#4	G#4	G4	G4	F4	F4
	Diff	0	0	0	0	0	0	0	0
Measure 14	Correct	D#4	D#4	F4	F4	G4	G4	D#4	D#4
	DSK	D#5	D#5	F4	F4	G4	G4	D#5	D#5
	Diff	12	12	0	0	0	0	12	12
Measure 15	Correct	C5	C5	C5	C5	A#4	A#4	A#4	G#4
	DSK	C5	C5	C5	C5	A#4	A#4	A#4	G#4
	Diff	0	0	0	0	0	0	0	0
Measure 16	Correct	G#4	-	F4	-	D#4	D#4	D#4	D#4
	DSK	G4	G4	F4	F4	D#5	D#5	D#5	D#4
	Diff	-1	#N/A	0	#N/A	12	12	12	0

Table 12 - Deck the Halls Verse 4 Results

From the above results, the DSK was able to successfully determine 93 of 128 notes which correspond to being correct 73% of the time. It also gets the same notes wrong fairly consistently. Most of the notes that it got wrong were D and D#.

To make things a little more complicated, drums and other musical enhancements were added to the original version of "Deck the Halls." The remixed was also played for the DSK and the DSK produced the following output shown in Table 13.

Measure 1	A_4	-	-	G_4	-	C4	-	C4
Measure 2	-	-	-	-	-	-	-	-
Measure 3	-	G4	-	-	-	-	-	C6
Measure 4	-	-	-	-	-	-	-	-
Measure 5	-	-	-	-	-	-	-	-
Measure 6	-	-	-	-	-	-	-	-
Measure 7	-	-	G_4	-	-	-	-	C6
Measure 8	-	C4	-	-	-	-	-	-
Measure 9	-	-	C6	C6	-	-	-	-
Measure 10	-	-	-	-	-	-	-	-
Measure 11	-	-	-	-	-	D5	-	-
Measure 12	-	-	C6	-	C6	-	-	C6
Measure 13	-	-	-	-	-	-	-	-
Measure 14	-	-	-	-	-	-	-	-
Measure 15	-	-	-	C5	-	-	C6	-
Measure 16	-	C4	-	-	-	-	-	-

Table 13 - Output of Remixed "Deck the Halls"

The DSK correctly identified 4 of 128 notes. In other words, it was correct 3% of the time and though that most of the music was just noise. Clearly, the system works perfectly for pure sinusoidal signals but has many problems when dealing with instruments and complex music.

To fix these issues, various algorithms could be designed to determine what instrument is the dominate instrument. Then a filter could be applied to the signal to isolate the desired data. The data could be run through the Goertzel algorithm or possibly another algorithm. The complexity required by the system might require more advanced hardware than the C6712 DSK can provide. Or maybe the FFT would be more applicable when properly implemented in the embedded system.

Included Files

The Audio Samples can be used to listen to different sounds that were either generated by the board or sent to the board for analysis. It is a web page that included all of the sounds as “audio” objects for easy playback and is best viewed in Firefox or Google Chrome. The sounds are saved as *.ogg files for better compression; however, I did not pay attention to the encoding setting when I created the files so I do not know if they are encoded with the lossless FLAC codec or lossy Vorbis codec. My guess is that are the lossy Vorbis codec, which could be the cause of some errors in interpreting the music on the DSK.

The Sheet Music folder contains files that were used to create the musical scores featured in the report as well as to render the actual sound.

The Goertzel Simulation Tool was written by me. It is best viewed in Firefox or Google Chrome. It requires libraries not written by me released under the MIT license. Those libraries are included.

The CCStudioFiles folder contains all of the source files and project files required to rebuild the projects in CCStudio and the source of the Visual Basic host program. There are 5 subfolders, one for each CCStudio project.

Project 1 has the CCStudio Files used to generate a tone on the C6713 DSK.

Project 2 has the CCStudio Files that implement the Goertzel Algorithm and output the results to the C6713 DSK LEDs.

Project 3 has the CCStudio Files that implement the Goertzel Algorithm and output the result via RTDX. The VB NOTE folder contains the source of the Visual Basic program that listens to the RTDX channel and outputs the data from the DSK.

Project 4 has the files that output more complicated “songs” from the DSK.

Project 5 is basically the most complex and up-to-date version of project 3. It has the most up-to-date Visual Basic host application code and C6713 DSK target application code.

The Diagrams folder contains additional diagrams and pictures used in the report and presentation.

Report.docx is the soft copy of this report.

Presentation.pptx is the presentation to go along with the report.

Credits

Sounds were recorded using: Audacity 1.3.13-beta (Unicode) http://audacity.sourceforge.net/

Musical scores were written and rendered using: musescore v 1.1 r 4611: http://musescore.org/

Diagrams were created using diagramly: http://www.diagram.ly/

Visual Basic code was written using Microsoft Visual Basic 6.0

Report written with Microsoft Word 2007

Presentation created with Microsoft Power Point 2007

Bibliography

1. Formula For Frequency Table. *Physics of Music Notes.* [Online] [Cited: November 20, 2011.] http://www.phy.mtu.edu/~suits/NoteFreqCalcs.html.
2. **Chassaing, Rulph.** *Digital Signal Processing and Applications with the C6713 and C6416 DSK.* Hoboken, New Jersey : John Wiley & Sons, Inc., 2005.
3. **Jones, Douglas L.** Goertzel's Algorithm. *CONNEXIONS.* [Online] [Cited: November 23, 2011.] http://cnx.org/content/m12024/latest/.
4. **Banks, Kevin.** The Goertzel Algorithm. *EE|Times.* [Online] August 28, 2002. [Cited: November 23, 2011.] http://www.eetimes.com/design/embedded/4024443/The-Goertzel-Algorithm.

License

Source Code

Notes.h – Used in Various CCStudio Projects

```
#define     C0    16.35
#define     C_0   17.32
#define     D0    18.35
#define     D_0   19.45
#define     E0    20.6
#define     F0    21.83
#define     F_0   23.12
#define     G0    24.5
#define     G_0   25.96
#define     A0    27.5
#define     A_0   29.14
#define     B0    30.87
#define     C1    32.7
#define     C_1   34.65
#define     D1    36.71
#define     D_1   38.89
#define     E1    41.2
#define     F1    43.65
#define     F_1   46.25
#define     G1    49
#define     G_1   51.91
#define     A1    55
#define     A_1   58.27
#define     B1    61.74
#define     C2    65.41
#define     C_2   69.3
#define     D2    73.42
#define     D_2   77.78
#define     E2    82.41
#define     F2    87.31
#define     F_2   92.5
#define     G2    98
#define     G_2   103.83
#define     A2    110
#define     A_2   116.54
#define     B2    123.47
#define     C3    130.81
#define     C_3   138.59
#define     D3    146.83
#define     D_3   155.56
#define     E3    164.81
#define     F3    174.61
#define     F_3   185
#define     G3    196
#define     G_3   207.65
#define     A3    220
#define     A_3   233.08
#define     B3    246.94
#define     C4    261.63
#define     C_4   277.18
#define     D4    293.66
#define     D_4   311.13
```

```
#define     E4    329.63
#define     F4    349.23
#define     F_4   369.99
#define     G4    392
#define     G_4   415.3
#define     A4    440
#define     A_4   466.16
#define     B4    493.88
#define     C5    523.25
#define     C_5   554.37
#define     D5    587.33
#define     D_5   622.25
#define     E5    659.26
#define     F5    698.46
#define     F_5   739.99
#define     G5    783.99
#define     G_5   830.61
#define     A5    880
#define     A_5   932.33
#define     B5    987.77
#define     C6    1046.5
#define     C_6   1108.73
#define     D6    1174.66
#define     D_6   1244.51
#define     E6    1318.51
#define     F6    1396.91
#define     F_6   1479.98
#define     G6    1567.98
#define     G_6   1661.22
#define     A6    1760
#define     A_6   1864.66
#define     B6    1975.53
#define     C7    2093
#define     C_7   2217.46
#define     D7    2349.32
#define     D_7   2489.02
#define     E7    2637.02
#define     F7    2793.83
#define     F_7   2959.96
#define     G7    3135.96
#define     G_7   3322.44
#define     A7    3520
#define     A_7   3729.31
#define     B7    3951.07
#define     C8    4186.01
#define     C_8   4434.92
#define     D8    4698.64
#define     D_8   4978.03
```

P01 - Tone.c – Generates “DO RE MI FA SO LA TI DO” on C6713 DSK

```
#include <math.h>          /* required for cos function*/
#include "tonecfg.h"       /* auto-generated by CCStudio */
#include "dsk6713.h"       /* board support library */
#include "dsk6713_aic23.h" /* required for using the codec */
#include "notes.h"         /* pre-computed musical note frequencies */

/* We will be setting the coded to run at 8kHz which gives a
 * sample period of 1/8000 = 0.000125s */
#define SAMPLE_PERIOD 0.000125
#define PI 3.14159265358979323846264338327950

/* The cosOut outputs a cosine of frequency "freq" for "duration" seconds
 * to the codec with handle "hCodec"
 * This is basically a utility function that busy-waits while generating
 * and outputting a cosine wave */
void cosOut(double freq, double duration, DSK6713_AIC23_CodecHandle hCodec){
      double i; int sample;
      for (i = 0; i < duration; i+=SAMPLE_PERIOD){
            sample = (int)(2048.0*cos(2*PI*freq*i));
        // Send a sample to the left channel
        while (!DSK6713_AIC23_write(hCodec, sample ));
        //Send a sample to the right channel
        while (!DSK6713_AIC23_write(hCodec, sample ));
    }
      return;
}

/* Codec configuration settings */
DSK6713_AIC23_Config config = DSK6713_AIC23_DEFAULTCONFIG;

void main(){
    DSK6713_AIC23_CodecHandle hCodec;
    int i;
    /* Initialize the board support library, must be called first */
    DSK6713_init();
    /* Start the codec, set sample rate to 8kHz */
    hCodec = DSK6713_AIC23_openCodec(0, &config);
      DSK6713_AIC23_setFreq(hCodec, DSK6713_AIC23_FREQ_8KHZ);
      /* Output a bunch of notes 2 times */
      for(i=0;i<2;i++){
            cosOut(C4,0.5,hCodec);//DO
            cosOut(D4,0.5,hCodec);//RAY
            cosOut(E4,0.5,hCodec);//MI
            cosOut(F4,0.5,hCodec);//FA
            cosOut(G4,0.5,hCodec);//SO
            cosOut(A4,0.5,hCodec);//LA
            cosOut(B4,0.5,hCodec);//TI
            cosOut(C5,0.5,hCodec);//DOH!
      }
    /* Close the codec */
    DSK6713_AIC23_closeCodec(hCodec);
}
```

P02 - Tone.c – Runs Goertzel Algorithm, Outputs Data to LEDs

```
#include <math.h> /* required for cos function*/
#include "tonecfg.h" /* auto-generated by CCStudio */
#include "dsk6713.h" /* board support library */
#include "dsk6713_aic23.h" /* required for using the codec for audio io*/
#include "dsk6713_led.h" /* required for working with the LEDs */
#include "notes.h" /* pre-computed musical note frequencies for ease */
/* We will be setting the coded to run at 8kHz which gives a
 * sample period of 1/8000 = 0.000125s */
#define SAMPLE_PERIOD 0.000125
#define PI 3.14159265358979323846264338327950
/* Codec configuration settings */
DSK6713_AIC23_Config config = DSK6713_AIC23_DEFAULTCONFIG;
/* Setup the variables which will be used for Goertzel analysis */
#define NOTE_COUNT 8    //we will work only 8 notes
#define BLOCK_SIZE 1600 //samples to process before output & reset
double Z1[NOTE_COUNT];  //the 1st delay element for each note
double Z2[NOTE_COUNT];  //the 2nd delay element for each note
double  S[NOTE_COUNT];  //the current value for each note
double RF[NOTE_COUNT];  //the actual frequency for each note
double NF[NOTE_COUNT];  //the normalized frequency for each note
double RW[NOTE_COUNT];  //2*cos(2*pi*normalizedFreq) for each note
double PO[NOTE_COUNT];  //power of each note
int i;                  //the iteration count
int f;                  //which note we're on
int maxNote;            //which note has the most power
double maxP;            //the max power measured
Uint32 sampleRead; //place to store value read from codec
double sampleDbl;  //used to convert value read from codec to double

void main(){
    DSK6713_AIC23_CodecHandle hCodec;
    /* Initialize the board support library, must be called first */
    DSK6713_init();
    /* Initialize the LED module of the BSL */
    DSK6713_LED_init();
    /* Start the codec, set sample rate to 8kHz */
    hCodec = DSK6713_AIC23_openCodec(0, &config);
    DSK6713_AIC23_setFreq(hCodec, DSK6713_AIC23_FREQ_8KHZ);

      /* Initialize everything to the starting parameters */
      RF[0]=C4;
      RF[1]=D4;
      RF[2]=E4;
      RF[3]=F4;
      RF[4]=G4;
      RF[5]=A4;
      RF[6]=B4;
      RF[7]=C5;
      for(f=0; f<NOTE_COUNT; f++){
            Z1[f]=0;
            Z2[f]=0;
            NF[f]=RF[f]*SAMPLE_PERIOD;
            RW[f]=2*cos(2*PI*NF[f]);
      }
      while(1){
            /* Run Goertzel Algorithm */
            for(i=0; i<BLOCK_SIZE; i++){
                  //read one channel
                  while(!DSK6713_AIC23_read(hCodec, &sampleRead)){}
                  sampleDbl = ((double)sampleRead)/16383.0;
                  for(f=0; f<NOTE_COUNT; f++){
                        S[f]  = sampleDbl + RW[f]*Z1[f] - Z2[f];
                        Z2[f] = Z1[f];
                        Z1[f] = S[f];
                  }
                  //read other channel & ignore it
                  while(!DSK6713_AIC23_read(hCodec, &sampleRead)){}
            }
            //Compute the power & which note has the most
            //set maxP to 40000.0 because when there is no input signal PO[f]
            //has values that max around 40000
            maxP = 40000.0;
            maxNote = -1;
            for(f=0; f<NOTE_COUNT; f++){
                  //calculate power, dont bother taking the square root
                  PO[f] = Z2[f]*Z2[f] + Z1[f]*Z1[f] - RW[f]*Z1[f]*Z2[f];
                  if(PO[f] > maxP){
                        maxP = PO[f];
                        maxNote = f;
                  }
                  //reset
                  Z1[f]=0.0;
                  Z2[f]=0.0;
            }
            //turn off LEDs
            DSK6713_LED_off(0);
            DSK6713_LED_off(1);
            DSK6713_LED_off(2);
            DSK6713_LED_off(3);
            switch(maxNote){
                  case 0:  DSK6713_LED_on(0); break;
                  case 1:  DSK6713_LED_on(1); break;
                  case 2:  DSK6713_LED_on(0); DSK6713_LED_on(1); break;
                  case 3:  DSK6713_LED_on(2); break;
                  case 4:  DSK6713_LED_on(0); DSK6713_LED_on(2); break;
                  case 5:  DSK6713_LED_on(1); DSK6713_LED_on(2); break;
                  case 6:  DSK6713_LED_on(0);
                           DSK6713_LED_on(1); DSK6713_LED_on(2); break;
                  case 7:  DSK6713_LED_on(3); break;
                  default: break;
            }
      }

    /* Close the codec */
    DSK6713_AIC23_closeCodec(hCodec);
}
```

P03 - Target.h – Used in RTDX Programs on Target, Made by TI

```
/**************************************************************************
* $RCSfile: target.h,v $
* $Revision: 1.1 $
* $Date: 2000/09/19 21:49:28 $
* Copyright (c) 2000 Texas Instruments Incorporated
*
* C6x specific initialization details
**************************************************************************/
#ifndef __TARGET_H
#define __TARGET_H
#include <c6x.h>                /* IER,ISR,CSR registers                */

/*    RTDX is interrupt driven on the C6x.
      So enable the interrupts now, or it won't work.
*/

#define IER_NMIE  0x00000002
#define CSR_GIE     0x00000001
#define TARGET_INITIALIZE() \
      IER |= 0x00000001 | IER_NMIE; \
      CSR |= CSR_GIE;

#endif /* __TARGET_H    */
```

P03 - Tone.c – Runs Goertzel, Outputs to LEDs and RTDX

```
#include <math.h> /* required for cos function*/
#include "tonecfg.h" /* auto-generated by CCStudio */
#include "dsk6713.h" /* board support library */
#include "dsk6713_aic23.h" /* required for using the codec for audio io */
#include "dsk6713_led.h" /* required for working with the LEDs */
#include "notes.h" /* pre-computed musical note frequencies for ease */
#include <rtdx.h>   /* For RTDX communication*/
#include "target.h" /* RTDX setup */
/* Declare and initialize an output channel called "ochan" */
RTDX_CreateOutputChannel(ochan);
/* We will be setting the coded to run at 8kHz which gives a
 * sample period of 1/8000 = 0.000125s */
#define SAMPLE_PERIOD 0.000125
#define PI 3.14159265358979323846264338327950
/* Codec configuration settings */
DSK6713_AIC23_Config config = DSK6713_AIC23_DEFAULTCONFIG;
/* Setup the variables which will be used for Goertzel analysis */
#define NOTE_COUNT 8    //we will work only 8 notes
#define BLOCK_SIZE 1600 //samples to process before output & reset
double Z1[NOTE_COUNT];  //the 1st delay element for each note
double Z2[NOTE_COUNT];  //the 2nd delay element for each note
double  S[NOTE_COUNT];  //the current value for each note
double RF[NOTE_COUNT];  //the actual frequency for each note
double NF[NOTE_COUNT];  //the normalized frequency for each note
double RW[NOTE_COUNT];  //2*cos(2*pi*normalizedFreq) for each note
double PO[NOTE_COUNT];  //power of each note
int i;                  //the iteration count
int f;                  //which note we're on
int maxNote;            //which note has the most power
double maxP;            //the max power measured
Uint32 sampleRead; //place to store value read from codec
double sampleDbl;  //used to convert value read from codec to double
void main(){
    DSK6713_AIC23_CodecHandle hCodec;
    /* Initialize the board support library, must be called first */
    DSK6713_init();
    /* Initialize the LED module of the BSL */
    DSK6713_LED_init();
    /* Start the codec, set sample rate to 8kHz */
    hCodec = DSK6713_AIC23_openCodec(0, &config);
    DSK6713_AIC23_setFreq(hCodec, DSK6713_AIC23_FREQ_8KHZ);
    /* RTDX Setup */
    TARGET_INITIALIZE();
    RTDX_enableOutput(&ochan);
      /* Initialize everything to the starting parameters */
      RF[0]=C4;
      RF[1]=D4;
      RF[2]=E4;
      RF[3]=F4;
      RF[4]=G4;
      RF[5]=A4;
      RF[6]=B4;
      RF[7]=C5;
      for(f=0; f<NOTE_COUNT; f++){
            Z1[f]=0;
            Z2[f]=0;
            NF[f]=RF[f]*SAMPLE_PERIOD;
            RW[f]=2*cos(2*PI*NF[f]);
      }
      while(1){
            /* Run Goertzel Algorithm */
            for(i=0; i<BLOCK_SIZE; i++){
                  //read one channel
                  while(!DSK6713_AIC23_read(hCodec, &sampleRead)){}
                  sampleDbl = ((double)sampleRead)/16383.0;
                  for(f=0; f<NOTE_COUNT; f++){
                        S[f]  = sampleDbl + RW[f]*Z1[f] - Z2[f];
                        Z2[f] = Z1[f];
                        Z1[f] = S[f];
                  }
                  //read other channel & ignore it
                  while(!DSK6713_AIC23_read(hCodec, &sampleRead)){}
            }
            maxP = 40000.0;
            maxNote = -1;
            for(f=0; f<NOTE_COUNT; f++){
                  //calculate power, dont bother taking the square root
                  PO[f] = Z2[f]*Z2[f] + Z1[f]*Z1[f] - RW[f]*Z1[f]*Z2[f];
                  if(PO[f] > maxP){
                        maxP = PO[f];
                        maxNote = f;
                  }
                  //reset
                  Z1[f]=0.0;
                  Z2[f]=0.0;
            }
            //turn off LEDs
            DSK6713_LED_off(0);
            DSK6713_LED_off(1);
            DSK6713_LED_off(2);
            DSK6713_LED_off(3);
            switch(maxNote){
                  case 0:  DSK6713_LED_on(0); break;
                  case 1:  DSK6713_LED_on(1); break;
                  case 2:  DSK6713_LED_on(0); DSK6713_LED_on(1); break;
                  case 3:  DSK6713_LED_on(2); break;
                  case 4:  DSK6713_LED_on(0); DSK6713_LED_on(2); break;
                  case 5:  DSK6713_LED_on(1); DSK6713_LED_on(2); break;
                  case 6:  DSK6713_LED_on(0); DSK6713_LED_on(1);
                           DSK6713_LED_on(2); break;
                  case 7:  DSK6713_LED_on(3); break;
                  default: break;
            }
            //write to RTDX
            if (!RTDX_write(&ochan,&maxNote,sizeof(maxNote))){/*ERROR*/}
      }
    /* Stop RTDX */       RTDX_disableOutput(&ochan);
    /* Close the codec */ DSK6713_AIC23_closeCodec(hCodec);
}
```

P03 - Visual Basic 1 – Source Code of First VB Host App

```
Option Explicit
'''''''''''''''''''''''''''''''''''''''''''''Utility Variables
Public connected As Integer                '1 if RTDX channel has been opened
Public lastMsg As String                   'for debugging purposes

'''''''''''''''''''''''''''''''''''''''''''''Needed for RTDX/CCStudio setup
Private CCSetup As Object                  'Used To Access CCStudio functions
Private Boards As Object                   'List of Available boards
Private Board As Object                    'Board
Private Processors As Object               'List of available processors
Private Processor As Object                'Processor
Public CurrentSelectedBoard As String      'The name of the board
Public CurrentSelectedProcessor As String  'Which processor on the board
Dim rtdx As Object                         'Acutal RTDX object

'''''''''''''''''''''''''''''''''''''''''''''RTDX OLE API Status Return codes
Const SUCCESS = &H0                         'Method call successful
Const FAIL = &H80004005                     'Method call failure
Const ENoDataAvailable = &H8003001E         'No data is currently available
Const EEndOfLogFile = &H80030002            'End of log file


'Utility function to output debugging messages to lower text area
Private Sub LogMessage(message As String)
  If lastMsg <> message Then
    tbStatus.Text = tbStatus.Text & message & vbNewLine
    lastMsg = message
  End If
End Sub


'Utility function that takes the value read from the board, converts it into
'useful text and displays it to the user
'The program on the board was designed to send 0 through 8 to represent
'different notes and -1 when it could not make a decision
'This converts those integers to a more meaningful string
Private Sub ProcessData(ReadVal As Long)
  Dim NOTES() As Variant
  NOTES() = Array("DO", "RE", "MI", "FA", "SO", "LA", "TI", "DOH")
  If ReadVal > -1 Then
    outputArea.Text = outputArea.Text & NOTES(ReadVal) & ","
  End If
End Sub


'When the program starts
Private Sub Form_Load()
  connected = 0 'state that we don't have an RTDX channel open
  If (GetAvailableBoards) Then     ' Get Available Boards and Processors
    list_Boards.Selected(0) = True ' Set the selected board to 0
  End If
End Sub
```

```
'When the program ends
Private Sub Form_Unload(Cancel As Integer)
  If connected = 1 Then     'If we have the RTDX channel open
    DisconnectFromBoard   'Close the RTDX channel
  End If
  Set CCSetup = Nothing     'Cleanup
  Set Boards = Nothing    'Cleanup
  Set Board = Nothing     'Cleanup
  Set Processors = Nothing  'Cleanup
  Set Processor = Nothing   'Cleanup
End Sub


'Function to update the list of boards that can be connected to
Private Function GetAvailableBoards() As Boolean
  Dim status As Long
  Dim BoardName As String

  ' Initialize Lists
  list_Boards.Clear
  list_Processors.Clear

  ' Instantiate the Code Composer Setup SystemSetup coclass and obtain a
  ' pointer to the ISystemSetup interface
  Set CCSetup = CreateObject("CodeComposerSetup.SystemSetup")

  ' Get a pointer to the IBoards interface
  status = CCSetup.GetBoards(Boards)

  ' Loop through the available boards, get the names of the boards,
  ' and add the board names to the boards list control
  For Each Board In Boards
    ' Get the board name
    status = Board.GetName(BoardName)
    ' Append board name to the board list
    list_Boards.AddItem (BoardName)
  Next

  ' return True
  GetAvailableBoards = True

End Function
```

```
'Function to update the list of processors which can be used
Private Function GetAvailableProcessors(SelectedBoardName As String) As
Boolean

  Dim status As Long
  Dim ProcessorName As String

  ' Get a pointer to the IBoard interface for the selected
  ' board
  status = CCSetup.GetBoardByName(SelectedBoardName, Board)


  ' Get a pointer to the IProcessors interface
  status = Board.GetProcessors(Processors)

  ' Loop through the available processors, get the names of the
  ' processors, and add the processors to the processors list
  ' control
  For Each Processor In Processors
    ' Get the processor name
    status = Processor.GetName(ProcessorName)

    ' Append processor name to the processor list
    list_Processors.AddItem (ProcessorName)
  Next

  ' Return True
  GetAvailableProcessors = True
End Function

'When the "connect" button is clicked
Private Sub btConnect_Click()
  If connected = 1 Then     'If we have the RTDX channel open
    DisconnectFromBoard   'Close the RTDX channel
  End If
  ConnectToBoard        'Open the RTDX channel
  btConnect.Enabled = False   'Disable the "connect" button
  btDisconnect.Enabled = True 'Enable the "disconnect" button
End Sub

'When the "disconnect" button is clicked
Private Sub btDisconnect_Click()
  DisconnectFromBoard      'Close the RTDX channel
  btDisconnect.Enabled = False 'Disable the "disconnect" button
  btConnect.Enabled = True   'Enable the "connect" button
End Sub

Private Sub btListRefresh_Click()
  ' Get Available Boards and Processors
  If (GetAvailableBoards) Then
    ' Set the selected board to 0
    list_Boards.Selected(0) = True
  End If
End Sub
```

```
'When a board is selected
Private Sub list_Boards_Click()
  ' Clear processor list
  list_Processors.Clear
  ' Get current selected board
  CurrentSelectedBoard = list_Boards.List(list_Boards.ListIndex)
  ' Get available processors for that board
  If (GetAvailableProcessors(CurrentSelectedBoard)) Then
    ' Set the selected processor to 0
    list_Processors.Selected(0) = True
  End If
End Sub

'When a processor is selected
Private Sub list_Processors_Click()
  ' Get current selected processor
  CurrentSelectedProcessor = list_Processors.List(list_Processors.ListIndex)
End Sub

'Function which tries to open the RTDX channel on the selected board/pro
Private Sub ConnectToBoard()
  LogMessage "Info  - Attempting to connect to board " & CurrentSelectedBoard
  LogMessage "Info  - Attempting to connect to processor " &
CurrentSelectedProcessor
  Dim status As Long
  ' Get application objects
  Set rtdx = CreateObject("RTDX")
  status = rtdx.SetProcessor(CurrentSelectedBoard, CurrentSelectedProcessor)
  If (status <> SUCCESS) Then
    LogMessage "Error - Set Processor failed"
    Exit Sub
  End If
  'open target's input channel
  '"ochan" must agree with RTDX_CreateOutputChannel(ochan);
  'from target source code
  status = rtdx.Open("ochan", "R")
  Select Case status
  Case Is = SUCCESS
    connected = 1
    LogMessage "Info  - Opened RTDX channel for reading"
  Case Is = FAIL
    LogMessage "Error - Unable to open channel for RTDX communications"
    Exit Sub
  Case Else
    LogMessage "Info  - Unknown return value from openning RTDX channel"
    Exit Sub
  End Select
End Sub
```

```
'Function which tries to close the RTDX channel on the selected board/proc
Private Sub DisconnectFromBoard()
  Dim status As Long
  ' close target's input channel
  status = rtdx.Close()
  Select Case status
    Case Is = SUCCESS
      LogMessage "Info  - Successfully closed RTDX channel"
    Case Is = FAIL
      LogMessage "Error - Unable to close RTDX channel"
    Case Else
      LogMessage "Info  - Unknown return value from closing RTDX channel"
  End Select
  connected = 0
  Set rtdx = Nothing           ' kill RTDX OLE object
End Sub

'Function to check for new RTDX data every XX milliseconds
Private Sub Timer1_Timer()
  'Dim ReadValue As Variant
  Dim ReadValue As Long
  Dim status As Long
  If connected = 1 Then
    'status = rtdx.ReadSAI2(ReadValue)
    'status = rtdx.ReadSAF4(ReadValue)
    'Do
      status = rtdx.ReadI4(ReadValue)
      Select Case status
        Case Is = SUCCESS
          ProcessData ReadValue
        Case Is = FAIL
          LogMessage "Error - Reading data failed " & ReadValue
        Case Is = ENoDataAvailable
          LogMessage "Error - No data available"
        Case Is = EEndOfLogFile
          LogMessage "Info  - Reached end of log file"
        Case Else
          LogMessage "Error - Unknown error reading RTDX channel - " &
ReadValue
      End Select
    'Loop Until status = EEndOfLogFile
  End If
End Sub
```

P04 - Sequence.h – Plays All Notes From A0 to D#8

```
cosOut(C0   ,0.25,hCodec);
cosOut(C_0  ,0.25,hCodec);
cosOut(D0   ,0.25,hCodec);
cosOut(D_0  ,0.25,hCodec);
cosOut(E0   ,0.25,hCodec);
cosOut(F0   ,0.25,hCodec);
cosOut(F_0  ,0.25,hCodec);
cosOut(G0   ,0.25,hCodec);
cosOut(G_0  ,0.25,hCodec);
cosOut(A0   ,0.25,hCodec);
cosOut(A_0  ,0.25,hCodec);
cosOut(B0   ,0.25,hCodec);
cosOut(C1   ,0.25,hCodec);
cosOut(C_1  ,0.25,hCodec);
cosOut(D1   ,0.25,hCodec);
cosOut(D_1  ,0.25,hCodec);
cosOut(E1   ,0.25,hCodec);
cosOut(F1   ,0.25,hCodec);
cosOut(F_1  ,0.25,hCodec);
cosOut(G1   ,0.25,hCodec);
cosOut(G_1  ,0.25,hCodec);
cosOut(A1   ,0.25,hCodec);
cosOut(A_1  ,0.25,hCodec);
cosOut(B1   ,0.25,hCodec);
cosOut(C2   ,0.25,hCodec);
cosOut(C_2  ,0.25,hCodec);
cosOut(D2   ,0.25,hCodec);
cosOut(D_2  ,0.25,hCodec);
cosOut(E2   ,0.25,hCodec);
cosOut(F2   ,0.25,hCodec);
cosOut(F_2  ,0.25,hCodec);
cosOut(G2   ,0.25,hCodec);
cosOut(G_2  ,0.25,hCodec);
cosOut(A2   ,0.25,hCodec);
cosOut(A_2  ,0.25,hCodec);
cosOut(B2   ,0.25,hCodec);
cosOut(C3   ,0.25,hCodec);
cosOut(C_3  ,0.25,hCodec);
cosOut(D3   ,0.25,hCodec);
cosOut(D_3  ,0.25,hCodec);
cosOut(E3   ,0.25,hCodec);
cosOut(F3   ,0.25,hCodec);
cosOut(F_3  ,0.25,hCodec);
cosOut(G3   ,0.25,hCodec);
cosOut(G_3  ,0.25,hCodec);
cosOut(A3   ,0.25,hCodec);
cosOut(A_3  ,0.25,hCodec);
cosOut(B3   ,0.25,hCodec);
cosOut(C4   ,0.25,hCodec);
cosOut(C_4  ,0.25,hCodec);
cosOut(D4   ,0.25,hCodec);
cosOut(D_4  ,0.25,hCodec);
cosOut(E4   ,0.25,hCodec);
cosOut(F4   ,0.25,hCodec);
cosOut(F_4  ,0.25,hCodec);
cosOut(G4   ,0.25,hCodec);
cosOut(G_4  ,0.25,hCodec);
cosOut(A4   ,0.25,hCodec);
cosOut(A_4  ,0.25,hCodec);
cosOut(B4   ,0.25,hCodec);
cosOut(C5   ,0.25,hCodec);
cosOut(C_5  ,0.25,hCodec);
cosOut(D5   ,0.25,hCodec);
cosOut(D_5  ,0.25,hCodec);
cosOut(E5   ,0.25,hCodec);
cosOut(F5   ,0.25,hCodec);
cosOut(F_5  ,0.25,hCodec);
cosOut(G5   ,0.25,hCodec);
cosOut(G_5  ,0.25,hCodec);
cosOut(A5   ,0.25,hCodec);
cosOut(A_5  ,0.25,hCodec);
cosOut(B5   ,0.25,hCodec);
cosOut(C6   ,0.25,hCodec);
cosOut(C_6  ,0.25,hCodec);
cosOut(D6   ,0.25,hCodec);
cosOut(D_6  ,0.25,hCodec);
cosOut(E6   ,0.25,hCodec);
cosOut(F6   ,0.25,hCodec);
cosOut(F_6  ,0.25,hCodec);
cosOut(G6   ,0.25,hCodec);
cosOut(G_6  ,0.25,hCodec);
cosOut(A6   ,0.25,hCodec);
cosOut(A_6  ,0.25,hCodec);
cosOut(B6   ,0.25,hCodec);
cosOut(C7   ,0.25,hCodec);
cosOut(C_7  ,0.25,hCodec);
cosOut(D7   ,0.25,hCodec);
cosOut(D_7  ,0.25,hCodec);
cosOut(E7   ,0.25,hCodec);
cosOut(F7   ,0.25,hCodec);
cosOut(F_7  ,0.25,hCodec);
cosOut(G7   ,0.25,hCodec);
cosOut(G_7  ,0.25,hCodec);
cosOut(A7   ,0.25,hCodec);
cosOut(A_7  ,0.25,hCodec);
cosOut(B7   ,0.25,hCodec);
cosOut(C8   ,0.25,hCodec);
cosOut(C_8  ,0.25,hCodec);
cosOut(D8   ,0.25,hCodec);
cosOut(D_8  ,0.25,hCodec);
```

P04 - Sequence_deck.h – Plays “Deck the Halls”

```
cosOut(A_4,0.25,hCodec);
cosOut(A_4,0.25,hCodec);
cosOut(A_4,0.25,hCodec);
cosOut(G_4,0.25,hCodec);
cosOut(G4,0.25,hCodec);
cosOut(G4,0.25,hCodec);
cosOut(F4,0.25,hCodec);
cosOut(F4,0.25,hCodec);
cosOut(D_5,0.25,hCodec);
cosOut(D_5,0.25,hCodec);
cosOut(F4,0.25,hCodec);
cosOut(F4,0.25,hCodec);
cosOut(G4,0.25,hCodec);
cosOut(G4,0.25,hCodec);
cosOut(D_5,0.25,hCodec);
cosOut(D_5,0.25,hCodec);
cosOut(F4,0.25,hCodec);
cosOut(G4,0.25,hCodec);
cosOut(G_4,0.25,hCodec);
cosOut(F4,0.25,hCodec);
cosOut(G4,0.25,hCodec);
cosOut(G4,0.25,hCodec);
cosOut(G4,0.25,hCodec);
cosOut(F4,0.25,hCodec);
cosOut(D_5,0.25,hCodec);
cosOut(D_5,0.25,hCodec);
cosOut(D5,0.25,hCodec);
cosOut(D5,0.25,hCodec);
cosOut(D_5,0.25,hCodec);
cosOut(D_5,0.25,hCodec);
cosOut(D_5,0.25,hCodec);
cosOut(D_4,0.25,hCodec);
cosOut(A_4,0.25,hCodec);
cosOut(A_4,0.25,hCodec);
cosOut(A_4,0.25,hCodec);
cosOut(G_4,0.25,hCodec);
cosOut(G4,0.25,hCodec);
cosOut(G4,0.25,hCodec);
cosOut(F4,0.25,hCodec);
cosOut(F4,0.25,hCodec);
cosOut(D_5,0.25,hCodec);
cosOut(D_5,0.25,hCodec);
cosOut(F4,0.25,hCodec);
cosOut(F4,0.25,hCodec);
cosOut(G4,0.25,hCodec);
cosOut(G4,0.25,hCodec);
cosOut(D_5,0.25,hCodec);
cosOut(D_5,0.25,hCodec);
cosOut(F4,0.25,hCodec);
cosOut(G4,0.25,hCodec);
cosOut(G_4,0.25,hCodec);
cosOut(F4,0.25,hCodec);
cosOut(G4,0.25,hCodec);
cosOut(G4,0.25,hCodec);
cosOut(G4,0.25,hCodec);
cosOut(F4,0.25,hCodec);
cosOut(D_5,0.25,hCodec);
cosOut(D_5,0.25,hCodec);
cosOut(D5,0.25,hCodec);
cosOut(D5,0.25,hCodec);
cosOut(D_5,0.25,hCodec);
cosOut(D_5,0.25,hCodec);
cosOut(D_5,0.25,hCodec);
cosOut(D_4,0.25,hCodec);
cosOut(F4,0.25,hCodec);
cosOut(F4,0.25,hCodec);
cosOut(F4,0.25,hCodec);
cosOut(G4,0.25,hCodec);
cosOut(G_4,0.25,hCodec);
cosOut(G_4,0.25,hCodec);
cosOut(F4,0.25,hCodec);
cosOut(F4,0.25,hCodec);
cosOut(G4,0.25,hCodec);
cosOut(G4,0.25,hCodec);
cosOut(G4,0.25,hCodec);
cosOut(G_4,0.25,hCodec);
cosOut(A_4,0.25,hCodec);
cosOut(A_4,0.25,hCodec);
cosOut(F4,0.25,hCodec);
cosOut(F4,0.25,hCodec);
cosOut(G4,0.25,hCodec);
cosOut(G_4,0.25,hCodec);
cosOut(A_4,0.25,hCodec);
cosOut(A_4,0.25,hCodec);
cosOut(C5,0.25,hCodec);
cosOut(D5,0.25,hCodec);
cosOut(D_5,0.25,hCodec);
cosOut(D_5,0.25,hCodec);
cosOut(D5,0.25,hCodec);
cosOut(D5,0.25,hCodec);
cosOut(C5,0.25,hCodec);
cosOut(C5,0.25,hCodec);
cosOut(A_4,0.25,hCodec);
cosOut(A_4,0.25,hCodec);
cosOut(A_4,0.25,hCodec);
cosOut(A_4,0.25,hCodec);
cosOut(A_4,0.25,hCodec);
cosOut(A_4,0.25,hCodec);
cosOut(A_4,0.25,hCodec);
cosOut(G_4,0.25,hCodec);
cosOut(G4,0.25,hCodec);
cosOut(G4,0.25,hCodec);
cosOut(F4,0.25,hCodec);
cosOut(F4,0.25,hCodec);
cosOut(D_5,0.25,hCodec);
cosOut(D_5,0.25,hCodec);
cosOut(F4,0.25,hCodec);
cosOut(F4,0.25,hCodec);
cosOut(G4,0.25,hCodec);
cosOut(G4,0.25,hCodec);
cosOut(D_5,0.25,hCodec);
cosOut(D_5,0.25,hCodec);
cosOut(C5,0.25,hCodec);
cosOut(C5,0.25,hCodec);
cosOut(C5,0.25,hCodec);
cosOut(C5,0.25,hCodec);
cosOut(A_4,0.25,hCodec);
cosOut(A_4,0.25,hCodec);
cosOut(A_4,0.25,hCodec);
cosOut(G_4,0.25,hCodec);
cosOut(G4,0.25,hCodec);
cosOut(G4,0.25,hCodec);
cosOut(F4,0.25,hCodec);
cosOut(F4,0.25,hCodec);
cosOut(D_5,0.25,hCodec);
cosOut(D_5,0.25,hCodec);
cosOut(D_5,0.25,hCodec);
cosOut(D_4,0.25,hCodec);
```

P04 - Tone.c – Plays a Sequence of Notes

```
#include <math.h>          /* required for cos function*/
#include "tonecfg.h"       /* auto-generated by CCStudio */
#include "dsk6713.h"       /* board support library */
#include "dsk6713_aic23.h" /* required for using the codec for audio io */
#include "notes.h"         /* pre-computed musical note frequencies */

/* We will be setting the coded to run at 8kHz which gives a
 * sample period of 1/8000 = 0.000125s */
#define SAMPLE_PERIOD 0.000125
#define PI 3.14159265358979323846264338327 95

/* The cosOut outputs a cosine of frequency "freq" for "duration" seconds
 * to the codec with handle "hCodec"
 * This is basically a utility function that busy-waits while generating
 * and outputting a cosine wave */
void cosOut(double freq, double duration, DSK6713_AIC23_CodecHandle hCodec){
      double i; int sample;
      for (i = SAMPLE_PERIOD; i < duration; i+=SAMPLE_PERIOD){
            sample = (int)(2048.0*cos(2*PI*freq*i));
            // Send a sample to the left channel
            while (!DSK6713_AIC23_write(hCodec, sample ));
            //Send a sample to the right channel
            while (!DSK6713_AIC23_write(hCodec, sample ));
      }
      return;
}

/* Codec configuration settings */
DSK6713_AIC23_Config config = DSK6713_AIC23_DEFAULTCONFIG;

void main(){
    DSK6713_AIC23_CodecHandle hCodec;

      int i;

    /* Initialize the board support library, must be called first */
    DSK6713_init();

    /* Start the codec, set sample rate to 8kHz */
    hCodec = DSK6713_AIC23_openCodec(0, &config);
      DSK6713_AIC23_setFreq(hCodec, DSK6713_AIC23_FREQ_8KHZ);

      /* Output all the notes 2 times */
      for(i=0;i<2;i++){
            //#include "sequence.h"
      }
      //play Deck the Halls, based on the notes the DSK identified
      #include "sequence_deck.h"

    /* Close the codec */
    DSK6713_AIC23_closeCodec(hCodec);
}
```

P05 – Note_setup.h – Sets up Notes that will be Listened for

```
RF[   0      ]     =      C0    ;
RF[   1      ]     =      C_0   ;
RF[   2      ]     =      D0    ;
RF[   3      ]     =      D_0   ;
RF[   4      ]     =      E0    ;
RF[   5      ]     =      F0    ;
RF[   6      ]     =      F_0   ;
RF[   7      ]     =      G0    ;
RF[   8      ]     =      G_0   ;
RF[   9      ]     =      A0    ;
RF[   10     ]     =      A_0   ;
RF[   11     ]     =      B0    ;
RF[   12     ]     =      C1    ;
RF[   13     ]     =      C_1   ;
RF[   14     ]     =      D1    ;
RF[   15     ]     =      D_1   ;
RF[   16     ]     =      E1    ;
RF[   17     ]     =      F1    ;
RF[   18     ]     =      F_1   ;
RF[   19     ]     =      G1    ;
RF[   20     ]     =      G_1   ;
RF[   21     ]     =      A1    ;
RF[   22     ]     =      A_1   ;
RF[   23     ]     =      B1    ;
RF[   24     ]     =      C2    ;
RF[   25     ]     =      C_2   ;
RF[   26     ]     =      D2    ;
RF[   27     ]     =      D_2   ;
RF[   28     ]     =      E2    ;
RF[   29     ]     =      F2    ;
RF[   30     ]     =      F_2   ;
RF[   31     ]     =      G2    ;
RF[   32     ]     =      G_2   ;
RF[   33     ]     =      A2    ;
RF[   34     ]     =      A_2   ;
RF[   35     ]     =      B2    ;
RF[   36     ]     =      C3    ;
RF[   37     ]     =      C_3   ;
RF[   38     ]     =      D3    ;
RF[   39     ]     =      D_3   ;
RF[   40     ]     =      E3    ;
RF[   41     ]     =      F3    ;
RF[   42     ]     =      F_3   ;
RF[   43     ]     =      G3    ;
RF[   44     ]     =      G_3   ;
RF[   45     ]     =      A3    ;
RF[   46     ]     =      A_3   ;
RF[   47     ]     =      B3    ;
RF[   48     ]     =      C4    ;
RF[   49     ]     =      C_4   ;
RF[   50     ]     =      D4    ;
RF[   51     ]     =      D_4   ;
RF[   52     ]     =      E4    ;
RF[   53     ]     =      F4    ;
RF[   54     ]     =      F_4   ;
```

```
RF[   55     ]      =      G4     ;
RF[   56     ]      =      G_4    ;
RF[   57     ]      =      A4     ;
RF[   58     ]      =      A_4    ;
RF[   59     ]      =      B4     ;
RF[   60     ]      =      C5     ;
RF[   61     ]      =      C_5    ;
RF[   62     ]      =      D5     ;
RF[   63     ]      =      D_5    ;
RF[   64     ]      =      E5     ;
RF[   65     ]      =      F5     ;
RF[   66     ]      =      F_5    ;
RF[   67     ]      =      G5     ;
RF[   68     ]      =      G_5    ;
RF[   69     ]      =      A5     ;
RF[   70     ]      =      A_5    ;
RF[   71     ]      =      B5     ;
RF[   72     ]      =      C6     ;
RF[   73     ]      =      C_6    ;
RF[   74     ]      =      D6     ;
RF[   75     ]      =      D_6    ;
RF[   76     ]      =      E6     ;
RF[   77     ]      =      F6     ;
RF[   78     ]      =      F_6    ;
RF[   79     ]      =      G6     ;
RF[   80     ]      =      G_6    ;
RF[   81     ]      =      A6     ;
RF[   82     ]      =      A_6    ;
RF[   83     ]      =      B6     ;
RF[   84     ]      =      C7     ;
RF[   85     ]      =      C_7    ;
RF[   86     ]      =      D7     ;
RF[   87     ]      =      D_7    ;
RF[   88     ]      =      E7     ;
RF[   89     ]      =      F7     ;
RF[   90     ]      =      F_7    ;
RF[   91     ]      =      G7     ;
RF[   92     ]      =      G_7    ;
RF[   93     ]      =      A7     ;
RF[   94     ]      =      A_7    ;
RF[   95     ]      =      B7     ;
```

P05 – Tone.c – Target Code

```c
#include <math.h> /* required for cos function*/
#include "tonecfg.h" /* auto-generated by CCStudio */
#include "dsk6713.h" /* board support library */
#include "dsk6713_aic23.h" /* required for using the codec for audio io */
#include "dsk6713_led.h" /* required for working with the LEDs */
#include "notes.h" /* pre-computed musical note frequencies for ease */
#include <rtdx.h>   /* For RTDX communication*/
#include "target.h" /* RTDX setup */

/* Declare and initialize an output channel called "ochan" */
RTDX_CreateOutputChannel(ochan);

/* We will be setting the coded to run at 8kHz which gives a
 * sample period of 1/8000 = 0.000125s */
#define SAMPLE_PERIOD 0.000125
#define PI 3.14159265358979323846264338327950

/* Codec configuration settings */
DSK6713_AIC23_Config config = DSK6713_AIC23_DEFAULTCONFIG;

/* Setup the variables which will be used for Goertzel analysis */
#define NOTE_COUNT 96   //we will work only some notes
#define BLOCK_SIZE 2000 //samples to process before output & reset
double Z1[NOTE_COUNT];  //the 1st delay element for each note
double Z2[NOTE_COUNT];  //the 2nd delay element for each note
double  S[NOTE_COUNT];  //the current value for each note
double RF[NOTE_COUNT];  //the actual frequency for each note
double NF[NOTE_COUNT];  //the normalized frequency for each note
double RW[NOTE_COUNT];  //2*cos(2*pi*normalizedFreq) for each note
double PO[NOTE_COUNT];  //power of each note
int i;                  //the iteration count
int f;                  //which note we're on
int maxNote;            //which note has the most power
double maxP;            //the max power measured
double totP;            //the total power measured
Uint32 sampleRead; //place to store value read from codec
double sampleDbl;  //used to convert value read from codec to double

void main(){
    DSK6713_AIC23_CodecHandle hCodec;

    /* Initialize the board support library, must be called first */
    DSK6713_init();
      /* Initialize the LED module of the BSL */
    DSK6713_LED_init();

    /* Start the codec, set sample rate to 8kHz */
    hCodec = DSK6713_AIC23_openCodec(0, &config);
    DSK6713_AIC23_setFreq(hCodec, DSK6713_AIC23_FREQ_8KHZ);

    /* RTDX Setup */
    TARGET_INITIALIZE();
    RTDX_enableOutput(&ochan);
```

```
      /* Initialize everything to the starting parameters */
      #include "note_setup.h"
      for(f=0; f<NOTE_COUNT; f++){
            Z1[f]=0;
            Z2[f]=0;
            NF[f]=RF[f]*SAMPLE_PERIOD;
            RW[f]=2*cos(2*PI*NF[f]);
      }

      while(1){
            /* Run Goertzel Algorithm */
            for(i=0; i<BLOCK_SIZE; i++){
                  //read one channel
                  while(!DSK6713_AIC23_read(hCodec, &sampleRead)){}
                  sampleDbl = ((double)sampleRead)/16383.0;
                  for(f=0; f<NOTE_COUNT; f++){
                        S[f]  = sampleDbl + RW[f]*Z1[f] - Z2[f];
                        Z2[f] = Z1[f];
                        Z1[f] = S[f];
                  }
                  //compute the total power of the signal
                  //this will be used to normalize the measurements later
                  totP += sampleDbl*sampleDbl;
                  //read other channel & ignore it
                  while(!DSK6713_AIC23_read(hCodec, &sampleRead)){}
            }
            //this number was empirically determined
            //maximum normalized power squared must be more than:
            maxP = 50.0;
            maxNote = -1;
            for(f=0; f<NOTE_COUNT; f++){
                  //calculate power, dont bother taking the square root
                  //but do normalize it
                  PO[f]=(Z2[f]*Z2[f] + Z1[f]*Z1[f] - RW[f]*Z1[f]*Z2[f])/totP;
                  if(PO[f] > maxP){
                        maxP = PO[f];
                        maxNote = f;
                  }
                  //reset
                  Z1[f]=0.0;
                  Z2[f]=0.0;
            }
            totP = 0;

            //write to RTDX
            if ( !RTDX_write( &ochan, &maxNote, sizeof(maxNote) ) ) {
            //if ( !RTDX_write( &ochan, &maxP, sizeof(maxP) ) ) {
            //ERROR
        }
      }
      /* Stop RTDX */
      RTDX_disableOutput(&ochan);
    /* Close the codec */
    DSK6713_AIC23_closeCodec(hCodec);
}
```

P05 – VB Host Code

```
Option Explicit
'''''''''''''''''''''''''''''''''''''''''''''Utility Variables
Public connected As Integer                 '1 if RTDX channel has been
opened
Public lastMsg As String                    'for debugging purposes

'''''''''''''''''''''''''''''''''''''''''''''Needed for RTDX/CCStudio setup
Private CCSetup As Object                   'Used To Access CCStudio
functions
Private Boards As Object                    'List of Available boards
Private Board As Object                     'Board
Private Processors As Object                'List of available processors
Private Processor As Object                 'Processor
Public CurrentSelectedBoard As String       'The name of the board
Public CurrentSelectedProcessor As String   'Which processor on the board
Dim rtdx As Object                          'Acutal RTDX object

'''''''''''''''''''''''''''''''''''''''''''''RTDX OLE API Status Return codes
Const SUCCESS = &H0                         'Method call successful
Const FAIL = &H80004005                     'Method call failure
Const ENoDataAvailable = &H8003001E         'No data is currently available
Const EEndOfLogFile = &H80030002            'End of log file

'Utility function to output debugging messages to lower text area
Private Sub LogMessage(message As String)
    If lastMsg <> message Then
        tbStatus.SelStart = Len(tbStatus.Text)
        tbStatus.SelText = message & vbNewLine
        lastMsg = message
    End If
End Sub
```

```
'Utility function that takes the value read from the board, converts it into
'useful text and displays it to the user
'The program on the board was designed to send 0 through 8 to represent
'different notes and -1 when it could not make a decision
'This converts those integers to a more meaningful string
Private Sub ProcessData(ReadVal As Long)
    Dim NOTES() As Variant
    NOTES() = Array("C0", "C_0", "D0", "D_0", "E0", "F0", "F_0", "G0", "G_0",
"A0", "A_0", "B0", "C1", "C_1", "D1", "D_1", "E1", "F1", "F_1", "G1", "G_1",
"A1", "A_1", "B1", "C2", "C_2", "D2", "D_2", "E2", "F2", "F_2", "G2", "G_2",
"A2", "A_2", "B2", "C3", "C_3", "D3", "D_3", "E3", "F3", "F_3", "G3", "G_3",
"A3", "A_3", "B3", "C4", "C_4", "D4", "D_4", "E4", "F4", "F_4", "G4", "G_4",
"A4", "A_4", "B4", "C5", "C_5", "D5", "D_5", "E5", "F5", "F_5", "G5", "G_5",
"A5", "A_5", "B5", "C6", "C_6", "D6", "D_6", "E6", "F6", "F_6", "G6", "G_6",
"A6", "A_6", "B6", "C7", "C_7", "D7", "D_7", "E7", "F7", "F_7", "G7", "G_7",
"A7", "A_7", "B7", "C8", "C_8", "D8", "D_8")
    If ReadVal = -1 Then
        outputArea.SelStart = Len(outputArea.Text)
        outputArea.SelText = "-"
    End If
    If ReadVal > -1 Then
        outputArea.SelStart = Len(outputArea.Text)
        outputArea.SelText = NOTES(ReadVal) & ","
    End If
End Sub

'When the program starts
Private Sub Form_Load()
    connected = 0 'state that we don't have an RTDX channel open
    If (GetAvailableBoards) Then       ' Get Available Boards and Processors
        list_Boards.Selected(0) = True ' Set the selected board to 0
    End If
End Sub

'When the program ends
Private Sub Form_Unload(Cancel As Integer)
    If connected = 1 Then       'If we have the RTDX channel open
        DisconnectFromBoard     'Close the RTDX channel
    End If
    Set CCSetup = Nothing       'Cleanup
    Set Boards = Nothing        'Cleanup
    Set Board = Nothing         'Cleanup
    Set Processors = Nothing    'Cleanup
    Set Processor = Nothing     'Cleanup
End Sub
```

```
'Function to update the list of boards that can be connected to
Private Function GetAvailableBoards() As Boolean
    Dim status As Long
    Dim BoardName As String

    ' Initialize Lists
    list_Boards.Clear
    list_Processors.Clear

    ' Instantiate the Code Composer Setup SystemSetup coclass and obtain a
    ' pointer to the ISystemSetup interface
    Set CCSetup = CreateObject("CodeComposerSetup.SystemSetup")

    ' Get a pointer to the IBoards interface
    status = CCSetup.GetBoards(Boards)

    ' Loop through the available boards, get the names of the boards,
    ' and add the board names to the boards list control
    For Each Board In Boards
        ' Get the board name
        status = Board.GetName(BoardName)

        ' Append board name to the board list
        list_Boards.AddItem (BoardName)
    Next

    ' return True
    GetAvailableBoards = True

End Function

'Function to update the list of processors which can be used
Private Function GetAvailableProcessors(SelectedBoardName As String) As
Boolean
    Dim status As Long
    Dim ProcessorName As String
    ' Get a pointer to the IBoard interface for the selected
    ' board
    status = CCSetup.GetBoardByName(SelectedBoardName, Board)
    ' Get a pointer to the IProcessors interface
    status = Board.GetProcessors(Processors)
    ' Loop through the available processors, get the names of the
    ' processors, and add the processors to the processors list
    ' control
    For Each Processor In Processors
        ' Get the processor name
        status = Processor.GetName(ProcessorName)
        ' Append processor name to the processor list
        list_Processors.AddItem (ProcessorName)
    Next
    ' Return True
    GetAvailableProcessors = True
End Function
```

```
'When the "connect" button is clicked
Private Sub btConnect_Click()
    If connected = 1 Then       'If we have the RTDX channel open
        DisconnectFromBoard     'Close the RTDX channel
    End If
    ConnectToBoard              'Open the RTDX channel
    btConnect.Enabled = False   'Disable the "connect" button
    btDisconnect.Enabled = True 'Enable the "disconnect" button
End Sub

'When the "disconnect" button is clicked
Private Sub btDisconnect_Click()
    DisconnectFromBoard          'Close the RTDX channel
    btDisconnect.Enabled = False 'Disable the "disconnect" button
    btConnect.Enabled = True     'Enable the "connect" button
End Sub

Private Sub btListRefresh_Click()
    ' Get Available Boards and Processors
    If (GetAvailableBoards) Then
        ' Set the selected board to 0
        list_Boards.Selected(0) = True
    End If
End Sub

'When a board is selected
Private Sub list_Boards_Click()
    ' Clear processor list
    list_Processors.Clear
    ' Get current selected board
    CurrentSelectedBoard = list_Boards.List(list_Boards.ListIndex)
    ' Get available processors for that board
    If (GetAvailableProcessors(CurrentSelectedBoard)) Then
        ' Set the selected processor to 0
        list_Processors.Selected(0) = True
    End If
End Sub

'When a processor is selected
Private Sub list_Processors_Click()
    ' Get current selected processor
    CurrentSelectedProcessor =
list_Processors.List(list_Processors.ListIndex)
End Sub
```

```
'Function which tries to open the RTDX channel on the selected board/proc
Private Sub ConnectToBoard()
  LogMessage "Info  - Attempting to connect to board " & CurrentSelectedBoard
  LogMessage "Info  - Attempting to connect to processor " &
CurrentSelectedProcessor
   Dim status As Long
   ' Get application objects
   Set rtdx = CreateObject("RTDX")
   status = rtdx.SetProcessor(CurrentSelectedBoard, CurrentSelectedProcessor)
   If (status <> SUCCESS) Then
        LogMessage "Error - Set Processor failed"
        Exit Sub
   End If

    'open target's input channel
    '"ochan" must agree with RTDX_CreateOutputChannel(ochan);
    'from target source code
    status = rtdx.Open("ochan", "R")
    Select Case status
    Case Is = SUCCESS
        connected = 1
        LogMessage "Info  - Opened RTDX channel for reading"
    Case Is = FAIL
        LogMessage "Error - Unable to open channel for RTDX communications"
        Exit Sub
    Case Else
        LogMessage "Info  - Unknown return value from openning RTDX channel"
        Exit Sub
    End Select
End Sub

'Function which tries to close the RTDX channel on the selected board/proc
Private Sub DisconnectFromBoard()
    Dim status As Long
    ' close target's input channel
    status = rtdx.Close()
    Select Case status
        Case Is = SUCCESS
          LogMessage "Info  - Successfully closed RTDX channel"
        Case Is = FAIL
          LogMessage "Error - Unable to close RTDX channel"
        Case Else
          LogMessage "Info  - Unknown return value from closing RTDX channel"
    End Select
    connected = 0
    Set rtdx = Nothing                     ' kill RTDX OLE object
End Sub
```

```
'Function to check for new RTDX data every XX milliseconds
Private Sub Timer1_Timer()
    'Dim ReadValue As Variant
    Dim ReadValue As Long 'rtdx.ReadI4(ReadValue)
    'Dim ReadValue As Int  'rtdx.ReadI2(ReadValue)
    'Dim ReadValue As Single 'rtdx.ReadF4(ReadValue)
    'Dim ReadValue As Double  'rtdx.ReadF8(ReadValue)
    Dim status As Long
    If connected = 1 Then
        'status = rtdx.ReadSAI2(ReadValue)
        'status = rtdx.ReadSAF4(ReadValue)
        'Do
            status = rtdx.ReadI4(ReadValue)
            'status = rtdx.ReadF8(ReadValue)
            Select Case status
                Case Is = SUCCESS
                    ProcessData ReadValue
                    'LogMessage "Data: " & ReadValue
                Case Is = FAIL
                    LogMessage "Error - Reading data failed " & ReadValue
                Case Is = ENoDataAvailable
                    LogMessage "Error - No data available"
                Case Is = EEndOfLogFile
                    LogMessage "Info  - Reached end of log file"
                Case Else
                   LogMessage "Error - Unknown error reading RTDX channel - "
& ReadValue
            End Select
        'Loop Until status = EEndOfLogFile
    End If
End Sub
```

Goertzel Algorithm Simulation Web Page Code:

```
<!DOCTYPE html>
<html lang="en" >
 <head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
    <meta name="description" content="Goertzel Algorithm Simulation">
    <meta name="author" content="Andrew Ippoliti">
    <title>Goertzel Stuff</title>
    <script language="javascript" type="text/javascript"
      src="./flot/jquery.js"></script>
    <script language="javascript" type="text/javascript"
      src="./flot/jquery.flot.js"></script>
    <script language="javascript" type="text/javascript"
      src="./jquery.flot.navigate.js"></script>
      <style type="text/css">
            #container {width: 640px; margin: auto; background-color: #FFF;}
            body {background-color: #FFF; text-align: center;}
      </style>
 </head>
    <body><div id="container">
    <h1>Goertzel Demonstration</h1>
      <div><label for="IN-SIGNAL">Signal:</label>
           <textarea id="IN-SIGNAL">cos(2*PI*440*t)</textarea></div>
      <div><label for="IN-SAMP-FREQ">Sampling Frequency (Hz):</label>
           <input type="text" value="8000" id="IN-SAMP-FREQ"/></div>
      <div><label for="IN-TEST-FREQ">Test Frequency (Hz):</label>
           <input type="text" value="440" id="IN-TEST-FREQ"/></div>
      <div><label for="IN-NUM-SAMP">Number of Samples:</label>
           <input type="text" value="1660" id="IN-NUM-SAMP"/></div>
      <div><button id="BT-UPDATE" style="width:100%;"
           onclick="UPDATE();">Update</button></div>
      <H2>The time plot of the signal:</H2>
      <div id="timeplot" style="width:480px;height:240px"></div>
      <H2>The progression of the Goertzel Algorthim:</H2>
      <div id="GoertzelRT" style="width:480px;height:240px"></div>
      <div>Real Time, Low Memory Output Magnitude:
           <span id="OUT-RT-MAG"></span></div>
      <div>The Block Goertzel Ouput Magnitude:
           <span id="OUT-BLOCK-MAG"></span></div>
    </div>
```

```
<script type="text/javascript">
/* The following code is written by Andrew Ippoliti
 * It depends on flot (not written by Andrew) for graphing
 */
function calcTime(signal,sampFreq,sampleCount){
    var T = 1/sampFreq;
    /* utility funcs */
    var sin=Math.sin;
    var cos=Math.cos;
    var PI=Math.PI;
    function u(t){return t>0};
    function SqW(f,d,t){
        var T = 1/f;
        if((t%T)/T<d){
            return 0.5;
        }else{
            return -0.5;
        }
    };
    /* optimize function so it it "compiled" */
    eval("var func = function(t){ return "+signal+";}");
    var d1 = [];
    var t=0;
    for (var i = 0; i < sampleCount; i+=1){
        t += T;
        d1.push([t, func(t)]);
    }
    return d1;
}
/* GoertzelBlock - Applies Goertzel Algorithm to a Block of Data
 * data is an array of [time,sample] pairs
 * testFreq is the frequency that is being tested for
 * sampleFreq is the rate at which the data was sampled
 * procLen is the number of samples that should be processed */
function GoertzelBlock(data, testFreq, sampleFreq, procLen) {
    /* Z1 is the previous value, Z2 is the previous previous value
     * S is the current value, i is the iteration */
    var Z1=0.0, Z2=0.0, S = 0.0, i = 0;
    /* Compute the normalized frequency that we are looking for */
    var nomFreq = testFreq / sampleFreq;
    /* realW is the real part of the complex exponential */
    var realW = 2*Math.cos(2*Math.PI*nomFreq);
    /* imagW is the imaginary part of the complex exponential,
     * Note: imagW is not needed for the magnitude computation */
    var imagW = 1*Math.sin(2*Math.PI*nomFreq);
    /* iterate over all samples */
    for (i=0; i<procLen; i++) {
        S = data[i][1] + realW * Z1 - Z2; //calculate current
        Z2 = Z1; //update previous previous
        Z1 = S;  //update previous
    }
    /* Return the power, this isn't normalized so it can be rather high */
    return Math.sqrt(Z2*Z2+Z1*Z1-realW*Z1*Z2);
}
```

```
function GenRTGF(data, testFreq, sampleFreq, procLen) {
    var Z1=0.0, Z2=0.0, S = 0.0, i = 0;
    var nomFreq = testFreq / sampleFreq;
    var realW = 2*Math.cos(2*Math.PI*nomFreq);
    var imagW = 1*Math.sin(2*Math.PI*nomFreq);
    /* power is the 'output' of the algorithm at a particular iteration */
    var power = 0;
    /* powTotal can be used to compute the acutal power in the signal
     * which can then be used to normalize the power computed */
    var powTot = 0;
    /* outData will store an array of [iteration, output] pairs so that
     * the data can be graphed, instead of just outputting the final value */
    var outData = [];
    for (i=0; i<procLen; i++) {
        S = data[i][1] + realW * Z1 - Z2; //calculate current
        Z2 = Z1; //update previous previous
        Z1 = S;  //update previous
        /* calculate the power */
        power = Z2*Z2+Z1*Z1-realW*Z1*Z2;
        /* increment total power */
        powTot += data[i][1]*data[i][1];
        /* add output to array */
        outData.push([i,Math.sqrt(power)]);
        /* for normailized power */
        //power = Math.sqrt(power/(powTot*i));
        //outData.push([i,power]);
    }
    /* Return the power, this isn't normalized so it can be rather high */
    return outData;
}

function UPDATE(){
    var signal = document.getElementById("IN-SIGNAL").value;
    var sampleRate = parseInt(document.getElementById("IN-SAMP-FREQ").value);
    var testFreq = parseInt(document.getElementById("IN-TEST-FREQ").value);
    var sampleCount = parseInt(document.getElementById("IN-NUM-SAMP").value);
    var timeData = calcTime(signal,sampleRate,sampleCount);
    $.plot($("#timeplot"), [
        {
            data: timeData,
            label: "Time Signal",
            color: "#0000CC",
            lines: { show: true, steps: true }
        }
    ]);
    var gbOut = GoertzelBlock(timeData,testFreq,sampleRate,timeData.length);
    document.getElementById("OUT-BLOCK-MAG").innerHTML = gbOut;
    var gbRT = GenRTGF(timeData,testFreq,sampleRate,timeData.length);
    document.getElementById("OUT-RT-MAG").innerHTML = gbRT[gbRT.length-1][1];
    var gbRT1 = GenRTGF(timeData,testFreq*Math.pow(2,
        1/12),sampleRate,timeData.length);
    var gbRT2 = GenRTGF(timeData,testFreq*Math.pow(2,
        -1/12),sampleRate,timeData.length);
    $.plot($("#GoertzelRT"), [
        {
            data: gbRT1,
            label: "Half-Step Above: "+
                    Math.round(testFreq*Math.pow(2,1/12))+"Hz",
            color: "#CC0000",
            lines: { show: true, steps: true }
        },
        {
            data: gbRT,
            label: "Test Frequency: "+testFreq+"Hz",
            color: "#333333",
            lines: { show: true, steps: true }
        },
        {
            data: gbRT2,
            label: "Half-Step Below: "+
                   Math.round(testFreq*Math.pow(2,-1/12))+"Hz",
            color: "#00CC00",
            lines: { show: true, steps: true }
        }],
        {legend: { position: 'nw' }}
    );
}
</script>

</body>
</html>
```