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

[Abstract 2](#_Toc311109836)

[Requirements 2](#_Toc311109837)

[Generating Tones 3](#_Toc311109838)

[Analyzing Tones via the Fourier Transform 5](#_Toc311109839)

[Frequency Analysis via the Goertzel Algorithm 6](#_Toc311109840)

[Detecting Notes with the C6713 DSK 10](#_Toc311109841)

[Communicating with the Host Computer via RTDX 14](#_Toc311109842)

[Extending the Possibilities – Detecting More Frequencies 22](#_Toc311109843)

[Extending the Possibilities – Detecting Different Instruments 23](#_Toc311109844)

[Extending the Possibilities – More Complicated Songs 25](#_Toc311109845)

[Included Files 28](#_Toc311109846)

[Credits 29](#_Toc311109847)

[Bibliography 29](#_Toc311109848)

[License 29](#_Toc311109849)

[Source Code 30](#_Toc311109850)

[Notes.h – Used in Various CCStudio Projects 30](#_Toc311109851)

[P01 - Tone.c – Generates “DO RE MI FA SO LA TI DO” on C6713 DSK 32](#_Toc311109852)

[P02 - Tone.c – Runs Goertzel Algorithm, Outputs Data to LEDs 33](#_Toc311109853)

[P03 - Target.h – Used in RTDX Programs on Target, Made by TI 35](#_Toc311109854)

[P03 - Tone.c – Runs Goertzel, Outputs to LEDs and RTDX 36](#_Toc311109855)

[P03 - Visual Basic 1 – Source Code of First VB Host App 38](#_Toc311109856)

[P04 - Sequence.h – Plays All Notes From A0 to D#8 43](#_Toc311109857)

[P04 - Sequence\_deck.h – Plays “Deck the Halls” 45](#_Toc311109858)

[P04 - Tone.c – Plays a Sequence of Notes 48](#_Toc311109859)

[P05 – Note\_setup.h – Sets up Notes that will be Listened for 49](#_Toc311109860)

[P05 – Tone.c – Target Code 51](#_Toc311109861)

[P05 – VB Host Code 53](#_Toc311109862)

[Goertzel Algorithm Simulation Web Page Code: 59](#_Toc311109863)

# 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 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 fn of a note n half-steps away from a reference note of frequency f0 is f0 times the a to the n. The symbol a is the twelfth root of 2 or 21/12. Musicians typically select A4 with frequency 440Hz as the reference note. That equation was used to generate all of the frequencies in “notes.h” as #define noteName noteFrequency.



Equation - Musical Note Frequencies

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

|  |  |  |  |
| --- | --- | --- | --- |
| AllowedNotes.png | | | |
| **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 - 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 3.1415926535897932384626433832795  /\* 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 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 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. Note that N is the total number of samples and k is the index of the frequency bin.



Equation - 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.

|  |
| --- |
| #define PI 3.1415926535897932384626433832795  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 - 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 . The realization of the difference equation is shown in Figure 1 - Goertzel Algorithm Realization.

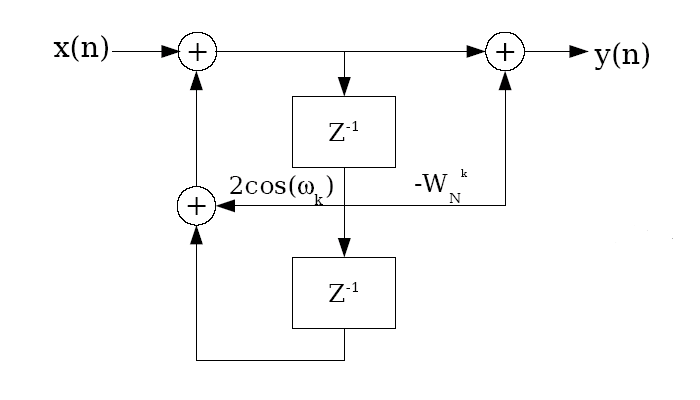


Figure - Goertzel Algorithm Realization

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 fS 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 . 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 - 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 - Simulation of Detecting a 1760 Hz Component in a Pure 1760 Hz Cosine Wave

As shown in Table 1 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.

|  |  |
| --- | --- |
| **Simulation #2 – Detecting A1 in a Pure A1 Tone** | |
| Sampling Frequency (Hz) | 8000 |
| Test Frequency (Hz) | 55 |
| Number of Samples | 1600 |
| Input Signal: cos(2\*PI\*55\*t) | |
|  | |
| Progression of the Algorithm | |
|  | |
| Algorithm Output | 799.9999999999801 |

Table - 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 2 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 1 and Table 2. 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 - 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 3.1415926535897932384626433832795  /\* 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 - 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 - 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 - 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.

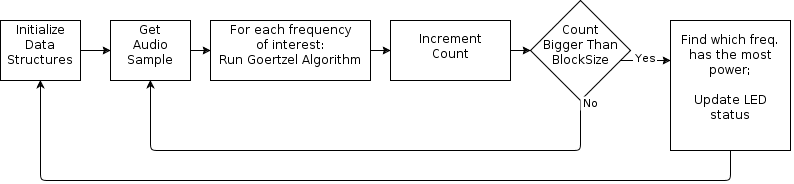


Figure - Main Processing Loop

|  |
| --- |
| /\* 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 - Main Processing Loop Code

Table 3 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 - 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 4 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 - 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 - 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 - 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 - 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 - 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 - 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.

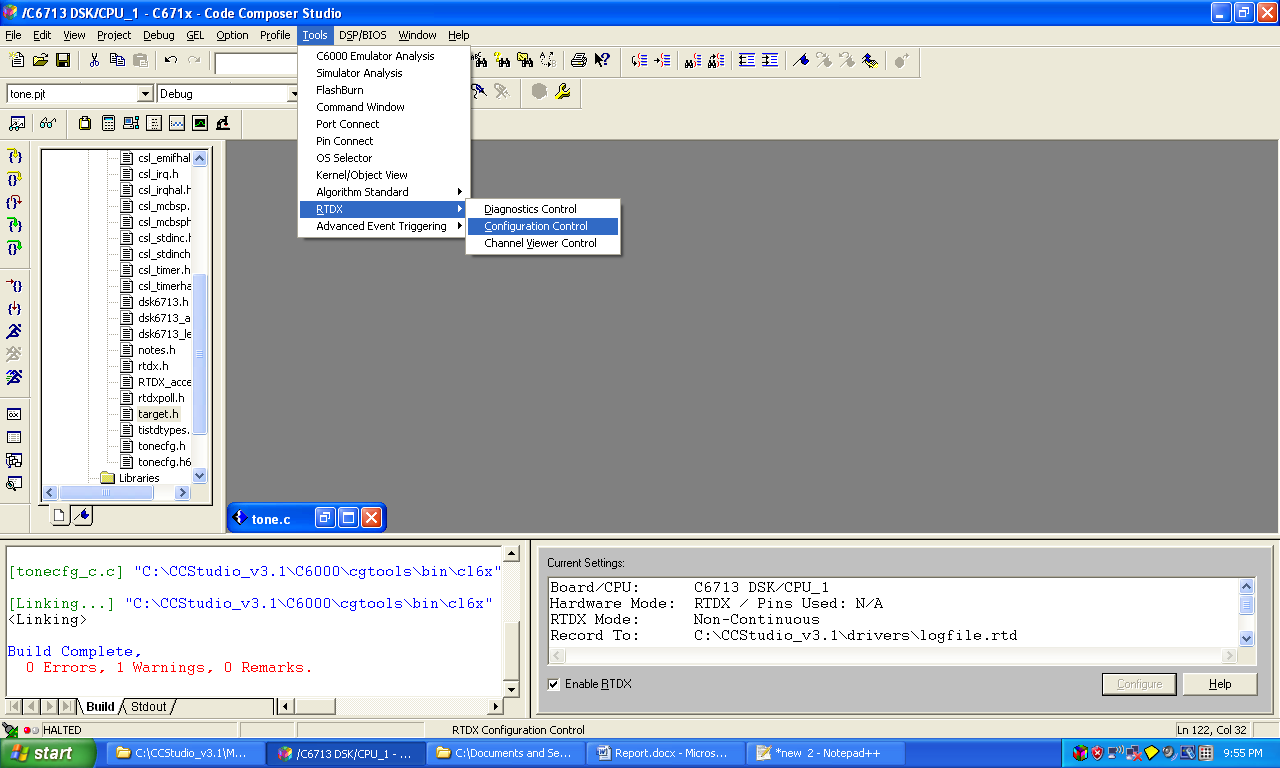


Figure - Accessing the RTDX Configuration Control

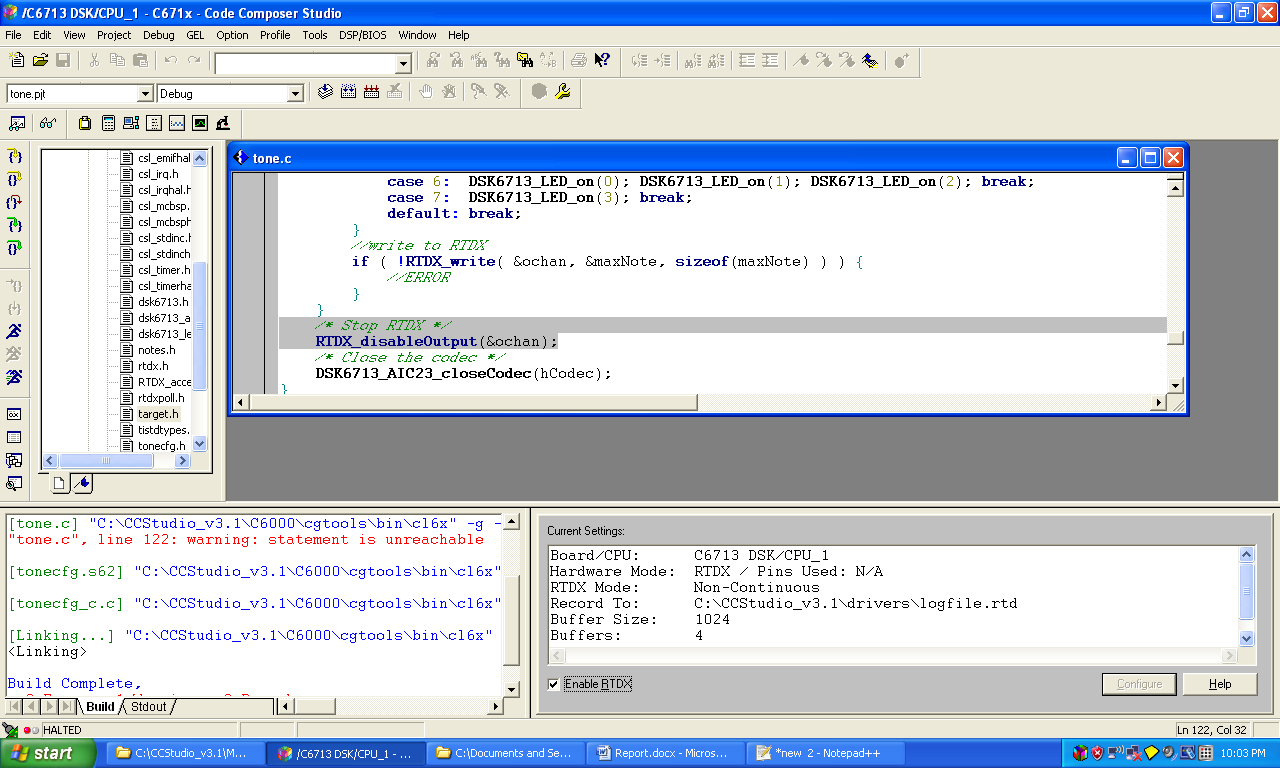


Figure - RTDX Current Settings Panel

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.

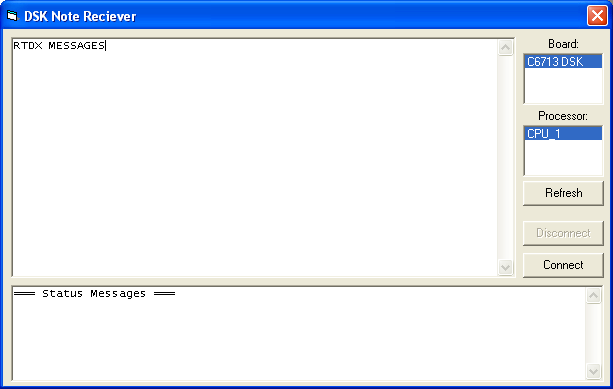


Figure - Interface for Program which will Receive RTDX Data

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 - 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 - 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 - 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 - 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 - 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 - 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 - 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 - 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 - 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 - 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 - 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 – 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 - 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.

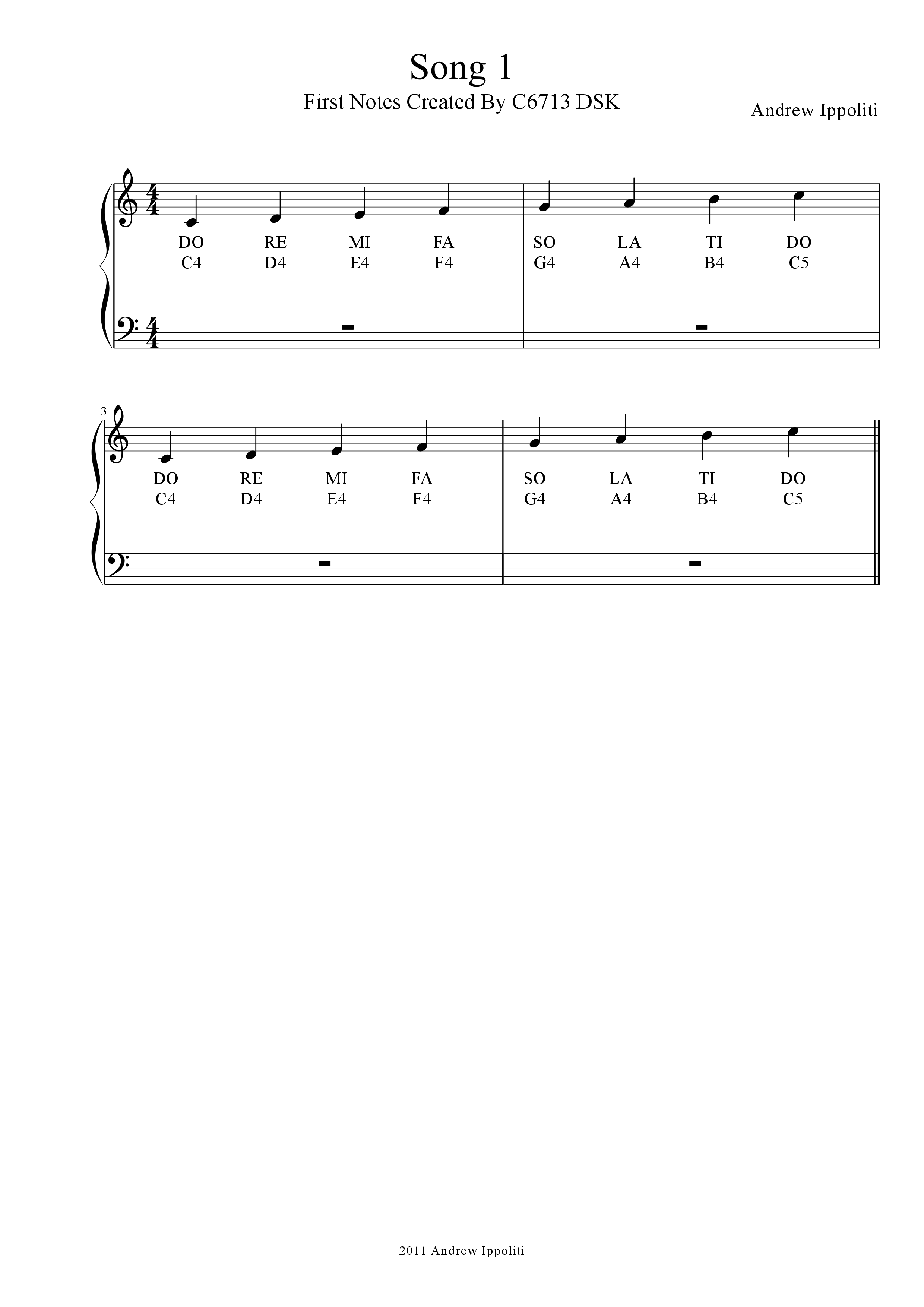


Figure - Simple Melody

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **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 - 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 - 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.

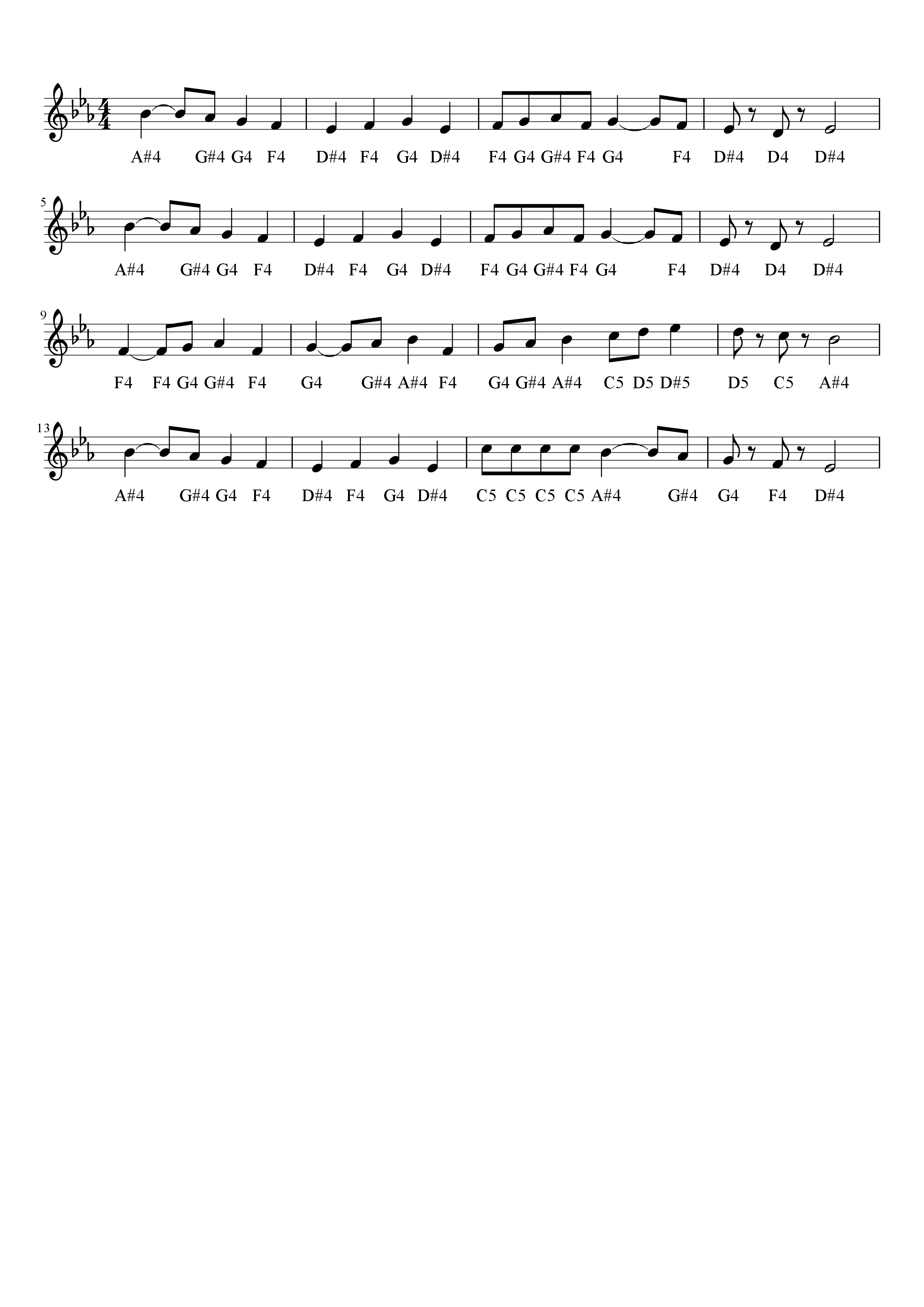


Figure - Musical Score for "Deck the Halls"

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **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 - 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 - 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 - 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 - 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 - 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](AudioSamples/Index.html) 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](SheetMusic) 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](Simulation/Index.html) 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](CCStudioFiles) 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](CCStudioFiles/P01_TONE) has the CCStudio Files used to generate a tone on the C6713 DSK.

[Project 2](CCStudioFiles/P02_GOERTZEL) has the CCStudio Files that implement the Goertzel Algorithm and output the results to the C6713 DSK LEDs.

[Project 3](CCStudioFiles/P03_VB_NOTE) has the CCStudio Files that implement the Goertzel Algorithm and output the result via RTDX. The [VB NOTE](CCStudioFiles/P03_VB_NOTE/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](CCStudioFiles/P04_ALL_NOTE) has the files that output more complicated “songs” from the DSK.

[Project 5](CCStudioFiles/P05_VB_ALL_NOTE) 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

Most of the code is © Andrew Ippoliti 2011, all rights reserved and all liabilities disclaimed. Some portions of code are property of Texas Instruments and/or others.

# 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 3.1415926535897932384626433832795

/\* 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 3.1415926535897932384626433832795

/\* 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 3.1415926535897932384626433832795

/\* 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 succussful

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 3.1415926535897932384626433832795

/\* 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

#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 3.1415926535897932384626433832795

/\* 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 succussful

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>