
DSP First
Lab 04: Synthesis of Sinusoidal Signals - Music Synthesis

Pre-Lab and Warm-Up: You should read at least the Pre-Lab and Warm-up sections of this lab assignment and go over all exercises in the Pre-Lab section before going to your assigned lab session.

Verification: The Warm-up section of each lab must be completed **during your assigned Lab time** and the steps marked *Instructor Verification* must also be signed off **during the lab time**. One of the laboratory instructors must verify the appropriate steps by signing on the **Instructor Verification** line. When you have completed a step that requires verification, simply demonstrate the step to the TA or instructor. Turn in the completed verification sheet to your TA when you leave the lab.

Lab Report: It is only necessary to turn in a report on Section 4 with graphs and explanations. You are asked to **label** the axes of your plots and include a title for every plot. In order to keep track of plots, include your plot *inlined*¹ within your report. If you are unsure about what is expected, ask the TA who will grade your report.

1 Introduction

This lab includes a project on music synthesis with sinusoids. The piece, *Fugue #2 for the Well-Tempered Clavier* by Bach² has been selected for doing the synthesis program. The project requires an extensive programming effort and should be documented with a complete **formal** lab report. A good report should include the following items: a cover sheet, commented LabVIEW code, explanations of your approach, conclusions and any additional tweaks that you implemented for the synthesis. Since the project must be evaluated by listening to the quality of the synthesized song, the criteria for judging a good song are given at the end of this lab description. If you would like to try other songs, the **DSP First CD-ROM** includes information about alternative tunes: *Minuet in G*, *Fur Elise*, *Beethoven's Fifth*, *Jesu*, *Joy of Man's Desiring* and *Twinkle, Twinkle, Little Star*. The music synthesis will be done with sinusoidal waveforms of the form

$$x(t) = \sum_k A_k \cos(\omega_k t + \phi_k) \quad (1)$$

so it will be necessary to establish the connection between musical notes, their frequencies, and sinusoids. A secondary objective of the lab is the challenge of trying to add other features to the synthesis in order to improve the subjective quality for listening. Students who take this challenge will be motivated to learn more about the spectral representation of signals — a topic that underlies this entire course.

¹ Remember, if you want to include a waveform plot, right-click and select **Data Operations»Export Simplified Image...**

²See <http://www.naxos.com/mainsite/default.asp?pn=Composers&char=B&ComposerID=52>.

2 Pre-Lab

In this lab, the periodic waveforms and music signals will be created with the intention of playing them out through a speaker (or headphones). Therefore, it is necessary to take into account the fact that a conversion is needed from the digital samples, which are numbers stored in the computer memory to the actual voltage waveform that will be amplified for the speakers.

2.1 LabVIEW Review

You learned several LabVIEW techniques during the last lab. Here is a list of some of them: How to use probes.

- How to convert a VI to a sub VI
 - Adding terminals to a sub VI
 - Adding a sub VI to a block diagram
- How to use a for loop
 - Iterating over an array
 - Using N to specify the number of iterations
 - Using i
 - Using shift registers to carry values from one iteration to another.

2.2 Theory of Sampling

Chapter 4 treats sampling in detail, but this lab is usually done prior to lectures on sampling, so we provide a quick summary of essential facts here. The idealized process of sampling a signal and the subsequent reconstruction of the signal from its samples is depicted in Figure 1.

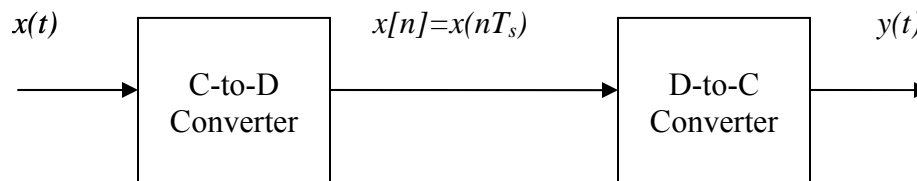


Figure 1: Sampling and reconstruction of a continuous-time Signal.

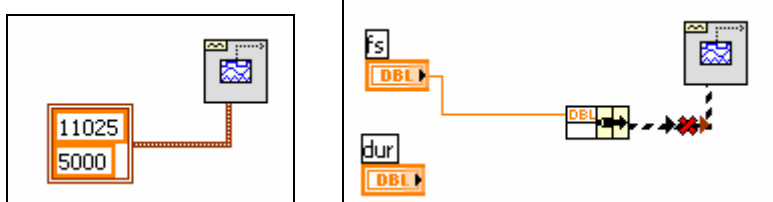
This figure shows a continuous-time input signal $x(t)$, which is sampled by the continuous-to-discrete (C-to-D) converter to produce a sequence of samples $x[n]=x(nT_s)$, where n is the integer sample index and T_s is the sampling period. The sampling rate is $f_s=1/T_s$ where the units are samples per second. As described in Chapter 4 of the text, the ideal discrete-to-continuous (D-to-C) converter takes the input samples and interpolates a smooth curve between them. The *Sampling Theorem* tells us that if the input signal $x(t)$ is a sum of sine waves, then the output $y(t)$ will be equal to the input $x(t)$ if the sampling rate is more than twice the highest frequency f_{max} in the input, i.e., $f_s > 2 f_{max}$. In other words, if we *sample fast enough* then there will be no problems synthesizing the continuous audio signals from $x[n]$.

2.3 D-to-A Conversion

Most computers have a built-in analog-to-digital (A-to-D) converter and a digital-to-analog (D-to-A) converter (usually on the sound card). These hardware systems are physical realizations of the idealized concepts of C-to-D and D-to-C converters respectively, but for purposes of this lab we will assume that the hardware A/D and D/A are perfect realizations.

The digital-to-analog conversion process has a number of aspects, but in its simplest form the only thing we need to worry about at this point is that the time spacing (T_s) between the signal samples must correspond to the rate of the D-to-A hardware that is being used. From LabVIEW, the sound output is done by the **soundcs** VI or the **tripleDisplay** VI which both support a variable D-to-A sampling rate if the hardware on the machine has such capability. A convenient choice for the D-to-A conversion rate is 11025 samples per second³, so $T_s = 1/11025$ seconds; another common choice is 8000 samples/sec. Both of these rates satisfy the requirement of *sampling fast enough* as explained in the next section. In fact, most piano notes have relatively low frequencies, so an even lower sampling rate could be used. If you are using **soundcs** or **tripleDisplay**, the input will be scaled automatically for the D-to-A converter.

- a) The ideal C-to-D converter is, in effect, being implemented whenever we take samples of a continuous-time formula, e.g., $x(t)$ at $t = t_n$. This is done for us in LabVIEW by the **Function Generator VI**. It first makes a vector of times, and then evaluates the formula for the continuous-time signal at the sample times, i.e., $x[n] = x(nT_s)$ if $t_n = nT_s$. This assumes perfect knowledge of the input signal, but we have already been doing it this way in previous labs. To begin, create a vector x_1 of samples of a sinusoidal signal with $A_1=100$, $\omega_1=2\pi(800)$, and $\phi_1=-\pi/3$. Use a sampling rate of 11025 samples/second, and compute a total number of samples equivalent to a time duration of 0.5 seconds. You may find it helpful to recall that in LabVIEW the **sampling info** input lets you set the sampling rate and number of samples.

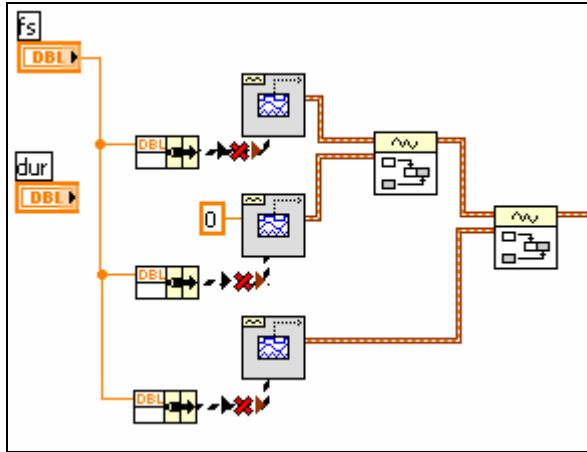


In the left diagram the sampling rate is set to 11025 and there are 5000 samples. The right diagram shows how to use a **Bundle**⁴ to access the sampling rate and number of samples separately. You need to figure out how to calculate the number of samples given fs and dur . You should use the `sin_syn` function from a previous lab for this part. Use **tripleDisplay** to look at and play the resulting vector through the D-to-A converter of the your computer, assuming that the hardware can support the $fs = 11025$ Hz rate. Listen to the output.

- b) Now create another vector x_2 of samples of a second sinusoidal signal (0.8 secs. in duration) for the case $A_2=80$, $\omega_2=2\pi(1200)$, and $\phi_2=+\pi/4$. Listen to the signal reconstructed from these samples. How does its sound compare to the signal in part (a)?
- c) **Concatenate** the two signals x_1 and x_2 with a short duration of 0.1 seconds of silence in between. You should be able to use something like:

³ This sampling rate is one quarter of the rate (44,100 Hz) used in audio CD players.

⁴ Hint: right-click the background of the block diagram and then click **Search**. Type **Bundle** to find it.



where the right-most two blocks are **Append Waveforms VIs**⁵. Note the middle function generator has amplitude of **0**. Determine the correct value of **dur** to make 0.1 seconds of silence. Listen to this new signal to verify that it is correct.

- d) To verify that the concatenation operation was done correctly in the previous part, attach a **tripleDisplay**. This will plot a huge number of points, but it will show the “envelope” of the signal and verify that the amplitude changes from 100 to zero and then to 80 at the correct times.

2.4 Debugging Skills

Testing and debugging code is a big part of any programming job, as you know if you have been staying up late on the first labs. Almost any modern programming environment provides a *debugger* so that break-points can be set and variables examined in the middle of program execution. Of course, many programmers insist on using the old-fashioned method of inserting print statements in the middle of their code (which is rather hard in a graphical language like LabVIEW). This is akin to riding a tricycle to commute around Atlanta.

Do the exercise in the **Debugging** folder to learn some simple, yet powerful techniques.

2.5 Piano Keyboard

Section 4 of this lab consists of synthesizing the notes of a well known piece of music.⁶ Since these signals require sinusoidal tones to represent piano notes, a quick introduction to the frequency layout of the piano keyboard is needed. On a piano (Figure 2), the keyboard is divided into octaves--the notes in one octave being twice the frequency of the notes in the next

⁵ You can find this VI by right-clicking on the background of the block diagram view and selecting **Search**.

lower octave.

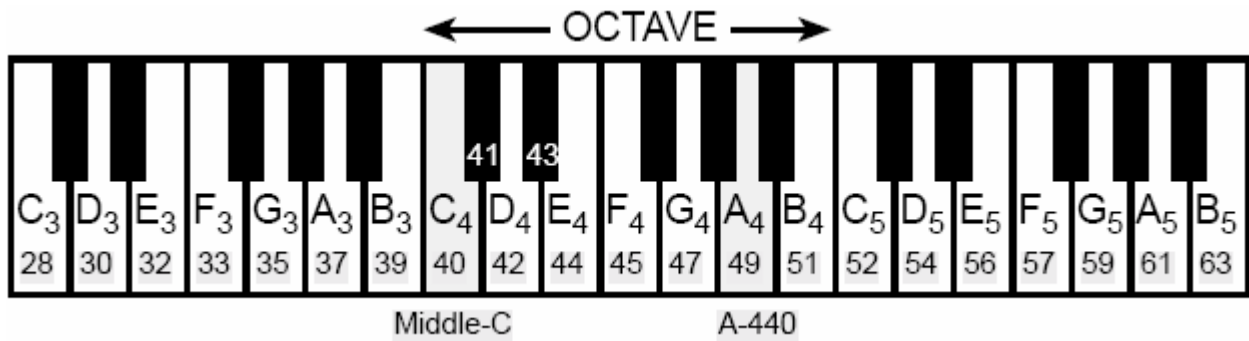


Figure 2 Layout of a piano keyboard. Key numbers are shaded. The notation C₄ means the C-key in the fourth octave.

For example, the reference note is the A above middle-C which is usually called A-440 (or A₄) because its frequency is 440 Hz. (In this lab, we are using the number 40 to represent middle C. This is somewhat arbitrary; for instance, the Musical Instrument Digital Interface (MIDI) standard represents middle C with the number 60). Each octave contains 12 notes (5 black keys and 7 white) and the ratio between the frequencies of the notes is constant between successive notes. As a result, this ratio must be $2^{1/12}$. Since middle C is 9 keys below A-440, its frequency is approximately 261 Hz. Consult chapter 9 for even more details.

Musical notation shows which notes are to be played and their relative timing (half, quarter, or eighth). Figure 3 shows how the keys on the piano correspond to notes drawn in musical notation. The white keys are all labeled as A, B, C, D, E, F, and G; but the black keys are denoted with “sharps” or “flats.” A sharp such as A[#] is one key number larger than A; a flat is one key lower, e.g., A₄^b is key number 48.

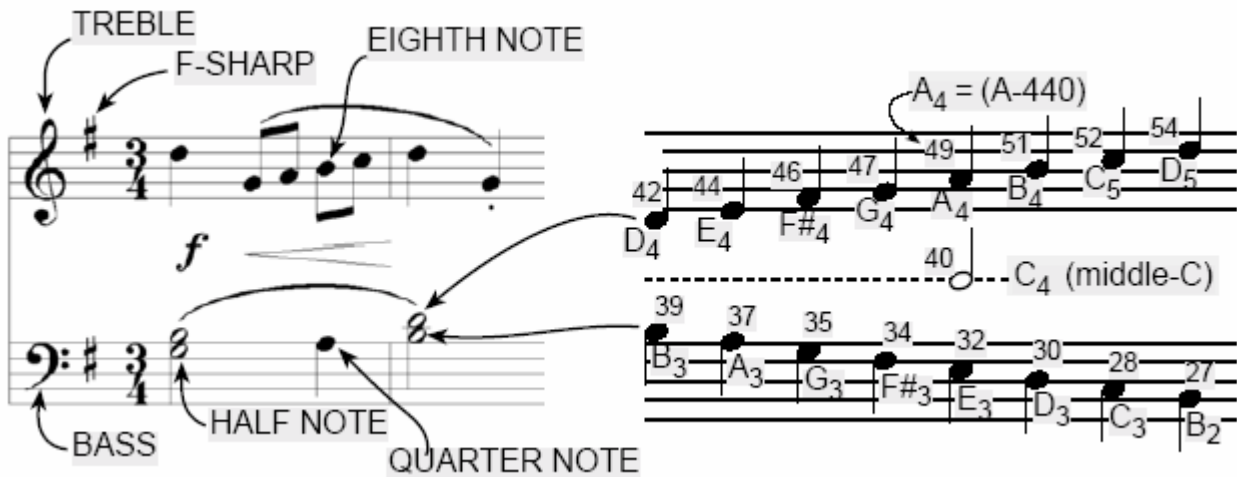


Figure 3 Musical notation is a time-frequency diagram where vertical position indicates which note is to be played. Notice that the shape of the note defines it as a half, quarter or eighth note, which in turn defines the duration of the sound.

⁶ If you have little or no experience reading music, don't be intimidated. Only a little music knowledge is needed to carry out this lab. On the other hand, the experience of working in an application area where you must quickly acquire knowledge is a valuable one. Many real-world engineering problems have this flavor, especially in signal processing which has such a broad applicability in diverse areas such as geophysics, medicine, radar, speech, etc.

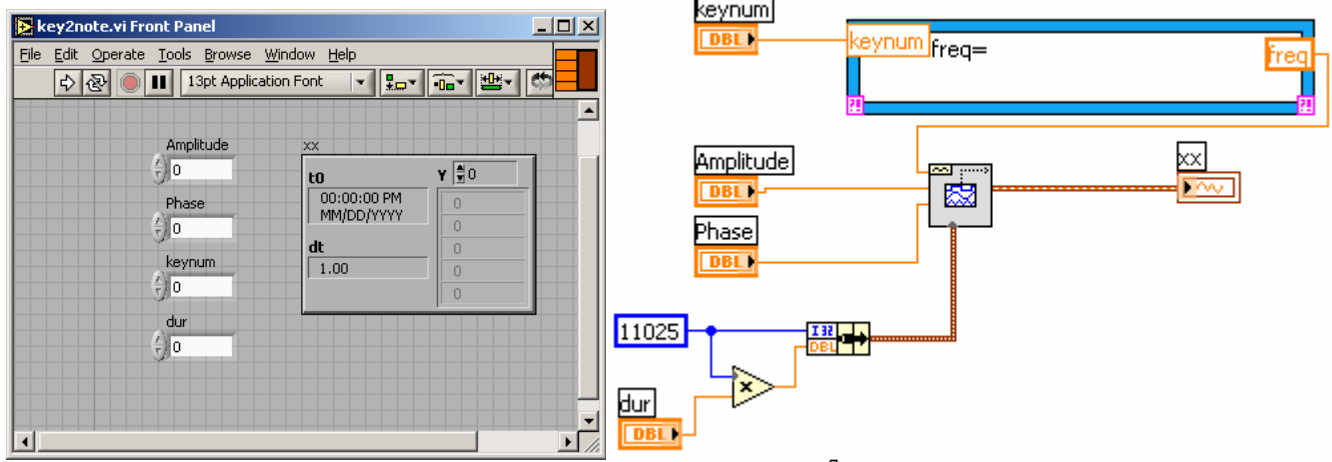
Another interesting relationship is the ratio of fifths and fourths as used in a chord. Strictly speaking the fifth note should be 1.5 times the frequency of the base note. For middle-C the fifth is G, but the frequency of G is about 392 Hz which is not exactly 1.5 times 261.6. It is very close, but the slight detuning introduced by the ratio $2^{1/12}$ gives a better sound to the piano overall. This innovation in tuning is called “equally-tempered” or “well-tempered” and was introduced in Germany in the 1760's and made famous by J. S. Bach in the “Well Tempered Clavier,” which is the source of the song for this lab.

You can use the ratio $2^{1/12}$ to calculate the frequency of notes anywhere on the piano keyboard. For example, the E-flat above middle-C (black key number 43) is 6 keys below A-440, so its frequency should be $f_{43} = 440 \times 2^{-6/12} = 440 / \sqrt{2} \approx 311$ Hertz.

3 Warm-up

3.1 Note Frequency Function

Now write a VI to produce a desired note for a given duration. Your VI should be in the form of a sub VI called `key2note.vi`. Your VI should have the following form:



The block to the right of **keynum** is a **MathScript Node** block⁷. Once you have placed the MathScript Node on the block diagram, right-click on the left edge and select **Add Input** and type **keynum** to create the keynum input. Do the same select **Add Output** on the right side to add the **freq** output. Type in this block the formula given above to determine the frequency for a sinusoid in terms of its key number. You should start from a reference note (middle-C or A-440 is recommended) and solve for the frequency based on this reference.

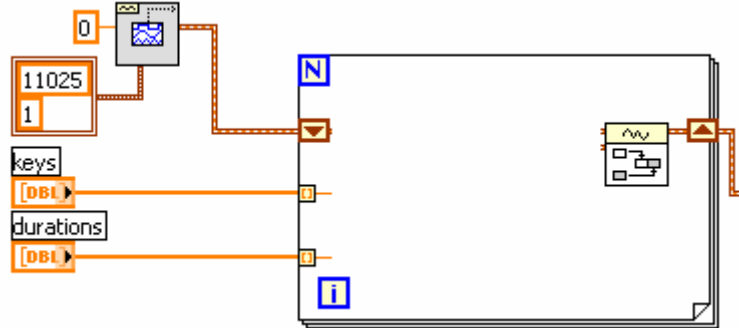
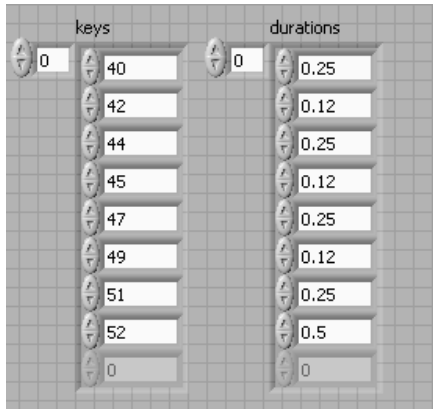
Attach a **tripleDisplay** and select **Play Sound** to listen to your note. Does it sound right?

Instructor Verification (separate page)

3.2 Synthesize a Scale

In a previous section you completed the `key2note.vi` VI which synthesizes the correct sinusoidal signal for a particular key number. Now, use that function to finish the following incomplete VI that will play scales:

⁷ This is a handy block to know about since you can enter several line of MATLAB-like code in it.




The **Append Waveforms** VI (right-most above) takes two waveforms and appends them. Review for a moment what the **for loop** does and what the brown triangles on the left and right of the for loop do. The function generator on the upper left is used to generate a single sample zero valued waveform to serve as a starting point for the output. It feeds into the shift register of the loop. Each time through the loop a new note is generated and appended to the previous notes. Finally the waveform for the entire scale emerges from the right of the for loop. You need to figure out how to add the **key2note** VI so the scale will play. Attach a **tripleDisplay** and listen to your scale. Does it sound right? Try changing some note numbers and durations. Play a bit. Save this VI as **PlayScale.vi**.

Instructor Verification (separate page)

3.3 Spectrogram

In this part, you will display the spectrogram of the scale synthesized in the previous section. Remember that the spectrogram displays an image that shows the *frequency* content of the synthesized *time* signal. Its horizontal axis is time and its vertical axis is frequency.

a) Generate the signal for the scale with the **PlayScale** VI from above.

Click the **Spectrogram** tab. Zoom in to see the progression of the consecutive notes in the scale (click the middle of the three buttons on the bottom left ) and identify the note A-440 in your spectrogram. Below the spectrum display are some values that control how the display is computed. The middle one is the *window length* which could be varied to get different looking spectrograms. The spectrogram is able to “see” the separate spectrum lines with a longer window length, e.g., 1024 or 2048⁸.

Instructor Verification (separate page)

4 Lab Exercise: Synthesis of Musical Notes

The audible range of musical notes consists of well-defined frequencies assigned to each note in a musical score. **Five different pieces are given in the book**, but we have chosen a different one for the synthesis program in this lab. Before starting the project, make sure that you have a working knowledge of the relationship between a musical score, key number and frequency. The following steps are an overview of the process of actually synthesizing the music. Read it

⁸ Usually the window length is chosen to be a power of two, because a special algorithm called the FFT is used in the computation. The fastest FFT programs are those where the signal length is a power of 2.

through, but don't do it now. Rather we will walk you through each step in greater detail in the sections that follow.

- a) Determine a sampling frequency that will be used to play out the sound through the D-to-A system of the computer. This will dictate the time T_s between samples of the sinusoids.
- b) Determine the total time duration needed for each note, and also determine the frequency (in hertz) for each note (see Figure 2 and the discussion of the well-tempered scale in the warm-up.) A comma-separated data file called **WellTempered.csv** is provided with this information stored in columns; this contains the portion of the piece needed for this lab. A second file called **WellTemperedShort.csv** has the same information for the first few measures of the piece; you may find this useful for initial debugging. **Both of these files can be found in the LabVIEW files link.**
- c) Synthesize the waveform as a combination of sinusoids, and play it out through the computer's built-in speaker or headphones.
- d) Make a plot of a few periods of two or three of the sinusoids to illustrate that you have the correct frequency (or period) for each note.
- e) Include a spectrogram of a portion of your synthesized music--probably about 1 or 2 secs--so that you can illustrate the fact that you have all the different notes. This piece has many sixteenth notes, so a window length of 512 might be the best choice. In addition, the spectrogram will scale the frequency axis to run from zero to half the sampling frequency, so it might be useful to “zoom in” on the region where the notes are.

4.1 Spectrogram of the Music

Musical notation describes how a song is composed of different frequencies and when they should be played. This representation can be considered to be a *time-frequency* representation of the signal that synthesizes the song. In LabVIEW we can compute a time-frequency representation from the signal itself. This is called the spectrogram, and its implementation with the LabVIEW VI **tripleDisplay**. **To aid your understanding of music and its connection to frequency content, a LabVIEW VI is available so that you can visualize the spectrogram along with musical notation. This GUI also has the capability to synthesize music from a list of notes, but these notes are given in “standard” musical notation, not key number. For more information, consult the `{\tt help}` on `{\tt musicgui.m}`.**

4.2 Fugue #2 for the Well-Tempered Clavier

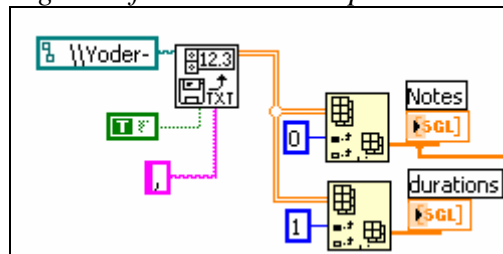
Fugue #2 for the Well-Tempered Clavier is one of those pieces of classical music that everyone has heard, but nobody particularly remembers its name. The first few measures are shown in Figure 4, and all the measures that you must synthesize can be found on the **website**.



Figure 4 First few measures of the piece *Fugue #2 for the Well-Tempered Clavier*. You must synthesize the entire portion of the *Fugue #2 for the Well-Tempered Clavier* given in *WellTempered.csv* by using sinusoids.⁹

4.3 Data File for Notes

The comma-separated files *WellTempered.csv* and *Well TemperedShort.csv* contain note a duration information for the *Fugue #2 for the Well-Tempered Clavier*.



The block diagram above shows how to read the file. The left-most block is a **Read from Spreadsheet File** block. The top left constant is the path to the file to be read. The True/False constant tells it to transpose the output so that we can get to the columns of data easily. The bottom , (comma) tells it the values are separated by commas. The right two blocks are **Index Array** blocks. The top block gets the 0th column of data (the key numbers) the bottom block gets the 1st column which is the duration data. You could add another column to the spreadsheet that contains amplitude data. If so, it would be easy to add another **Index Array** block to access that data too.

4.3.1 Timing

Musicians often think of the tempo, or speed of a song, in terms of “beats per minute” or BPM, where the beats are usually quarter notes. To the right of *Allegretto moderato* in Figure 4 you will see the BPM for this song. You should write the code so that the BPM is a control that can be changed easily. For example, you might let the BPM be a control on the front panel. Computer programs which let musicians record, modify, and play back notes played on a keyboard or other electronic instrument are called “sequencers.”¹⁰ The timing resolution of a sequencer is usually measured in “pulses per quarter note,” or PPQ. In this lab, we will employ four pulses per quarter note. A real commercial sequencer would have a much higher PPQ to encapsulate the subtle timing nuances of a real human playing a real instrument. The starting times and durations of notes in the music file provided to you are specified in terms of “pulses,” so it will be helpful helpful to compute the number of “seconds per pulse,” for instance via:

⁹Use sinusoids sampled at 11025 samples/sec (a lower sampling rate could be used if you have a computer with limited memory).

¹⁰ Popular commercial sequencers include Mark of the Unicorn's Digital Performer, Emagic's Logic Audio, Steinberg's Cubase and Opcode's Studio Vision.

```

beats_per_second = bpm/60;
seconds_per_beat = 1/beats_per_second;
seconds_per_pulse = seconds_per_beat / 4;

```

If the tempo is defined only once, then it could be changed: for example, setting bpm = 240 would make the whole piece play twice as fast. This is useful for debugging. Another timing issue is related to the fact that when a musical instrument is played, the notes are not continuous. Therefore, inserting very short pauses between notes usually improves the musical sound because it imitates the natural transition that a musician must make from one note to the next. An envelope (discussed below) can accomplish the same thing.

4.4 Assessment

After you finish the project, assess the quality of your synthesized result. Suggest some other features that could be incorporated into your program if you had more time to work on it. Some commonly used improvements are described in the next section.

4.5 Musical Tweaks

The musical passage is likely to sound very artificial, because it is created from pure sinusoids. Therefore, you should try to improve the quality of the sound by incorporating some modifications. For example, one improvement comes from using an “envelope,” where you multiply each pure tone signal by an envelope $E(t)$ so that it fades in and out.

$$x(t) = E(t) \cos(2\pi f_{key}t + \phi)$$

If an envelope is used it, should “fade in” quickly and fade out more slowly. An envelope such as a half-cycle of a sine wave $\sin(\pi/dur)$ is simple to program, but it sounds poor because it does not turn on quickly enough, so simultaneous notes of different durations no longer appear to begin at the same time. A standard way to define the envelope function is to divide $E(t)$ into four sections: attack (A), delay (D), sustain (S), and release (R). Together these are called ADSR. The attack is a quickly rising front edge, the delay is a small short-duration drop, the sustain is more or less constant and the release drops quickly back to zero. Figure 5 shows a linear approximation to the ADSR profile. Look for the **Arbitrary Wave VI** or **Ramp Pattern VI** for a way to create linearly increasing and decreasing values.

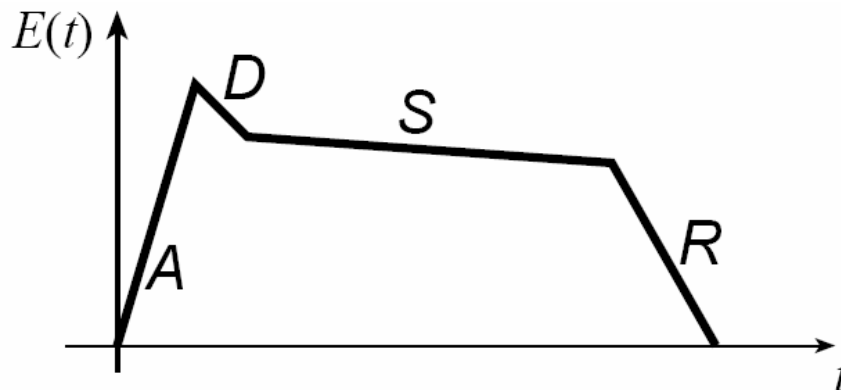


Figure 5 ADSR profile for an envelope function $E(t)$.

Some other issues that affect the quality of your synthesis include relative timing of the notes, correct durations for tempo, rests (pauses) in the appropriate places, relative amplitudes to emphasize certain notes and make others soft, and harmonics. Since true piano sounds have a second and third harmonic content, and we have been studying harmonics, this modification is relatively simple, but be careful to make the amplitudes of the harmonics smaller than the fundamental frequency component.¹¹

Furthermore, if you include too many higher harmonics, you might violate the sampling theorem and cause *aliasing*. You should experiment to see what sounds best.

4.6 Programming Tips

You may want to modify your *key2note* VI to accept additional parameters describing amplitude, duration, etc. In addition, you might choose to add an envelope and/or harmonics. Chords are created on a computer by simply adding the signal vectors of several notes. Although we have provided a LabVIEW file containing the note values and durations for *Fugue #2 for the Well-Tempered Clavier*, you are free to modify the duration values or add notes if you think it will improve the quality of the synthesized sound.

5 LabVIEW things to remember for future labs:

- Using a **Function Generator**, adjusting the sampling rate and number of samples using a **Bundle**.
- Appending waveforms with **Append Waveforms**.
- Debugging, setting breakpoints.
- Displaying Spectrograms, setting the window length.
- Reading comma separated value files.
- MathScript
- ... What else?

¹¹ In the early 80's, a company called Digital Keyboards produced a commercial synthesizer called the Synergy in which the user created sounds via "additive synthesis" by specifying the envelopes of individual frequency components. This is a quite powerful, albeit tedious and challenging way to create realistic sounds. American composer Wendy Carlos (best known for *Switched-On Bach* and her score for *A Clockwork Orange*) used it extensively in her score for *Tron*. See <http://www.synthmuseum.com/synergy/synergy01.html>.

Lab 04
Instructor Verification Sheet

For each verification, be prepared to explain your answer and respond to other related questions that the lab TA's or professors might ask. Turn this page in at the end of your lab period.

Name: _____ Date of Lab: _____

Part 3.1 Complete and demonstrate the VI `key2note.vi`:

Verified: _____ Date/Time: _____

Part 3.2 Complete and demonstrate the script file `PlayScale.vi`:

Verified: _____ Date/Time: _____

Part 3.3 Demonstrate the spectrogram of the scale generated by `PlayScale.vi`:

Verified: _____ Date/Time: _____

Sound Evaluation Criteria

Does the file play notes? All Notes _____ Most _____ Treble only _____

Overall Impression: _____

Excellent:

Enjoyable sound, good use of extra features such as harmonics, envelopes, etc.

Good:

Bass and Treble clefs synthesized and in sync, few errors, one or two special features.

OK:

Basic sinusoidal synthesis, including the bass, with only a few errors.

Poor:

No bass notes, or treble and bass not synchronized, many wrong notes.