

Analog Shield Application Note 3: Polyphonic Music Player

Revised June 3, 2014

Author: William J. Esposito, Doctoral Candidate and Gregory T.A. Kovacs, Professor of Electrical Engineering, Stanford University

This document was edited and adapted for MPIDE by Digilent. The original document was prepared by William J. Esposito and Gregory T.A. Kovacs at Stanford University.

Overview

The Analog Shield Polyphonic Music Player plays a converted 4-voice music file through normal headphones or a stereo system.

Hardware

The Polyphonic Music Player is built around the Arduino™ and uses the Digilent Analog Shield* for signal output, combining the four channels with a simple resistive mixer.

The Arduino UNO™ R3 can be found at: <http://store.arduino.cc/>

The Analog Shield can be found at:
<http://www.digilentinc.com/analogshield>

A full description of the resistive mixer is found below, but the parts requirements include (for monaural output):

- 4x 330 Ω Resistors
- 1x 3.3 k Ω Resistor
- 1x 0.047 μ F Capacitor
- 1x 3.5mm Headphone Jack

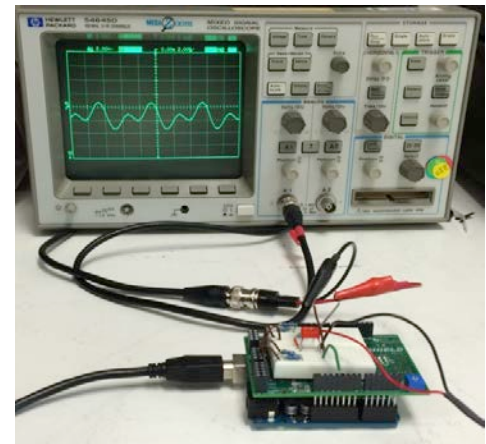


Figure 1. Polyphonic music Player with an oscilloscope.



Figure 2. Prototype Analog Shield.

**The Analog Shield was developed as a collaboration between Texas Instruments® and the Kovacs / Giovangrandi lab at Stanford University.*

Building the Demo

In order to build the Polyphonic Music Player, two nonstandard Arduino libraries are necessary. For a guide on Arduino libraries and how to install them, go to <http://arduino.cc/en/Guide/Libraries>.

The libraries required for the visualizer are:

- The TimerOne library (for the DDS sample clock) <https://code.google.com/p/arduino-timerone/>
- The Analog Shield library (for analog input) <http://diligentinc.com/analogshield>

In case these libraries move in the future, the Adafruit® LCD libraries are linked from the Adafruit Touch LCD Shield tutorial at <https://learn.adafruit.com/rgb-lcd-shield/>.

Once these libraries are downloaded and unzipped in four Arduino/libraries folders, be sure to re-open the Arduino IDE as it will not recognize libraries added while it is open. Once that is done, attach the Analog Shield to the Arduino Uno. Next, open the “Polyphonic_music” sketch with the Arduino IDE and upload it to the Arduino Uno. Once the demo is uploaded, one tone will be output from each of the four DAC outputs on the analog shield (D0-D3). Included with the player is an example song, which plays when the sketch is uploaded.

At this point, tones can be heard directly at the outputs by connecting a headphone (and a current limiting resistor of at least 330 Ω). To produce a combined sound output, a simple filter will have to be constructed, which will be described next.

Connecting the Output

The most straightforward way to output all four voices at one time is to use resistors to build a simple four input low-pass mixer / filter at the output of the Analog Shield’s DAC. It is crucial that all four independent resistors be used before the junction instead of one after; as the presence of the resistors ensures that there is no short circuit between outputs of the DAC. These circuits will work well for headphones, and a lower value or a potentiometer can be substituted for R5 (and R6) if adjustable volume is desired. Figure 3 below is the monaural mixer circuit schematic for headphones.

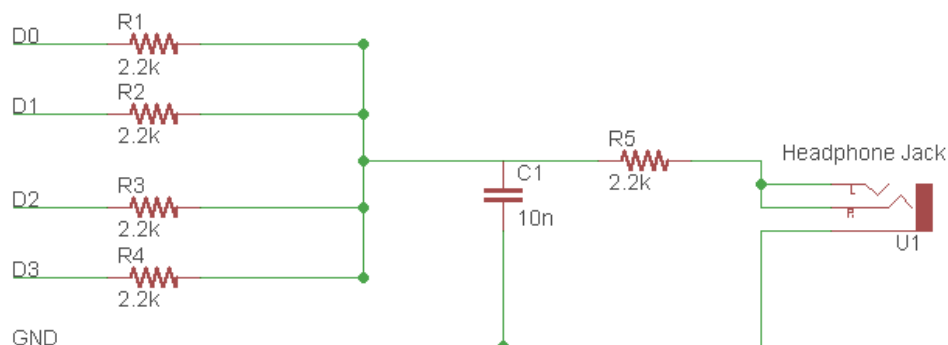


Figure 3. Monaural output mixer for headphones.

An alternative configuration allows stereo output, although the left and right channels are separated by voice rather than arranged as the composer may have intended.

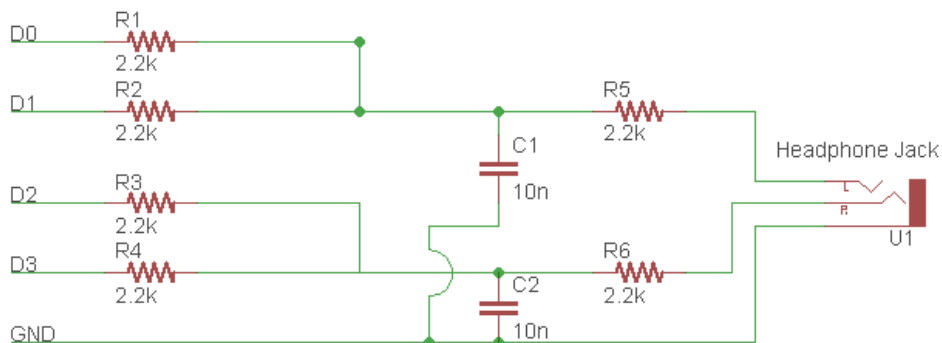


Figure 4. Pseudo-stereo output headphone mixer.

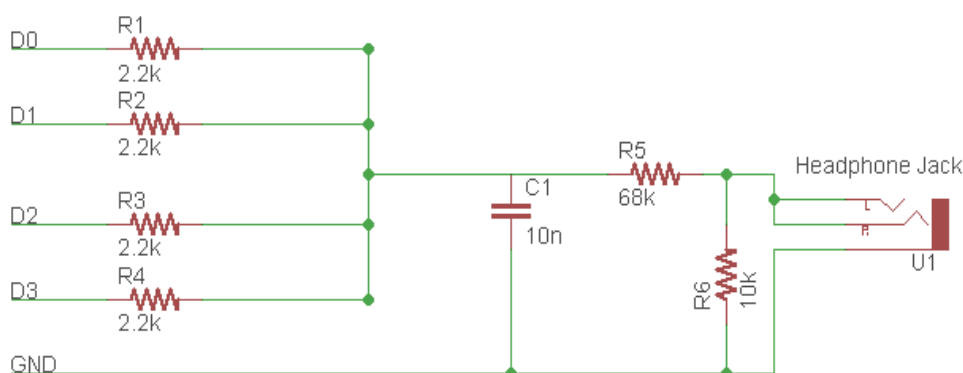


Figure 5. Mixer for driving line input on a home stereo.

For driving a stereo receiver, this output may deliver too much power. Be sure to turn the volume to the minimum setting and then adjust upward. If the primary output of the player is going to be a stereo, the circuit described in Fig. 5 would be preferable.

Once a mixer has been connected to the music Player, a headphone or the line input of a stereo can be connected to the polyphonic demo and tested. Beware that the input is above standard line input voltages (line level) and a larger resistor is recommended for R5/6

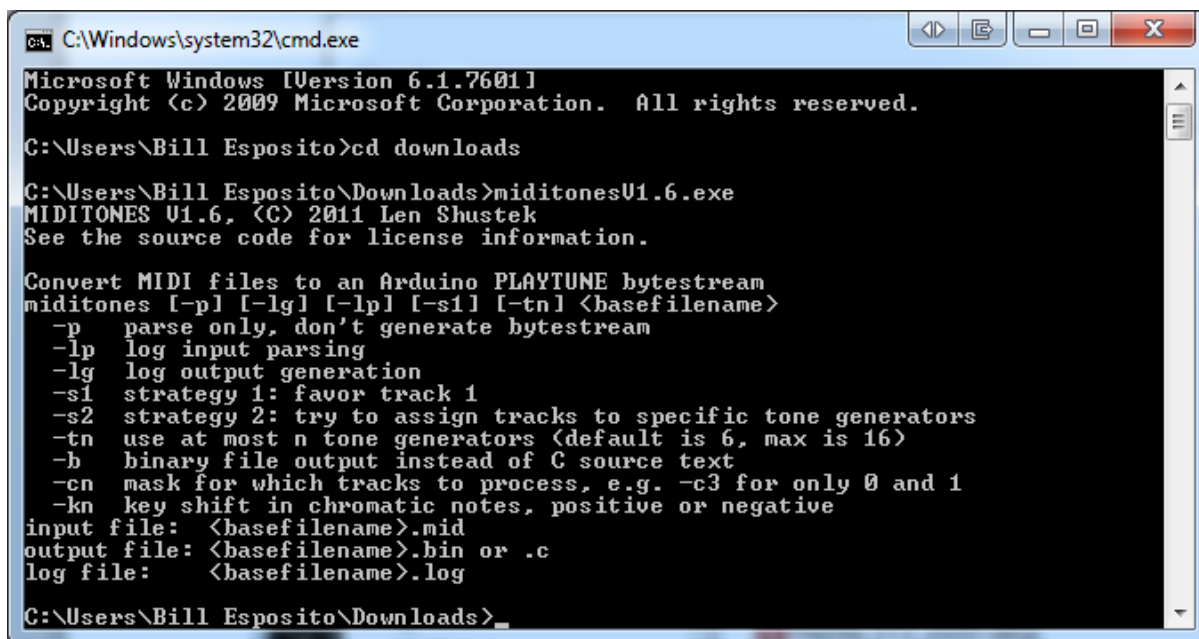
A simple resistive mixer is a low quality circuit and it is only due to the buffers on the outputs of the Analog Shield DAC that such a simple circuit is possible here. A more detailed discussion of mixer circuits (along with several higher quality mixer circuits that can be built with components found in most any parts kit) has been written by Rod Elliot and can be found at <http://sound.westhost.com/articles/audio-mixing.htm>.

How the Music Is Stored

The Polyphonic Music Player stores its output sound in the form of an array stored in program memory. Using this larger part of memory allows a much longer song to be stored, but prevents the score to be modified in real time by the player itself. Changing the output song requires a new upload to the Arduino Uno. Using program memory is described in detail at <http://arduino.cc/en/Reference/PROGMEM>.

In order to create a new song, an array of commands and delays must be constructed. Instructions perform operations such as starting a tone generator, stopping a tone generator, ending the track, or restarting the track from the beginning. Between events, delays are inserted to pace the music.

The command to start a tone generator is of the format '0x9n' where 'n' is a number between 0 and 3 indicating which tone generator to start. This command must be followed by an integer indicating which MIDI note to start the generator at. There are 128 MIDI notes which span five octaves, from 8 Hz to 12.5 kHz. A full listing of MIDI notes can be found at <http://www.tonalsoft.com/pub/news/pitch-bend.aspx>.



```

C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\Bill Esposito>cd downloads
C:\Users\Bill Esposito\Downloads>miditonesU1.6.exe
MIDITONES U1.6, (C) 2011 Len Shustek
See the source code for license information.

Convert MIDI files to an Arduino PLAYTUNE bytestream
miditones [-pl [-lg] [-lp] [-s1] [-tn] <basefilename>
  -p  parse only, don't generate bytestream
  -lp  log input parsing
  -lg  log output generation
  -s1  strategy 1: favor track 1
  -s2  strategy 2: try to assign tracks to specific tone generators
  -tn  use at most n tone generators (default is 6, max is 16)
  -b   binary file output instead of C source text
  -cn  mask for which tracks to process, e.g. -c3 for only 0 and 1
  -kn  key shift in chromatic notes, positive or negative
input file: <basefilename>.mid
output file: <basefilename>.bin or .c
log file:   <basefilename>.log
C:\Users\Bill Esposito\Downloads>

```

Figure 6. Miditones command line interface.

Screenshot of command prompt running on Microsoft Windows 7.

For example, in order to start generator 1 playing midi note 60 (middle C, 262 Hz), add 0x91, 60 to the array.

The command to stop a tone generator is of the format '0x8n', which simply stops the appropriate tone generator. For example, to stop tone generator number 0, add 0x80 to the array.

Between events, there must be delays. These delays come in the form of two integers. These numbers represent a 15 bit integer counting milliseconds until the next event (this means that the maximum number of milliseconds that can be delayed is 32767). For example, to add a delay of 2 seconds, 0x07, 0xd0 would be added to the array.

Finally, to stop or loop the playback, the command is 0xf0 (to stop) and 0xe0 to loop the output back to the start of the track. A simple example output song would be:

```
{ 0x90, 20, 0x07, 0xD0, 0x91, 30, 0x03, 0xE8, 0x92, 40, 0x05, 0xDC, 0x80, 0x05, 0xDC, 0xF0}
```

This corresponds to the sequence of commands:

```
{START GENERATOR 0, MIDI NOTE 20, DELAY 2000ms, START GENERATOR 1, MIDI NOTE 30, DELAY 1000ms, START  
GENERATOR 2, MIDI NOTE 40, DELAY 1500ms, STOP GENERATOR 0, DELAY 1500ms, END TRACK}
```

Converting a MIDI for the Player

A utility called Miditytes exists which takes a MIDI file as input and outputs an array in the format used by the player. The program is open source and available on Google Code™ at <https://code.google.com/p/miditytes>. Unfortunately, the version on Google Code is not compiled for OS X™, so included with the polyphonic player download are Windows® and Mac® compiled binaries of Miditytes along with the source code.

The primary limitation of the Polyphonic MIDI player is that it only has four voices. Although this is better than the best internal Arduino tone generator, there are many midi files with more than four voices. Be aware that complex tracks may have “missing” portions if too many voices are cut out in the conversion process.

Once a suitable midi file is ready, download the Miditytes program from Google Code. It is easiest to download Miditytes to a default directory, such as “downloads,” as it will be easier to find and use on the command line.

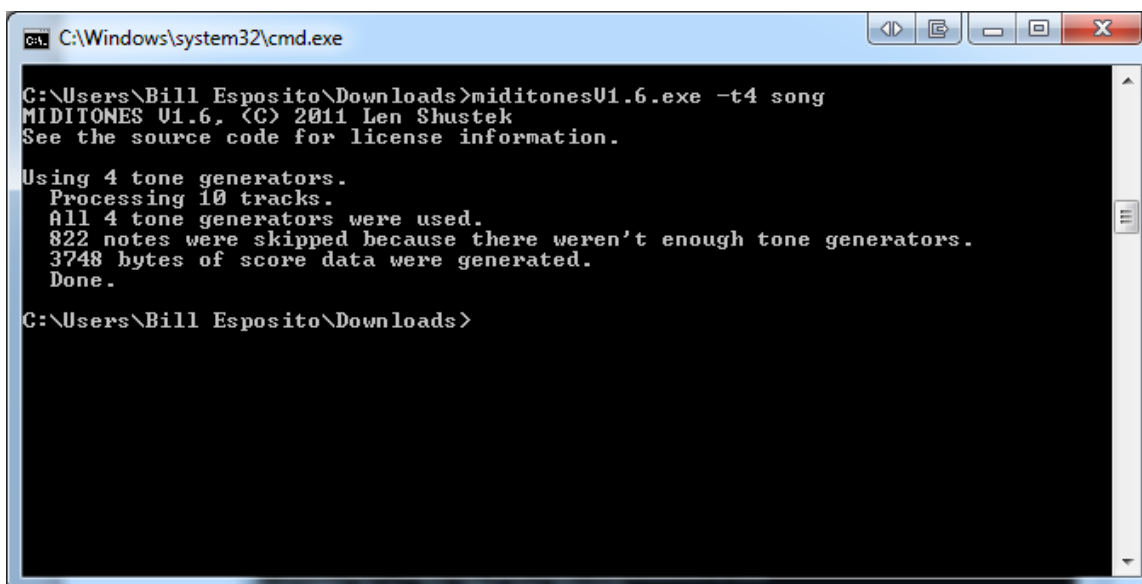
“Miditytes” must be run from a command window. The command window can be found by searching for CMD on Windows). From the command window, navigate to the folder which contains Miditytes (if the command window opens in “C:\Users\<your username>\”, typing “cd Downloads” will work).

Try executing Miditytes with no optional parameters by typing “miditytesV1.6.exe”. Miditytes will display a manual page which explains various useful options.

To convert a file, execute Miditytes with the filename as a parameter. It is also useful to limit the number of output voices to four with the “-t4” option. Four is the maximum number of channels that the Midi Player will output, and additional channels will waste memory but not play back sound. For example, for a file “song.mid”, with a four voice limit, the command would be:

```
miditytesV1.6.exe -t4 song
```

Including the .mid extension to the file name will cause an error. The output will look like Fig. 7 below:



```
C:\Windows\system32\cmd.exe

C:\Users\Bill Esposito\Downloads>miditonesV1.6.exe -t4 song
MIDITONES V1.6, (C) 2011 Len Shustek
See the source code for license information.

Using 4 tone generators.
Processing 10 tracks.
All 4 tone generators were used.
822 notes were skipped because there weren't enough tone generators.
3748 bytes of score data were generated.
Done.

C:\Users\Bill Esposito\Downloads>
```

Figure 7. Executing Miditones.

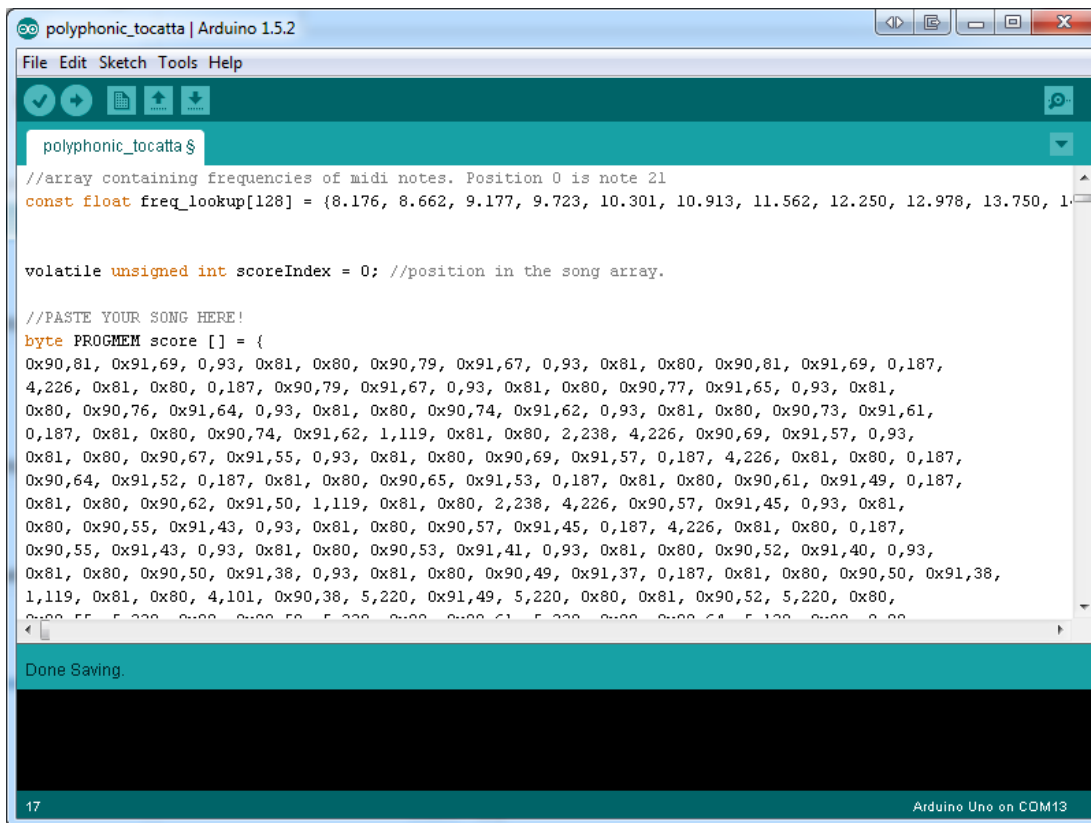
Screenshot of command prompt running on Microsoft Windows 7.

Once Miditones reports that it has produced an output, it will be stored in a file with the same name as the input midi, but as a .c extension (in the example, “song.c”). This file can be opened with any text editor (Notepad® will do), and the text is ready to copy into the polyphonic MIDI player Arduino sketch.

Adding Music to the Player

To copy the song into the Arduino sketch, replace the array named ‘score’ that starts on line 17 of the sketch, just below the comment “PASTE YOUR SONG HERE” and above the comment “Analog Shield Library.” Figure 8 below displays the insertion location for the sketch.

Once the new song is pasted into the sketch, it is ready to upload. Note that a long or particularly complex music file can cause the Arduino to run out of memory and compilation to fail.



```

polyphonic_tocatta | Arduino 1.5.2
File Edit Sketch Tools Help
polyphonic_tocatta $
//array containing frequencies of midi notes. Position 0 is note 21
const float freq_lookup[128] = {8.176, 8.662, 9.177, 9.723, 10.301, 10.913, 11.562, 12.250, 12.978, 13.750, 14.643, 15.556, 16.497, 17.468, 18.469, 19.500, 20.561, 21.651, 22.771, 23.921, 25.101, 26.311, 27.551, 28.831, 30.151, 31.511, 32.911, 34.351, 35.831, 37.351, 38.911, 40.511, 42.151, 43.831, 45.551, 47.311, 49.111, 50.951, 52.831, 54.751, 56.711, 58.711, 60.751, 62.831, 64.951, 67.111, 69.311, 71.551, 73.831, 76.151, 78.511, 80.911, 83.351, 85.831, 88.351, 90.911, 93.511, 96.151, 98.831, 101.551, 104.311, 107.111, 110.0, 112.9, 115.9, 118.9, 121.9, 125.0, 128.1, 131.3, 134.6, 138.0, 141.4, 144.9, 148.4, 152.0, 155.6, 159.3, 163.1, 166.9, 170.8, 174.8, 178.8, 182.9, 187.1, 191.3, 195.6, 199.9, 204.3, 208.8, 213.3, 217.9, 222.5, 227.2, 231.9, 236.7, 241.5, 246.4, 251.3, 256.3, 261.3, 266.4, 271.5, 276.7, 281.9, 287.2, 292.5, 297.9, 303.3, 308.8, 314.3, 319.9, 325.6, 331.3, 337.1, 343.0, 348.9, 354.9, 361.0, 367.1, 373.3, 379.5, 385.8, 392.1, 398.5, 405.0, 411.5, 418.1, 424.8, 431.5, 438.3, 445.2, 452.2, 459.3, 466.5, 473.7, 481.0, 488.3, 495.7, 503.2, 510.7, 518.3, 526.0, 533.7, 541.5, 549.4, 557.4, 565.5, 573.6, 581.8, 590.1, 598.5, 607.0, 615.6, 624.3, 633.1, 642.0, 651.0, 660.1, 669.3, 678.6, 688.0, 697.5, 707.1, 716.8, 726.6, 736.5, 746.5, 756.7, 767.0, 777.4, 787.9, 798.5, 809.3, 820.1, 831.1, 842.2, 853.4, 864.7, 876.1, 887.7, 899.4, 911.3, 923.3, 935.4, 947.6, 959.9, 972.4, 985.0, 997.8, 1010.7, 1023.8, 1037.0, 1050.4, 1063.9, 1077.5, 1091.3, 1105.3, 1119.4, 1133.7, 1148.1, 1162.7, 1177.4, 1192.3, 1207.4, 1222.6, 1238.0, 1253.6, 1269.3, 1285.2, 1301.3, 1317.5, 1333.9, 1350.4, 1367.1, 1383.9, 1400.9, 1418.1, 1435.5, 1453.1, 1470.8, 1488.7, 1506.8, 1525.0, 1543.4, 1561.9, 1580.6, 1599.4, 1618.4, 1637.5, 1656.8, 1676.3, 1695.9, 1715.7, 1735.7, 1755.8, 1776.1, 1796.6, 1817.2, 1838.0, 1858.9, 1879.9, 1901.1, 1922.5, 1944.0, 1965.7, 1987.5, 2009.5, 2031.7, 2054.0, 2076.5, 2099.1, 2121.9, 2144.8, 2167.9, 2191.2, 2214.7, 2238.3, 2262.1, 2286.1, 2310.2, 2334.5, 2358.9, 2383.5, 2408.2, 2433.1, 2458.1, 2483.3, 2508.6, 2534.1, 2559.7, 2585.4, 2611.3, 2637.3, 2663.5, 2689.8, 2716.3, 2742.9, 2769.7, 2796.6, 2823.7, 2850.9, 2878.3, 2905.8, 2933.4, 2961.2, 2989.1, 3017.2, 3045.4, 3073.8, 3102.3, 3131.0, 3159.8, 3188.8, 3217.9, 3247.2, 3276.6, 3306.2, 3335.9, 3365.8, 3395.8, 3425.9, 3456.1, 3486.5, 3517.0, 3547.6, 3578.3, 3609.1, 3640.0, 3671.0, 3702.2, 3733.5, 3764.9, 3796.4, 3828.0, 3859.7, 3891.5, 3923.4, 3955.4, 3987.5, 4019.7, 4052.0, 4084.4, 4116.9, 4149.5, 4182.2, 4215.0, 4247.9, 4280.9, 4314.0, 4347.2, 4380.5, 4413.9, 4447.4, 4481.0, 4514.7, 4548.5, 4582.4, 4616.4, 4650.5, 4684.7, 4719.0, 4753.4, 4787.9, 4822.5, 4857.2, 4892.0, 4926.9, 4961.9, 4997.0, 5032.2, 5067.5, 5102.9, 5138.4, 5174.0, 5209.7, 5245.5, 5281.4, 5317.4, 5353.5, 5389.7, 5426.0, 5462.4, 5498.9, 5535.5, 5572.2, 5609.0, 5645.9, 5682.9, 5720.0, 5757.2, 5794.5, 5831.9, 5869.4, 5907.0, 5944.7, 5982.5, 6020.4, 6058.4, 6096.5, 6134.7, 6173.0, 6211.4, 6250.0, 6288.7, 6327.5, 6366.4, 6405.4, 6444.6, 6483.9, 6523.3, 6562.9, 6602.6, 6642.4, 6682.3, 6722.3, 6762.4, 6802.6, 6842.9, 6883.3, 6923.8, 6964.4, 7005.1, 7045.9, 7086.8, 7127.8, 7168.9, 7210.1, 7251.4, 7292.8, 7334.3, 7375.9, 7417.6, 7459.4, 7501.3, 7543.3, 7585.4, 7627.6, 7669.9, 7712.3, 7754.8, 7797.4, 7840.1, 7882.9, 7925.8, 7968.8, 8011.9, 8055.1, 8098.4, 8141.8, 8185.3, 8228.9, 8272.6, 8316.4, 8360.3, 8404.3, 8448.4, 8492.6, 8536.9, 8581.3, 8625.8, 8670.4, 8715.1, 8759.9, 8804.8, 8849.8, 8894.9, 8940.1, 8985.4, 9030.8, 9076.4, 9122.1, 9167.9, 9213.8, 9259.8, 9305.9, 9352.1, 9398.4, 9444.8, 9491.3, 9538.0, 9584.8, 9631.7, 9678.7, 9725.8, 9773.0, 9820.3, 9867.7, 9915.2, 9962.8, 10010.5, 10058.3, 10106.2, 10154.2, 10202.3, 10250.5, 10298.8, 10347.2, 10395.7, 10444.3, 10493.0, 10541.8, 10590.7, 10639.7, 10688.8, 10738.0, 10787.3, 10836.7, 10886.2, 10935.8, 10985.5, 11035.3, 11085.2, 11135.2, 11185.3, 11235.5, 11285.8, 11336.2, 11386.7, 11437.3, 11488.0, 11538.8, 11589.7, 11640.7, 11691.8, 11743.0, 11794.3, 11845.7, 11897.2, 11948.8, 12000.5, 12052.3, 12104.2, 12156.2, 12208.3, 12260.5, 12312.8, 12365.2, 12417.7, 12470.3, 12523.0, 12575.8, 12628.7, 12681.7, 12734.8, 12788.0, 12841.3, 12894.7, 12948.2, 13001.8, 13055.5, 13109.3, 13163.2, 13217.2, 13271.3, 13325.5, 13379.8, 13434.2, 13488.7, 13543.3, 13598.0, 13652.8, 13707.7, 13762.7, 13817.8, 13873.0, 13928.3, 13983.7, 14039.2, 14094.8, 14150.5, 14206.3, 14262.2, 14318.2, 14374.3, 14430.5, 14486.8, 14543.2, 14599.7, 14656.3, 14713.0, 14769.8, 14826.7, 14883.7, 14940.8, 14998.0, 15055.3, 15112.7, 15170.2, 15227.8, 15285.5, 15343.3, 15401.2, 15459.2, 15517.3, 15575.5, 15633.8, 15692.2, 15750.7, 15809.3, 15868.0, 15926.8, 15985.7, 16044.7, 16103.8, 16163.0, 16222.3, 16281.7, 16341.2, 16400.8, 16460.5, 16520.3, 16580.2, 16640.2, 16700.3, 16760.5, 16820.8, 16881.2, 16941.7, 17002.3, 17063.0, 17123.8, 17184.7, 17245.7, 17306.8, 17368.0, 17429.3, 17490.7, 17552.2, 17613.8, 17675.5, 17737.3, 17799.2, 17861.2, 17923.3, 17985.5, 18047.8, 18110.2, 18172.7, 18235.3, 18298.0, 18360.8, 18423.7, 18486.7, 18549.8, 18613.0, 18676.3, 18739.7, 18803.2, 18866.8, 18930.5, 18994.3, 19058.2, 19122.2, 19186.3, 19250.5, 19314.8, 19379.2, 19443.7, 19508.3, 19573.0, 19637.7, 19702.5, 19767.4, 19832.4, 19897.5, 19962.7, 20028.0, 20093.4, 20158.9, 20224.5, 20290.2, 20356.0, 20421.9, 20487.9, 20554.0, 20620.2, 20686.5, 20752.9, 20819.4, 20886.0, 20952.7, 21019.5, 21086.4, 21153.4, 21220.5, 21287.7, 21355.0, 21422.4, 21490.0, 21557.7, 21625.5, 21693.4, 21761.4, 21829.5, 21897.7, 21966.0, 22034.4, 22102.9, 22171.5, 22240.2, 22309.0, 22377.9, 22446.9, 22516.0, 22585.2, 22654.5, 22723.9, 22793.4, 22863.0, 22932.7, 23002.5, 23072.4, 23142.4, 23212.5, 23282.7, 23353.0, 23423.4, 23493.9, 23564.5, 23635.2, 23706.0, 23776.9, 23847.9, 23919.0, 23990.2, 24061.5, 24132.9, 24204.4, 24276.0, 24347.7, 24419.5, 24491.4, 24563.4, 24635.5, 24707.7, 24779.9, 24852.2, 24924.6, 25000.0, 25075.4, 25150.9, 25226.5, 25302.1, 25377.8, 25453.5, 25529.3, 25605.1, 25681.0, 25756.9, 25832.9, 25908.9, 25985.0, 26061.1, 26137.3, 26213.5, 26289.8, 26366.1, 26442.5, 26518.9, 26595.4, 26671.9, 26748.5, 26825.1, 26901.8, 26978.5, 27055.2, 27132.0, 27208.8, 27285.6, 27362.4, 27439.2, 27516.0, 27592.8, 27669.6, 27746.4, 27823.2, 27900.0, 27976.8, 28053.6, 28130.4, 28207.2, 28284.0, 28360.8, 28437.6, 28514.4, 28591.2, 28668.0, 28744.8, 28821.6, 28898.4, 28975.2, 29051.9, 29128.7, 29205.5, 29282.3, 29359.0, 29435.8, 29512.5, 29589.3, 29666.0, 29742.8, 29819.5, 29896.3, 29973.0, 30049.8, 30126.5, 30203.3, 30279.9, 30356.6, 30433.3, 30509.9, 30586.6, 30663.3, 30739.9, 30816.6, 30893.3, 30969.9, 31046.6, 31123.3, 31199.9, 31276.6, 31353.3, 31429.9, 31506.6, 31583.3, 31659.9, 31736.6, 31813.3, 31889.9, 31966.6, 32043.3, 32119.9, 32196.6, 32273.3, 32349.9, 32426.6, 32503.3, 32579.9, 32656.6, 32733.3, 32809.9, 32886.6, 32963.3, 33039.9, 33116.6, 33193.3, 33269.9, 33346.6, 33423.3, 33499.9, 33576.6, 33653.3, 33729.9, 33806.6, 33883.3, 33959.9, 34036.6, 34113.3, 34189.9, 34266.6, 34343.3, 34419.9, 34496.6, 34573.3, 34649.9, 34726.6, 34803.3, 34879.9, 34956.6, 35033.3, 35109.9, 35186.6, 35263.3, 35339.9, 35416.6, 35493.3, 35569.9, 35646.6, 35723.3, 35799.9, 35876.6, 35953.3, 36029.9, 36106.6, 36183.3, 36259.9, 36336.6, 36413.3, 36489.9, 36566.6, 36643.3, 36719.9, 36796.6, 36873.3, 36949.9, 37026.6, 37103.3, 37179.9, 37256.6, 37333.3, 37409.9, 37486.6, 37563.3, 37639.9, 37716.6, 37793.3, 37869.9, 37946.6, 38023.3, 38100.0, 38176.6, 38253.3, 38329.9, 38406.6, 38483.3, 38559.9, 38636.6, 38713.3, 38790.0, 38866.6, 38943.3, 39019.9, 39096.6, 39173.3, 39250.0, 39326.6, 39403.3, 39479.9, 39556.6, 39633.3, 39710.0, 39786.6, 39863.3, 39939.9, 40016.6, 40093.3, 40170.0, 40246.6, 40323.3, 40400.0, 40476.6, 40553.3, 40629.9, 40706.6, 40783.3, 40860.0, 40936.6, 41013.3, 41090.0, 41166.6, 41243.3, 41320.0, 41396.6, 41473.3, 41550.0, 41626.6, 41703.3, 41780.0, 41856.6, 41933.3, 42010.0, 42086.6, 42163.3, 42240.0, 42316.6, 42393.3, 42470.0, 42546.6, 42623.3, 42700.0, 42776.6, 42853.3, 42930.0, 43006.6, 43083.3, 43160.0, 43236.6, 43313.3, 43390.0, 43466.6, 43543.3, 43620.0, 43696.6, 43773.3, 43850.0, 43926.6, 44003.3, 44080.0, 44156.6, 44233.3, 44310.0, 44386.6, 44463.3, 44540.0, 44616.6, 44693.3, 44770.0, 44846.6, 44923.3, 45000.0, 45076.6, 45153.3, 45230.0, 45306.6, 45383.3, 45460.0, 45536.6, 45613.3, 45690.0, 45766.6, 45843.3, 45920.0, 46000.0, 46079.9, 46158.8, 46238.8, 46318.7, 46398.7, 46478.6, 46558.6, 46638.5, 46718.5, 46798.4, 46878.4, 46958.3, 47038.3, 47118.2, 47198.2, 47278.1, 47358.1, 47438.0, 47518.0, 47597.9, 47677.9, 47757.8, 47837.8, 47917.7, 48000.0, 48080.0, 48160.0, 48240.0, 48320.0, 48400.0, 48480.0, 48560.0, 48640.0, 48720.0, 48800.0, 48880.0, 48960.0, 49040.0, 49120.0, 49200.0, 49280.0, 49360.0, 49440.0, 49520.0, 49600.0, 49680.0, 49760.0, 49840.0, 49920.0, 50000.0, 50080.0, 50160.0, 50240.0, 50320.0, 50400.0, 50480.0, 50560.0, 50640.0, 50720.0, 50800.0, 50880.0, 50960.0, 51040.0, 51120.0, 51200.0, 51280.0, 51360.0, 51440.0, 51520.0, 51600.0, 51680.0, 51760.0, 51840.0, 51920.0, 52000.0, 52080.0, 52160.0, 52240.0, 52320.0, 52400.0, 52480.0, 52560.0, 52640.0, 52720.0, 52800.0, 52880.0, 52960.0, 53040.0, 53120.0, 53200.0, 53280.0, 53360.0, 53440.0, 53520.0, 53600.0, 53680.0, 53760.0, 53840.0, 53920.0, 54000.0, 54080.0, 54160.0, 54240.0, 54320.0, 54400.0, 54480.0, 54560.0, 54640.0, 54720.0, 54800.0, 54880.0, 54960.0, 55040.0, 55120.0, 55200.0, 55280.0, 55360.0, 55440.0, 55520.0, 55600.0, 55680.0, 55760.0, 55840.0, 55920.0, 56000.0, 56080.0, 56160.0, 56240.0, 56320.0, 56400.0, 56480.0, 56560.0, 56640.0, 56720.0, 56800.0, 56880.0, 56960.0, 57040.0, 57120.0, 57200.0, 57280.0, 57360.0, 57440.0, 57520.0, 57600.0, 57680.0, 57760.0, 57840.0, 57920.0, 58000.0, 58080.0, 58160.0, 58240.0, 58320.0, 58400.0, 58480.0, 58560.0, 58640.0, 58720.0, 58800.0, 58880.0, 58960.0, 59040.0, 59120.0, 59200.0, 59280.0, 59360.0, 59440.0, 59520.0, 59600.0, 59680.0, 59760.0, 59840.0, 59920.0, 60000.0, 60080.0, 60160.0, 60240.0, 60320.0, 60400.0, 60480.0, 60560.0, 60640.0, 60720.0, 60800.0, 60880.0, 60960.0, 61040.0, 61120.0, 61200.0, 61280.0, 61360.0, 61440.0, 61520.0, 61600.0, 61680.0, 61760.0, 61840.0, 61920.0, 62000.0, 62080.0, 62160.0, 62240.0, 62320.0, 62400.0, 62480.0, 62560.0, 62640.0, 62720.0, 62800.0, 62880.0, 62960.0, 63040.0, 63120.0, 63200.0, 63280.0, 63360.0, 63440.0, 63520.0, 63600.0, 63680.0, 63760.0, 63840.0, 63920.0, 64000.0, 64080.0, 64160.0, 64240.0, 64320.0, 64400.0, 64480.0, 64560.0, 64640.0, 64720.0, 64800.0, 64880.0, 64960.0, 65040.0, 65120.0, 65200.0, 65280.0, 65360.0, 65440.0, 65520.0, 65600.0, 65680.0, 65760.0, 65840.0, 65920.0, 66000.0, 66080.0, 66160.0, 66240.0, 66320.0, 66400.0, 66480.0, 66560.0, 66640.0, 66720.0, 66800.0, 66880.0, 66960.0, 67040.0, 67120.0, 67200.0, 67280.0, 67360.0, 67440.0, 67520.0, 67600.0, 67680.0, 67760.0, 67840.0, 67920.0, 68000.0, 68080.0, 68160.0, 68240.0, 68320.0, 68400.0, 68480.0, 68560.0, 68640.0, 68720.0, 68800.0, 68880.0, 68960.0, 69040.0, 69120.0, 69200.0, 69280.0, 69360.0, 69440.0, 69520.0, 69600.0, 69680.0, 69760.0, 69840.0, 69920.0, 70000.0, 70080.0, 70160.0, 70240.0, 70320.0, 70400.0, 70480.0, 70560.0, 70640.0, 70720.0, 70800.0, 70880.0, 70960.0, 71040.0, 71120.0, 71200.0, 71280.0, 71360.0, 71440.0, 71520.0, 71600.0, 71680.0, 71760.0, 71840.0, 71920.0, 72000.0, 72080.0, 72160.0, 72240.0, 72320.0, 72400.0, 72480.0, 72560.0, 72640.0, 72720.0, 72800.0, 72880.0, 72960.0, 73040.0, 73120.0, 73200.0, 73280.0, 73360.0, 73440.0, 73520.0, 73600.0, 73680.0, 73760.0, 73840.0, 73920.0, 74000.0, 74080.0, 74160.0, 74240.0, 74320.0, 74400.0, 74480.0, 74560.0, 74640.0, 74720.0, 74800.0, 74880.0, 74960.0, 75040.0, 75120.0, 75200.0, 75280.0, 75360.0, 75440.0, 75520.0, 75600.0, 75680.0, 75760.0, 75840.0, 75920.0, 76000.0, 76080.0, 76160.0, 76240.0, 76320.0, 76400.0, 76480.0, 76560.0, 76640.0, 76720.0, 76800.0, 76880.0, 76960.0, 77040.0, 77120.0, 77200.0, 77280.0, 77360.0, 77440.0, 77520.0, 77600.0, 77680.0, 77760.0, 77840.0, 77920.0, 78000.0, 78080.0, 78160.0, 78240.0, 78320.0, 78400.0, 78480.0, 78560.0, 78640.0, 78720.0, 78800.0, 78880.0, 78960.0, 79040.0, 79120.0, 79200.0, 79280.0, 79360.0, 79440.0,
```

```

/* Direct Digital Synthesis with LCD Output */
/* 16 bit polyphonic DDS for music generation.*/
//12 bit long sine table
PROGMEM prog_uint16_t isinTable16[] = { <full sine wave here> };

const float freq_lookup[128] = {<frequencies go here>}; //freqs of midi
notes.
volatile unsigned int scoreIndex = 0; //position in the song array.
byte PROGMEM score [] = {<music goes here>}; //PASTE YOUR SONG HERE!

#include <analogShield.h> //Analog Shield Library
#include <avr/pgmspace.h> //for storing sine and song
#include <TimerOne.h> //Timer

//DDS variables
volatile unsigned int tuningWord[4] = {0};
volatile unsigned int phaseAccumulator[4] = {0}; //16 bits
volatile unsigned long sampleCount = 0;
volatile unsigned long nextEvent = 0;

void setup(){}

void loop()
//takes user input and the goes into DDS forever.
{
  Timer1.initialize(50); //Setup the timer for DDS
  Timer1.attachInterrupt(dds); //start DDS
  while (1) { //stay inside this loop to reduce jitter
    TIMSK0 = 0x00; //Turn off timers
    TIMSK2 = 0x00; //to reduce jitter
    //enough samples have passed time to do something
    if(sampleCount > nextEvent)
    {
      //first a byte from the array.
      //This will set your nextEvent delay
      byte event = pgm_read_word(score + scoreIndex);
      scoreIndex++;
      if(event < 0x80)
        //If the highest bit is zero.
        //It is a time and we want to get a delay.
        {
          unsigned int delayMS =
            highByte(event) +
            lowByte(pgm_read_word(score + scoreIndex));

          scoreIndex++;
          nextEvent += 20 * delayMS; //actually 19.770
        }
      else //some sort of command
      {
        if((event & 0xF0) == 0x80) //start channel
        {
          event = event & 0x03;
          tuningWord[event] = 0;
        }
        else if((event & 0xF0) == 0x90) //start channel
        {
          event = event & 0x03;
          //get midi Note
          byte note = pgm_read_word(score + scoreIndex);

```



```

        scoreIndex++; //increment table pointer
        //convert midi note into frequency
        float newFreq = freq_lookup[note];
        //set appropriate channel;
        tuningWord[event] = (unsigned int)(newFreq * 3.28);
    }
    else if(event == 0xF0) //end of song sentinel; Stop
    {
        tuningWord[0] = 0;
        tuningWord[1] = 0;
        tuningWord[2] = 0;
        tuningWord[3] = 0;
        while(1);
        {};
    }
    else if(event == 0xE0) //end of song sentinel; Repeat
    {
        scoreIndex = 0;
    }
}
}; //wait forever.
}
//DDS code
void dds() //Direct Digital Synthesis Lives Here.
{
    unsigned int value[4];
    unsigned int tempPhase[4];
    //loop unrolled by hand because apparently the ATMEGA compiler
    //doesn't understand the word 'optimize'.
    //increment the phase accumulator by the value
    //stored in the tuning word.
    phaseAccumulator[0] += tuningWord[0]; //16 bits
    phaseAccumulator[1] += tuningWord[1]; //16 bits
    phaseAccumulator[2] += tuningWord[2]; //16 bits
    phaseAccumulator[3] += tuningWord[3]; //16 bits

    //top 12 bits of each accumulator
    tempPhase[0] = (unsigned int)(phaseAccumulator[0] >> 4);
    tempPhase[1] = (unsigned int)(phaseAccumulator[1] >> 4);
    tempPhase[2] = (unsigned int)(phaseAccumulator[2] >> 4);
    tempPhase[3] = (unsigned int)(phaseAccumulator[3] >> 4);
    //look up values
    value[0] = pgm_read_word(isinTable16 + tempPhase[0]);
    value[1] = pgm_read_word(isinTable16 + tempPhase[1]);
    value[2] = pgm_read_word(isinTable16 + tempPhase[2]);
    value[3] = pgm_read_word(isinTable16 + tempPhase[3]);

    //write the result to the output.
    analog.write(value[0], value[1], value[2], value[3], true);
    //count samples
    sampleCount++;
}

```

Potential Improvements

The music player is an extremely satisfying application of the Analog Shield. It would be worth additional investigation to attempt to connect the Arduino to an SD card, which would provide hours of playback. Another improvement worth investigating is “soft voices”. That is, implementing software multiplexing of two voices on each channel. This could allow for an 8 voice music player, although the cost would be a further reduction in bandwidth of the synthesized output. Another interesting modification would be to use four amplifiers and four speakers to output each voice on its own physical channel.

Disclaimer: This code and circuit was developed by William Esposito, Ph.D. Candidate in Electrical Engineering, Stanford University, in the Kovacs/Giovangrandi Laboratory in collaboration with Texas Instruments, Incorporated. All code herein is free and open source, but is provided as-is with no warranties implied or provided. Use of this code and the associated documentation is at the user’s own risk.