Keyboard Based Polyphonic Music Synthesizer

Daniel Poracky, E.E.
Amauri Lopez, C.E.
ELC 343
Dr. Pearlstein

Overview

Objective: design and build a device that would mimic the actions of a music synthesizer as close as possible.

This device needed to be polyphonic in nature,

Challenge Elements

Three or more pushbuttons

Interrupt Service Routine

Digital to Analog Converter

Lowpass Filter

What is a synthesizer?

electronic musical instrument that generates electric signals that are converted to sound through instrument amplifiers and loudspeakers.

Robert Moog, first designer to popularize voltage control, with some of his analog synthesizers

In Short: How it was done

Look up frequency for music note

Create waveform in software (square, sine, triangle, etc.)

Assign push buttons in hardware to different waves

Check for button press, include in total output if pushed

Throw total to DAC

Output result from DAC to speaker

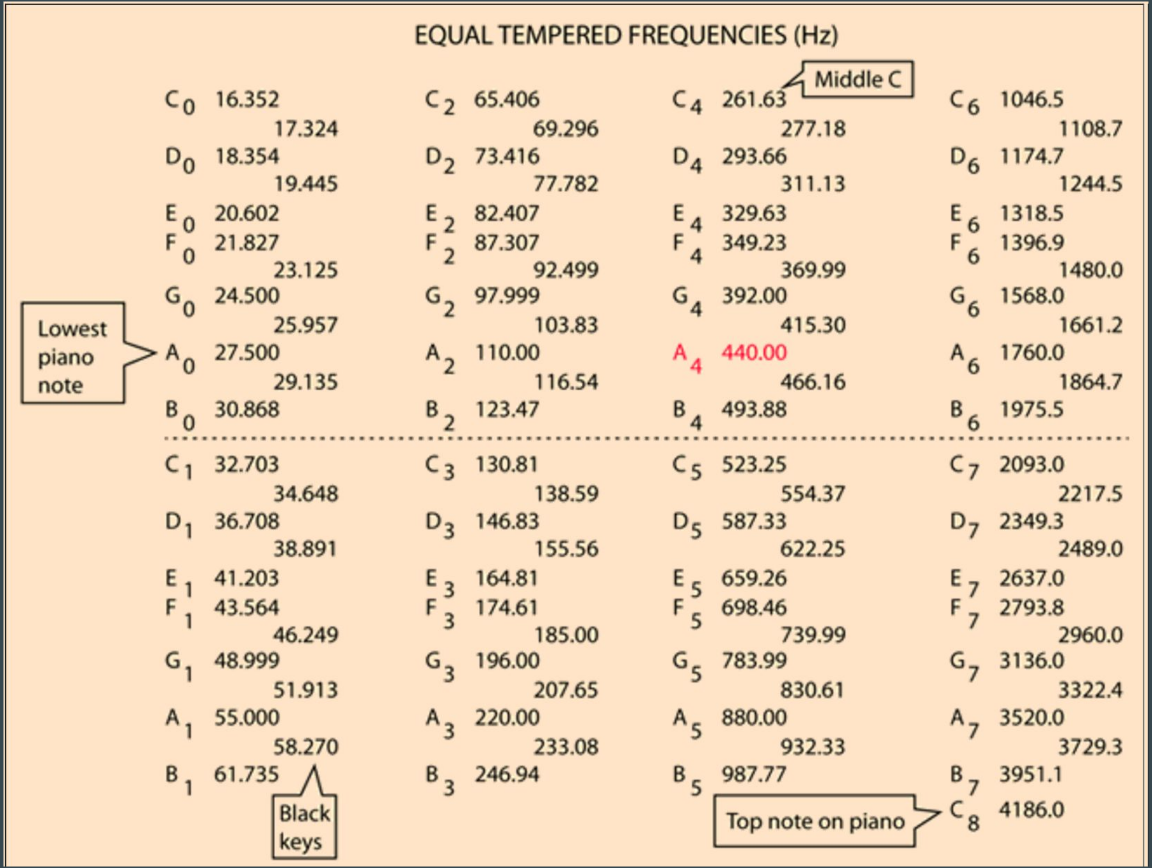
To produce many sounds, in general, we needed to research what characteristics of these sounds we could use to differentiate between them.

Being a music synthesizer, each sound should not be random and should resemble a musical note.

Each musical note has a certain frequency, or amount of wave cycles per second, associated with it.

Notes and their associated frequencies would eventually be used to generate our waveforms.

Equal Tempered Scale



EQUAL TEMPERED FREQUENCIES (Hz)

	C_0	16.352		C_2	65.406		C_4	261.63		Middle C	C_6	1046.5	
			17.324			69.296			277.18				1108.7
	D_0	18.354		D_2	73.416		D_4	293.66			D_6	1174.7	
			19.445			77.782			311.13				1244.5
	E_0	20.602		E_2	82.407		E_4	329.63			E_6	1318.5	
	F_0	21.827		F_2	87.307		F_4	349.23			F_6	1396.9	
			23.125			92.499			369.99				1480.0
	G_0	24.500		G_2	97.999		G_4	392.00			G_6	1568.0	
			25.957			103.83			415.30				1661.2
Lowest piano note	A_0	27.500		A_2	110.00		A_4	440.00			A_6	1760.0	
			29.135			116.54			466.16				1864.7
	B_0	30.868		B_2	123.47		B_4	493.88			B_6	1975.5	
	C_1	32.703		C_3	130.81		C_5	523.25			C_7	2093.0	
			34.648			138.59			554.37				2217.5
	D_1	36.708		D_3	146.83		D_5	587.33			D_7	2349.3	
			38.891			155.56			622.25				2489.0
	E_1	41.203		E_3	164.81		E_5	659.26			E_7	2637.0	
	F_1	43.564		F_3	174.61		F_5	698.46			F_7	2793.8	
			46.249			185.00			739.99				2960.0
	G_1	48.999		G_3	196.00		G_5	783.99			G_7	3136.0	
			51.913			207.65			830.61				3322.4
	A_1	55.000		A_3	220.00		A_5	880.00			A_7	3520.0	
			58.270			233.08			932.33				3729.3
	B_1	61.735	Black keys	B_3	246.94		B_5	987.77			B_7	3951.1	
										Top note on piano	C_8	4186.0	

Equal Tempered Scale

The note A, which is above middle C, has a frequency of 440Hz.

Widely recognized pitch standard for tuning instruments and calibration equipment

This was used as a reference to establish a relationship between it and the next note in a musical series.

Finding Our Keys

440 Hz was used as a reference with established relationships between it and the next note in a musical series.

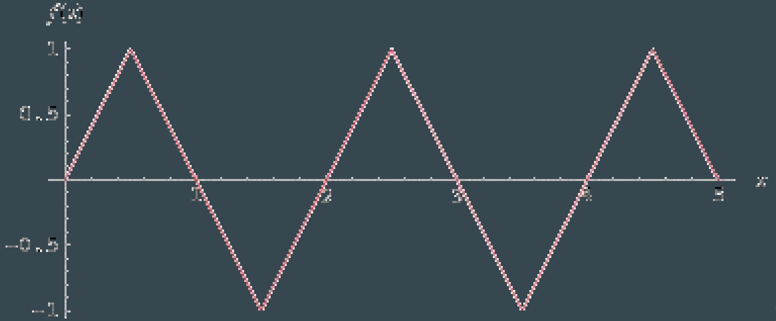
A derived mathematical relationship

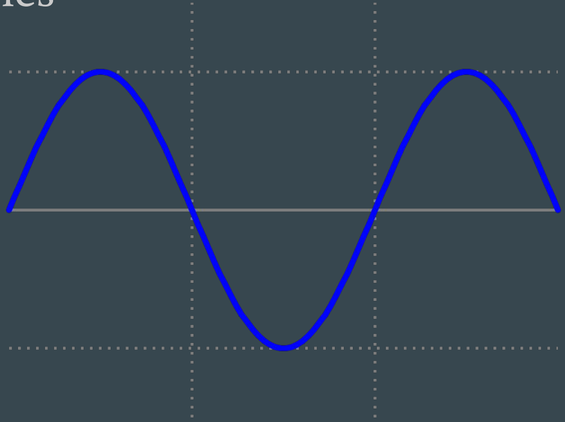
$$f_n = 2^{\frac{1}{x}} * (f_{n-1})$$

describes the differences in frequencies between each half-step (x=12) and full-step (x=6) on a keyboard.

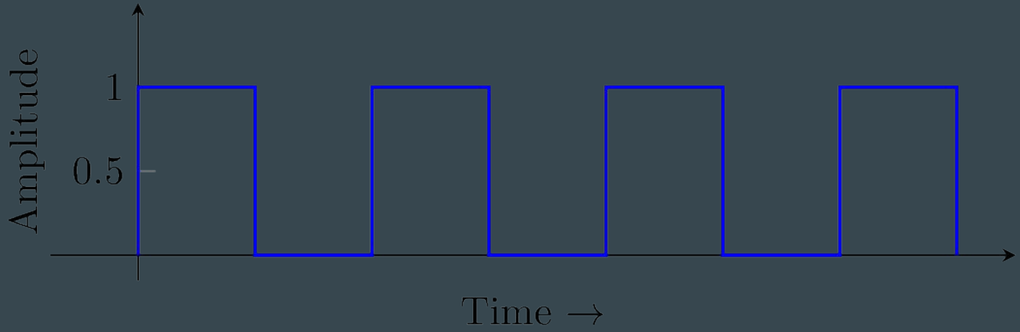
Choosing a Waveform

Different types of waveforms would create different tones





Square wave



Generating Our Square Wave

At every moment in time we needed to generate square waves for how many buttons were pressed.

These square waves were summed to obtain a total resultant wave, and then this total was scaled by a constant to achieve maximum gain.

We chose to scale our resultant by 64, a random number we felt was high enough and appropriate.

Generating Our Square Wave cont.

A sampling frequency (fs) of 12kHz was selected

Calculate ratio between sampling frequency and the period of the note ($t=1/f$)

This determined how many times a note can be repeated within our sampling frequency

This determined how often a value would need to be generated in relation to the clock

Generating Our Square Wave cont.

Mathematical representation:

The decimal portion of the ratio was taken (t-y)

Checked if decimal portion was above or below 0.5

Above 0.5 : Return 0

Below 0.5 : Return 1

Square Wave

Generating Our Sine Wave

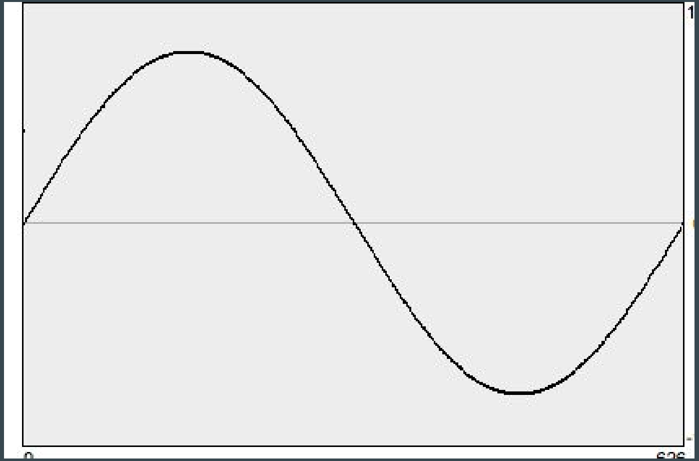
Same initial steps as generation of square wave (ratio, finding decimal)

Sine lookup table was generated for 256 values within one period

Searched through this table to return values directly towards input to VDAC.

Decimal part * 256 = element in sine table that we’re looking for. (di)

return sinetable[di]
Where di = (int) ((t-y)*256)



Outputting a Waveform to Speaker

After generated either square or sine wave

Check to see which button is being pressed

Output corresponding waveform to VDAC

Multiple buttons pushed: output sum of the waveforms

Connect speaker to output

Obstacles Faced/Relevant Info

Pushbuttons and onboard pushbutton - slightly unreliable

Generating entire sine wave within every tick of 12 kHz clock was challenging
- -especially for summed waves from multiple buttons pushed
- -tried to optimize for speed as much as possible. Managed two button presses.

Initially, high frequency noise disrupted output of multiple push button presses
- -added low pass filter from output of VDAC to speaker

PSoC board offered limited breadboard space

Questions