SUPERCOLLIDER DX-7 EMULATOR

Supercollider DX-7 Emulator

Aziz Ege Gonul, 14285

June, 27 2016 – August, 19 2016

STEIM (the STudio for Electro-Instrumental Music)

Mrs. Kristina Andersen (Company Advisor)

September, 14 2016

Sabanci University

Faculty of Engineering and Natural Sciences

Electronics Engineering

Abstract

Currently I am completing a Bachelor of Science degree in Electronics Engineering at the Sabanci University in Turkey. I am in my senior year and therefore I have to conduct 8 weeks internship before my graduation. My project company is STEIM and it is an independent electronic music centre. It is known for its intuitive use of technology to create new musical interfaces. During my internship I engaged in ongoing projects which would most often be based on electronics and technology. I also developed my own project which will be based upon FM synthesis using Supercollider programming language. It is based upon the circuit architecture of the vintage Yamaha synthesisers. It is mapped to the MIDI controller for further modulations. It is an extremely complex system with more than 100 parameters and feedback loops. Since I am coming from an engineering background and have in-depth knowledge about the subject, I overcame technical and theoretical difficulties. Every step is well-documented for my internship report and also for my portfolio. This document shows step-by-step how I managed to build DX-7 Supercollider Emulator.

Table of Contents

Introduction

This report documents the work which has been done during the 8-weeks of internship at STEIM (the STudio for Electro-Instrumental Music), Amsterdam under the supervision of Mrs. Kristina Andersen. During this internship, I have worked mainly with FM synthesis and Yamaha DX-7 Synthesizer. I wanted to do my internship project around FM to gain knowledge and to learn, improve and develop new sets of skills in the area of programming and signal processing. Thereby, one of my goals was to improve my sound processing knowledge. On the other hand, I am planning to study at the area of Music Technology Master's program. Therefore, choosing internship at STEIM paved the way for acquiring new set of skills and expanding my knowledge in this field. The report first shall give an overview of the tasks which was completed during the period of internship with technical details. Then, the results obtained shall be discussed and analysed.

Company/ Institution Background

According to company website (2016), STEIM is an independent electronic music centre unique in its dedication to live performance. They encourage musical instrument design concepts, using both high tech and low technology solutions. According to Wikipedia (n.d.) page, they existed since 1969 and they are financially supported by the Dutch Ministry of Culture. STEIM is very well known at the area of electronic music and especially in the area of electronic instrument design. It is not a commercially inclined institution but they have had commercially successful products such as “Crackle Box” which is an instrument that makes sounds using touch plates with very little cost. It is rather a small-sized company with a big vision. They have 8-10 core staff members active during the year but they do have graduate program in collaboration with the Institute of Sonology. Therefore, the workspace is occupied with the graduate students, interns, staff and some collaborators of active projects. Their building is at the centre of Amsterdam and it is rather large building with three different studios, dedicated hardware and software rooms and also offices for staffs. I mostly worked in their hardware laboratory during my internship.

Internship Project: Description and Analysis

Part 1: Project Overview

My project was based on the very famous music synthesiser of the 80’s, namely Yamaha DX-7. My main goal in this project was to program the sound engine from scratch using programming languages such as Supercollider and C and Java. I chose this project because of the popularity of this synthesiser. It is the first digital synthesizer which was affordable, so it became one of the most sold synthesiser of the history (Shepard, 2013). Creating complex sounds using FM synthesis is found by John Chowning who was a graduate student at Stanford University back then. “John Chowning, has brought Stanford millions of dollars in patent license fees — second only in terms of revenue generated to Stanford’s basic biotechnology patents, second moreover at Stanford, whose licensing income far outstrips that of any other US university.” (Johnson, 1994, p.1). Since it is a very famous synthesiser it has a very large user database with extensive resources. I have had extensive knowledge about the sound generation techniques like subtractive synthesis or additive synthesis and also I have taken the course called EE-313 (Introduction to Communication Systems) which had a chapter on Frequency Modulation, but FM synthesis of the sounds was always something different to me. So I saw this internship project as a chance to break the barrier of sound generation techniques. My final aim was to load original DX-7 presets to my program and having the exact same output.

steim
DX7
DIGITAL PROGRAMMABLE ALGORITHM SYNTHESIZER
MEMORY PROTECT
INTERNAL OFF

Yamaha DX-7

Part 2: Frequency Modulation Synthesis

Most of the sounds we hear during the daily life are complex. They contain wide variety of partials (Partial is the single sine tone at a particular frequency). Partials changing over time; make up the tone colour. For example when we hear single piano note, it is a very complex summation of partials changing over time. It first starts very intense in terms of loudness and fades over time. In order to create these complex sounds, frequency modulation synthesis most often use one simple waveform, to modulate another simple waveform's frequency. This modulation can create stunningly complex and varied sound. This modulation can be seen from the Figure 1 (Chowning,1973, p. 527) and Figure 2 (Schottstaedt,n.d.). Figure 1 displays what happens when we increase modulation depth of sinusoidal and Figure 2 shows an example of a FM complex sound generation, in the format of spectrogram. Figure 2 shows time, frequency and loudness in its x, y and z plane.

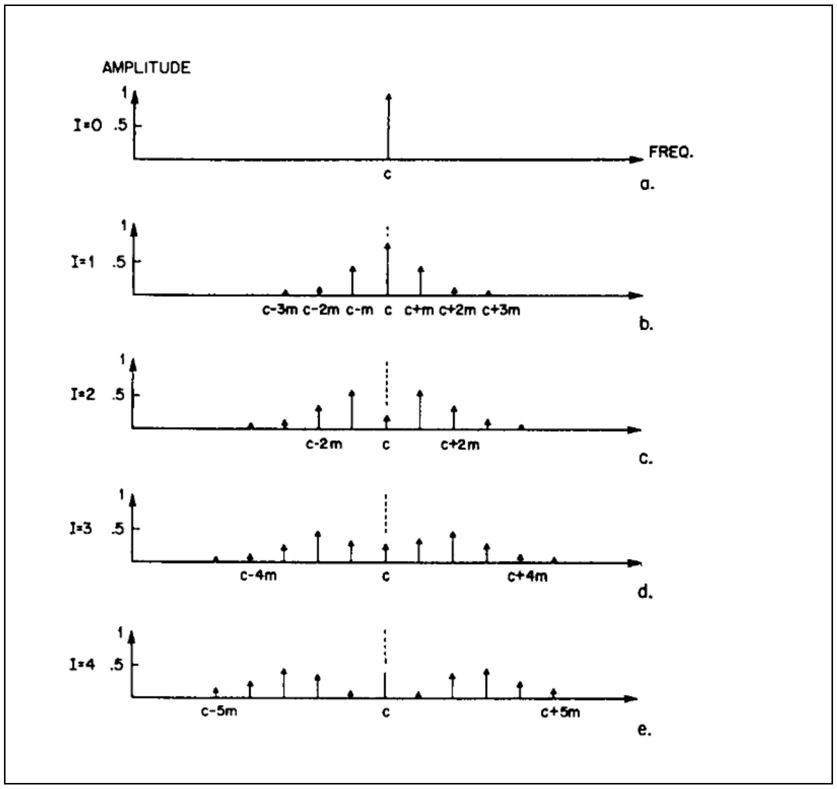


Figure 1. Modulation Depth of Sinusoidal

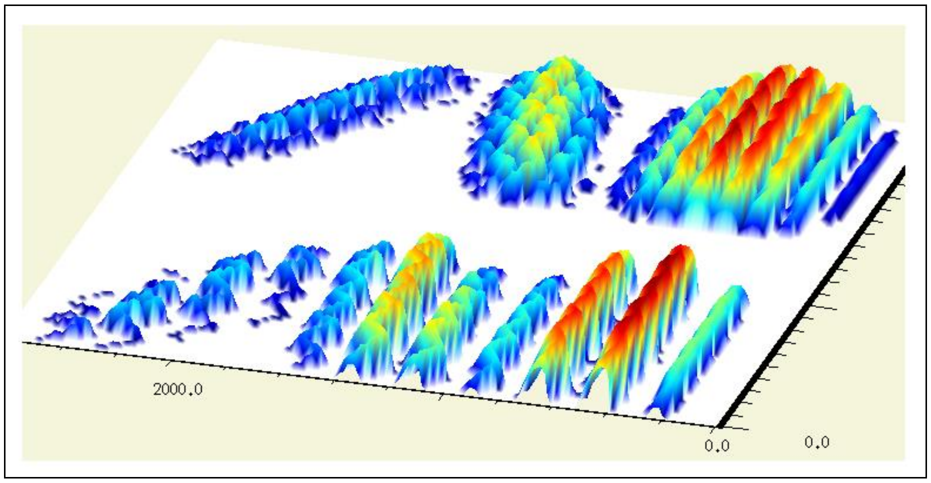


Figure 2. FM Complex Sound Generation

Part 3: Yamaha DX-7 Details

For the DX-7, there are some confusions regarding the FM synthesis. It uses phase modulation rather than frequency modulation but essentially they both are very similar. Phase modulation is almost exactly same as frequency modulation but modulation depth is the integral of FM but this makes very little difference because DX-7 uses only sine waves and integral of a sine wave is again sine wave. So it is possible to use both terms interchangeably. DX-7 has 6 oscillators each of them can produce only sine wave. In DX-7 terminology oscillators are called operators and I will use the word operator throughout this document. Sound generation technique of DX-7 is based on the modulation matrix of these oscillators. These matrixes are called algorithms. Also, some operators do have feedback chain which essentially means that operator is modulating its own frequency. It is possible to see feedback chain in Figure 3 (Chowning & Bristol, 1986). Feedback looped operators often sound chaotic and very noisy. There are at total 149 parameters. 6 Operators have same structure so some parameters of 149 are identical.

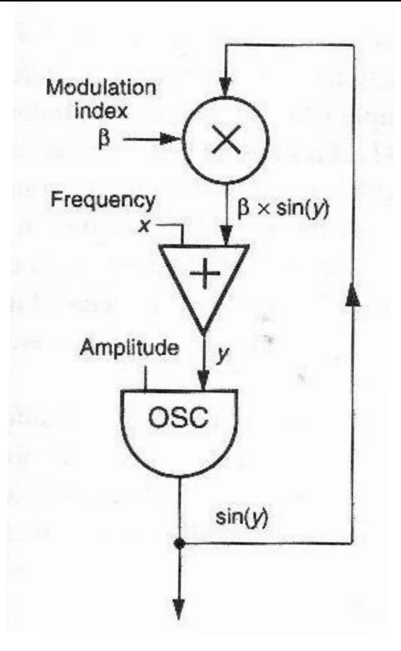


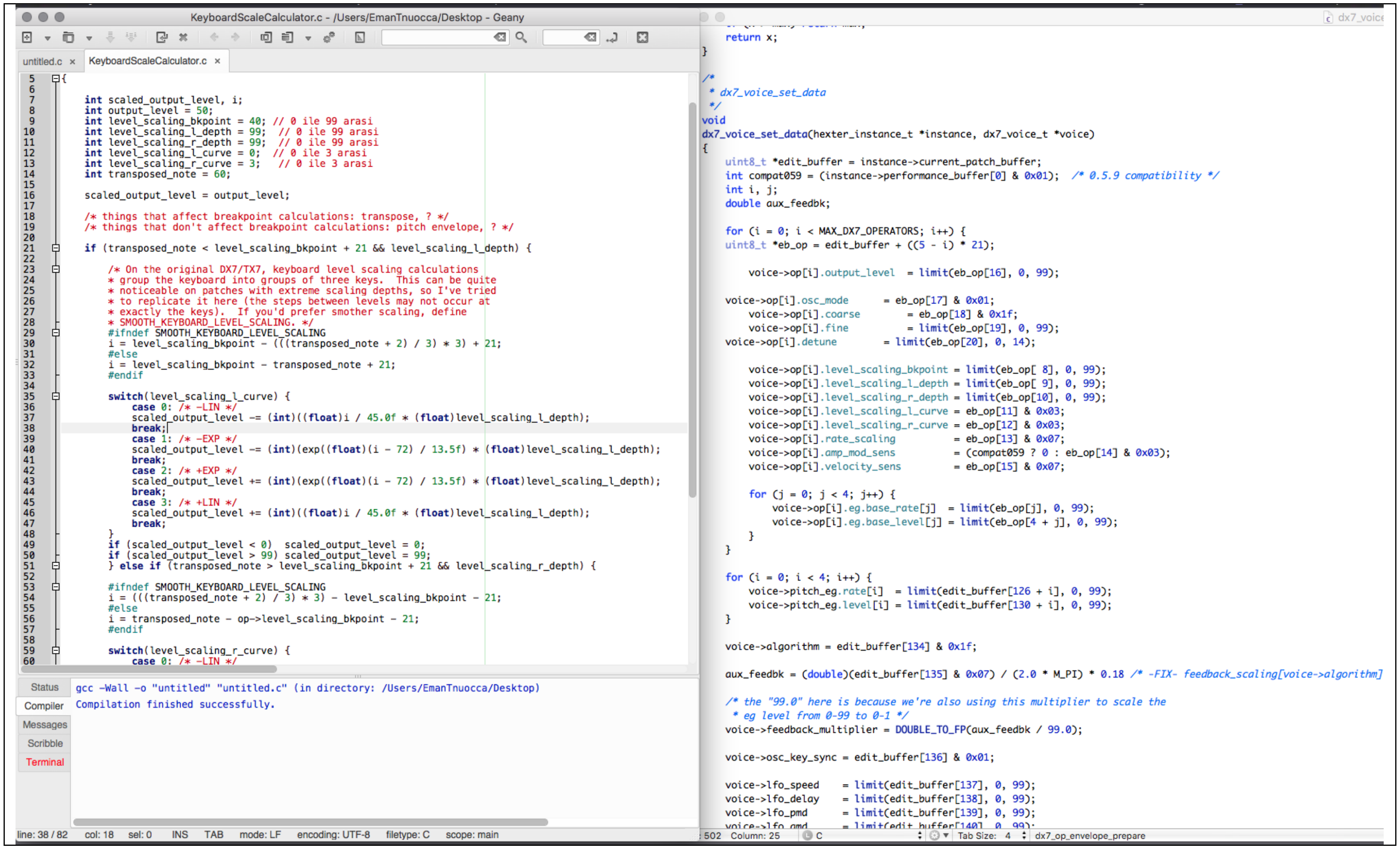
Figure 3. Feedback Chain

Part 4: Yamaha DX-7 Envelope Generation

There is plenty of information available which is highly related to hardware of DX-7. Also there is thorough information from the service manuals of the instrument. Main thing to understand with DX-7 is that there are only 6 sine oscillators for the use sound generation and there is nothing else to use, no filters, no complex waveforms etc. The volume level of operators is actually making all the difference to sound and so there are very different complex schemes which affect the volume level. One of them is envelope generation. Every six operator has its own envelope generation circuit which is essentially some Attack-Decay-Sustain-Release type signal multiplied with VCA of the operator. These envelope generation circuits are pair of VLSI chips, namely YM28190 EGS chips. These chips respond very differently from what I have seen previously. So in order replicate them with programming I treat them as block-box and gathered lots of data's with different parameters. Eventually I was able to come up with formula to catch an approximate response. There were difficulties such as rising signals and falling signals behaved differently. They both have 95 decibel dynamic range but with the same parameter values attacks are faster than decay and they somewhat have different response. So I have to come up with two different formulas. It is possible to find envelope generation formulas in the function of EnvCal at the Supercollider Appendix 1.

Part 5: Other Volume Parameters

Envelope parameters are the major part of the sound generation part and at total they cover 48 parameters (6 copies of 6 operators of 8 envelope values). There are other functions such as keyboard scale value which is something essentially decreasing and increasing volume over the keyboard range. Each 6 operator has its own keyboard scale parameters. In order to implement it, I have used some other implementations of the DX-7. I have found very good project called Dexed, which is another DX-7 emulator at the android platform. I found the code relating the scaling function, inside of the Dexed project and isolated it to analyse its output value. It was written in C code, as it is depicted in Figure 4. After I got the whole structure, I coded it in the Supercollider. There is also output level for each oscillator and by analysing it every 2 ticks of parameters are equivalent to 1.5 decibel especially when parameters are between 30 and 99. This is displayed on Figure 5. There is also velocity control for each operator which again affects the output level.


KeyboardScaleCalculator.c - /Users/EmanTnuocca/Desktop - Geany
untitled.c
KeyboardScaleCalculator.c
Status
Compiler
Messages
Scribble
Terminal
line: 38 / 82 col: 18 sel: 0 INS TAB mode: LF encoding: UTF-8 filetype: C scope: main
dx7_voice
502 Column: 25 C Tab Size: 4 dx7_op_envelope_prepare

```c
{

    int scaled_output_level, i;
    int output_level = 50;
    int level_scaling_bkpoint = 40; // 0 ile 99 arasi
    int level_scaling_l_depth = 99;  // 0 ile 99 arasi
    int level_scaling_r_depth = 99;  // 0 ile 99 arasi
    int level_scaling_l_curve = 0;  // 0 ile 3 arasi
    int level_scaling_r_curve = 3;   // 0 ile 3 arasi
    int transposed_note = 60;

    scaled_output_level = output_level;

    /* things that affect breakpoint calculations: transpose, ? */
    /* things that don't affect breakpoint calculations: pitch envelope, ? */

    if (transposed_note < level_scaling_bkpoint + 21 && level_scaling_l_depth) {

        /* On the original DX7/TX7, keyboard level scaling calculations
        * group the keyboard into groups of three keys.  This can be quite
        * noticeable on patches with extreme scaling depths, so I've tried
        * to replicate it here (the steps between levels may not occur at
        * exactly the keys).  If you'd prefer smother scaling, define
        * SMOOTH_KEYBOARD_LEVEL_SCALING. */
        #ifndef SMOOTH_KEYBOARD_LEVEL_SCALING
        i = level_scaling_bkpoint - (((transposed_note + 2) / 3) * 3) + 21;
        #else
        i = level_scaling_bkpoint - transposed_note + 21;
        #endif

        switch(level_scaling_l_curve) {
            case 0: /* -LIN */
            scaled_output_level -= (int)((float)i / 45.0f * (float)level_scaling_l_depth);
            break;
            case 1: /* -EXP */
            scaled_output_level -= (int)(exp((float)(i - 72) / 13.5f) * (float)level_scaling_l_depth);
            break;
            case 2: /* +EXP */
            scaled_output_level += (int)(exp((float)(i - 72) / 13.5f) * (float)level_scaling_l_depth);
            break;
            case 3: /* +LIN */
            scaled_output_level += (int)((float)i / 45.0f * (float)level_scaling_l_depth);
            break;
        }
        if (scaled_output_level < 0)  scaled_output_level = 0;
        if (scaled_output_level > 99) scaled_output_level = 99;
        } else if (transposed_note > level_scaling_bkpoint + 21 && level_scaling_r_depth) {

        #ifndef SMOOTH_KEYBOARD_LEVEL_SCALING
        i = (((transposed_note + 2) / 3) * 3) - level_scaling_bkpoint - 21;
        #else
        i = transposed_note - op->level_scaling_bkpoint - 21;
        #endif

        switch(level_scaling_r_curve) {
            case 0: /* -LIN */
```

```
gcc -Wall -o "untitled" "untitled.c" (in directory: /Users/EmanTnuocca/Desktop)
Compilation finished successfully.
```

```c
    return x;
}

/*
 * dx7_voice_set_data
 */
void
dx7_voice_set_data(hexter_instance_t *instance, dx7_voice_t *voice)
{
    uint8_t *edit_buffer = instance->current_patch_buffer;
    int compat059 = (instance->performance_buffer[0] & 0x01);  /* 0.5.9 compatibility */
    int i, j;
    double aux_feedbk;

    for (i = 0; i < MAX_DX7_OPERATORS; i++) {
    uint8_t *eb_op = edit_buffer + ((5 - i) * 21);

        voice->op[i].output_level  = limit(eb_op[16], 0, 99);

    voice->op[i].osc_mode       = eb_op[17] & 0x01;
        voice->op[i].coarse         = eb_op[18] & 0x1f;
        voice->op[i].fine           = limit(eb_op[19], 0, 99);
    voice->op[i].detune         = limit(eb_op[20], 0, 14);

        voice->op[i].level_scaling_bkpoint = limit(eb_op[ 8], 0, 99);
        voice->op[i].level_scaling_l_depth = limit(eb_op[ 9], 0, 99);
        voice->op[i].level_scaling_r_depth = limit(eb_op[10], 0, 99);
        voice->op[i].level_scaling_l_curve = eb_op[11] & 0x03;
        voice->op[i].level_scaling_r_curve = eb_op[12] & 0x03;
        voice->op[i].rate_scaling          = eb_op[13] & 0x07;
        voice->op[i].amp_mod_sens          = (compat059 ? 0 : eb_op[14] & 0x03);
        voice->op[i].velocity_sens         = eb_op[15] & 0x07;

        for (j = 0; j < 4; j++) {
            voice->op[i].eg.base_rate[j]  = limit(eb_op[j], 0, 99);
            voice->op[i].eg.base_level[j] = limit(eb_op[4 + j], 0, 99);
        }
    }

    for (i = 0; i < 4; i++) {
        voice->pitch_eg.rate[i]  = limit(edit_buffer[126 + i], 0, 99);
        voice->pitch_eg.level[i] = limit(edit_buffer[130 + i], 0, 99);
    }

    voice->algorithm = edit_buffer[134] & 0x1f;

    aux_feedbk = (double)(edit_buffer[135] & 0x07) / (2.0 * M_PI) * 0.18 /* -FIX- feedback_scaling[voice->algorithm]

    /* the "99.0" here is because we're also using this multiplier to scale the
     * eg level from 0-99 to 0-1 */
    voice->feedback_multiplier = DOUBLE_TO_FP(aux_feedbk / 99.0);

    voice->osc_key_sync = edit_buffer[136] & 0x01;

    voice->lfo_speed    = limit(edit_buffer[137], 0, 99);
    voice->lfo_delay    = limit(edit_buffer[138], 0, 99);
    voice->lfo_pmd      = limit(edit_buffer[139], 0, 99);
    voice->lfo_amd      = limit(edit_buffer[140], 0, 99);
```

Figure 4. My Scale Calculator

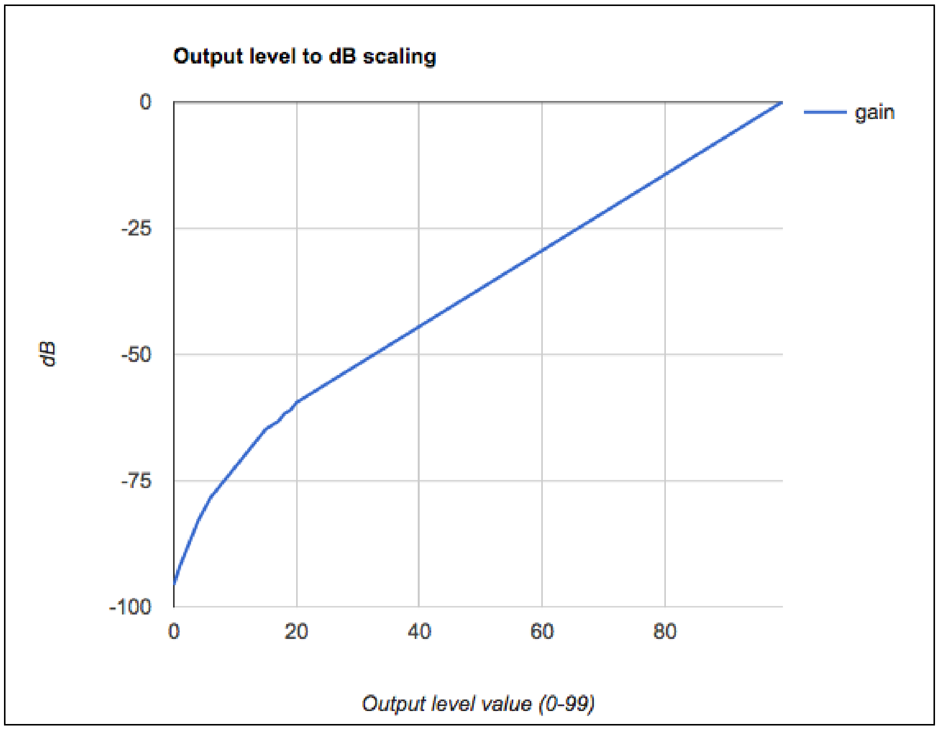


Figure 5. Output Level to dB Scaling

Part 6: Low Frequency Oscillators, Pitch Modulation Envelope

There are some functions which affect all oscillators at the same time. One of them is pitch envelope generator. It was very straight forward again I have studied Dexed C code source files in order to adapt my own pitch envelope. Once I have grasped for one node of the envelope, I was able to implement it for the whole pitch modulation envelope. Whole process of pitch modulation implementation took my entire day which was very fast compared to VCA envelope generator which took my couple of weeks before I get 99% approximate results. For the low frequency oscillator, I didn’t use built in Supercollider oscillators. I was able to write whole LFO waveforms in a buffer and it read from there. It was much more efficient. I have used Fourier Series to produce formulas. It is possible to find all related functions in the Appendix 1 of Supercollider code. Figure 6 shows my own produced waves and Figure 7 (Chowning & Bristol, 1986) shows the internal DX-7 waveforms.

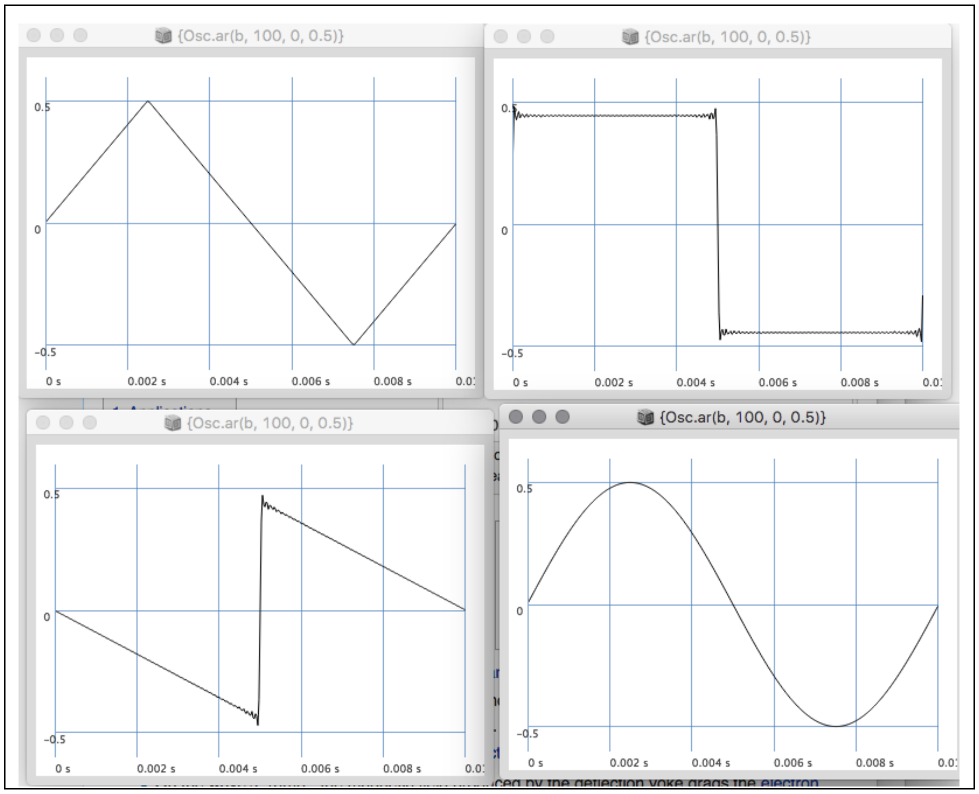


Figure 6. My LFO Waveforms

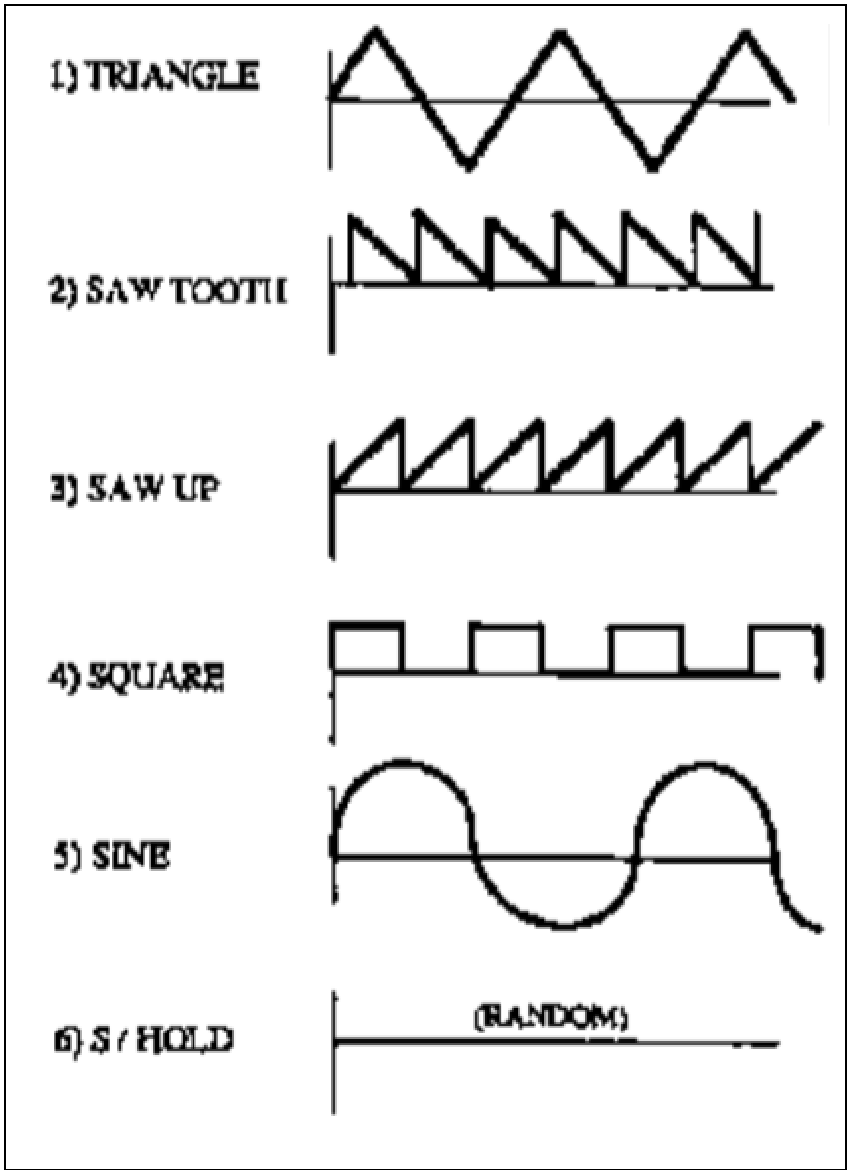


Figure 7. DX-7 LFO Waveshapes

Internship Experience: Observations and Analysis

My internship real world experience was mostly about programming and improving the existing code. I have used some of the knowledge which I gained at Sabanci University. For example signal processing was an important part in this project and I have used lots of things related to the EE-311, EE-312 and EE-313 courses. I have also attended workshops and helped other people's project. Some of them needed circuit knowledge and some again needed signal processing. Before the internship I have worked extra on the programming language Supercollider and collected extensive resources about the DX-7 synthesiser. At the STEIM, I found original DX-7 and it saved cost because I was considering buying a second-hand DX-7. After the workshop, I think my programming skills were increased.

Conclusion

At the end of the internship, I have accomplished what I have expected. I added extra features like loading original DX-7 presets to Supercollider. At the end, sounds were extremely similar. It was hard to distinguish one from the other. Some original DX-7 presets were not exactly accurate but I have found the reasons for that and noted all of them. Hopefully in the future I will able to debug them.

References

Chowning, John M. (1973). The synthesis of complex audio spectra by means of frequency modulation. *Journal of the Audio Engineering Society*. Volume 21, Number 7. 526-534.

Chowning, John M. & Bristol, David. (1986). *FM theory & applications: by musicians for musicians.* Tokyo Japan, Yamaha Music Foundation.

Johnstone, Robert. (1994). *The sound of one chip clapping: Yamaha and FM synthesis*. MIT Japan Program: Science, Technology, Management, Centre for International Studies, Massachusetts Institute of Technology. MIT JP 94-09

Schottstaedt, Bill (n.d.). An introduction to FM. *CCRMA Stanford.* Retrieved September 10, 2016, from https://ccrma.stanford.edu/software/snd/snd/fm.html

Shepard, Brian K. (2013). *Refining sound: A practical guide to synthesis and synthesisers*. London. Oxford University Press.

STEIM, (2016). Whats STEIM. Retrieved September 10, 2016, from http://steim.org/what-is-steim/

STEIM. (n.d.). In Wikipedia. Retrieved September 11, 2016, from https://en.wikipedia.org/wiki/STEIM

Yamaha DX7. (n.d.). In Wikipedia. Retrieved September 11, 2016, from https://en.wikipedia.org/wiki/Yamaha_DX7

Appendix 1 - CODE of the Supercollider DX-7 Emulator

```
(
	var
	coarseArrR =
[0.5,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31],
	coarseArrF =
	[
		1, 10, 100, 1000,
		1, 10, 100, 1000,
		1, 10, 100, 1000,
		1, 10, 100, 1000,
		1, 10, 100, 1000,
		1, 10, 100, 1000,
		1, 10, 100, 1000,
		1, 10, 100, 1000],
	fineArrR = [
		1.00,1.01,1.02,1.03,1.04,1.05,1.06,1.07,1.08,1.09,
		1.10,1.11,1.12,1.13,1.14,1.15,1.16,1.17,1.18,1.19,
		1.20,1.21,1.22,1.23,1.24,1.25,1.26,1.27,1.28,1.29,
		1.30,1.31,1.32,1.33,1.34,1.35,1.36,1.37,1.38,1.39,
		1.40,1.41,1.42,1.43,1.44,1.45,1.46,1.47,1.48,1.49,
		1.50,1.51,1.52,1.53,1.54,1.55,1.56,1.57,1.58,1.59,
		1.60,1.61,1.62,1.63,1.64,1.65,1.66,1.67,1.68,1.69,
		1.70,1.71,1.72,1.73,1.74,1.75,1.76,1.77,1.78,1.79,
		1.80,1.81,1.82,1.83,1.84,1.85,1.86,1.87,1.88,1.89,
		1.90,1.91,1.92,1.93,1.94,1.95,1.96,1.97,1.98,1.99],

	fineArrF =

	[

		1.000,1.023,1.047,1.072,1.096,1.122,1.148,1.175,1.202,1.230,

		1.259,1.288,1.318,1.349,1.380,1.413,1.445,1.479,1.514,1.549,

		1.585,1.622,1.660,1.698,1.738,1.778,1.820,1.862,1.905,1.950,

		1.995,2.042,2.089,2.138,2.188,2.239,2.291,2.344,2.399,2.455,

		2.512,2.570,2.630,2.692,2.716,2.818,2.884,2.951,3.020,3.090,

		3.162,3.236,3.311,3.388,3.467,3.548,3.631,3.715,3.802,3.890,

		3.981,4.074,4.169,4.266,4.365,4.467,4.571,4.677,4.786,4.898,

		5.012,5.129,5.248,5.370,5.495,5.623,5.754,5.888,6.026,6.166,

		6.310,6.457,6.607,6.761,6.918,7.079,7.244,7.413,7.586,7.762,

		7.943,8.128,8.318,8.511,8.718,8.913,9.120,9.333,9.550,9.772],

	voice_eg_rate_rise_duration = [	39.638000, 37.013000, 34.388000, 31.763000,
27.210500,	22.658000, 20.408000, 18.158000, 15.908000, 14.557000,	13.206000,
12.108333, 11.010667, 9.913000, 8.921000,	7.929000, 7.171333, 6.413667, 5.656000,
5.307000,	4.958000, 4.405667, 3.853333, 3.301000, 2.889000,	2.477000, 2.313000,
2.149000, 1.985000, 1.700500,	1.416000, 1.274333, 1.132667, 0.991000, 0.909000,
0.827000, 0.758000, 0.689000, 0.620000, 0.558000,	0.496000, 0.448667, 0.401333,
0.354000, 0.332000,	0.310000, 0.275667, 0.241333, 0.207000, 0.180950,	0.154900,
0.144567, 0.134233, 0.123900, 0.106200,	0.088500, 0.079667, 0.070833, 0.062000,
0.056800,	0.051600, 0.047300, 0.043000, 0.038700, 0.034800,	0.030900, 0.028000,
0.025100, 0.022200, 0.020815,	0.019430, 0.017237, 0.015043, 0.012850, 0.011230,
0.009610, 0.009077, 0.008543, 0.008010, 0.006960,	0.005910, 0.005357, 0.004803,
0.004250, 0.003960,	0.003670, 0.003310, 0.002950, 0.002590, 0.002420,	0.002250,
0.002000, 0.001749, 0.001499, 0.001443,	0.001387, 0.001242, 0.001096, 0.000951,
0.000815,	0.000815, 0.000815, 0.000815, 0.000815, 0.000815,	0.000815, 0.000815,
0.000815, 0.000815, 0.000815,	0.000815, 0.000815, 0.000815, 0.000815, 0.000815,
0.000815, 0.000815, 0.000815, 0.000815, 0.000815,	0.000815, 0.000815, 0.000815,
0.000815, 0.000815,	0.000815, 0.000815, 0.000815],

	dx7_voice_eg_rate_rise_percent = [	0.000010, 0.000010, 0.000010, 0.000010,
0.000010,	0.000010, 0.000010, 0.000010, 0.000010, 0.000010,	0.000010, 0.000010,
0.000010, 0.000010, 0.000010,    0.000010, 0.000010, 0.000010, 0.000010, 0.000010,
0.000010, 0.000010, 0.000010, 0.000010, 0.000010,    0.000010, 0.000010, 0.000010, 0.000010,
0.000010,    0.000010, 0.000010, 0.005007, 0.010005, 0.015003,    0.020000, 0.028000,
0.036000, 0.044000, 0.052000,    0.060000, 0.068000, 0.076000, 0.084000, 0.092000,
0.100000, 0.108000, 0.116000, 0.124000, 0.132000,    0.140000, 0.150000, 0.160000, 0.170000,
0.180000,    0.190000, 0.200000, 0.210000, 0.220000, 0.230000,    0.240000, 0.251000,
0.262000, 0.273000, 0.284000,    0.295000, 0.306000, 0.317000, 0.328000, 0.339000,
0.350000, 0.365000, 0.380000, 0.395000, 0.410000,    0.425000, 0.440000, 0.455000, 0.470000,
0.485000,    0.500000, 0.520000, 0.540000, 0.560000, 0.580000,    0.600000, 0.620000,
0.640000, 0.660000, 0.680000,    0.700000, 0.732000, 0.764000, 0.796000, 0.828000,
0.860000, 0.895000, 0.930000, 0.965000, 1.000000,    1.000000, 1.000000, 1.000000, 1.000000,
1.000000,    1.000000, 1.000000, 1.000000, 1.000000, 1.000000,    1.000000, 1.000000,
1.000000, 1.000000, 1.000000,    1.000000, 1.000000, 1.000000, 1.000000, 1.000000,
1.000000, 1.000000, 1.000000, 1.000000, 1.000000,    1.000000, 1.000000, 1.000000],

        velTable = [-99.0,-10.295511,-9.709229,-9.372207,-9.121093,-8.629703,-8.441805,-
8.205647,-7.810857,-7.653259,-7.299901,-7.242308,-6.934396, 6.727051,-6.594723,-6.427755,-
6.275133,-6.015212,-5.843023,-5.828787,-5.725659,-5.443202,-5.421110,-5.222133,-5.160615,-
5.038265,-4.948225,-4.812105,    -4.632120,-4.511531,-4.488645,-4.370043,-4.370610,-
4.058591,-4.066902,-3.952988,-3.909686,-3.810096,-3.691883,-3.621306,-3.527286,-3.437519,-
3.373512,-3.339195,-3.195983, -3.167622, -3.094788, -2.984045,-2.937463, -2.890713, -
2.890660, -2.691874,-2.649229, -2.544696, -2.498147, -2.462573,-2.396637, -2.399795, -
2.236338, -2.217625, -2.158336, -2.135569, -1.978521, -1.913965,-1.937082, -1.752275, -
1.704013, -1.640514,-1.598791, -1.553859, -1.512187, -1.448088,-1.450443, -1.220567, -
1.182340, -1.123139,-1.098469, -1.020642, -0.973039, -0.933279,-0.938035, -0.757380, -
0.740860, -0.669721,-0.681526, -0.555390, -0.519321, -0.509318,-0.456936, -0.460622, -
0.290578, -0.264393,-0.252716, -0.194141, -0.153566, -0.067842,-0.033402, -0.054947,
0.012860, 0.000000,-0.009715,  0.236054,  0.273956,  0.271968,0.330177,  0.345427,
0.352333,  0.433861,0.442952,  0.476411,  0.539632,  0.525355,0.526115,  0.707022, 0.701551,
0.734875,0.739149,  0.794320,  0.801578,  0.814225,0.818939,  0.897102,  0.895082,
0.927998,0.929797,  0.956112,  0.956789,  0.958121],

        /* This table converts LFO speed to frequency in Hz. It is based on

        * interpolation of Jamie Bullock's measurements. */

        dx7_voice_lfo_frequency = [

                0.062506,  0.124815,  0.311474,  0.435381,  0.619784,

                0.744396,  0.930495,  1.116390,  1.284220,  1.496880,

                1.567830,  1.738994,  1.910158,  2.081322,  2.252486,
                2.423650,  2.580668,  2.737686,  2.894704,  3.051722,
                3.208740,  3.366820,  3.524900,  3.682980,  3.841060,
                3.999140,  4.159420,  4.319700,  4.479980,  4.640260,
                4.800540,  4.953584,  5.106628,  5.259672,  5.412716,
                5.565760,  5.724918,  5.884076,  6.043234,  6.202392,
                6.361550,  6.520044,  6.678538,  6.837032,  6.995526,
                7.154020,  7.300500,  7.446980,  7.593460,  7.739940,
                7.886420,  8.020588,  8.154756,  8.288924,  8.423092,
                8.557260,  8.712624,  8.867988,  9.023352,  9.178716,
                9.334080,  9.669644, 10.005208, 10.340772, 10.676336,
                11.011900, 11.963680, 12.915460, 13.867240, 14.819020,
                15.770800, 16.640240, 17.509680, 18.379120, 19.248560,
                20.118000, 21.040700, 21.963400, 22.886100, 23.808800,
                24.731500, 25.759740, 26.787980, 27.816220, 28.844460,
                29.872700, 31.228200, 32.583700, 33.939200, 35.294700,
                36.650200, 37.812480, 38.974760, 40.137040, 41.299320,
                42.461600, 43.639800, 44.818000, 45.996200, 47.174400,
                47.174400, 47.174400, 47.174400, 47.174400, 47.174400,
                47.174400, 47.174400, 47.174400, 47.174400, 47.174400,
                47.174400, 47.174400, 47.174400, 47.174400, 47.174400,
                47.174400, 47.174400, 47.174400, 47.174400, 47.174400,
                47.174400, 47.174400, 47.174400, 47.174400, 47.174400,
                47.174400, 47.174400, 47.174400],
        /* This table converts pitch modulation sensitivity to semitones at full
        * modulation (assuming a perfectly linear pitch mod depth to pitch
        	* relationship).  It is from a simple averaging of Jamie Bullock's
        	* TX-data-1/PMD and TX-data-2/ENV data, and ignores the apparent ~0.1
        	* semitone positive bias that Jamie observed. [-FIX- smbolton: my
        	* inclination would be to call this bias, if it's reproducible, a
        	* non-desirable 'bug', and _not_ implement it in hexter. And, at
        	* least for my own personal build, I'd change that PMS=7 value to a
        	* full octave, since that's one thing that's always bugged me about
        	* my TX7.  Thoughts? ] */
        	dx7_voice_pms_to_semitones= [ //8
        		0.0, 0.450584, 0.900392, 1.474744,
        		2.587385, 4.232292, 6.982097, /* 11.722111 */ 12.0],
        	/* This table converts amplitude modulation depth to output level
        	* reduction at full modulation with an amplitude modulation sensitivity
        	* of 3.  It was constructed from regression of a very few data points,
        	* using this code:
        	*   perl -e 'for ($i = 0; $i <= 99; $i++) { printf " %f,\n", exp($i * 0.0428993 - 0.285189);
        }' >x.c
        	* and is probably rather rough in its accuracy. -FIX- */
        	dx7_voice_amd_to_ol_adjustment = [ //100
        		0.0, 0.784829, 0.819230, 0.855139, 0.892622, 0.931748,
        		0.972589, 1.015221, 1.059721, 1.106171, 1.154658, 1.205270,
        		1.258100, 1.313246, 1.370809, 1.430896, 1.493616, 1.559085,
        		1.627424, 1.698759, 1.773220, 1.850945, 1.932077, 2.016765,
        		2.105166, 2.197441, 2.293761, 2.394303, 2.499252, 2.608801,
        		2.723152, 2.842515, 2.967111, 3.097167, 3.232925, 3.374633,
        		3.522552, 3.676956, 3.838127, 4.006362, 4.181972, 4.365280,
        		4.556622, 4.756352, 4.964836, 5.182458, 5.409620, 5.646738,
        		5.894251, 6.152612, 6.422298, 6.703805, 6.997652, 7.304378,
        		7.624549, 7.958754, 8.307609, 8.671754, 9.051861, 9.448629,
        		9.862789, 10.295103, 10.746365, 11.217408, 11.709099,
        		12.222341, 12.758080, 13.317302, 13.901036, 14.510357,
        		15.146387, 15.810295, 16.503304, 17.226690, 17.981783,
        		18.769975, 19.592715, 20.451518, 21.347965, 22.283705,
        		23.260462, 24.280032, 25.344294, 26.455204, 27.614809,
        		28.825243, 30.088734, 31.407606, 32.784289, 34.221315,
        		35.721330, 37.287095, 38.921492, 40.627529, 42.408347,
        		44.267222, 46.207578, 48.232984, 50.347169, 52.75],
        /* This table converts pitch envelope level parameters into the
        * actual pitch shift in semitones.  For levels [17,85], this is
        * just ((level - 50) / 32 * 12), but at the outer edges the shift
        * is exagerated to 0 = -48 and 99 => 47.624.  This is based on
        * measurements I took from my TX7. */
        dx7_voice_pitch_level_to_shift = [ //128
        		-48.000000, -43.497081, -38.995993, -35.626132, -31.873615,
        		-28.495880, -25.500672, -22.872620, -20.998167, -19.496961,
        		-18.373238, -17.251065, -16.122139, -15.375956, -14.624487,
        		-13.876516, -13.126351, -12.375000, -12.000000, -11.625000,
        		-11.250000, -10.875000, -10.500000, -10.125000, -9.750000,
        		-9.375000, -9.000000, -8.625000, -8.250000, -7.875000,
        		-7.500000, -7.125000, -6.750000, -6.375000, -6.000000,
              -5.625000, -5.250000, -4.875000, -4.500000, -4.125000,
              -3.750000, -3.375000, -3.000000, -2.625000, -2.250000,
              -1.875000, -1.500000, -1.125000, -0.750000, -0.375000, 0.000000,
              0.375000, 0.750000, 1.125000, 1.500000, 1.875000, 2.250000,
              2.625000, 3.000000, 3.375000, 3.750000, 4.125000, 4.500000,
              4.875000, 5.250000, 5.625000, 6.000000, 6.375000, 6.750000,
              7.125000, 7.500000, 7.875000, 8.250000, 8.625000, 9.000000,
              9.375000, 9.750000, 10.125000, 10.500000, 10.875000, 11.250000,
              11.625000, 12.000000, 12.375000, 12.750000, 13.125000,
              14.251187, 15.001922, 16.126327, 17.250917, 18.375718,
              19.877643, 21.753528, 24.373913, 27.378021, 30.748956,
              34.499234, 38.627888, 43.122335, 47.624065, 48.0, 48.0, 48.0,
              48.0, 48.0, 48.0, 48.0, 48.0, 48.0, 48.0, 48.0, 48.0, 48.0,
              48.0, 48.0, 48.0, 48.0, 48.0, 48.0, 48.0, 48.0, 48.0, 48.0,
              48.0, 48.0, 48.0, 48.0, 48.0],
        /*

        adepth = dx7_voice_amd_to_ol_adjustment[voice->lfo_amd];

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%



        /* full-scale amp mod for adepth and edepth should be 52.75 and
        * their sum _must_ be limited to less than 128, or bad things will happen! */
        if (adepth > 127.5f) adepth = 127.5f;
        if (adepth + mdepth > 127.5f)

        mdepth = 127.5f - adepth;

        if (adepth + mdepth + edepth > 127.5f)

        edepth = 127.5f - (adepth + mdepth);


        %%%%%%%%%%%%%%%%%%%%%%%%%%


        double freq; PITCH RELATED


        voice->last_port_tuning = *instance->tuning;


        instance->fixed_freq_multiplier = *instance->tuning / 440.0;


        freq = voice->pitch_eg.value + voice->portamento.value +

        instance->pitch_bend -

        instance->lfo_value_for_pitch *

        (voice->pitch_mod_depth_pmd * FP_TO_DOUBLE(voice->lfo_delay_value) +

        voice->pitch_mod_depth_mods);


        voice->last_pitch = freq;


        freq += (double)(limit_note(voice->key + voice->transpose - 24));


        /* -FIX- this maybe could be optimized */

        /*      a lookup table of 8k values would give ~1.5 cent accuracy,
        	*	but then would interpolating that be faster than exp()? */
        	freq = *instance->tuning * exp((freq - 69.0) * M_LN2 / 12.0);


        	return freq;


        	*/
        	vr = Array.fill(256, 63),
        	nova = ParGroup.new,
        	noteArrayInferno = Array.newClear(128),
        	defme, defjamHead, betass = 0, headno, defPitchEnv,
        	envCal, dumm, selector = [66, 48, 54, 60, 66],
        	feedbackSel = [
        		35, 7, 35, 33, 35, 34, 35, 21,
        		7, 14, 35,  7, 35, 35,  7, 35,
        		7, 14, 35, 14, 14, 35, 35, 35,
        		35,35, 14, 28, 35, 28, 35, 35
        	],
        	feedbackIndex = [0, 0.0066, 0.0077, 0.011, 0.0153, 0.018, 0.0226, 0.063],
        	abc = Bus.audio(s,2),
        	waveform_selector,
        	wf = Array2D.fromArray(32,42, [ //algorithm matrix
        		/*Alg #1*/ /*1*/0,(pi * 4),0,0,0,0,/*2*/0,0,0,0,0,0,/*3*/0,0,0,(pi *
        4),0,0,/*4*/0,0,0,0,(pi * 4),0,/*5*/0,0,0,0,0,(pi * 4),/*6*/0,0,0,0,0,(pi * 4),/*V*/1,0,1,0,0,0,
        		/*Alg #2*/ /*1*/0,(pi * 4),0,0,0,0,/*2*/0,(pi * 4),0,0,0,0,/*3*/0,0,0,(pi *
        4),0,0,/*4*/0,0,0,0,(pi * 4),0,/*5*/0,0,0,0,0,(pi * 4),/*6*/0,0,0,0,0,0,/*V*/1,0,1,0,0,0,
        		/*Alg #3*/ /*1*/0,(pi * 4),0,0,0,0,/*2*/0,0,(pi *
        4),0,0,0,/*3*/0,0,0,0,0,0,/*4*/0,0,0,0,(pi * 4),0,/*5*/0,0,0,0,0,(pi * 4),/*6*/0,0,0,0,0,(pi *
        4),/*V*/1,0,0,1,0,0,

        		/*Alg #4*/ /*1*/0,(pi * 4),0,0,0,0,/*2*/0,0,(pi *
        4),0,0,0,/*3*/0,0,0,0,0,0,/*4*/0,0,0,0,(pi * 4),0,/*5*/0,0,0,0,0,(pi * 4),/*6*/0,0,0,(pi *
        4),0,0,/*V*/1,0,0,1,0,0,

        		/*Alg #5*/ /*1*/0,(pi * 4),0,0,0,0,/*2*/0,0,0,0,0,0,/*3*/0,0,0,(pi *
        4),0,0,/*4*/0,0,0,0,0,0,/*5*/0,0,0,0,0,(pi * 4),/*6*/0,0,0,0,0,(pi * 4),/*V*/1,0,1,0,1,0,

        		/*Alg #6*/ /*1*/0,(pi * 4),0,0,0,0,/*2*/0,0,0,0,0,0,/*3*/0,0,0,(pi *
        4),0,0,/*4*/0,0,0,0,0,0,/*5*/0,0,0,0,0,(pi * 4),/*6*/0,0,0,0,(pi * 4),0,/*V*/1,0,1,0,1,0,

        		/*Alg #7*/ /*1*/0,(pi * 4),0,0,0,0,/*2*/0,0,0,0,0,0,/*3*/0,0,0,(pi * 4),(pi *
        4),0,/*4*/0,0,0,0,0,0,/*5*/0,0,0,0,0,(pi * 4),/*6*/0,0,0,0,0,(pi * 4),/*V*/1,0,1,0,0,0,

        		/*Alg #8*/ /*1*/0,(pi * 4),0,0,0,0,/*2*/0,0,0,0,0,0,/*3*/0,0,0,(pi * 4),(pi *
        4),0,/*4*/0,0,0,(pi * 4),0,0,/*5*/0,0,0,0,0,(pi * 4),/*6*/0,0,0,0,0,0,/*V*/1,0,1,0,0,0,

        		/*Alg #9*/ /*1*/0,(pi * 4),0,0,0,0,/*2*/0,(pi * 4),0,0,0,0,/*3*/0,0,0,(pi * 4),(pi *
        4),0,/*4*/0,0,0,0,0,0,/*5*/0,0,0,0,0,(pi * 4),/*6*/0,0,0,0,0,0,/*V*/1,0,1,0,0,0,

        		/*Alg #10*/ /*1*/0,(pi * 4),0,0,0,0,/*2*/0,0,(pi * 4),0,0,0,/*3*/0,0,(pi *
        4),0,0,0,/*4*/0,0,0,0,(pi * 4),(pi * 4),/*5*/0,0,0,0,0,0,/*6*/0,0,0,0,0,0,/*V*/1,0,0,1,0,0,

        		/*Alg #11*/ /*1*/0,(pi * 4),0,0,0,0,/*2*/0,0,(pi *
        4),0,0,0,/*3*/0,0,0,0,0,0,/*4*/0,0,0,0,(pi * 4),(pi * 4),/*5*/0,0,0,0,0,0,/*6*/0,0,0,0,0,(pi *
        4),/*V*/1,0,0,1,0,0,

        		/*Alg #12*/ /*1*/0,(pi * 4),0,0,0,0,/*2*/0,(pi * 4),0,0,0,0,/*3*/0,0,0,(pi * 4),(pi *
        4),(pi * 4),/*4*/0,0,0,0,0,0,/*5*/0,0,0,0,0,0,/*6*/0,0,0,0,0,0,/*V*/1,0,1,0,0,0,

        		/*Alg #13*/ /*1*/0,(pi * 4),0,0,0,0,/*2*/0,0,0,0,0,0,/*3*/0,0,0,(pi * 4),(pi * 4),(pi
        * 4),/*4*/0,0,0,0,0,0,/*5*/0,0,0,0,0,0,/*6*/0,0,0,0,0,(pi * 4),/*V*/1,0,1,0,0,0,

        		/*Alg #14*/ /*1*/0,(pi * 4),0,0,0,0,/*2*/0,0,0,0,0,0,/*3*/0,0,0,(pi *
        4),0,0,/*4*/0,0,0,0,(pi * 4),(pi * 4),/*5*/0,0,0,0,0,0,/*6*/0,0,0,0,0,(pi * 4),/*V*/1,0,1,0,0,0,

        		/*Alg #15*/ /*1*/0,(pi * 4),0,0,0,0,/*2*/0,(pi * 4),0,0,0,0,/*3*/0,0,0,(pi *
        4),0,0,/*4*/0,0,0,0,(pi * 4),(pi * 4),/*5*/0,0,0,0,0,0,/*6*/0,0,0,0,0,0,/*V*/1,0,1,0,0,0,

        		/*Alg #16*/ /*1*/0,(pi * 4),(pi * 4),0,(pi * 4),0,/*2*/0,0,0,0,0,0,/*3*/0,0,0,(pi *
        4),0,0,/*4*/0,0,0,0,0,0,/*5*/0,0,0,0,0,(pi * 4),/*6*/0,0,0,0,0,(pi * 4),/*V*/1,0,0,0,0,0,
        		/*Alg #17*/ /*1*/0,(pi * 4),(pi * 4),0,(pi * 4),0,/*2*/0,(pi *
        4),0,0,0,0,/*3*/0,0,0,(pi * 4),0,0,/*4*/0,0,0,0,0,0,/*5*/0,0,0,0,0,(pi *
        4),/*6*/0,0,0,0,0,0,/*V*/1,0,0,0,0,0,

        		/*Alg #18*/ /*1*/0,(pi * 4),(pi * 4),(pi * 4),0,0,/*2*/0,0,0,0,0,0,/*3*/0,0,(pi *
        4),0,0,0,/*4*/0,0,0,0,(pi * 4),0,/*5*/0,0,0,0,0,(pi * 4),/*6*/0,0,0,0,0,0,/*V*/1,0,0,0,0,0,

        		/*Alg #19*/ /*1*/0,(pi * 4),0,0,0,0,/*2*/0,0,(pi *
        4),0,0,0,/*3*/0,0,0,0,0,0,/*4*/0,0,0,0,0,(pi * 4),/*5*/0,0,0,0,0,(pi * 4),/*6*/0,0,0,0,0,(pi *
        4),/*V*/1,0,0,1,1,0,

        		/*Alg #20*/ /*1*/0,0,(pi * 4),0,0,0,/*2*/0,0,(pi * 4),0,0,0,/*3*/0,0,(pi *
        4),0,0,0,/*4*/0,0,0,0,(pi * 4),(pi * 4),/*5*/0,0,0,0,0,0,/*6*/0,0,0,0,0,0,/*V*/1,1,0,1,0,0,

        		/*Alg #21*/ /*1*/0,0,(pi * 4),0,0,0,/*2*/0,0,(pi * 4),0,0,0,/*3*/0,0,(pi *
        4),0,0,0,/*4*/0,0,0,0,0,(pi * 4),/*5*/0,0,0,0,0,(pi * 4),/*6*/0,0,0,0,0,0,/*V*/1,1,0,1,1,0,

        		/*Alg #22*/ /*1*/0,(pi * 4),0,0,0,0,/*2*/0,0,0,0,0,0,/*3*/0,0,0,0,0,(pi *
        4),/*4*/0,0,0,0,0,(pi * 4),/*5*/0,0,0,0,0,(pi * 4),/*6*/0,0,0,0,0,(pi * 4),/*V*/1,0,1,1,1,0,

        		/*Alg #23*/ /*1*/0,0,0,0,0,0,/*2*/0,0,(pi *
        4),0,0,0,/*3*/0,0,0,0,0,0,/*4*/0,0,0,0,0,(pi * 4),/*5*/0,0,0,0,0,(pi * 4),/*6*/0,0,0,0,0,(pi *
        4),/*V*/1,1,0,1,1,0,

        		/*Alg #24*/ /*1*/0,0,0,0,0,0,/*2*/0,0,0,0,0,0,/*3*/0,0,0,0,0,(pi *
        4),/*4*/0,0,0,0,0,(pi * 4),/*5*/0,0,0,0,0,(pi * 4),/*6*/0,0,0,0,0,(pi * 4),/*V*/1,1,1,1,1,0,

        		/*Alg #25*/ /*1*/0,0,0,0,0,0,/*2*/0,0,0,0,0,0,/*3*/0,0,0,0,0,0,/*4*/0,0,0,0,0,(pi *
        4),/*5*/0,0,0,0,0,(pi * 4),/*6*/0,0,0,0,0,(pi * 4),/*V*/1,1,1,1,1,0,

        		/*Alg #26*/ /*1*/0,0,0,0,0,0,/*2*/0,0,(pi *
        4),0,0,0,/*3*/0,0,0,0,0,0,/*4*/0,0,0,0,(pi * 4),(pi * 4),/*5*/0,0,0,0,0,0,/*6*/0,0,0,0,0,(pi *
        4),/*V*/1,1,0,1,0,0,

        		/*Alg #27*/ /*1*/0,0,0,0,0,0,/*2*/0,0,(pi * 4),0,0,0,/*3*/0,0,(pi *
        4),0,0,0,/*4*/0,0,0,0,(pi * 4),(pi * 4),/*5*/0,0,0,0,0,0,/*6*/0,0,0,0,0,0,/*V*/1,1,0,1,0,0,

        		/*Alg #28*/ /*1*/0,(pi * 4),0,0,0,0,/*2*/0,0,0,0,0,0,/*3*/0,0,0,(pi *
        4),0,0,/*4*/0,0,0,0,(pi * 4),0,/*5*/0,0,0,0,(pi * 4),0,/*6*/0,0,0,0,0,0,/*V*/1,0,1,0,0,1,

        		/*Alg #29*/ /*1*/0,0,0,0,0,0,/*2*/0,0,0,0,0,0,/*3*/0,0,0,(pi *
        4),0,0,/*4*/0,0,0,0,0,0,/*5*/0,0,0,0,0,(pi * 4),/*6*/0,0,0,0,0,(pi * 4),/*V*/1,1,1,0,1,0,

        		/*Alg #30*/ /*1*/0,0,0,0,0,0,/*2*/0,0,0,0,0,0,/*3*/0,0,0,(pi *
        4),0,0,/*4*/0,0,0,0,(pi * 4),0,/*5*/0,0,0,0,(pi * 4),0,/*6*/0,0,0,0,0,0,/*V*/1,1,1,0,0,1,
                      /*Alg #31*/
        /*1*/0,0,0,0,0,0,/*2*/0,0,0,0,0,0,/*3*/0,0,0,0,0,0,/*4*/0,0,0,0,0,0,/*5*/0,0,0,0,0,(pi *
        4),/*6*/0,0,0,0,0,(pi * 4),/*V*/1,1,1,1,1,0,

                      /*Alg #32*/
        /*1*/0,0,0,0,0,0,/*2*/0,0,0,0,0,0,/*3*/0,0,0,0,0,0,/*4*/0,0,0,0,0,0,/*5*/0,0,0,0,0,0,/*6*/0,0,0,0,0,(
        pi * 4),/*V*/1,1,1,1,1,1



              ]),

              lfoSqr = Buffer.alloc(s, 512, 1),

              lfoSawDown = Buffer.alloc(s, 512, 1),

              lfoSawUp = Buffer.alloc(s, 512, 1),

              lfoSin = Buffer.alloc(s, 512, 1),

              lfoTri = Buffer.alloc(s, 512, 1),

              busMe = Bus.audio(s, 1);



              lfoSqr.sine3(Array.series(50, 1, 2), 1/Array.series(50, 1, 2)); //square waveform

              lfoTri.sine3(Array.series(50, 1, 2), ((1/(Array.series(50, 1, 2).squared)) * [1,-1]));
        //triangle waveform

              lfoSawDown.sine3(Array.series(100, 1, 1), (1/(Array.series(100, 1, 1)) * [-1,1]),4pi);
        //sawtooth waveform down

              lfoSawUp.sine3(Array.series(100, 1, 1), (1/(Array.series(100, 1, 1)) * [-1,1]),3pi);
        //sawtooth waveform rising

              lfoSin.sine3([1],[1]); //sine waveform

              waveform_selector = [lfoTri, lfoSawDown, lfoSawUp, lfoSqr, lfoSin, lfoSin];

              SynthDef(\InfEfx, {

                      arg outBus=0, pitch, gate,

                      lfoGet1 = 0, lfoGet2=1,

                      lfo_speed, lfo_wave,
        	lfo_delay, lfo_pmd, lfo_amd, lfo_sync, lfo_pms, pitchCons,
        	eg_r1, eg_r2, eg_r3, eg_r4, eg_L1, eg_L2, eg_L3, eg_L4;
        	//
        	var lfo, pitchenv, output, randomlfo, multiPitch;
        	//
        	FreeSelf.kr(gate);
        	randomlfo = LFNoise0.ar(lfo_speed, 1 * lfoGet1);
        	lfo = Osc.ar(lfo_wave,lfo_speed, mul:1 * lfoGet2);
        	multiPitch = pitchCons * lfo;
        	//lfo = DC.ar(0.1);
        	//Line.ar(3.3 snlik max delay time!);
        	//XLine.kr(0.0001,1,4.5);
        	//envgen gelmesi gerekio delay icin, trig
        	//sync off ise ve hic acilmadi ise random atmasi gerekio phase'e
        	//PHASE PART++
        	//tri en tepeden basliyor
        	//sawdown en asagidan baslio
        	//pms 4, pmd 40, +-1 semitone yapiyor
        	//sawup en yukardan
        	//square en asaidan
        	//sine sifirdan baslio
        	Out.ar(busMe, lfo)
        }).add;
        SynthDef(\Inferno, {
        	arg outBus=0, pitch, gate = 1,
        		lfospd = 1,//header

        		envPL0, envPL1, envPL2, envPL3, envPL4, envPR0, envPR1, envPR2, envPR3,

        	coars2, fine_2,coars3, fine_3,coars4, fine_4,coars5, fine_5,coars6, fine_6,

        		env1L0, env1L1, env1L2, env1L3, env1L4, env1R0, env1R1, env1R2, env1R3,
        env1C0, env1C1, env1C2, env1C3,

        		env2L0, env2L1, env2L2, env2L3, env2L4, env2R0, env2R1, env2R2, env2R3,
        env2C0, env2C1, env2C2, env2C3,

        		env3L0, env3L1, env3L2, env3L3, env3L4, env3R0, env3R1, env3R2, env3R3,
        env3C0, env3C1, env3C2, env3C3,

        		env4L0, env4L1, env4L2, env4L3, env4L4, env4R0, env4R1, env4R2, env4R3,
        env4C0, env4C1, env4C2, env4C3,

        		env5L0, env5L1, env5L2, env5L3, env5L4, env5R0, env5R1, env5R2, env5R3,
        env5C0, env5C1, env5C2, env5C3,

        		env6L0, env6L1, env6L2, env6L3, env6L4, env6R0, env6R1, env6R2, env6R3,
        env6C0, env6C1, env6C2, env6C3,

        		noteBlok1, noteBlok2, noteBlok3, noteBlok4, noteBlok5, noteBlok6,

        		dn0, dn1, dn2, dn3, dn4, dn5,

        		dn6, dn7, dn8, dn9, dn10, dn11,

        		dn12, dn13, dn14, dn15, dn16, dn17,

        		dn18, dn19, dn20, dn21, dn22, dn23,

        		dn24, dn25, dn26, dn27, dn28, dn29,

        		dn30, dn31, dn32, dn33, dn34, dn35,

        		dn36, dn37, dn38, dn39, dn40, dn41,

        		detun1, detun2, detun3, detun4, detun5, detun6,

        		//below is the general,

        		osc_sync, transpose,

        		gate1=1, gate1Rel = 1, amp=0.1, totVol, ampL1, ampL2, fltAtk;
```

```
			//
			var ctls, mods, chans, out,
			envAmp1, envEnv1, envAmp2, envEnv2, envAmp3, envEnv3 ,envAmp4,
envEnv4, envAmp5, envEnv5, envAmp6, envEnv6, dca, envAmpP, envEnvP;
			//
			//fdb = Array.fill(36, 1),
			//fdbNo = 1,
			//fdbInd = 1,
			//lfo = Osc.ar(lfo_wave, 10, 0, 0.5);
			//pitchenv=
			//fdb[fdbNo] = fdbInd;
			envEnvP = Env.new([ envPL0, envPL1, envPL2, envPL3, envPL4],
[envPR0,envPR1,envPR2,envPR3], 0, 3);
			envAmpP = EnvGen.kr(envEnvP, gate, doneAction:0);
			//
			envEnv1 = Env.new([(-1 * env1L0).dbamp ,(-1 * env1L1).dbamp, (-1 *
env1L2).dbamp, (-1 * env1L3).dbamp, (-1 * env1L4).dbamp],
[env1R0,env1R1,env1R2,env1R3], [env1C0,env1C1,env1C2,env1C3], 3);
			envAmp1 = EnvGen.kr(envEnv1, gate, doneAction:0 );
			envEnv2 = Env.new([(-1 * env2L0).dbamp ,(-1 * env2L1).dbamp, (-1 *
env2L2).dbamp, (-1 * env2L3).dbamp, (-1 * env2L4).dbamp],
[env2R0,env2R1,env2R2,env2R3], [env2C0,env2C1,env2C2,env2C3], 3);
			envAmp2 = EnvGen.kr(envEnv2, gate, doneAction:0 );
			envEnv3 = Env.new([(-1 * env3L0).dbamp ,(-1 * env3L1).dbamp, (-1 *
env3L2).dbamp, (-1 * env3L3).dbamp, (-1 * env3L4).dbamp],
[env3R0,env3R1,env3R2,env3R3], [env3C0,env3C1,env3C2,env3C3], 3);
			envAmp3 = EnvGen.kr(envEnv3, gate, doneAction:0 );
			envEnv4 = Env.new([(-1 * env4L0).dbamp ,(-1 * env4L1).dbamp, (-1 *
env4L2).dbamp, (-1 * env4L3).dbamp, (-1 * env4L4).dbamp],
[env4R0,env4R1,env4R2,env4R3], [env4C0,env4C1,env4C2,env4C3], 3);

			envAmp4 = EnvGen.kr(envEnv4, gate, doneAction:0 );

			envEnv5 = Env.new([(-1 * env5L0).dbamp ,(-1 * env5L1).dbamp, (-1 *
env5L2).dbamp, (-1 * env5L3).dbamp, (-1 * env5L4).dbamp],
[env5R0,env5R1,env5R2,env5R3], [env5C0,env5C1,env5C2,env5C3], 3);

			envAmp5 = EnvGen.kr(envEnv5, gate, doneAction:0 );

			envEnv6 = Env.new([(-1 * env6L0).dbamp ,(-1 * env6L1).dbamp, (-1 *
env6L2).dbamp, (-1 * env6L3).dbamp, (-1 * env6L4).dbamp],
[env6R0,env6R1,env6R2,env6R3], [env6C0,env6C1,env6C2,env6C3], 3);

			envAmp6 = EnvGen.kr(envEnv6, gate, doneAction:0 );

		envEnv1.test.plot;

			ctls = [

				[coars1 * fine_1 * ((pitch * noteBlok1) + ((detun1-7)/32)) *
(envAmpP.midiratio),  0, envAmp1 ],

				[coars2 * fine_2 * ((pitch * noteBlok2) + ((detun2-7)/32)) *
(envAmpP.midiratio),  0, envAmp2 ],

				[coars3 * fine_3 * ((pitch * noteBlok3) + ((detun3-7)/32)) *
(envAmpP.midiratio),  0, envAmp3 ],

				[coars4 * fine_4 * ((pitch * noteBlok4) + ((detun4-7)/32)) *
(envAmpP.midiratio),  0, envAmp4 ],

				[coars5 * fine_5 * ((pitch * noteBlok5) + ((detun5-7)/32)) *
(envAmpP.midiratio),  0, envAmp5 ],

				[coars6 * fine_6 * ((pitch * noteBlok6) + ((detun6-7)/32)) *
(envAmpP.midiratio),  0, envAmp6 ]

			];

			mods = [

				[dn0, dn1, dn2, dn3, dn4, dn5],

				[dn6, dn7, dn8, dn9, dn10, dn11],
                [dn12, dn13, dn14, dn15, dn16, dn17],
                [dn18, dn19, dn20, dn21, dn22, dn23],
                [dn24, dn25, dn26, dn27, dn28, dn29],
                [dn30, dn31, dn32, dn33, dn34, dn35]
        ];
        chans = [0, 1, 2, 3, 4, 5];
        out = FM7.ar(ctls, mods).slice(chans) * -12.dbamp;
        out = Mix.new([
                (out[0] * 1*  dn36),
                (out[1] * 1 * dn37),
                (out[2] * 1 * dn38),
                (out[3] * 1 * dn39),
                (out[4] * 1 * dn40),
                (out[5] * 1 * dn41),
        ]);
        //filterlar filan hep buraya gelcek
        DetectSilence.ar(out, doneAction:2);
        //out = out * Lag2.ar(In.ar(busMe),0.01);
        //Out.ar([0,1], 0.5 * In.ar(busMe)); //deneme
        Out.ar(abc, out.dup); //orj
        Out.ar(outBus, out.dup); //orj

}).add;
envCal = { arg // Area of hardcore envelope calculation function.
        vr1, //Level Current
        vr2, //Level Next
        vrx, //Speed
        vrv, //Volume
        velo,//Velocity
        velSens,
        level_scaling_bkpoint,
        level_scaling_l_depth,
        level_scaling_r_depth,
        level_scaling_l_curve,
        level_scaling_r_curve,
        transposed_note,
        rate_scaling,//implemented but not tuned!
        amp_mod_sens;//not sure if i put this globa
        var dummy, velFin, env_L0, env_L1, env_R0, vrvDec, endR0, dbStr, dbEnd;
        if (((transposed_note < (level_scaling_bkpoint + 21)) && (level_scaling_l_depth
> 0)), {
                dummy = (level_scaling_bkpoint - (((transposed_note + 2) / 3).asInt * 3) +
21);
                case
                { level_scaling_l_curve == 0 } // -lin
                {vrv = (vrv - ((dummy / 45) * level_scaling_l_depth).asInt)}
                { level_scaling_l_curve == 1 } // -exp
                {vrv = (vrv - (((dummy - 72) / 13.5).exp * level_scaling_l_depth).asInt)}
                { level_scaling_l_curve == 2 } // +exp
                {vrv = (vrv + (((dummy - 72) / 13.5).exp * level_scaling_l_depth).asInt)}
                { level_scaling_l_curve == 3 } // +lin
                {vrv = (vrv + ((dummy / 45) * level_scaling_l_depth).asInt)};

        });

        if (((transposed_note > (level_scaling_bkpoint + 21)) && (level_scaling_r_depth
> 0)), {

                dummy = (((transposed_note + 2) / 3).asInt * 3) - level_scaling_bkpoint -
21;

                case

                { level_scaling_r_curve == 0 }   // -lin

                {vrv = (vrv - ((dummy / 45) * level_scaling_r_depth).asInt)}

                { level_scaling_r_curve == 1 }   // -exp

                {vrv = (vrv - (((dummy - 72) / 13.5).exp * level_scaling_r_depth).asInt)}

                { level_scaling_r_curve == 2 }   // +exp

                {vrv = (vrv + (((dummy - 72) / 13.5).exp * level_scaling_r_depth).asInt)}

                { level_scaling_r_curve == 3 }   // +lin

                {vrv = (vrv + ((dummy / 45) * level_scaling_r_depth).asInt)};

        });

        rate_scaling = ((rate_scaling * (transposed_note - 21) / (126.0 - 21.0) * 127.0 /
128.0 * 6.5 )- 0.5).asInt;

        vrx = (rate_scaling + vrx).clip(0,99);

        vrv = vrv.clip(0,99);

        velFin = (velTable[velo] * velSens);

        //~vrv = ((~vrv+(~velTable[~velo] * ~velSens)).clip(0,127));//old version

        env_L0 = case

        {(100 > vr1) && (vr1 > 20)}   { (64 * (14 + ((vr1) >> 1)))}

        {(21 > vr1)  && (vr1 > 16)}   { (64 * (4 + vr1))}

        {(17 > vr1)  && (vr1 > 5)}    { (64 * (5 + vr1))}
```

```
{(6 > vr1)   && (vr1 > (-1))} { (64 * (2 * vr1))}
{99 < vr1}  { (64 * (14 + (99 >> 1)))};
env_L1 = case
{(100 > vr2) && (vr2 > 20)}   { (64 * (14 + ((vr2) >> 1)))}
{(21 > vr2)  && (vr2 > 16)}   { (64 * (4 + vr2))}
{(17 > vr2)  && (vr2 > 5)}    { (64 * (5 + vr2))}
{(6 > vr2)   && (vr2 > (-1))} { (64 * (2 * vr2 + 5))}
{99 < vr2}  { (64 * (14 + (99 >> 1)))};


vrvDec = ((vrv+ velFin).clip(0,127));//is this correct, limit 99?
env_R0 = (0.2819*(2.pow((vrx)*0.16))); // attack ise -1? decay ise + 0.5?
vrvDec = case
{(128 > vrvDec) && (vrvDec > 20)}   {(32 * (vrvDec + 28))}
{(21 > vrvDec)  && (vrvDec > 16)}   {(32 * (vrvDec + 17))}
{(17 > vrvDec)  && (vrvDec > 5)}    {(32 * (vrvDec + 11))}
{(6 > vrvDec)   && (vrvDec > (-1))} {(32 * (vrvDec))};
dbStr = ((99 - (vr1 + vrv - 99 + velFin)) / 1.33333333);


dbEnd = ((99 - (vr2 + vrv - 99 + velFin)) / 1.333333);
endR0 = if( (env_L0 > env_L1),
        {
                ((((env_L0 + vrvDec - 4064).clip(1,9000) - (env_L1 + vrvDec -
4064).clip(1,9000))* 0.0235) / env_R0)
        },
                {
                        (voice_eg_rate_rise_duration[vrx] *
(dx7_voice_eg_rate_rise_percent[((((vr2 * vrv) / 99) + velFin).clip(0,99)).asInt] -
dx7_voice_eg_rate_rise_percent[((((vr1 * vrv) / 99) + velFin).clip(0,99)).asInt]))
                }
        );
        //[dbStr, dbEnd, endR0].postln;
        [dbStr, dbEnd, endR0];


    };
    defjamHead = { arg a1; var abc = [], cba = [],  algo, fdbIndNo = Array.fill(42,
1),lfoDepth, lfoGet1, lfoGet2;


        /*abc = abc ++
        [


        ];*/
        /*abc =[
        \dn,        (vr[128]).clip(0,31), //algo
        \fdbNo,     feedbackSel[vr[128]],
        \fdbInd,    2 * (vr[129]).clip(0,7),
        //\fdb,    (vr[129]).clip(0,7),
        /*\keySyn,    vr[2].linlin(0,127,0.01,1),
        \transp,    vr[3].linexp(0,127,1,400),
        \lfospd,    vr[4].linexp(0,127,0.0001,5),
        \lfowav,    vr[5].linexp(0,127,0.01,30),
		\lfodly,   vr[5].linexp(0,127,0.01,30),
		\lfopmd,   vr[5].linexp(0,127,0.01,30),
		\lfopms,   vr[5].linexp(0,127,0.01,30),
		\lfoamd,   vr[5].linexp(0,127,0.01,30),
		\lfokey,   vr[5].linexp(0,127,0.01,30),
		\peRt1,    vr[5].linexp(0,127,0.01,30),
		\peRt2,    vr[5].linexp(0,127,0.01,30),
		\peRt3,    vr[5].linexp(0,127,0.01,30),
		\peRt4,    vr[5].linexp(0,127,0.01,30),
		\peLvl1,    vr[5].linexp(0,127,0.01,30),
		\peLvl2,    vr[5].linexp(0,127,0.01,30),
		\peLvl3,    vr[5].linexp(0,127,0.01,30),
		\peLvl4,    vr[5].linexp(0,127,0.01,30),
		//
		\totVol, vr[7].linlin(0,127,-24,24)*/];*/


		if((a1 == 1),
			{
				algo = (vr[128]).clip(0,31);
				fdbIndNo[feedbackSel[algo]] =  feedbackIndex[(vr[129])] *
(vr[129]);
				42.do(
					{
						arg x;
						abc = abc ++ [\dn ++ x, (wf[algo,x] *
fdbIndNo[x])];
```

```
				});
				abc;
			},
			{
				if(vr[133] == 5, {lfoGet1=1; lfoGet2 =0}, {lfoGet1 = 0; lfoGet2
=1 });
				lfoDepth = (dx7_voice_pms_to_semitones[vr[138]]) * (vr[135] /
99);
				lfoDepth.postln;
				cba = [
					\lfo_speed, dx7_voice_lfo_frequency[vr[132]],
					\lfo_wave, (waveform_selector[vr[133]]).bufnum, //random
signal icinde yap,
					//\lfoDepth, lfoDepth,
					/* sifirdan baslasin because we need to reset the LFO phase
*/
					\lfoGet1, lfoGet1,
					\lfoGet2, lfoGet2,
					\lfo_delay, ((vr[136]).clip(0,99)).linlin(0,99,0.01,3)//lineer
ramp ile carp
					//\lfo_amd,
					//\lfo_sync,

				];
				//cba.postln;
				cba;
				//a y.removeAt(1); ['a', 'b', 'c'].do({ arg item, i; [i, item].postln; });
```

```
                }
            );
        };
        defPitchEnv = { arg rate_1, rate_2, rate_3, rate_4, level_1, level_2, level_3, level_4;
            var endy,
            envPL0, envPL1, envPL2, envPL3, envPL4, envPR0, envPR1, envPR2,envPR3;
            //functions start here
            envPL0 = dx7_voice_pitch_level_to_shift[level_4];
            envPL1 = dx7_voice_pitch_level_to_shift[level_1];
            envPL2 = dx7_voice_pitch_level_to_shift[level_2];
            envPL3 = dx7_voice_pitch_level_to_shift[level_3];
            envPL4 = dx7_voice_pitch_level_to_shift[level_4];
            envPR0 = exp((rate_1 - 70.337897) / -25.580953) * abs((envPL1 - envPL0) /
    96.0) * 2.45;
            envPR1 = exp((rate_2 - 70.337897) / -25.580953) * abs((envPL2 - envPL1) /
    96.0) * 2.45;
            envPR2 = exp((rate_3 - 70.337897) / -25.580953) * abs((envPL3 - envPL2) /
    96.0) * 2.45;
            envPR3 = exp((rate_4 - 70.337897) / -25.580953) * abs((envPL4 - envPL3) /
    96.0) * 2.45;
            endy = [
                \envPL0, envPL0,
                \envPL1, envPL1,
                \envPL2, envPL2,
                \envPL3, envPL3,
                \envPL4, envPL4,
                \envPR0, envPR0,
				\envPR1, envPR1,
				\envPR2, envPR2,
				\envPR3, envPR3
			];
			//endy.postln;
			endy;
		};
		defme = { arg a1,b1;
			var a , bil, ptchEnv,
			envL  = Array2D.new(6,5),
			envR  = Array2D.new(6,4),
			//selector = [66, 48, 54, 60, 66],
			envC  = Array2D.new(6,4);
			6.do(
				{arg y;
					4.do(
						{
							arg x;//selector = [66, 48, 54, 60, 66],
							dumm = envCal.value(
								vr[selector[x] + y],//vr1, //Level Current
								vr[selector[x + 1] + y],//vr2, //Level Next
								(99-vr[(24 + (x * 6)) + y]),//vrx, //Speed
								vr[6 + y],//vrv, //Volume
								b1,//velo,//Velocity
								vr[108 + y],//velSens,
                                        vr[78 + y],//level_scaling_bkpoint,
                                        vr[90 + y],//level_scaling_l_depth,
                                        vr[102 + y],//level_scaling_r_depth,
                                        vr[84 + y],//level_scaling_l_curve,
                                        vr[96 + y],//level_scaling_r_curve,
                                        (a1 + 3 + vr[131] - 24),//transposed_note,
                                        vr[72 + y],//rate_scaling,//not implemented
yer
                                        vr[18 + y]//amp_mod_sens;//no
                                   );
                                   envL[y,x] = dumm[0];
                                   envR[y,x] = dumm[2];
                                   if(dumm[0] > dumm[1], { envC[y,x] = 3}
,{envC[y,x] = -3});
                         });
                         envL[y,4] = envL[y,0];
                    }
               );
               //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
               bil = defjamHead.value(1);
               a =[
                    \pitch, (a1 + vr[131] - 24 ).midicps,
                    \amp, b1.linlin(0,127,-18,18)];
               6.do({
                    arg c;
                    var trCoarse, trFine, trDetune, bloke;//176
```

```
        if(vr[176 + c] == 0, //midi CC 128 + 48 fixed frequency
            {
                trCoarse = coarseArrR[vr[0 + c]];
                trFine   = fineArrR[vr[114 + c]];
                trDetune = vr[12 + c];
                bloke = 1;
            },
            {
                trCoarse = coarseArrF[vr[0 + c]];
                trFine   = fineArrF[vr[114 + c]];
                trDetune = 39;
                bloke = 0;
        });
        a = a ++ [
            \coars ++ (c + 1).asString, trCoarse,
            \fine_ ++ (c + 1).asString, trFine,
            \detun ++ (c + 1).asString, trDetune,
            \noteBlok ++ (c + 1).asString, bloke,
        ];

    });
    6.do({arg k;
        a = a ++ [
            \coars ++ (k + 1).asString, coarseArrR[vr[0 + k]],
            \fine_ ++ (k + 1).asString, fineArrR[vr[114 + k]],
                    \env ++ (k + 1).asString ++ "L0", envL[k,0],
                    \env ++ (k + 1).asString ++ "L1", envL[k,1],
                    \env ++ (k + 1).asString ++ "L2", envL[k,2],
                    \env ++ (k + 1).asString ++ "L3", envL[k,3],
                    \env ++ (k + 1).asString ++ "L4", envL[k,4],
                    \env ++ (k + 1).asString ++ "R0", envR[k,0],
                    \env ++ (k + 1).asString ++ "R1", envR[k,1],
                    \env ++ (k + 1).asString ++ "R2", envR[k,2],
                    \env ++ (k + 1).asString ++ "R3", envR[k,3],
                    \env ++ (k + 1).asString ++ "C0", envC[k,0],
                    \env ++ (k + 1).asString ++ "C1", envC[k,1],
                    \env ++ (k + 1).asString ++ "C2", envC[k,2],
                    \env ++ (k + 1).asString ++ "C3", envC[k,3],
            ];

});
//a.postln;
/*a =[
\pitch, a1.midicps,
\amp, b1.linlin(0,127,-18,18),
\coars1, coarseArrR[vr[0]],
\fine_1, fineArrR[vr[114]],
\env1L0, envL[0,0],
\env1L1, envL[0,1],
\env1L2, envL[0,2],
                \env1L3, envL[0,3],

                \env1L4, envL[0,4],

                \env1R0, envR[0,0],

                \env1R1, envR[0,1],

                \env1R2, envR[0,2],

                \env1R3, envR[0,3],

                \env1C0, envC[0,0],

                \env1C1, envC[0,1],

                \env1C2, envC[0,2],

                \env1C3, envC[0,3],

                ];*/

                ptchEnv = defPitchEnv.value((99-vr[139]),(99-vr[140]),(99-vr[141]),(99-
vr[142]),vr[143],vr[144],vr[145],vr[146]);

                a = bil ++ a ++ ptchEnv;

                //a.postln;

                a;


        };

        MIDIdef.noteOn(\InfernoController, {arg vel, note;

                //150.do({ arg c; ("cc#" ++ c.asString ++ " = " ++ vr[c]).postln}); //general cc
printer, rates are not inverted.

                if(noteArrayInferno[note] !== nil,noteArrayInferno[note].free);

                if(vr[137] == 1, {//lfo sync controller

                        if(betass == 1,   {

                                headno.free;

                                betass = 1;
                                headno = Synth.before(nova,\InfEfx, defjamHead.value(0));
                        },
                        {
                                betass = 1;
                                headno = Synth.before(nova, \InfEfx, defjamHead.value(0))
                        }
                )},
                {
                        if(betass == 0,   {betass = 1;headno = Synth.before(nova ,\InfEfx,
defjamHead.value(0)).onFree{betass = 0}});
                });
                noteArrayInferno[note] = Synth(\Inferno ,defme.value(note, vel),nova);
                //noteArrayInferno[note] = Synth(\Inferno ,defme.value(note, vel), nova);
//orjburayi addtotail diye degistirdin
                ~test_Inferno = 1;
        },srcID:~midiInINST4,chan: 3);
        MIDIdef.noteOff(\noteoffmykeyInferno, {arg vel, note;
                noteArrayInferno[note].set(\gate,0);
                noteArrayInferno[note] = nil;
        },srcID:~midiInINST4,chan:3);
        MIDIdef.cc(\InfernoCC1, {arg ...args;
                args[0].postln;
                vr[args[1]] = args[0];
                //headno.setControls(defjamHead.value(0));
                //nova.setControls(defme.value(0)); // simdilik kapali, cpu intensive olmamasi
icin
```

```
	},(0..127),srcID:~midiInINST4,chan:3);
	MIDIdef.cc(\InfernoCC2, {arg ...args;
		args[0].postln;
		vr[args[1] + 128] = args[0];
		//nova.setControls(defme.value(0)); // simdilik kapali, cpu intensive olmamasi
icin
	},(0..127),srcID:~midiInINST4,chan:4);
	SynthDef(\InfernoDiskout, {arg bufnum;
		DiskOut.ar(bufnum, In.ar(abc,2));
	}).add;
	~midiOutInferno = MIDIOut.newByName("Cirklon", "Port 4");//.latency_(0.001);
	)


	//BUILT PATCH STORAGE HERE && ARRAY GELICEK BURAYA
	~midiOutInferno = MIDIOut.newByName("Cirklon", "Port 4").latency_(0.001);
	(
	~midiOutInferno.control(chan: 3, ctlNum: 0, val: 9);
	)


	//multitrack is not yet implemented
	//degerleri test ederken, her zaman bir if function olsun, deger yukarida ise diye
	//((((8096 - ((64 * (14 + (vr[3] >> 1))) + (32 * (99 + 28)))) * 0.0235) /
(0.2819*(2.pow(24*0.16)))) / (2 + (((64 * (14 + (99 >> 1))) / 256).floor)));
	//Keyboard rate scaling icin, complete dx7'de sayfa 130'a bak
	//inferno head'e patch load icin ikinci parametre ile lock yap, degeri 50den buyuk ise,
yollasin, boylece cakisma olmasin, load up kisminda
```

```
//ensonuncusunda boyle yap,

//\coars1, if( ~dene > 5, coarseArrR[vr[0]], coarseArrR[vr[0]+4]),

// eger algoritim degisiriyorsa, output osc sayisina bagli olarak output level scale oluyor


(

{

	var ctls, mods, chans;

	ctls = [

		[1000, 0, Decay.ar(Impulse.ar(5), 0.2)],

		[310, 0, Decay.ar(Impulse.ar(5), 0.2)],

		[800, 0, 0],

		[1200, 0, 0],

		[1500, 0, 0],

		[1800, 0, 0]

	];

	mods = [

		[2,MouseX.kr(0,50), 0, 0, 0, 0],

		[0, 0.4, 0, 0, 0, 0],

		[0, 0, 0, 0, 0, 0],

		[0, 0, 0, 0, 0, 0],

		[0, 0, 0, 0, 0, 0],

		[0, 0, 0, 0, 0, 0]

	];

	chans = [0, 0];

	FM7.ar(ctls, mods).slice(chans) * -12.dbamp;
}.play;

)

//No 99 to 127 ratio, just let it bigger value,

/*

For each 6 oscillator

(1)  Osc Coarse -- "OP1_FreqCoarse" = 0-32

(2)  Osc Fine -- "OP1_FreqFine" = 0-99

3)  Osc Detune -- "OP1_Detune" = 0-14

4)  Mod Sens -- "OP1_LFO_AmplitudeModSens" = 0-4

(5)  Env Rate-1 -- "OP1_EG_R1_reversed"  = 0-99

(6)  Env Rate-2 -- "OP1_EG_R2_reversed"  = 0-99

(7)  Env Rate-3 -- "OP1_EG_R3_reversed"  = 0-99

(8)  Env Rate-4 -- "OP1_EG_R4_reversed"  = 0-99

9)  Env Level-1-- "OP1_EG_L1"                  = 0-99

10) Env Level-2-- "OP1_EG_L2"                  = 0-99

11) Env Level-3-- "OP1_EG_L3"                  = 0-99

12) Env Level-4-- "OP1_EG_L4"                  = 0-99

13) Scaletel -- "OP1_KRS"                = 0-7

14) Keyboard Breakpoint -- "OP1_KLS_BreakPoint" = 0-99

15) L Curve      -- "OP1_KLS_LeftCurve"  = 0-3

16) L Depth      -- "OP1_KLS_LeftDepth"  = 0-99

17) R Curve      -- "OP1_KLS_RightCurve" = 0-3

18) R Depth      -- "OP1_KLS_RightDepth" = 0-99

19) Velocity     --"OP1_KeybVelSens"  = 0-7

20) Output Volume--"OP1_OutputLevel"= 0-99
```

21) Osc Fix Ratio -- "OP1_FreqMode" = 0-1

%%%%%->Break_

CC# 0<=>5 : Osc Coarse -------> DONE!

CC# 6<=>11 : Output Volume -------> DONE!

CC# 12<=>17 : Osc Detune -------> DONE!

CC# 18<=>23 : Mod Sens

CC# 24<=>29 : Env Rate-1 -------> DONE!

CC# 30<=>35 : Env Rate-2 -------> DONE!

CC# 36<=>41 : Env Rate-3 -------> DONE!

CC# 42<=>47 : Env Rate-4 -------> DONE!

CC# 48<=>53 : Env Level-1 -------> DONE!

CC# 54<=>59 : Env Level-2 -------> DONE!

CC# 60<=>65 : Env Level-3 -------> DONE!

CC# 66<=>71 : Env Level-4 -------> DONE!

CC# 72<=>77 : Scaletel -------> DONE!

CC# 78<=>83 : Keyboard Breakpoint -------> DONE!

CC# 84<=>89 : L Curve -------> DONE!

CC# 90<=>95 : L Depth -------> DONE!

CC# 96<=>101 : R Curve -------> DONE!

CC# 102<=>107 : R Depth -------> DONE!

CC# 108<=>113 : Velocity -------> DONE!

CC# 114<=>119 : Osc Fine -------> DONE!

CC# 5<=>0 : Osc Fix Ratio %% Connected Osc Coarse %% -------> DONE!

=================

Global

1) Algorithm -- "Algorithm"= 0-31 -------> DONE!

2) Feedback -- "Feedback" = 0-7 -------> DONE!

3) Osc Key Sync -- "Oscillator_Sync" = 0-1

4) Transpose -- "Transpose" = 0-48

5) LFO speed -- "LFO_Speed" = 0-99 -------> DONE!

6) LFO wave -- "LFO_Waveform" = 0-5 // triangle, saw down, saw up, square, sine, S&H

7) LFO delay -- "LFO_Delay" = 0-99

8) LFO PMD -- "LFO_PMD" = 0-99

9) LFO AMD -- "LFO_AMD" = 0-99

10)LFO Key Sync -- "LFO_Sync" = 0-1 -------> DONE!

11)Pitch EG Rate-1 -- "Pitch_EG_R1_reversed" = 0-99 -------> DONE!

12)Pitch EG Rate-2 -- "Pitch_EG_R2_reversed" = 0-99 -------> DONE!

13)Pitch EG Rate-3 -- "Pitch_EG_R3_reversed" = 0-99 -------> DONE!

14)Pitch EG Rate-4 -- "Pitch_EG_R4_reversed" = 0-99 -------> DONE!

15)Pitch EG Level-1-- "Pitch_EG_L1" = 0-99 -------> DONE!

16)Pitch EG Level-2-- "Pitch_EG_L2" = 0-99 -------> DONE!

17)Pitch EG Level-3-- "Pitch_EG_L3" = 0-99 -------> DONE!

18)Pitch EG Level-4-- "Pitch_EG_L4" = 0-99 -------> DONE!

19)LFO PMS -- "LFO_PitchModSens" = 0-7

==============.no need

Function Parameter

1)Portament Mode,

2)Portament Glisanddi

3)Portamento Time

Aftertouch

Breath

Foot Ctrl

Mod Wheel

PitchBend Range

Mono/Poly

*/

//27 JUNE 2016 10:41:19 (ECT +2)//