

Pentatonic scale																							
	Note	Midi	Hz			Note	Midi	Hz			Note	Midi	Hz			Note	Midi	Hz			Note	Midi	Hz
LOW octave					BASS octave					Middle octave					Treble octave					High octave			
C major					C major					C major					C major					C major			
	C	C1	32.7		C	C2	65.4			C		C3	130.8		C		C4	261.6		C		C5	523.3
	D	D1	36.7		D	D2	73.4			D		D3	146.8		D		D4	293.7		D		D5	587.3
	E	E1	41.2		E	E2	82.4			E		E3	164.8		E		E4	329.6		E		E5	659.3
	G	G1	49		G	G2	97.9			G		G3	196		G		G4	392		G		G5	784
	A	A1	55		A	A2	110			A		A3	220		A		A4	440		A		A5	880
	C	C2	65.4		C	C3	130.8			C		C4	261.6		C		C5	523.3		C		C6	1046.5
C minor					C minor					C minor					C minor					C minor			
	C	C1	32.7		C	C2	65.4			C		C3	130.8		C		C4	261.6		C		C5	523.3
	E♭	E♭1	38.9		E♭	E♭2	77.7			E♭		D#3	155.6		E♭		D#4	311.1		E♭		D#5	622.3
	F	F1	43.7		F	F2	87.3			F		F3	174.6		F		F4	349.2		F		F5	698.5
	G	G1	49		G	G2	97.9			G		G3	196		G		G4	392		G		G5	784
	B♭	B♭1	58.3		B♭	B♭2	116.5			B♭		G#3	207.7		B♭		A#4	466.2		B♭		A#5	932.3
	C	C2	65.4		C	C3	130.8			C		C4	261.6		C		C5	523.3		C		C6	1046.5
G Major					G Major					G Major					G Major					G Major			
	G	G1	49		G		G2	98		G		G3	196		G		G4	392		G		G5	784
	A	A1	55		A		A2	110		A		A3	220		A		A4	440		A			
	B	B1	61.7		B		B2	123.5		B		B3	246.9		B		B4	493.9		B			
	D	D2	73.4		D		D3	146.8		D		D4	293.7		D		D5	587.3		D			
	E	E2	82.4		E		E3	164.8		E		E4	329.6		E		E5	659.3		E			
	G	G2	98		G		G3	196		G		G4	392		G		G5	784		G			
G Minor					G Minor					G Minor					G Minor					G Minor			
	G	G1	49		G		G2	98		G		G3	196		G		G4	392		G		G5	784
	B	B1	61.7		B		B2	123.5		B		B3	246.9		B		B4	493.9		B			
	C	C2	65.4		C		C3	130.8		C		C4	261.6		C		C5	523.3		C			
	D	D2	73.4		D		D3	146.8		D		D4	293.7		D		D5	587.3		D			
	F	F2	87.3		F		F3	174.6		F		F4	349.2		F		F5	698.5		F			
	G	G2	98		G		G3	196		G		G4	392		G		G5	784		G			
D major					D major					D major					D major					D major			
	D	D1	36.7		D	D2	73.4			D		D3	146.8		D		D4	293.7		D		D5	587.3
	E	E1	41.2		E	E2	82.4			E		E3	164.8		E		E4	293.7		E		E5	659.3
	F#	F#1	46.2		F#	F#2	92.4			F#		F#3	185		F#		F#4	370		F#		F#5	740
	A	A1	55		A	A2	110			A		A3	220		A		A4	440		A		A5	880
	B	B1	61.7		B	B2	123.4			B		B3	246.9		B		B4	493.9		B		B5	987.8
	D	D2	73.4		D	D3	146.8			D		D4	293.7		D		D5	587.3		D		D6	1174.7
D minor					D minor					D minor					D minor					D minor			
	D	D1	36.7		D	D2	73.4			D		D3	146.8		D		D4	293.7		D		D5	587.3
	F	F1	43.7		F	F2	87.3			F		F3	174.6		F		F4	349.2		F		F5	698.5
	G	G1	49		G	G2	97.9			G		G3	196		G		G4	392		G		G5	784
	A	A1	55		A	A2	110			A		A3	220		A		A4	440		A		A5	880
	C	D2	65.4		C	D2	130.8			C		C4	261.6		C		C5	523.3		C		C6	1046.5
	D	D2	73.4		D	D3	146.8			D		D4	293.7		D		D5	587.3		D		D6	1174.7
A major					A major					A major					A major					A major			
	A	A1	55		A		A2	110		A		A3	220		A		A4	440		A			
	B	B0	61.7		B		B2	123.5		B		B3	246.9		B		B4	493.9		B			
	C#	C#1	69.2		C#		C#3	138.6		C#		C#4	277.2		C#		C#5	554.4		C#			
	E	E1	41.2		E		E3	164.8		E		E4	329.6		E		E5	659.3		E			
	F#	F#1	46.2		F#		F#3	185		F#		F#4	370		F#		F#5	740		F#			
	A	A2	110		A		A3	220		A		A4	440		A		A5	880		A			
A minor					A minor					A minor					A minor					A minor			
	A	A1	55		A		A2	110		A		A3	220		A		A4	440		A			
	C	B♭1	65.4		C♭		B2	123.5		C♭		B3	246.9		C♭		B4	493.9		C♭			
	D	D2	73.4		D		D3	146.8		D		D4	293.7		D		D5	587.3		D			
	E	E2	82.4		E		E3	164.8		E		E4	329.6		E		E5	659.3		E			
	G♭	F#2	92.5		G♭		F#3	185		G♭		F#4	370		G♭		F#5	740		G♭			

	A	A2	110		A	A3	220		A	A4	440		A	A5	880		A			
E major				E major				E major				E major				E major				
	E	E1	41.2		E	E2	82.4		E	E3	164.8		E	E4	329.6		E			
	F#	F#1	46.2		F#	F#2	92.5		F#	F#3	185		F#	F#4	370		F#			
	G#	G#1	51.9		G#	G#2	103.8		G#	G#3	207.7		G#	G#4	415.3		G#			
	B	B1	61.7		B	B2	123.5		B	B3	246.9		B	B4	493.9		B			
	C#	C#2	69.3		C#	C#3	138.6		C#	C#4	277.2		C#	C#5	554.4		C#			
	E	E2	82.4		E	E3	164.8		E	E4	329.6		E	E5	659.3		E			
E Minor				E Minor				E Minor				E Minor				E Minor				
	E	E1	41.2		E	E2	82.4		E	E3	164.8		E	E4	329.6		E			
	G	G1	49		G	G2	98		G	G3	196		G	G4	392		G			
	A	A1	55		A	A2	110		A	A3	220		A	A4	440		A			
	B	B1	61.7		B	B2	123.5		B	B3	246.9		B	B4	493.9		B			
	Db	C#2	69.3		Db	C#3	138.6		Db	C#4	277.2		Db	C#5	554.4		Db			
	E	E2	82.4		E	E3	164.8		E	E4	329.6		E	E5	659.3		E			
B Major				B Major				B Major				B Major				B Major				
	B	B1	61.7		B	B2	123.5		B	B3	246.9		B	B4	493.9		B			
	C#	C#2	69.3		C#	C#3	138.6		C#	C#4	277.2		C#	C#5	554.4		C#			
	D#	D#2	77.8		D#	D#3	155.6		D#	D#4	277.2		D#	D#5	622.3		D#			
	F#	F#2	92.5		F#	F#3	185		F#	F#4	370		F#	F#5	740		F#			
	G#	G#2	103.8		G#	G#3	207.7		G#	G#4	415.3		G#	G#5	830.6		G#			
	B	B2	123.5		B	B3	246.9		B	B4	493.9		B	B5	987.8		B			
B Minor				B Minor				B Minor				B Minor				B Minor				
	B	B1	61.7		B	B2	123.5		B	B3	246.9		B	B4	493.9		B			
	D	D2	73.4		D	D3	146.8		D	D4	293.7		D	D5	587.3		D			
	E	E2	82.4		E	E3	164.8		E	E4	329.6		E	E5	659.3		E			
	F#	F#2	92.5		F#	F#3	185		F#	F#4	370		F#	F#5	740		F#			
	A	A2	110		A	A3	220		A	A4	440		A	A5	880		A			
	B	B2	123.5		B	B3	246.9		B	B4	493.9		B	B5	987.8		B			
F# Major				F# Major				F# Major				F# Major				F# Major				
	F#	F#1	46.2		F#	F#2	92.5		F#	F#3	185		F#	F#4	370		F#			
	G#	G#1	51.9		G#	G#2	103.8		G#	G#3	207.7		G#	G#4	415.3		G#			
	A#	A#1	58.3		A#	A#2	116.5		A#	A#3	233.1		A#	A#4	466.2		A#			
	C#	C#2	69.3		C#	C#3	138.6		C#	C#4	277.2		C#	C#5	554.4		C#			
	D#	D#2	77.8		D#	D#3	155.6		D#	D#4	311.1		D#	D#5	622.3		D#			
	F#	F#2	92.5		F#	F#3	185		F#	F#4	370		F#	F#5	740		F#			
F# Minor				F# Minor				F# Minor				F# Minor				F# Minor				
	F#	F#1	46.2		F#	F#2	92.5		F#	F#3	185		F#	F#4	370		F#			
	A	A1	55		A	A2	110		A	A3	220		A	A4	440		A			
	B	B1	61.7		B	B2	123.5		B	B3	246.9		B	B4	493.9		B			
	C#	C#2	69.3		C#	C#3	138.6		C#	C#4	277.2		C#	C#5	554.4		C#			
	E	E2	82.4		E	E3	164.8		E	E4	329.6		E	E5	659.3		E			
	F#	F#2	92.5		F#	F#3	185		F#	F#4	370		F#	F#5	740		F#			
C# Major				C# Major				C# Major				C# Major				C# Major				
	C#	C#1	34.6		C#	C#2	69.3		C#	C#3	138.5		C#	C#4	277.2		C#	C#5	554.4	
	D#	D#1	38.9		D#	D#2	77.7		D#	D#3	146.8		D#	D#4	311.1		D#	D#5	622.3	
	F	F1	43.7		F	F2	87.3		F	F3	174.6		F	F4	349.2		F	F5	698.5	
	G#	G#1	51.9		G#	G#2	103.8		G#	G#3	207.6		G#	G#4	415.3		G#	G#5	830.6	
	A#	A#	58.3		A#	A#2	116.5		A#	A#3	233		A#	A#4	466.2		A#	A#5	932.3	
	C#	C#2	69.3		C#	C#3	138.5		C#	C#4	277.1		C#	C#5	554.4		C#	C#6	1108.7	
C# Minor				C# Minor				C# Minor				C# Minor				C# Minor				
	C#	C#1	34.6		C#	C#2	69.3		C#	C#3	138.5		C#	C#4	277.2		C#	C#5	554.4	
	E	E1	41.2		E	E2	82.4		E	E3	164.8		E	E4	329.6		E	E5	659.3	
	F#	F#1	46.2		F#	F#2	92.4		F#	F#3	184.9		F#	F#4	370		F#	F#5	740	
	G#	G#1	51.9		G#	G#2	103.8		G#	G#3	207.6		G#	G#4	369.9		G#	G#5	830.6	

	B	B1	61.7		B	B2	123.4		B	B3	246.9		B	B4	493.8		B	B5	987.8
	C#	C#2	69.3		C#	C#3	138.5		C#	C#4	277.1		C#	C#5	554.3		C#	C#6	1108.7
F Major				F Major				F Major				F Major				F Major			
	F	F1	43.7		F	F2	87.3		F	F3	174.6		F	F4	349.2		F		
	G	G1	49		G	G2	98		G	G3	196		G	G4	392		G		
	A	A1	55		A	A2	110		A	A3	220		A	A4	440		A		
	C	C2	65.4		C	C3	130.8		C	C4	261.6		C	C5	523.3		C		
	D	D2	73.4		D	D3	146.8		D	D4	293.7		D	D5	587.3		D		
	F	F2	87.3		F	F3	174.6		F	F4	349.2		F	F5	698.5		F		
F Minor				F Minor				F Minor				F Minor				F Minor			
	F	F1	43.7		F	F2	87.3		F	F3	174.6		F	F4	349.2		F		
	A	A1	55		A	A2	110		A	A3	220		A	A4	440		A		
	Bb	A#1	58.3		Bb	A#2	116.5		Bb	A#3	233.1		Bb	A#4	466.2		Bb		
	C	C2	65.4		C	C3	130.8		C	C4	261.6		C	C5	523.3		C		
	E	E2	82.4		E	E3	164.8		E	E4	329.6		E	E5	659.3		E		
	F	F2	87.3		F	F3	174.6		F	F4	349.2		F	F5	698.5		F		
Bb Major				Bb Major				Bb Major				Bb Major				Bb Major			
	Bb	A#1	58.3		Bb	A#2	116.5		Bb	A#3	233.1		Bb	A#4	466.2		Bb		
	C	C2	65.4		C	C3	130.8		C	C4	261.6		C	C5	523.3		C		
	D	D2	73.4		D	D3	146.8		D	D4	293.7		D	D5	587.3		D		
	F	F2	87.3		F	F3	174.6		F	F4	349.2		F	F5	698.5		F		
	G	G2	98		G	G3	196		G	G4	392		G	G5	784		G		
	Bb	A#2	116.5		Bb	A#3	233.1		Bb	A#4	466.2		Bb	A#5	932.3		Bb		
Bb Minor				Bb Minor				Bb Minor				Bb Minor				Bb Minor			
	Bb	A#1	58.3		Bb	A#2	116.5		Bb	A#3	233.1		Bb	A#4	466.2		Bb		
	D	D2	73.4		D	D3	146.8		D	D4	293.7		D	D5	587.3		D		
	E	E2	73.4		E	E3	164.8		E	E4	329.6		E	E5	659.3		E		
	F	F2	87.3		F	F3	174.6		F	F4	349.2		F	F5	698.5		F		
	A	A2	110		A	A3	220		A	A4	440		A	A5	880		A		
	Bb	A#2	116.5		Bb	A#3	233.1		Bb	A#4	466.2		Bb	A#5	932.3		Bb		
Eb Major				Eb Major				Eb Major				Eb Major				Eb Major			
	Eb	D#1	38.9		Eb	D#2	77.8		Eb	D#3	155.6		Eb	D#4	311.1		Eb		
	F	F1	43.7		F	F2	87.3		F	F3	174.6		F	F4	349.2		F		
	G	G1	49		G	G2	98		G	G3	196		G	G4	392		G		
	Bb	A#1	58.3		Bb	A#2	116.5		Bb	A#3	233.1		Bb	A#4	466.2		Bb		
	C	C2	65.4		C	C3	130.8		C	C4	261.6		C	C5	523.3		C		
	Eb	D#2	77.8		Eb	D#3	155.6		Eb	D#4	311.1		Eb	D#5	622.3		Eb		
Eb minor				Eb minor				Eb minor				Eb minor				Eb minor			
	Eb	D#1	38.9		Eb	D#2	77.7		Eb	E3	164.8		Eb	D#4	311.1		Eb		
	G	G1	49		G	G2	98		G	G3	196		G	G4	392		G		
	A	A1	55		A	A2	110		A	A3	220		A	A4	440		A		
	Bb	A#1	58.3		Bb	A#2	116.5		Bb	A#3	233.1		Bb	A#4	466.2		Bb		
	D	D2	73.4		D	D3	146.8		D	D4	293.7		D	D5	587.3		D		
	Eb	D#2	77.7		Eb	E3	164.8		Eb	D#4	311.1		Eb	D#5	622.3		Eb		
Ab Major				Ab Major				Ab Major				Ab Major				Ab Major			
	Ab	G#1	51.9		Ab	G#2	103.8		Ab	G#3	207.7		Ab	G#4	415.3		Ab		
	Bb	A#1	58.3		Bb	A#2	116.5		Bb	A#3	233.1		Bb	A#4	466.2		Bb		
	C	C2	65.4		C	C3	130.8		C	C4	261.6		C	C5	523.3		C		
	Eb	D#2	77.8		Eb	D#3	155.6		Eb	D#4	311.1		Eb	D#5	622.3		Eb		
	F	F2	87.3		F	F3	174.6		F	F4	349.2		F	F5	698.5		F		
	Ab	G#2	103.8		Ab	G#3	207.7		Ab	G#4	415.3		Ab	G#5	830.6		Ab		
Ab Minor				Ab Minor				Ab Minor				Ab Minor				Ab Minor			
	Ab	G#1	51.9		Ab	G#2	103.8		Ab	G#3	207.7		Ab	G#4	415.3		Ab		
	C	C2	65.4		C	C3	130.8		C	C4	261.6		C	C5	523.3		C		
	Db	C#2	69.3		Db	C#3	138.6		Db	C#4	277.2		Db	C#5	554.4		Db		
	Eb	D#2	77.8		Eb	D#3	155.6		Eb	D#4	311.1		Eb	D#5	622.3		Eb		
	Gb	F#2	92.5		Gb	F#3	185		Gb	F#4	370		Gb	F#5	740		Gb		
	Ab	G#2	103.8		Ab	G#3	207.7		Ab	G#4	415.3		Ab	G#5	830.6		Ab		


```

#include <MozziGuts.h> //https://sensorium.github.io/Mozzi/learn/
#include <Oscil.h> // oscillator
#include <Smooth.h>
#include <AutoMap.h> // maps unpredictable inputs to a range
#include <EEPROM.h>

#include <tables/cos2048_int8.h> // table for Oscils to play
#include <tables/sin2048_int8.h>
#include <tables/saw2048_int8.h>
#include <tables/square_no_alias_2048_int8.h>
#include <tables/triangle2048_int8.h>
#include <tables/triangle_dist_cubed_2048_int8.h>
#include <tables/triangle_dist_squared_2048_int8.h>
#include <tables/triangle_hermes_2048_int8.h>
#include <tables/triangle_valve_2048_int8.h>
#include <tables/triangle_valve_2_2048_int8.h>

#define NUM_TABLES 10
const int8_t *WAVE_TABLES[NUM_TABLES] = {COS2048_DATA , SIN2048_DATA,
SAW2048_DATA, SQUARE_NO_ALIAS_2048_DATA, TRIANGLE2048_DATA,
TRIANGLE_DIST_CUBED_2048_DATA, TRIANGLE_DIST_SQUARED_2048_DATA ,
TRIANGLE_HERMES_2048_DATA , TRIANGLE_VALVE_2048_DATA,
TRIANGLE_VALVE_2_2048_DATA};
#define TABLE_SIZE 2048

// desired carrier frequency max and min, for AutoMap
const int MIN_CARRIER_FREQ = 22;
const int MAX_CARRIER_FREQ = 440;

// frequency min max
const int MIN = 1;
const int MAX = 11;

// desired intensity max and min, for AutoMap, note they're inverted for reverse
dynamics
const int MIN_INTENSITY = 700;
const int MAX_INTENSITY = 10;

// desired mod speed max and min, for AutoMap, note they're inverted for reverse
dynamics
const int MIN_MOD_SPEED = 10000;
const int MAX_MOD_SPEED = 1;

AutoMap kMapCarrierFreq(0, 1023, MIN_CARRIER_FREQ, MAX_CARRIER_FREQ);
AutoMap kMapIntensity(0, 1023, MIN_INTENSITY, MAX_INTENSITY);
AutoMap kMapModSpeed(0, 1023, MIN_MOD_SPEED, MAX_MOD_SPEED);
AutoMap mapThis(0, 1023, MIN, MAX);

//rate of audio 128 is probably max
#define CONTROL_RATE 128

const uint8_t MOD_RATIO = 5; // brightness (harmonics)
long FMIntensity; // carries control info from updateControl to updateAudio

// smoothing for intensity to remove clicks on transitions
const float SMOOTHNESS = 0.95f;
Smooth <long> aSmoothIntensity(SMOOTHNESS);

```

```

// number of steps
#define SYNTHCOUNT 8

//each class stores setting for one synth
class SynthSetting {
public:
    int FRQ;
    int FMIntensity1;
    int FMIntensity2;
    float carrierFreq;
    int lastPlayedButton;
    int modFreq;
    int modSpeed;
    int FMIntensity2Mapped;
    Oscil<TABLE_SIZE, AUDIO_RATE> aCarrier;
    Oscil<TABLE_SIZE, AUDIO_RATE> aModulator;
    Oscil<TABLE_SIZE, CONTROL_RATE> kIntensityMod;
};

//create a array of synths
SynthSetting synSett[SYNTHCOUNT];

const byte LEDS[] = {
    37, 36, 35, 34, 33, 32, 31, 30,
    29, 28, 27, 26, 25, 24, 23, 22
};
#define NUMLEDS sizeof(LEDS)

const byte BUTTONS[] = {
    46, 47, 48, 49, 50, 51, 52, 53
};
#define NUMBUTTONS sizeof(BUTTONS)

#define DEBOUNCE 5
byte pressed[NUMBUTTONS], justpressed[NUMBUTTONS], justreleased[NUMBUTTONS];

//notes in pentatonic scale C C# D D# E F F# G
//const float NOTELOOKUP[] = {261.626, 277.183, 293.665, 311.127, 329.628,
349.228, 369.994, 391.995};

int scaleSelect = 0;
int octaveSelect = 0;

const float NOTELOOKUP[24][4][6] =
{
    {
        //C majour
        {32.7, 36.7, 41.2, 49, 55, 65.4},
        {65.4, 73.4, 82.4, 97.9, 110, 130.8},
        {130.8, 146.8, 164.8, 196, 220, 261.6},
        {261.6, 293.7, 329.6, 392, 440, 523.3}
    },
    {
        //C minour
        {32.7, 38.9, 43.7, 49, 58.3, 65.4},
        {65.4, 77.7, 87.3, 97.9, 116.5, 130.8},
        {130.8, 155.6, 174.6, 196, 207.7, 261.6},
    }
}

```

```

    {261.6, 311.1, 349.2, 392, 466.2, 523.3}

},
{
    //G maj
    {49, 55, 61.7, 73.4, 82.4, 98},
    {98, 110, 123.5, 146.8, 164.8, 196},
    {196, 220, 246.9, 293.7, 329.6, 392},
    {392, 440, 493.9, 587.3, 659.3, 784}
},
{
    //G Min
    {49, 61.7, 65.4, 73.4, 87.3, 98},
    {98, 123.5, 130.8, 146.8, 174.6, 196},
    {196, 246.9, 261.6, 293.7, 349.2, 392},
    {392, 493.9, 523.3, 587.3, 698.5, 784}
},
{
    //D maj
    {36.7, 41.2, 46.2, 55, 61.7, 73.4},
    {73.4, 82.4, 92.4, 110, 123.4, 146.8},
    {146.8, 164.8, 185, 220, 246.9, 293.7},
    {293.7, 293.7, 370, 440, 493.9, 587.3}
},
{
    //D Min
    {36.7, 43.7, 49, 55, 65.4, 73.4},
    {73.4, 87.3, 97.9, 110, 130.8, 146.8 },
    {146.8, 174.6, 196, 220, 261.6, 293.7},
    {293.7, 349.2, 392, 440, 523.3, 587.3}
},
{
    //A maj
    {55, 61.7, 69.2, 82.4, 92.4, 110},
    {110, 123.5, 138.6, 164.8, 185, 220},
    {220, 246.9, 277.2, 329.6, 370, 440},
    {440, 493.9, 554.4, 659.3, 740, 880}
},
{
    //A Min
    {55, 65.4, 73.4, 82.4, 92.5, 110},
    {110, 123.5, 146.8, 164.8, 185, 220},
    {220, 246.9, 293.7, 329.6, 370, 440},
    {440, 493.9, 587.3, 659.3, 740, 880}
},
{
    //E Maj
    {41.2, 46.2, 51.9, 61.7, 69.3, 82.4},
    {82.4, 92.5, 103.8, 123.5, 138.6, 164.8},
    {164.8, 185, 207.7, 246.9, 277.2, 329.6},
    {329.6, 370, 415.3, 493.9, 554.4, 659.3}
},
{
    //E Min
    {41.2, 49, 55, 61.7, 69.3, 82.4},
    {82.4, 98, 110, 123.5, 138.6, 164.8},
    {164.8, 196, 220, 246.9, 277.2, 329.6},

```

```

    {329.6, 392, 440, 493.9, 554.4, 659.3}
},
{
    //B Maj
    {61.7, 69.3, 77.8, 92.5, 103.8, 123.5},
    {123.5, 138.6, 155.6, 185, 207.7, 246.9},
    {246.9, 277.2, 277.2, 370, 415.3, 493.9},
    {493.9, 554.4, 622.3, 740, 830.6, 987.8}
},
{
    //B Min
    {61.7, 73.4, 82.4, 92.5, 110, 123.5},
    {123.5, 146.8, 164.8, 185, 220, 246.9},
    {246.9, 293.7, 329.6, 370, 440, 493.9},
    {493.9, 587.3, 659.3, 740, 880, 987.8}
},
{
    //F# Maj
    {46.2, 51.9, 58.3, 69.3, 77.8, 92.5},
    {92.5, 103.8, 116.5, 138.6, 155.6, 185},
    {185, 207.7, 233.1, 277.2, 311.1, 370},
    {370, 415.3, 466.2, 554.4, 622.3, 740}
},
{
    //F# Min
    {46.2, 55, 61.7, 69.3, 82.4, 92.5},
    {92.5, 110, 123.5, 138.6, 164.8, 185},
    {185, 220, 246.9, 277.2, 329.6, 370},
    {370, 440, 493.9, 554.4, 659.3, 740}
},
{
    //C# Maj
    {34.6, 38.9, 43.7, 51.9, 58.3, 69.3},
    {69.3, 77.7, 87.3, 103.8, 116.5, 138.5},
    {138.5, 146.8, 174.6, 207.6, 233, 277.1},
    {277.1, 311.1, 349.2, 415.3, 466.2, 554.4}
},
{
    //C# Min
    {34.6, 41.2, 46.2, 51.9, 61.7, 69.3},
    {69.3, 82.4, 92.4, 103.8, 123.4, 138.5},
    {138.5, 164.8, 184.9, 207.6, 246.9, 277.1},
    {277.1, 329.6, 370, 369.9, 493.8, 554.3}
},
{
    //F Maj
    {43.7, 49, 55, 65.4, 73.4, 87.3},
    {87.3, 98, 110, 130.8, 146.8, 174.6},
    {174.6, 196, 220, 261.6, 293.7, 349.2},
    {349.2, 392, 440, 523.3, 587.3, 698.5}
},
{
    //F Min
    {43.7, 55, 58.3, 65.4, 82.4, 87.3},
    {87.3, 110, 116.5, 130.8, 164.8, 174.6},
    {174.6, 220, 220, 261.6, 329.6, 349.2},
    {349.2, 440, 466.2, 523.3, 659.3, 698.5}
}

```

```

},
{
    //Bb Maj
    {58.3, 65.4, 73.4, 87.3, 98, 116.5},
    {116.5, 130.8, 146.8, 174.6, 196, 233.1},
    {233.1, 261.6, 293.7, 349.2, 392, 466.2},
    {466.2, 523.3, 587.3, 698.5, 784, 932.3}
},
{
    //Bb Min
    {58.3, 73.4, 73.4, 87.3, 110, 116.5},
    {116.5, 146.8, 164.8, 174.6, 174.6, 233.1},
    {233.1, 293.7, 329.6, 349.2, 440, 466.2},
    {466.2, 587.3, 659.3, 698.5, 880, 932.3}
},
{
    //Eb Maj
    {38.9, 43.7, 49, 58.3, 65.4, 77.8},
    {77.8, 87.3, 98, 116.5, 130.8, 155.6},
    {155.6, 174.6, 196, 233.1, 261.6, 311.1},
    {311.1, 349.2, 392, 466.2, 523.3, 622.3}
},
{
    //Eb Min
    {38.9, 49, 55, 58.3, 73.4, 77.7},
    {77.7, 98, 110, 116.5, 146.8, 164.8},
    {164.8, 196, 220, 233.1, 293.7, 311.1},
    {311.1, 392, 440, 466.2, 587.3, 622.3}
},
{
    //Ab Maj
    {51.9, 58.3, 65.4, 77.8, 87.3, 103.8},
    {103.8, 116.5, 130.8, 155.6, 174.6, 207.7},
    {207.7, 233.1, 261.6, 311.1, 349.2, 415.3},
    {415.3, 466.2, 523.3, 622.3, 698.5, 830.6}
},
{
    //Ab Min
    {51.9, 65.4, 69.3, 77.8, 92.5, 103.8},
    {103.8, 130.8, 138.6, 155.6, 185, 207.7},
    {207.7, 261.6, 277.2, 311.1, 370, 415.3},
    {415.3, 523.3, 554.4, 622.3, 740, 830.6}
}
};
//all delays in ms and are non bolocking
unsigned long prevMillSec = millis();
uint32_t seqDelay = 1000;

// decalre cariables for pot dead zone to make sure user wants to change input
uint16_t A0Last, A1Last , A2Last, A3Last, A4Last, A5Last, A6Last, A7Last =
0;
uint16_t A8Last, A9Last , A10Last, A11Last, A12Last, A13Last, A14Last, A15Last =
0;
uint16_t A0Cur, A1Cur, A2Cur, A3Cur, A4Cur, A5Cur, A6Cur, A7Cur =
0;
uint16_t A8Cur, A9Cur, A10Cur, A11Cur, A12Cur, A13Cur, A14Cur, A15Cur =
0;

```

```

//current sequencer step
int8_t seqStepNum = 0;

//the potentiometer selects what should be played or selected
uint8_t potModeSelect = 0;

//direction of steps
bool sequenceDirection = true;

//time needed for pot to be in region to trigger direction change
const uint32_t seqDelayReverse = 1000;

//indexes for led chasers
uint8_t bottomLedIndex = 0;
uint8_t topLedIndex = 15;

//are we in playback mode?
bool playback;

void setup() {

    //read stored synth settings
    EEPROM.get( 0, synSett );

    //init buttons
    for (int i = 0; i < NUMBUTTONS; i++) {
        pinMode(BUTTONS[i], INPUT);
        digitalWrite(BUTTONS[i], HIGH);
    }
    //Init leds
    for (int i = 0; i < 16; ++i) {
        pinMode(LEDSD[i], OUTPUT);
        digitalWrite(LEDSD[i], HIGH);
    }
    //init synths with waves
    for (byte i = 1; i <= SYNTHCOUNT; ++i) {
        synSett[i].aCarrier.setTable(WAVE_TABLES[0]);
        synSett[i].aModulator.setTable(WAVE_TABLES[0]);
        synSett[i].kIntensityMod.setTable(WAVE_TABLES[0]);
        synSett[i].lastPlayedButton = 0;
    }
    //start synth
    startMozzi(CONTROL_RATE);
    digitalWrite(LEDSD[0], HIGH);
    digitalWrite(LEDSD[15], HIGH);
}

void updateControl() {
    //makeing sure the leds at the end are always on
    digitalWrite(LEDSD[0], HIGH);
    digitalWrite(LEDSD[15], HIGH);

    check_switches();

    A0Cur = mozziaAnalogRead(A0); //mode select
    potModeSelect = map(A0Cur, 0, 1022, 0, 8); // so 9 states state 8 is playback

```

```

//set mode
if (potModeSelect == 8) {
    playback = true;
} else {
    playback = false;
}

A1Cur = mozziAnalogRead(A1); //speed of sequence
seqDelay = map(A1Cur, 0, 1023, 40, 1500); //40ms per stop to 1s per step

if (playback == true) {
    // change synth step if button pressed
    for (int k = 0; k < 8; k++) {
        if (pressed[k]) {
            for (byte i = 4; i < 12; i++) {
                digitalWrite(LEDs[i], LOW);
            }
            digitalWrite(LEDs[k + 4], HIGH);
        }
        if (pressed[k] && k == 0) {
            seqStepNum = k;
            prevMillSec = millis();
        } else if (pressed[k] && k == 1) {
            seqStepNum = k;
            prevMillSec = millis();
        } else if (pressed[k] && k == 2) {
            seqStepNum = k;
            prevMillSec = millis();
        } else if (pressed[k] && k == 3) {
            seqStepNum = k;
            prevMillSec = millis();
        } else if (pressed[k] && k == 4) {
            seqStepNum = k;
            prevMillSec = millis();
        } else if (pressed[k] && k == 5) {
            seqStepNum = k;
            prevMillSec = millis();
        } else if (pressed[k] && k == 6) {
            seqStepNum = k;
            prevMillSec = millis();
        } else if (pressed[k] && k == 7) {
            seqStepNum = k;
            prevMillSec = millis();
        }
    }
}

//save synth state if BUTTONS pressed
if (pressed[0] && pressed[7]) {
    EEPROM.put(0, synSett);
}

if (playback == false) {
    if (pressed[6] && pressed[7]) {
        synSett[1] = synSett[0];
        synSett[2] = synSett[0];
        synSett[3] = synSett[0];
    }
}

```

```

    synSett[4] = synSett[0];
    synSett[5] = synSett[0];
    synSett[6] = synSett[0];
    synSett[7] = synSett[0];
}

if (pressed[0] && pressed[1] && pressed[2]) {
    scaleSelect = 0;
    octaveSelect = 0;
}
if (pressed[0] && pressed[1] && justreleased[6]) {
    if (scaleSelect > 0) {
        scaleSelect--;
    }
} else if (pressed[0] && pressed[1] && justreleased[7]) {
    if (scaleSelect < 23) {
        scaleSelect++;
    }
} else if (justreleased[6]) {
    if (octaveSelect > 0) {
        octaveSelect--;
    }
} else if (justreleased[7]) {
    if (octaveSelect < 3) {
        octaveSelect++;
    }
}

if ((scaleSelect & 1) == 0) {
    digitalWrite(LED13, HIGH);
    digitalWrite(LED12, LOW);
} else {
    digitalWrite(LED13, LOW);
    digitalWrite(LED12, HIGH);
}

if (octaveSelect == 0) {
    digitalWrite(LED1, LOW);
    //digitalWrite(LED2, LOW);
    digitalWrite(LED3, LOW);
    digitalWrite(LED12, LOW);
    //digitalWrite(LED13, LOW);
    digitalWrite(LED14, LOW);

    digitalWrite(LED1, HIGH);
} else if (octaveSelect == 1) {
    digitalWrite(LED1, LOW);
    //digitalWrite(LED2, LOW);
    digitalWrite(LED3, LOW);
    digitalWrite(LED12, LOW);
    //digitalWrite(LED13, LOW);
    digitalWrite(LED14, LOW);

    digitalWrite(LED3, HIGH);
} else if (octaveSelect == 2) {

```

```

    digitalWrite(LED1, LOW);
    //digitalWrite(LED2, LOW);
    digitalWrite(LED3, LOW);
    digitalWrite(LED12, LOW);
    //digitalWrite(LED13, LOW);
    digitalWrite(LED14, LOW);

    digitalWrite(LED12, HIGH);
} else if (octiveSelect == 3) {
    digitalWrite(LED1, LOW);
    //digitalWrite(LED2, LOW);
    digitalWrite(LED3, LOW);
    digitalWrite(LED12, LOW);
    //digitalWrite(LED13, LOW);
    digitalWrite(LED14, LOW);

    digitalWrite(LED14, HIGH);
}
}

//get current time
unsigned long currentMillis = millis();

//reverse direction of steps if pot turned
if (A1Cur > 900) {
    if (currentMillis - prevMillSec >= seqDelayReverse) {
        sequenceDirection = !sequenceDirection;
    }
}

//make led pattern
if (currentMillis - prevMillSec >= seqDelay) {
    if (playback == true) {
        digitalWrite(LED[bottomLedIndex], LOW);
        digitalWrite(LED[topLedIndex], LOW);
        bottomLedIndex++;
        topLedIndex--;
        if (bottomLedIndex >= 4) {
            bottomLedIndex = 1;
        }
        if (topLedIndex <= 11) {
            topLedIndex = 14;
        }
        digitalWrite(LED[bottomLedIndex], HIGH);
        digitalWrite(LED[topLedIndex], HIGH);
    }
    if (playback == false) {
        prevMillSec = currentMillis;
    }
}

//needed for when selecting LEDS
if (playback == false) {
    for (byte i = 4; i < 12; i++) {
        digitalWrite(LED[i], LOW);
    }
}

```

```

}

// select what synth to edit or playback mode
switch (potModeSelect) {
  case 0:
    seqStepNum = 0;
    digitalWrite(LEDs[seqStepNum + 4], HIGH);
    analogUpdate(seqStepNum);
    break;
  case 1:
    seqStepNum = 1;
    digitalWrite(LEDs[seqStepNum + 4], HIGH);
    analogUpdate(seqStepNum);
    break;
  case 2:
    seqStepNum = 2;
    digitalWrite(LEDs[seqStepNum + 4], HIGH);
    analogUpdate(seqStepNum);
    break;
  case 3:
    seqStepNum = 3;
    digitalWrite(LEDs[seqStepNum + 4], HIGH);
    analogUpdate(seqStepNum);
    break;
  case 4:
    seqStepNum = 4;
    digitalWrite(LEDs[seqStepNum + 4], HIGH);
    analogUpdate(seqStepNum);
    break;
  case 5:
    seqStepNum = 5;
    digitalWrite(LEDs[seqStepNum + 4], HIGH);
    analogUpdate(seqStepNum);
    break;
  case 6:
    seqStepNum = 6;
    digitalWrite(LEDs[seqStepNum + 4], HIGH);
    analogUpdate(seqStepNum);
    break;
  case 7:
    seqStepNum = 7;
    digitalWrite(LEDs[seqStepNum + 4], HIGH);
    analogUpdate(seqStepNum);
    break;
  case 8:
    if (currentMillis - prevMillSec >= seqDelay) {
      prevMillSec = currentMillis;
      //recerse step
      if (sequenceDirection == true) {
        if (seqStepNum >= 7) {
          seqStepNum = -1;
        }
        seqStepNum++;
      } else {
        if (seqStepNum < 0) {
          seqStepNum = 8;
        }
      }
    }
  }
}

```

```

        seqStepNum--;
    }
    for (byte i = 4; i < 12; i++) {
        digitalWrite(LED_S[i], LOW);
    }
    digitalWrite(LED_S[seqStepNum + 4], HIGH);
}
analogUpdate(0);
break;
}
//fm synth calculations and set up
synSett[seqStepNum].FMIntensity2 =
((long) synSett[seqStepNum].FMIntensity2Mapped * synSett[seqStepNum].FMIntensity1
* (synSett[seqStepNum].kIntensityMod.next() + 128)) >> 8; // shift back to range
after 8 bit multiply
synSett[seqStepNum].modFreq = synSett[seqStepNum].carrierFreq * MOD_RATIO *
synSett[seqStepNum].FRQ;

synSett[seqStepNum].aCarrier.setFreq(synSett[seqStepNum].carrierFreq);
synSett[seqStepNum].aModulator.setFreq(synSett[seqStepNum].modFreq);
synSett[seqStepNum].kIntensityMod.setFreq(synSett[seqStepNum].modSpeed);
}

int updateAudio() {
    //output constructed wave
    long modulation = aSmoothIntensity.next(synSett[seqStepNum].FMIntensity2) *
synSett[seqStepNum].aModulator.next();
    return synSett[seqStepNum].aCarrier.phMod(modulation);
}

void analogUpdate(int synthNum) {
    //setup mode
    //int A0Cur = mozziaAnalogRead(A0); // sequence choise
    // playback mode
    // modulation speed
    all
    //int A1Cur = mozziaAnalogRead(A1); //sequence speed
    //
    A2Cur = mozziaAnalogRead(A2); //modulation speed
    //
    A3Cur = mozziaAnalogRead(A3); //intensity 2
    //
    A4Cur = mozziaAnalogRead(A4); //intensity 1
    //
    A5Cur = mozziaAnalogRead(A5); //modulation frequency
    //
    A6Cur = mozziaAnalogRead(A6); //waveform select modulation
    //
    A7Cur = mozziaAnalogRead(A7); //waveform select carrier
    //

    A8Cur = mozziaAnalogRead(A8); // always frequency all
    A9Cur = mozziaAnalogRead(A9);
    A10Cur = mozziaAnalogRead(A10);
    A11Cur = mozziaAnalogRead(A11);
    A12Cur = mozziaAnalogRead(A12);
    A13Cur = mozziaAnalogRead(A13);
    A14Cur = mozziaAnalogRead(A14);
    A15Cur = mozziaAnalogRead(A15);

    // no matter what change the frequency of carrier
    if (abs(A15Last - A15Cur) > 25) {
        A15Last = A15Cur;
        synSett[0].carrierFreq = (float) kMapCarrierFreq(A15Cur);
    }
    if (abs(A14Last - A14Cur) > 25) {

```

```

    A14Last = A14Cur;
    synSett[1].carrierFreq = (float) kMapCarrierFreq(A14Cur);
}
if (abs(A13Last - A13Cur) > 25) {
    A13Last = A13Cur;
    synSett[2].carrierFreq = (float) kMapCarrierFreq(A13Cur);
}
if (abs(A12Last - A12Cur) > 25) {
    A12Last = A12Cur;
    synSett[3].carrierFreq = (float) kMapCarrierFreq(A12Cur);
}
if (abs(A11Last - A11Cur) > 25) {
    A11Last = A11Cur;
    synSett[4].carrierFreq = (float) kMapCarrierFreq(A11Cur);
}
if (abs(A10Last - A10Cur) > 25) {
    A10Last = A10Cur;
    synSett[5].carrierFreq = (float) kMapCarrierFreq(A10Cur);
}
if (abs(A9Last - A9Cur) > 25) {
    A9Last = A9Cur;
    synSett[6].carrierFreq = (float) kMapCarrierFreq(A9Cur);
}
if (abs(A8Last - A8Cur) > 25) {
    A8Last = A8Cur;
    synSett[7].carrierFreq = (float) kMapCarrierFreq(A8Cur);
}

//if we are in playback change the modulation speed
if (playback == true) {
    if (abs(A2Last - A2Cur) > 25) {
        A2Last = A2Cur;
        synSett[5].modSpeed = (float) kMapModSpeed(A2Cur) / 1000;
    }
    if (abs(A3Last - A3Cur) > 25) {
        A3Last = A3Cur;
        synSett[4].modSpeed = (float) kMapModSpeed(A3Cur) / 1000;
    }
    if (abs(A4Last - A4Cur) > 25) {
        A4Last = A4Cur;
        synSett[3].modSpeed = (float) kMapModSpeed(A4Cur) / 1000;
    }
    if (abs(A5Last - A5Cur) > 25) {
        A5Last = A5Cur;
        synSett[2].modSpeed = (float) kMapModSpeed(A5Cur) / 1000;
    }
    if (abs(A6Last - A6Cur) > 25) {
        A6Last = A6Cur;
        synSett[1].modSpeed = (float) kMapModSpeed(A6Cur) / 1000;
    }
    if (abs(A7Last - A7Cur) > 25) {
        A7Last = A7Cur;
        synSett[0].modSpeed = (float) kMapModSpeed(A7Cur) / 1000;
    }
}

//if we are in edit mode allow to fine tune synths
if (playback == false) {

```

```

if (abs(A2Last - A2Cur) > 40) {
    A2Last = A2Cur;
    //modulation speed
    synSett[synthNum].modSpeed = (float) kMapModSpeed(A2Cur) / 1000;
}
if (abs(A3Last - A3Cur) > 40) {
    A3Last = A3Cur;
    //modulation intensity 2
    synSett[synthNum].FMIntensity2Mapped = kMapIntensity(mapThis(A3Cur));
}
if (abs(A4Last - A4Cur) > 40) {
    A4Last = A4Cur;
    //modulation intensity 1
    synSett[synthNum].FMIntensity1 = mapThis(A1Cur);
}
if (abs(A5Last - A5Cur) > 40) {
    A5Last = A5Cur;
    //carrier frequency
    synSett[synthNum].FRQ = mapThis(A5Cur);
}
if (abs(A6Last - A6Cur) > 40) {
    A6Last = A6Cur;
    //wave forme select modulator
    synSett[synthNum].aModulator.setTable(WAVE_TABLES[map(A6Cur, 0, 1023, 0,
9)]];
}
if (abs(A7Last - A7Cur) > 40) {
    A7Last = A7Cur;
    //wavefrom select carrier
    synSett[synthNum].aCarrier.setTable(WAVE_TABLES[map(A7Cur, 0, 1023, 0,
9)]];
}

//over writing pot
for (int k = 0; k < 6; k++) { //two demintions iteration would be slower?
    if (pressed[k] && k == 0) {
        synSett[synthNum].lastPlayedButton = k;
        synSett[synthNum].carrierFreq =
NOTELOOKUP[scaleSelect][octiveSelect][synSett[synthNum].lastPlayedButton];
    } else if (pressed[k] && k == 1) {
        synSett[synthNum].lastPlayedButton = k;
        synSett[synthNum].carrierFreq =
NOTELOOKUP[scaleSelect][octiveSelect][synSett[synthNum].lastPlayedButton];
    } else if (pressed[k] && k == 2) {
        synSett[synthNum].lastPlayedButton = k;
        synSett[synthNum].carrierFreq =
NOTELOOKUP[scaleSelect][octiveSelect][synSett[synthNum].lastPlayedButton];
    } else if (pressed[k] && k == 3) {
        synSett[synthNum].lastPlayedButton = k;
        synSett[synthNum].carrierFreq =
NOTELOOKUP[scaleSelect][octiveSelect][synSett[synthNum].lastPlayedButton];
    } else if (pressed[k] && k == 4) {
        synSett[synthNum].lastPlayedButton = k;
        synSett[synthNum].carrierFreq =
NOTELOOKUP[scaleSelect][octiveSelect][synSett[synthNum].lastPlayedButton];
    } else if (pressed[k] && k == 5) {
        synSett[synthNum].lastPlayedButton = k;

```

```

        synSett[synthNum].carrierFreq =
NOTELOOKUP[scaleSelect][octiveSelect][synSett[synthNum].lastPlayedButton];
    }
    //else if (pressed[k] && k == 6) {
    //    synSett[synthNum].carrierFreq =
NOTELOOKUP[scaleSelect][octiveSelect][synSett[synthNum].lastPlayedButton];
    //} else if (pressed[k] && k == 7) {
    //    synSett[synthNum].carrierFreq =
NOTELOOKUP[scaleSelect][octiveSelect][synSett[synthNum].lastPlayedButton];
    //}
    }

    if (justreleased[6] || justreleased[7]){
        synSett[synthNum].carrierFreq =
NOTELOOKUP[scaleSelect][octiveSelect][synSett[synthNum].lastPlayedButton];
    }

}

}

void loop() {
    audioHook();
}

// adafruit non blocking switch checking
void check_switches()
{
    static byte previousstate[NUMBUTTONS];
    static byte currentstate[NUMBUTTONS];
    static long lasttime;
    byte index;

    if (millis() < lasttime) {
        // we wrapped around, lets just try again
        lasttime = millis();
    }

    if ((lasttime + DEBOUNCE) > millis()) {
        // not enough time has passed to debounce
        return;
    }
    // ok we have waited DEBOUNCE milliseconds, lets reset the timer
    lasttime = millis();

    for (index = 0; index < NUMBUTTONS; index++) {
        justpressed[index] = 0;          // when we start, we clear out the "just"
indicators
        justreleased[index] = 0;

        currentstate[index] = digitalRead(BUTTONS[index]);    // read the button

        if (currentstate[index] == previousstate[index]) {
            if ((pressed[index] == LOW) && (currentstate[index] == LOW)) {
                // just pressed
                justpressed[index] = 1;
            }
        }
    }
}

```

```

        else if ((pressed[index] == HIGH) && (currentstate[index] == HIGH)) {
            // just released
            justreleased[index] = 1;
        }
        pressed[index] = !currentstate[index]; // remember, digital HIGH means NOT
pressed
    }
    /*
        Serial.print("B");
        Serial.print(index);
        Serial.print(":");
        Serial.print(pressed[index], DEC);
        Serial.print(" ");
        Serial.println();
    */
    previousstate[index] = currentstate[index]; // keep a running tally of the
BUTTONS
    }
    //Serial.println();
}

```

Synth edit mode

Power On

Octave 1

Major

Octave 2

Sequence 1

Sequence 2

Sequence 3

Sequence 4

Sequence 5

Sequence 6

Sequence 7

Sequence 8

Octave 3

Minor

Octave 4

Power On

Carrier wave select

Modulator wave select

Modulator frequency

Intensity 1

Intensity 2

Modulation speed

Sequencer speed

Playback mode or Edit mode / Selector Synth 1-8

Carrier Frequency synth 1

Carrier Frequency synth 2

Carrier Frequency synth 3

Carrier Frequency synth 4

Carrier Frequency synth 5

Carrier Frequency synth 6

Carrier Frequency synth 7

Carrier Frequency synth 8

Note 1
Hold (Scale Select)

Note 2
Hold (Scale Select)

Note 3

Note 4

Note 5

Note 6

Octave Down Press (Scale Select Down)

Octave Up Press (Scale Select Up)

Synth playback mode

Power On

Octave 1

Major

Octave 2

Sequence 1

Sequence 2

Sequence 3

Sequence 4

Sequence 5

Sequence 6

Sequence 7

Sequence 8

Octave 3

Minor

Octave 4

Power On

Modulation speed 1

Modulation speed 2

Modulation speed 3

Modulation speed 4

Modulation speed 5

Modulation speed 6

Sequencer speed

Playback mode or edit mode synth 1-8

Carrier Frequency synth 1

Carrier Frequency synth 2

Carrier Frequency synth 3

Carrier Frequency synth 4

Carrier Frequency synth 5

Carrier Frequency synth 6

Carrier Frequency synth 7

Carrier Frequency synth 8

Sequence1

Sequence2

Sequence3

Sequence4

Sequence5

Sequence6

Sequence7

Sequence8

- Button Combinations:
- Scale Select:

Button 1 + Button 2 + (Button 7[Scale Down] or Button 8[Scale Up])
- Scale and Octave reset:

Button 1 + Button 2 + Button 3
- Copy synth 1 setting to all other synths:

Button 7 + Button 8
- Save settings:

Button 1 + Button 8
- Reverse step direction:

Move sequencer speed pot all the way to the right for 1 second