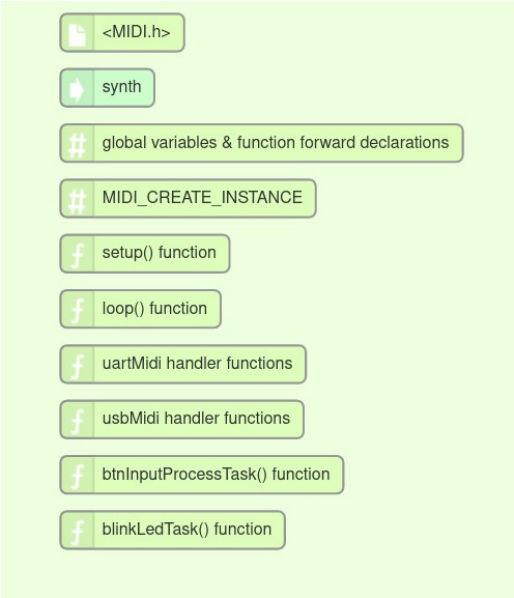
Mannicken polysynth meltdown

By jean-claude.feltes@education.lu

The polysynth is one of the examples in the Mannicken audio tool. It is really complex. To better understand, I tried to melt it down to the post important elements.

Main.cpp



global variables + declaration:

```
// LED + intervals
// buttons

#define KEYBOARD_NOTE_SHIFT_CORRECTION 21//-12

//uart_MIDI
void uartMidi_NoteOn(byte channel, byte note, byte velocity);
void uartMidi_NoteOff(byte channel, byte note, byte velocity);
void uartMidi_ControlChange(byte channel, byte control, byte value);
void uartMidi_PitchBend(byte channel, int value);

// the same for USBMIDI

void blinkLedTask(void);
void btnInputProcessTask(void);
```

MIDI_CREATE_INSTANCE:

```
MIDI_CREATE_INSTANCE(HardwareSerial, Serial1, MIDI);
```

setup():

```
void setup()
{
  AudioMemory(96);

  MIDI.begin();
  MIDI.setHandleNoteOn(uartMidi_NoteOn);
  MIDI.setHandleNoteOff(uartMidi_NoteOff);
  MIDI.setHandleControlChange(uartMidi_ControlChange);
  MIDI.setHandlePitchBend(uartMidi_PitchBend);

  // the same for USB MIDI

  synth.begin();

  // init buttons + LED
 }
```

loop():

```
void loop()
{
    usbMIDI.read();
    MIDI.read();

    btnInputProcessTask();
     blinkLedTask();
}
```

UART MIDI handler functions:

```
void uartMidi_NoteOn(byte channel, byte note, byte velocity) {
    note += KEYBOARD_NOTE_SHIFT_CORRECTION;
    velocity = 127 - velocity;
    synth.noteOn(note, velocity);
    usbMIDI.sendNoteOn(note, velocity, channel, 0);
}

void uartMidi_NoteOff(byte channel, byte note, byte velocity) {
    note += KEYBOARD_NOTE_SHIFT_CORRECTION;
    velocity = 127 - velocity;
    synth.noteOff(note);
    usbMIDI.sendNoteOff(note, velocity, channel, 0);
}

void uartMidi_ControlChange(byte channel, byte control, byte value) {
    usbMIDI.sendControlChange(control, value, channel, 0x00);
}

void uartMidi_PitchBend(byte channel, int value) {
    usbMIDI.sendPitchBend(value, channel, 0x00);
}
```

USB MIDI handler functions:

```
void usbMidi_NoteOn(byte channel, byte note, byte velocity) {
    synth.noteOn(note, velocity);
}

void usbMidi_NoteOff(byte channel, byte note, byte velocity) {
    synth.noteOff(note);
}

void usbMidi_PitchBend(byte channel, int value) {

}

void usbMidi_ControlChange(byte channel, byte control, byte value) {
    switch (control) { // cases 20-31,102-119 is undefined in midi spec
        case 64:
          if (value == 0)
            synth.deactivateSustain();
          else if (value == 127)
            synth.activateSustain();
          break;
        case 0:
          synth.set_InstrumentByIndex(value);
          break;
        case 20: // OSC A waveform select
          synth.set_OSC_A_waveform(value);          break;
        // the same for 21, 22 : OSC B, OSC C

        case 23:
          synth.set_OSC_A_pulseWidth(value);        break;
        // the same for 24, 25: OSC B, OSC C

        case 26:
          synth.set_OSC_A_phase(value);          break;
        // the same for 27, 28: OSC B, OSC C

        case 29:
          synth.set_OSC_A_amplitude(value);          break;
        // the same for 30, 31: OSC B, OSC C
        case 32: //("LSB for Control 0 (Bank Select)" @ midi spec.)
          synth.set_OSC_D_amplitude(value);         break;

        case 33:
          synth.set_mixVoices_gains(value);          break;
```

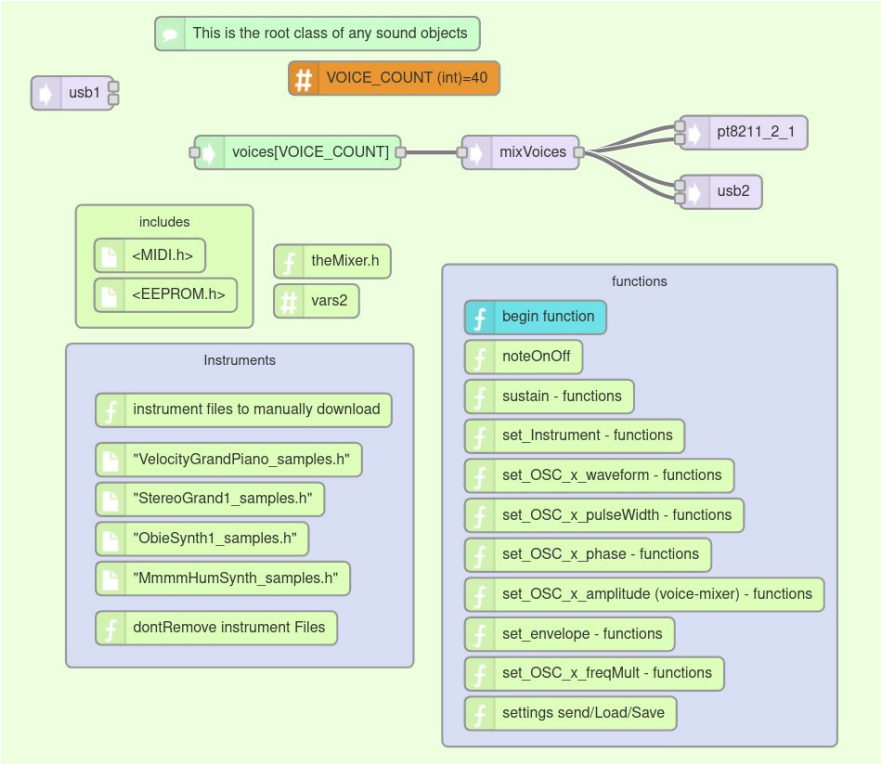
```
        case 100:
          synth.set_envelope_delay(value);          break;
        // 101, 102, 103, 104, 105   for  attack, hold, decay, sustain, release

        case 108:
          synth.set_OSC_A_freqMult(value);          break;
        // the same for 109, 110: OSC B, OSC C


        case 115: // set wavetable as primary (Piano mode)
          synth.SetWaveTable_As_Primary();          break;
        case 116:
          synth.SetWaveForm_As_Primary();          break;

        case 117: // EEPROM read settings
          synth.EEPROM_ReadSettings();          break;
        case 118: // EEPROM save settings
          synth.EEPROM_SaveSettings();          break;

        case 119: // get all values
          synth.sendAllSettings();        break;
    }
}
```

button process tasks:

```
void btnInputProcessTask(void)
{
  btnSustain = digitalRead(btnSustainPin);
  // the same for Sostenuto, SoftPedal, NextInstrument


    // Sustain pedal
    if ((btnSustain == LOW) && (btnSustainWasPressed == 0))
    {
        btnSustainWasPressed = 1;
        usbMIDI.sendControlChange(0x40, 0x7F, 0x00);
        synth.activateSustain();

        uint16_t memory_used = AudioMemoryUsageMax();
        uint16_t cpu_used = AudioProcessorUsageMax();
        uint8_t data[11];
        data[0] = 0x30 + memory_used/10000;
        data[1] = 0x30 + memory_used%10000/1000;
        data[2] = 0x30 + memory_used%10000%1000/100;
        data[3] = 0x30 + memory_used%10000%1000%100/10;
        data[4] = 0x30 + memory_used%10000%1000%100%10;
        data[5] = ':';
        data[6] = 0x30 + cpu_used/10000;
        data[7] = 0x30 + cpu_used%10000/1000;
        data[8] = 0x30 + cpu_used%10000%1000/100;
        data[9] = 0x30 + cpu_used%10000%1000%100/10;
        data[10] = 0x30 + cpu_used%10000%1000%100%10;
        usbMIDI.sendSysEx(11, data);
    }
    else if ((btnSustain == HIGH) && (btnSustainWasPressed == 1))
    {
        btnSustainWasPressed = 0;
        usbMIDI.sendControlChange(0x40, 0x00, 0x00);
        synth.deactivateSustain();
    }
    // Sostenuto Pedal
    if ((btnSostenuto == LOW) && (btnSostenutoWasPressed == 0))
    {
        btnSostenutoWasPressed = 1;
        usbMIDI.sendControlChange(0x42, 0x7F, 0x00);
    }
    else if ((btnSostenuto == HIGH) && (btnSostenutoWasPressed == 1))
    {
        btnSostenutoWasPressed = 0;
        usbMIDI.sendControlChange(0x42, 0x00, 0x00);
    }
    // Soft Pedal
    if ((btnSoftPedal == LOW) && (btnSoftPedalWasPressed == 0))
    {
        btnSoftPedalWasPressed = 1;
        usbMIDI.sendControlChange(0x43, 0x7F, 0x00);
    }
    else if ((btnSoftPedal == HIGH) && (btnSoftPedalWasPressed == 1))
    {
```

```
        btnSoftPedalWasPressed = 0;
        usbMIDI.sendControlChange(0x43, 0x00, 0x00);
    }

    // Next Instrument button
    if ((btnNextInstrument == LOW) && (btnNextInstrumentWasPressed == 0))
    {
        btnNextInstrumentWasPressed = 1;
        if (synth.currentWTinstrument == (InstrumentCount - 1)) synth.currentWTinstrument = 0;
        else synth.currentWTinstrument++;
        synth.set_InstrumentByIndex(synth.currentWTinstrument);
        usbMIDI.sendControlChange(0, synth.currentWTinstrument, 0x00);
    }
    else if ((btnNextInstrument == HIGH) && (btnNextInstrumentWasPressed == 1))
    {
        btnNextInstrumentWasPressed = 0;
    }
}
```

Synth



vars2:

```
#define NOTE_PRESSED_STATE_LED  6
#define NOTE_OVERFLOWN_LED      5
#define InstrumentCount 5

const float DIV127 = (1.0 / 127.0);
const float DIV100 = 0.01;
const float DIV64 = (1.0/64.0);
const float DIV360BY127 = (360.0/127.0);
const float DIV360BY120 = (3.0);

byte oscAwaveform = WAVEFORM_SINE;
// .. B + C

byte mixVoices_gains = 100;

byte oscAamp = 100;          //same: B C D
byte oscApulsewidth = 0;     //same: B C
byte oscAphase = 0;          //same: B C

byte envDelay = 0;           // the same for Attack, Hold, Decay, Sustain (=100)!, Release

byte currentWTinstrument = 0;

// global scope
```

```
byte oscApitchMult = 64; // set at middle           //same: B C
```

begin() function:

```
void begin()
{
    for (int i = 0; i < VOICE_COUNT; i++)
    {
        voices[i].begin();
        mixVoices.gain(i, 1.0f/VOICE_COUNT);
    }
    pinMode(NOTE_OVERFLOWN_LED, OUTPUT);      digitalWrite(NOTE_OVERFLOWN_LED, LOW);

    pinMode(NOTE_PRESSED_STATE_LED, OUTPUT);       digitalWrite(NOTE_PRESSED_STATE_LED, LOW);

    EEPROM_ReadSettings();

    set_InstrumentByIndex(1); // VelocityGrandPiano
}
```

Note on & Note Off:

```
void noteOn(byte note, byte velocity)
{
    digitalWrite(NOTE_PRESSED_STATE_LED, HIGH); //any note "pressed"
    // fist checks if this note is allready playing
    // it that is the case then it "reuses" this "slot"
    // this makes sure that not all "slots" is filled
    // with the same playing note
    // if the MIDI keyboard is for some reason
    // not sending a noteoff (my keyboard is sometimes glitching)
    // and when sending MIDI from my computer for fast playing songs
    for (int i = 0; i < VOICE_COUNT; i++)
    {
        // first check if the note was played recently
        if (voices[i].note == note)
        {
            voices[i].noteOn(note, velocity);
            digitalWrite(NOTE_OVERFLOWN_LED, LOW);
            return;
        }
    }
    // then if the note has not allready been played
    // // second see if there is any free "spot"
    for (int i = 0; i < VOICE_COUNT; i++)
    {
        if (voices[i].isNotPlaying())
        {
            voices[i].noteOn(note, velocity);
            return;
        }
    }
    digitalWrite(NOTE_OVERFLOWN_LED, HIGH); // this is a notification that there was no free spots
}
```

```
void noteOff(byte note)
{
    digitalWrite(NOTE_PRESSED_STATE_LED, LOW); //any note "released"
    for (int i = 0; i < VOICE_COUNT; i++)
    {
        if (voices[i].note == note)
        {
            voices[i].noteOff();
            return;
        }
    }
}
```

Sustain functions:

```
void activateSustain()
{
    for (int i = 0; i < VOICE_COUNT; i++)
    {
        voices[i].isSustain = 1;
```

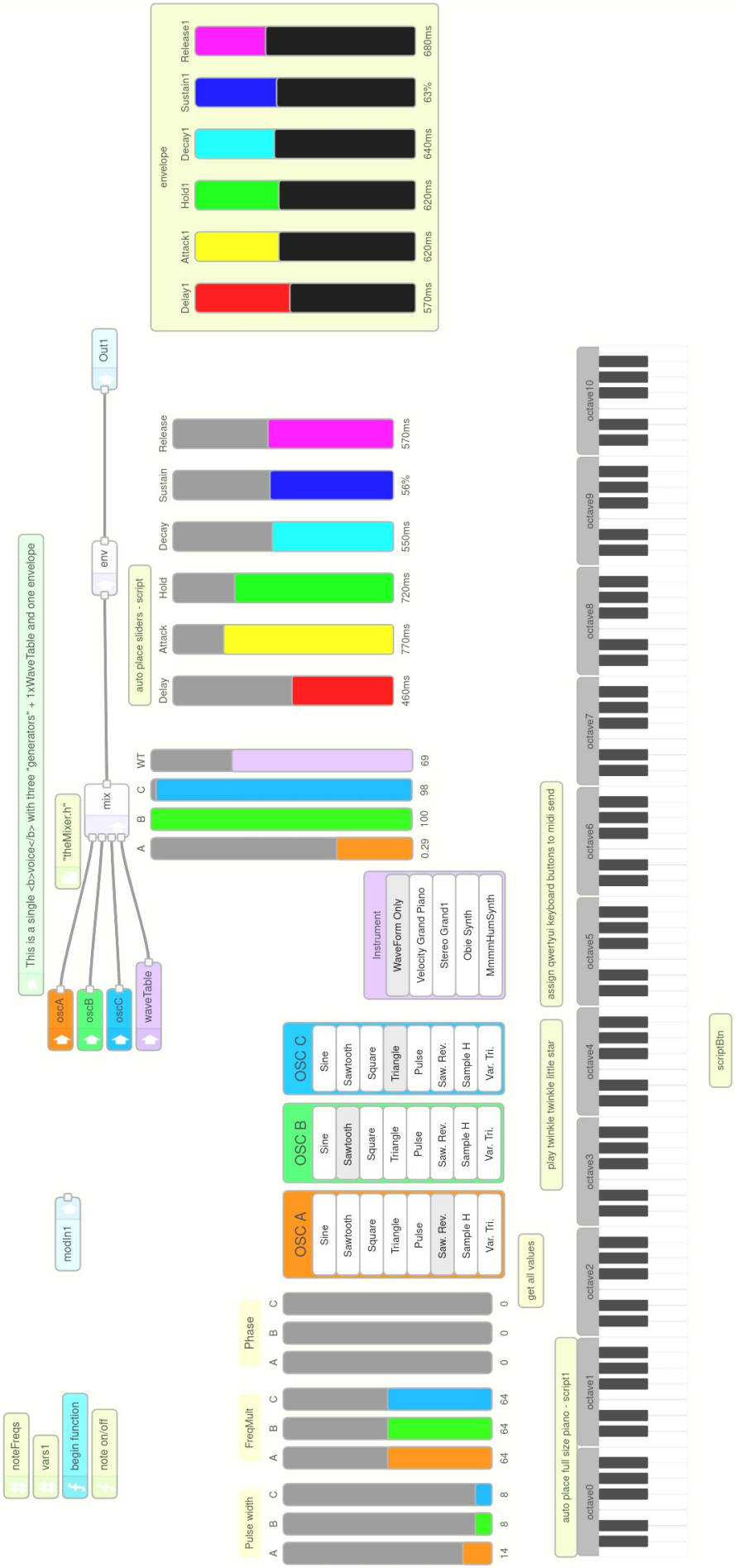
```
    }
}

void deactivateSustain()
{
    for (int i = 0; i < VOICE_COUNT; i++)
    {
        voices[i].isSustain = 0;
        if (!voices[i].isNoteOn)
            voices[i].noteOff();
    }
}
```

Set instrument functions:

```
void set_InstrumentByIndex(byte index)
{
    currentWTinstrument = index;
    switch(index)
    {
        case 0:     SetWaveForm_As_Primary();            break;
        case 1:     SetWaveTable_As_Primary(); set_Instrument(VelocityGrandPiano); break;
        case 2:     SetWaveTable_As_Primary(); set_Instrument(StereoGrand1);       break;
        case 3:     SetWaveTable_As_Primary(); set_Instrument(ObieSynth1);         break;
        case 4:     SetWaveTable_As_Primary(); set_Instrument(MmmmHumSynth);       break;
        default:            break;
    }
}

void set_Instrument(const AudioSynthWavetable::instrument_data &instrument)
{
    for (int i = 0; i< VOICE_COUNT; i++)
    {
        voices[i].waveTable.setInstrument(instrument);        voices[i].waveTable.amplitude(1.0);
    }
}

void set_mixVoices_gains(byte value)
{
    if (value > 100) value = 100;    mixVoices_gains = value;
    for (int i = 0; i < VOICE_COUNT; i++)
        mixVoices.gain(i, value*DIV100);
}

void SetWaveTable_As_Primary()
{
    set_OSC_A_amplitude(0);  // same for B, C
    set_OSC_D_amplitude(100);
    set_mixVoices_gains(100);
}

void SetWaveForm_As_Primary()
{
    set_OSC_A_amplitude(100);  // same for B, C
    set_OSC_D_amplitude(0);
    set_mixVoices_gains(3);
}
```

other functions:
set_OSC_x_waveform
set_OSC_x_pulseWidth
set_OSC_x_amplitude
set_OSC_x_freqMult
settings send / Load / Save

Voice


envelope
Delay1 570ms
Attack1 620ms
Hold1 620ms
Decay1 640ms
Sustain1 63%
Release1 680ms
Out1
env
mix
"theMixer.h"
This is a single voice with three "generators" + 1xWaveTable and one envelope
oscA
oscB
oscC
waveTable
auto place sliders - script
Delay 460ms
Attack 770ms
Hold 720ms
Decay 550ms
Sustain 56%
Release 570ms
A 0.29
B 100
C 98
WT 69
Instrument
WaveForm Only
Velocity Grand Piano
Stereo Grand1
Obie Synth
MmmmHumSynth
OSC A
OSC B
OSC C
Sine
Sawtooth
Square
Triangle
Pulse
Saw. Rev.
Sample H
Var. Tri.
get all values
assign qwertyui keyboard buttons to midi send
play twinkle twinkle little star
auto place full size piano - script1
octave0
octave1
octave2
octave3
octave4
octave5
octave6
octave7
octave8
octave9
octave10
Pulse width
FreqMult
Phase
A
B
C
noteFreqs
vars1
begin function
note on/off
modln1
scriptBtn

noteFreqs contains an array with the note frequencies:

```
const float noteFreqs[128] = {8.176, 8.662, 9.177, 9.723, 10.301, 10.913, 11.562, 12.25, 12.978, ...
12543.85};
```

vars1:

```
byte note = 0;
byte isNoteOn = 0;
byte isSustain = 0;
byte oscApitchMult = 64; // set at middle, also for B, C
float newAmp = 0.0;
```

begin function is empty

note on / off:

```
void noteOn(byte Note, byte velocity)
{
    float newAmp = 0.0f;
    if (Note >= sizeof(noteFreqs)) return;

    note = Note;
    isNoteOn = 1;

    newAmp = (float)velocity*(1.0f / 127.0f);

      oscA.frequency(GetBendedFreq(oscApitchMult));        // the same for B, C
      oscA.amplitude(newAmp);                              // the same for B, C
      waveTable.playNote(note, velocity);
      env.noteOn();
}

void noteOff()
{
    isNoteOn = 0;
    if (!isSustain)
    {
        env.noteOff();        waveTable.stop();
    }
}

bool isNotPlaying()
{
    if (!env.isActive())
        return true;
    else if(!waveTable.isPlaying())
        return true;
    else
        return false;
}

float GetBendedFreq(byte pitchMult)
{
    if (pitchMult < 64)
        return noteFreqs[note - 12*(64-pitchMult)];
    else if (pitchMult > 64)
        return noteFreqs[note + 12*(pitchMult-64)];
    else
        return noteFreqs[note];
}
```

graphic control elements:

send MIDI control codes,
for example OSCA – Sine:

```
var formatted = "midiSend(0xB0,20,"+d.selectedIndex+")";
RED.BiDirDataWebSocketBridge.SendToWebSocket(formatted);
```

The same is true for the other controls, like buttons, sliders, keyboard.
These elements are based on NodeRED.