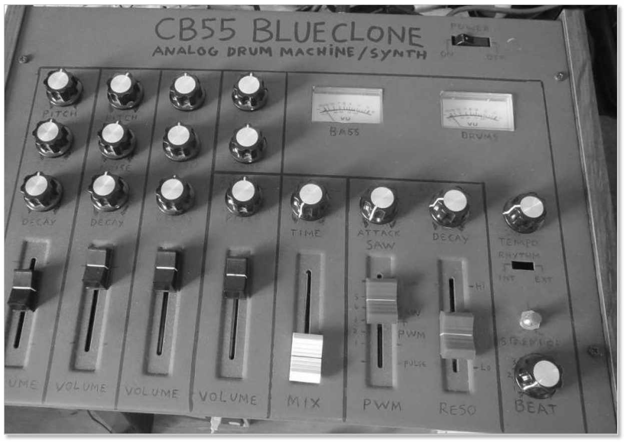


The Blue Clone

Pretty awesome analog drum machine (courtesy of CircuitBendersUK) and bass synthesizer! Very happy with this one.

The drums sounds are generated by a CB-55 board designed by CircuitBendersUK and modded by me. Check for info on the unmodded board **here**.

The bass synth is entirely of my own design, but is roughly based on a Roland Juno DCO.

An Arduino Uno generates trigger signals, DCO clock and takes care of the control and MIDI inputs.

Here's the quick demonstration video: **watch here**.

The Software

The code is heavily based on Drumique's, check that one out **here**.

The biggest difference is that Blue Clone also needs to generate the DCO clock, PWM and filter cutoff CV's.
Below's the entire bit of code for Blue Clone **(scroll further down for hardware).** Normally I have my code ordened and separated into different files (i.e. one for the drum machine, one for the bass synth, rhythms etc) but I've put it in one blob now for your convienience :):

```
#include "avr/pgmspace.h"

/*
 * IMPORTANT:
your arduino might not start if it receives data directly after a reset, because the bootloader thinks you want to upload a new progam.
you might need to unplug the midi-hardware until the board is running your program. that is when that statusLed turns on.

#############################################################################################################################################
MIDI MESSAGES:
midi messages start with one status byte followed by 1 _or_ 2 data bytes, depending on the command

example midi message: 144-36-100
  the status byte "144" tells us what to do. "144" means "note on".
  in this case the second bytes tells us which note to play (36=middle C)
  the third byte is the velocity for that note (that is how powerful the note was struck= 100)

example midi message: 128-36
  this message is a "note off" message (status byte = 128). it is followed by the note (data byte = 36)
  since "note off" messages don't need a velocity value (it's just OFF) there will be no third byte in this case
  NOTE: some midi keyboards will never send a "note off" message, but rather a "note on with zero velocity"

HARDWARE NOTE:
  The Midi Socket is connected to arduino RX (serial receive/in)
 */

 //bass synth
#define BASS_CLOCK 8    //PORTB 0
#define VAC_PWM 9
#define VAC_CUTOFF 10
#define PWM_OUT 11      //PORTB 3
#define SAW_OUT 12      //PORTB 4
#define MIDI_ACTIVE 13

//controls
#define ATTACK A5
#define DECAY A4
#define WAVE A3

unsigned int pitchcounter = 0;
uint16_t bassnote = 0;

//LFO
int LFO = 0;
int LFOspd = 1;
bool LFOdir = true;

//AD envelope
bool gate = false;
int ENV = 0;
int envatt = 8;
int envdec = 1;

uint16_t SYNTH_PITCHES[] =
{
  //C1  C#1 D1  D#1 E1  F1  F#1 G1  G#1 A1  A#1 B1
  959,  905,855,807,761,719,678,640,604,570,538,508,

  //C2  C#2 D2  D#2 E2  F2  F#2 G2  G#2 A2  A#2 B2
  480,  453,427,403,381,359,339,320,302,285,269,254,

  //C3  C#3 D3  D#3 E3  F3  F#3 G3  G#3 A3  A#3 B3
  240,  226,214,202,190,180,170,160,151,143,135,127,

  //C4  C#4 D4  D#4 E4  F4  F#4 G4  G#4 A4  A#4 B4
  120,  113,107,101,95, 90, 85, 80, 76, 71, 67, 64,

  //C5  C#5 D5  D#5 E5  F5  F#5 G5  G#5 A5  A#5 B5
  60,   57, 53, 50, 48, 45, 42, 40, 38, 36, 34, 32,

  //C6  C#6 D6  D#6 E6  F6  F#6 G6  G#6 A6  A#6 B6
  30,   28, 27, 25, 24, 22, 21, 20, 19, 18, 17, 16
};

const PROGMEM uint16_t BASSLINE_ROCK[16] =
{
  480, 240, 0, 0, 480, 240, 0, 0,
  480, 240, 0, 0, 480, 240, 160, 143
};

const PROGMEM uint16_t BASSLINE_POP[16] =
{
  381, 381, 381, 381, 381, 381, 339, 320,
  285, 285, 285, 285, 285, 320, 320, 339
};

const PROGMEM uint16_t BASSLINE_SWING[16] =
{
  761, 640, 0, 761, 480, 0, 480, 427,
  427, 427, 0, 427, 427, 0, 427, 640
};

const PROGMEM uint16_t BASSLINE_SLOWROCK[16] =
{
  285, 320, 381, 508, 0, 0, 0, 381,
  508, 453, 427, 508, 0, 0, 0, 0
};

const PROGMEM uint16_t BASSLINE_MARCH[16] =
{
  381, 0, 190, 0, 381, 0, 190, 0,
  381, 339, 190, 320, 381, 0, 190, 320
};

const PROGMEM uint16_t BASSLINE_RAP[16] =
{
  761, 761, 0, 761, 640, 0, 761, 761,
  678, 640, 0, 640, 0, 640, 640, 0
};

const PROGMEM uint16_t BASSLINE_METAL[16] =
{
  480, 480, 480, 0, 480, 480, 480, 0,
  508, 381, 381, 381, 427, 427, 427, 427
};

const PROGMEM uint16_t BASSLINE_ROCKTWO[16] =
{
  381, 381, 381, 480, 254, 254, 254, 320,
  285, 285, 285, 285, 190, 190, 190, 190
};

//RAM MEMORY TABLE
uint16_t bassline_table[16]   __attribute__ ((aligned(16))) =
{
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
};


//drum triggers
#define TRIG_KICK 3
#define TRIG_SNR 4
#define TRIG_HH 5
#define TRIG_RIM 6
#define TRIG_ACC 2

//controls
#define MIDIEXT 7
#define TEMPO A2
#define BEAT A0
#define STST A1
```

```
bool stbuttprev = false;
int tempo_prev = 0;
bool runnin = true;

//RHY1**********************************************************
bool RHY_ROCK_K[16] =
{true,false,false,false,true,false,false,false,
 true,false,false,false,true,false,false,false};

bool RHY_ROCK_S[16] =
{false,false,true,false,false,false,true,false,
 false,false,true,false,false,false,true,false};

bool RHY_ROCK_H[16] =
{true,true,true,true,true,true,true,true,
 true,true,true,true,true,true,true,true};

bool RHY_ROCK_R[16] =
{false,false,false,false,false,false,false,false,
 false,false,false,false,false,true,false,true};

bool RHY_ROCK_A[16] =
{true,false,true,false,true,false,true,false,
 true,false,true,false,true,false,true,false};

//RHY2**********************************************************
bool RHY_POP_K[16] =
{true,false,false,false,false,false,true,false,
 true,false,true,false,false,false,false,false};

bool RHY_POP_S[16] =
{false,false,false,false,true,false,false,false,
 false,false,false,false,true,false,false,false};

bool RHY_POP_H[16] =
{true,false,true,false,true,false,true,false,
 true,false,true,false,true,false,true,true};

bool RHY_POP_R[16] =
{false,false,false,false,false,false,false,false,
 false,false,false,false,false,false,false,false};

bool RHY_POP_A[16] =
{false,false,false,false,false,false,false,false,
 false,false,false,false,false,false,false,true};

//RHY3**********************************************************
bool RHY_SWING_K[16] =
{true,false,false,false,false,false,true,false,
 false,false,false,false,true,false,true,false};

bool RHY_SWING_S[16] =
{false,false,false,false,false,false,false,false,
 false,false,false,false,false,false,false,false};

bool RHY_SWING_H[16] =
{true,false,true,true,false,true,true,false,
 true,true,false,true,true,false,false,true};

bool RHY_SWING_R[16] =
{false,false,false,true,false,false,false,false,
 false,true,false,false,false,false,true,false};

bool RHY_SWING_A[16] =
{true,false,false,true,false,false,true,false,
 false,true,false,false,true,false,true,false};

//RHY4**********************************************************
bool RHY_SLOWROCK_K[16] =
{true,false,false,false,false,false,false,false,
 false,false,false,false,false,false,true,false};

bool RHY_SLOWROCK_S[16] =
{false,false,false,false,false,false,false,false,
 true,false,false,false,false,false,false,false};

bool RHY_SLOWROCK_H[16] =
{true,false,true,false,true,false,true,false,
 true,false,true,false,true,false,true,false};

bool RHY_SLOWROCK_R[16] =
{false,false,false,false,false,false,false,false,
 false,false,false,false,false,false,true,true};

bool RHY_SLOWROCK_A[16] =
{false,false,true,false,false,false,true,false,
 false,false,true,false,false,false,true,false};
```

```
//RHY5*********************************************************
bool RHY_MARCH_K[16] =
{true,false,false,false,true,false,false,false,
 true,false,false,false,true,false,false,false};

bool RHY_MARCH_S[16] =
{true,false,true,true,true,false,true,false,
 true,true,true,false,true,false,true,false};

bool RHY_MARCH_H[16] =
{true,false,false,false,false,false,false,false,
 false,false,false,false,false,false,false,false};

bool RHY_MARCH_R[16] =
{false,false,false,false,false,false,false,false,
 false,false,false,false,true,false,true,false};

bool RHY_MARCH_A[16] =
{true,false,false,false,true,false,true,false,
 false,false,true,false,false,false,true,false};

//RHY6*********************************************************
bool RHY_RAP_K[16] =
{true,false,false,false,true,false,false,false,
 true,false,false,false,true,false,false,true};

bool RHY_RAP_S[16] =
{false,false,true,false,false,false,true,false,
 false,false,true,false,false,false,true,false};

bool RHY_RAP_H[16] =
{false,true,false,true,false,true,false,true,
 false,true,false,true,false,true,false,true};

bool RHY_RAP_R[16] =
{false,false,false,false,false,false,false,false,
 false,false,true,true,false,true,true,false};

bool RHY_RAP_A[16] =
{false,false,false,false,false,false,false,false,
 false,false,false,false,true,false,true,false};

//RHY7*********************************************************
bool RHY_METAL_K[16] =
{true,true,true,false,true,true,true,false,
 true,true,true,false,true,true,true,false};

bool RHY_METAL_S[16] =
{false,false,false,false,false,false,false,false,
 true,false,false,false,false,false,false,false};

bool RHY_METAL_H[16] =
{true,false,true,false,true,false,true,false,
 true,false,true,false,true,false,true,false};

bool RHY_METAL_R[16] =
{false,false,false,false,false,false,false,false,
 false,false,false,false,false,false,false,false};

bool RHY_METAL_A[16] =
{false,false,false,false,false,false,false,false,
 false,false,false,false,false,false,false,false};

//RHY8*********************************************************
bool RHY_ROCKTWO_K[16] =
{true,false,false,false,true,true,false,false,
 true,false,false,false,true,true,false,false};

bool RHY_ROCKTWO_S[16] =
{false,false,true,false,false,false,true,false,
 false,false,true,false,false,false,true,false};

bool RHY_ROCKTWO_H[16] =
{true,true,true,true,true,true,true,true,
 true,true,true,true,true,true,true,true};

bool RHY_ROCKTWO_R[16] =
{false,true,false,true,false,true,false,true,
 false,true,false,true,false,true,false,true};

bool RHY_ROCKTWO_A[16] =
{true,false,true,false,true,false,true,false,
 true,false,true,false,true,false,true,false};

//Port manipulation macros
#define CLR(x,y)(x&=(~(1<<y)))
```

```
#define SET(x,y)(x|=(1<<y))

//globals
unsigned long beat_timer = 0;
unsigned long last_time = 0;
unsigned long last_time_s = 0;
unsigned long last_time_h = 0;
unsigned long last_time_r = 0;

int scancount = 0;

bool internal = false;
byte MIDImode = 0;
byte MIDImode_prev = 0;

int curr_step = 0;
const bool* kick_steps = RHY_ROCK_K;
bool* snr_steps = RHY_ROCK_S;
bool* hh_steps = RHY_ROCK_H;
bool* rim_steps = RHY_ROCK_R;
bool* acc_steps = RHY_ROCK_A;

//FUNCTIONS  ****************************************************************
void SetWaveform()
{
  int data = analogRead(WAVE);

  if(data < 400) //just a pulse
  {
    SET(PORTB, 3);
    CLR(PORTB, 4);
    LFOspd = 0;
    OCR1A = 0;
  }
  else if(data < 980) //PWM + SAW
  {
    SET(PORTB, 3);
    SET(PORTB, 4);
    LFOspd = map(data, 400, 979, 1, 5);
  }
  else //SAW
  {
    CLR(PORTB, 3);
    SET(PORTB, 4);
    LFOspd = 0;
    OCR1A = 0;
  }
}

void SetControls()
{
  //scan tempo 0-1023 mapped to 10-220 BPM, to millisecs, mapped to microsecs
  //((60,000 / BPM) / 4) * 1000
  int temp = 0;
  if((temp = analogRead(TEMPO)) != tempo_prev) //quite expensive so lets only do it when changed
  {
      tempo_prev = temp;
      beat_timer = ((60000 / (long)map(temp, 0, 1023, 10, 220)) / 4) * 1000;
  }

  //scan beat knob
  int tempb = analogRead(BEAT) >> 7; //8 beats
  if(tempb != temp)
  {
    temp = tempb;
    switch(temp)
    {
      case 0:
        kick_steps = RHY_ROCK_K;
        snr_steps = RHY_ROCK_S;
        hh_steps = RHY_ROCK_H;
        rim_steps = RHY_ROCK_R;
        acc_steps = RHY_ROCK_A;

        for(int i = 0; i < 16; i++)
        {
            bassline_table[i] = (uint16_t)pgm_read_word_near(BASSLINE_ROCK + i);
        }
      break;

      case 1:
        kick_steps = RHY_POP_K;
        snr_steps = RHY_POP_S;
        hh_steps = RHY_POP_H;
        rim_steps = RHY_POP_R;
        acc_steps = RHY_POP_A;
    for(int i = 0; i < 16; i++)
    {
        bassline_table[i] = (uint16_t)pgm_read_word_near(BASSLINE_POP + i);
    }
  break;

  case 2:
    kick_steps = RHY_SWING_K;
    snr_steps = RHY_SWING_S;
    hh_steps = RHY_SWING_H;
    rim_steps = RHY_SWING_R;
    acc_steps = RHY_SWING_A;

    for(int i = 0; i < 16; i++)
    {
        bassline_table[i] = (uint16_t)pgm_read_word_near(BASSLINE_SWING + i);
    }
  break;

  case 3:
    kick_steps = RHY_SLOWROCK_K;
    snr_steps = RHY_SLOWROCK_S;
    hh_steps = RHY_SLOWROCK_H;
    rim_steps = RHY_SLOWROCK_R;
    acc_steps = RHY_SLOWROCK_A;

    for(int i = 0; i < 16; i++)
    {
        bassline_table[i] = (uint16_t)pgm_read_word_near(BASSLINE_SLOWROCK + i);
    }
  break;

  case 4:
    kick_steps = RHY_MARCH_K;
    snr_steps = RHY_MARCH_S;
    hh_steps = RHY_MARCH_H;
    rim_steps = RHY_MARCH_R;
    acc_steps = RHY_MARCH_A;

    for(int i = 0; i < 16; i++)
    {
        bassline_table[i] = (uint16_t)pgm_read_word_near(BASSLINE_MARCH + i);
    }
  break;

  case 5:
    kick_steps = RHY_RAP_K;
    snr_steps = RHY_RAP_S;
    hh_steps = RHY_RAP_H;
    rim_steps = RHY_RAP_R;
    acc_steps = RHY_RAP_A;

    for(int i = 0; i < 16; i++)
    {
        bassline_table[i] = (uint16_t)pgm_read_word_near(BASSLINE_RAP + i);
    }
  break;

  case 6:
    kick_steps = RHY_METAL_K;
    snr_steps = RHY_METAL_S;
    hh_steps = RHY_METAL_H;
    rim_steps = RHY_METAL_R;
    acc_steps = RHY_METAL_A;

    for(int i = 0; i < 16; i++)
    {
        bassline_table[i] = (uint16_t)pgm_read_word_near(BASSLINE_METAL + i);
    }
  break;

  case 7:
    kick_steps = RHY_ROCKTWO_K;
    snr_steps = RHY_ROCKTWO_S;
    hh_steps = RHY_ROCKTWO_H;
    rim_steps = RHY_ROCKTWO_R;
    acc_steps = RHY_ROCKTWO_A;

    for(int i = 0; i < 16; i++)
    {
        bassline_table[i] = (uint16_t)pgm_read_word_near(BASSLINE_ROCKTWO + i);
    }
  break;

  default:
    kick_steps = RHY_ROCK_K;
    snr_steps = RHY_ROCK_S;
```

```
        hh_steps = RHY_ROCK_H;
        rim_steps = RHY_ROCK_R;
        acc_steps = RHY_ROCK_A;

        for(int i = 0; i < 16; i++)
        {
            bassline_table[i] = (uint16_t)pgm_read_word_near(BASSLINE_ROCK + i);
        }
      break;
    }
  }

  //scan attack
  envatt = 1023-analogRead(ATTACK); //1023 - 0

  //scan decay
  envdec = (1024-analogRead(DECAY)) >> 3; //128 - 0

  //scan start/stop
  if(!digitalRead(STST) && !stbuttprev)
  {
    runnin = !runnin;
    if(!runnin)
      curr_step = 0; //reset
    stbuttprev = true;
  }
  else if(digitalRead(STST))
    stbuttprev = false;

  //scan int/ext
  if(digitalRead(MIDIEXT))
    internal = true;
  else
    internal = false;

}


//MAIN FUNCTIONS ***********************************************************
void setup()
{
  pinMode(MIDI_ACTIVE, OUTPUT);
  digitalWrite(MIDI_ACTIVE, LOW);

  //60,000 / BPM = MS per 4th note
  //our sequencer always counts in 16th notes
  //so our final MS is: 4thMS/4
  //So for 120 BPM this would be: 125 ms = 125000 us
  beat_timer = 125000;
  last_time = micros();

  //3 = kick    (PORTD 3)
  //4 = snare   (PORTD 4)
  //5 = hh      (PORTD 5)
  //6 = rimshot (PORTD 6)
  //7 = accent  (PORTD 7)
  pinMode(TRIG_KICK, OUTPUT);
  pinMode(TRIG_SNR, OUTPUT);
  pinMode(TRIG_HH, OUTPUT);
  pinMode(TRIG_RIM, OUTPUT);
  pinMode(TRIG_ACC, OUTPUT);

  pinMode(STST, INPUT_PULLUP);
  digitalRead(STST); //set as digital input?

  pinMode(MIDIEXT, INPUT_PULLUP);

  digitalWrite(TRIG_KICK, LOW);
  digitalWrite(TRIG_SNR, LOW);
  digitalWrite(TRIG_HH, LOW);
  digitalWrite(TRIG_RIM, LOW);
  digitalWrite(TRIG_ACC, LOW);

  //bass synth
  //8 = clock           (PORTB 0)
  //9 = PWM vactrol     (OCR1A)
  //10 = cutoff vactrol (OCR1B)
  //11 = PWM out on/off (PORTB 3)
  //12 = saw out on/off (PORTB 4)
  pinMode(BASS_CLOCK, OUTPUT);
  pinMode(VAC_PWM, OUTPUT);
  pinMode(VAC_CUTOFF, OUTPUT);
  pinMode(PWM_OUT, OUTPUT);
  pinMode(SAW_OUT, OUTPUT);

  digitalWrite(BASS_CLOCK, LOW);
  digitalWrite(PWM_OUT, HIGH);
  digitalWrite(SAW_OUT, HIGH);

  //timer 1 is responsible for CV outputs
  TCCR1A = _BV(COM1A1) | _BV(COM1B1) | _BV(WGM10); //8-bit  fast
  TCCR1B = _BV(CS10); //32KHz
  TIMSK1 |= (1 << TOIE1); //overflow interrupt
  OCR1A = 0; //9 pwm
  OCR1B = 0; //10 cutoff

  sei();

  Serial.begin(31250);
  digitalWrite(MIDI_ACTIVE, HIGH);
}

void loop()
{
  //scan controsl
  scancount++;
  if(scancount > 1024)
  {
    scancount = 0;
    SetWaveform();
    SetControls();
  }

  if(internal) //internal beat machine
  {
    if(runnin)
    {
      unsigned long curr_time = micros(); //need to take care of overflow after ~70 minutes
      if(curr_time - last_time > beat_timer)
      {
          last_time = curr_time;

          bassnote = bassline_table[curr_step];
          gate = true;

          //set outputs
          if(kick_steps[curr_step])
            SET(PORTD, TRIG_KICK);
          else
            CLR(PORTD, TRIG_KICK);

          if(snr_steps[curr_step])
            SET(PORTD, TRIG_SNR);
          else
            CLR(PORTD, TRIG_SNR);

          if(hh_steps[curr_step])
            SET(PORTD, TRIG_HH);
          else
            CLR(PORTD, TRIG_HH);

          if(rim_steps[curr_step])
            SET(PORTD, TRIG_RIM);
          else
            CLR(PORTD, TRIG_RIM);

          if(acc_steps[curr_step])
            SET(PORTD, TRIG_ACC);
          else
            CLR(PORTD, TRIG_ACC);

          //step
          curr_step++;
          if(curr_step > 15)
              curr_step = 0;
      }

       //set pins to LOW after 10ms/10000us, to ensure trigger pulse width, regardless of tempo
      curr_time = micros();
      if(curr_time - last_time > 10000)
      {
         CLR(PORTD, TRIG_KICK);
         CLR(PORTD, TRIG_SNR);
         CLR(PORTD, TRIG_HH);
         CLR(PORTD, TRIG_RIM);
         CLR(PORTD, TRIG_ACC);
      }
    }
    else
    {
      bassnote = 0;
      gate = false;
    }
  }
```

```
else //external MIDI input
{
   unsigned long curr_time = micros(); //need to take care of overflow after ~70 minutes
   if(curr_time - last_time > 10000)
     CLR(PORTD, TRIG_KICK);
   if(curr_time - last_time_s > 10000)
     CLR(PORTD, TRIG_SNR);
   if(curr_time - last_time_h > 10000)
     CLR(PORTD, TRIG_HH);
   if(curr_time - last_time_r > 10000)
     CLR(PORTD, TRIG_RIM);

   if(Serial.available())
   {
      byte comm = Serial.read();

      //channel 0 = bass synth
      //starts at C1 = MIDI note 24

      //channel 10 = drums

      //note off = 1000nnnn (n = channel)
      //note on =  1001nnnn (n = channel)

      if(comm > 127) //message
      {
        if(comm == 144) //note on, channel 0
        {
          MIDImode = 1;
        }
        else if(comm == 153) //note on, channel 10 (9)
        {
          MIDImode = 3;
        }
        else if(comm == 128) //note off, //channel 0
        {
          MIDImode = 2;
        }
        else if(comm == 176) //channel message
        {
          MIDImode = 4;
        }
      }
      else //data
      {
        if(MIDImode == 1) //note on note
        {
          if(comm >= 24 && comm < 96)
          {
            bassnote = SYNTH_PITCHES[comm - 24];
            gate = true;
            MIDImode = 0;
          }
        }
        else if(MIDImode == 2) //note off note
        {
          bassnote = 0;
          MIDImode = 99;
          MIDImode_prev = 2;
        }
        else if(MIDImode == 3) //note off note
        {
          if(comm == 36) //kick
          {
            SET(PORTD, TRIG_KICK);
            last_time = micros();
          }
          else if(comm == 40) //snare
          {
            SET(PORTD, TRIG_SNR);
            last_time_s = micros();
          }
          else if(comm == 42) //high hat
          {
            SET(PORTD, TRIG_HH);
            last_time_h = micros();
          }
          else if(comm == 31) //rimshot
          {
            SET(PORTD, TRIG_RIM);
            last_time_r = micros();
          }

          MIDImode = 99;
          MIDImode_prev = 3;
        }
        else if(MIDImode == 4 && comm == 123) //all notes off
```

```
          {
            bassnote = 0;
            gate = false;
            MIDImode = 99;
            MIDImode_prev = 0;
          }
          else if(MIDImode == 99)
          {
            //misc messages & velocity stuff we skip
            MIDImode = MIDImode_prev;
          }
        }
      }
  }
}

//TIMER 1 interrupt for bass synth, 31372.55 Hz/32us
ISR(TIMER1_OVF_vect)
{
  //modulate PWM on bass synth
  if(LFOdir)
  {
    LFO+=LFOspd;
    if(LFO > 20000)
    {
      LFO = 20000;
      LFOdir = false;
    }
  }
  else
  {
    LFO-=LFOspd;
    if(LFO < 2049)
    {
      LFO = 2048;
      LFOdir = true;
    }
  }

  OCR1A = LFO >> 8;

  //env
  if(gate)
  {
    ENV += envatt;
    if(ENV > 32512)
      gate = false;
  }
  else if(ENV > envdec)
  {
    ENV -= envdec;
  }


  OCR1B = ENV >> 8;

  if(bassnote)
  {
    pitchcounter++;
    if(pitchcounter > bassnote)
    {
      pitchcounter = 0;
      //send a short pulse < 32us
      SET(PORTB, 0);
      delayMicroseconds(20);
      CLR(PORTB, 0);
    }
  }
}
```

The Hardware

The hardware is separated in 3 sections:
- Power supply, delivers +12V and +5V,
- Drum machine, the modded CB-55 board,
- Bass synth, DCO and VCF with various routing.

So let's go over each section.

THE POWER SUPPLY:
I used an old 15V AC wall wart, removed the transformer from it to be used as the mains transformer in Blue Clone. MUCH cheaper than having to buy a new transformer! I also salvaged a fuse holder with a proper fuse too. Don't want mishaps to be big mishaps :)
After that it's all pretty standard. AC gets rectified and smoothed to around 20V DC, then regulated down to 12V by a 7812. The 5V is regulated from that 12V by an

7805.

The 12V rail supplies most of the analog circuitry and the Arduino (on the Vin pin! The Arduino has its own 5V regulator).
The 5V rail supplies the 4016 switches, due to the 5V logic level used by Arduino.

THE DRUM SYNTH
I'm only going to list the various mods here, for details on/schematics of the board check the CircuitBenders link supplied at the top of the page.

Bass drum:
- BD decay: Replace R33 with a 10k pot.
- BD pitch: Replace R28 with a 500k pot.
- BD punch: Replace R35 with a 50k pot. This goes from a mellow tone to a punch in your face!

Snare Drum:
- SD decay: Replace R55 with a 100k pot.
- SD pitch: Replace R49 with a 50k pot.
- SD noise: Replace R52 with a 10k pot. Sets the noise decay.

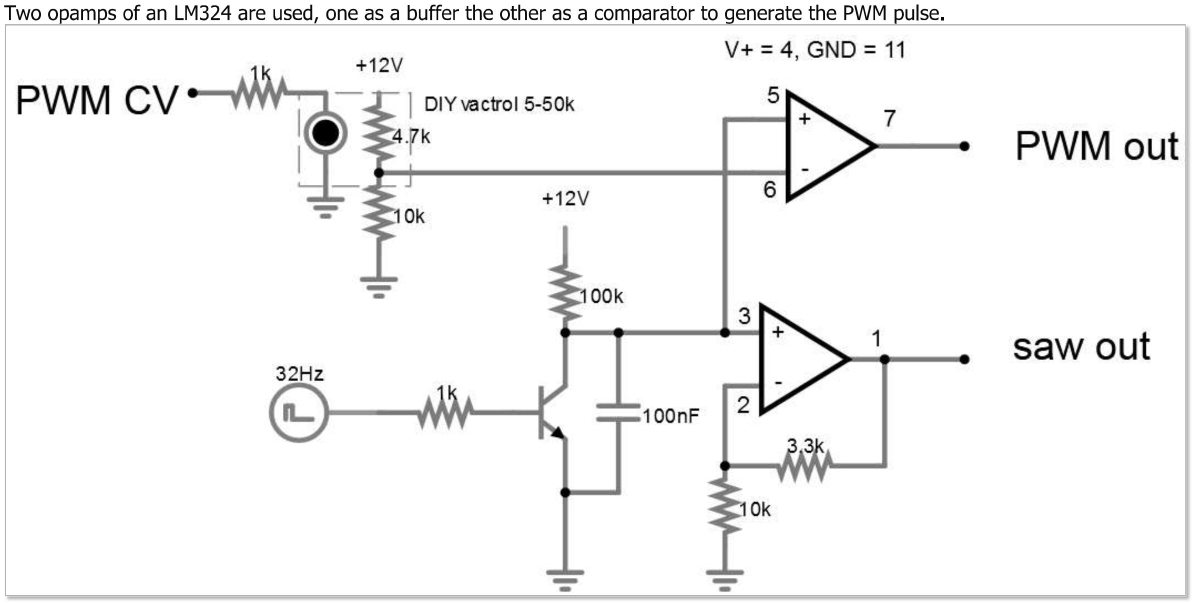
Hats:
- HH decay: Attach to the top pin of R67 a 500k pot and 100nF capacitor to ground.
- HH decay: Attach a 68nF capacitor to the middle leg of T7, and a 20k pot between the capacitor and ground.

Rimshot:
- RS pitch: Replace R43 with a 1M pot.

THE BASS SYNTH
I've kept this circuit as simple as possible, as there wasn't a lot of space left while keeping things neat.

The DCO is a very simple one: a capacitor is charged through a resistor and discharged through an parallel transistor. The Arduino generates pulses in the frequency of the note played to reset the capacitor. Of course there's the issue with a lower amplitude at higher notes but I found this not to be an issue for a bass synth wich doesn't play that big a range anyways. It can do 4 octaves quite well though!

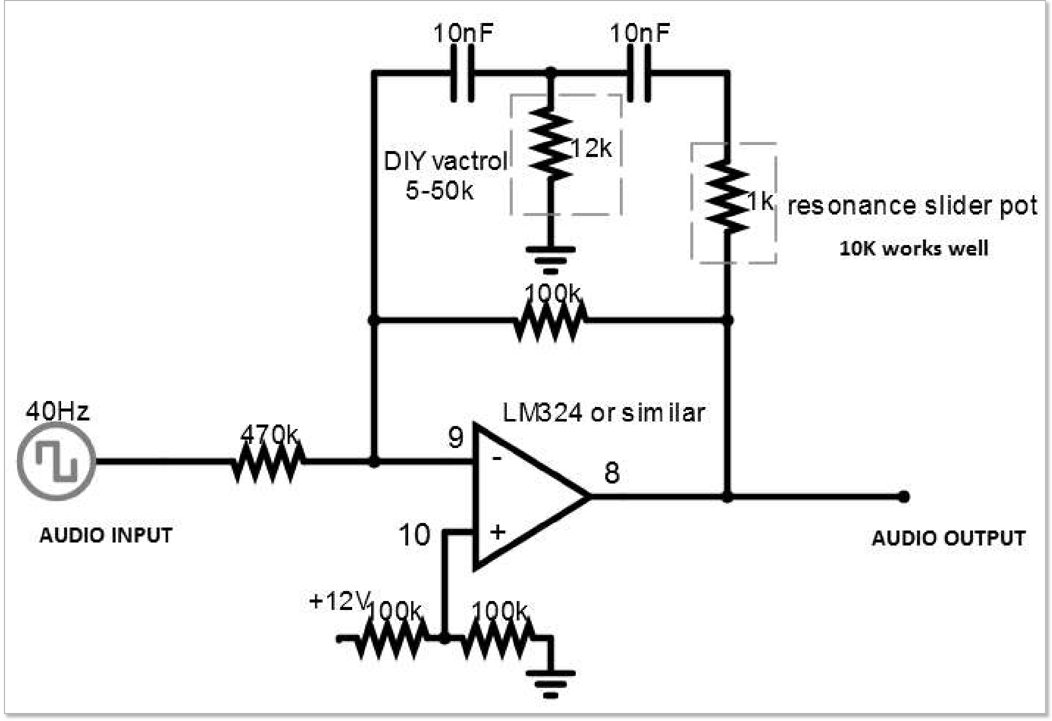
Two opamps of an LM324 are used, one as a buffer the other as a comparator to generate the PWM pulse.



The 2 signals go through an CD4016 analog switch, which is controlled by the Arduino and are then passively mixed to the input of the VCF.

The VCF is also very simple, I made my own vactrols from a 3mm red LED and an LDR that ranges from around 5 to 350 kilo ohms.
This allowed me to just put the vactrols in place where I needed a voltage controlled variable resistance.

The low pass filter itself is a very standard twin-T circuit. Simple but quite effective :)


10nF
10nF
DIY vactrol
5-50k
12k
1k
resonance slider pot
10K works well
100k
40Hz
470k
LM324 or similar
9
-
8
10
+
AUDIO INPUT
AUDIO OUTPUT
+12V
100k
100k