Listing 1. Listing szkicu Arduino, kontrolującego działanie sekwencera

#include <Adafruit\_NeoPixel.h>    // Biblioteka do obsługi Adafruit NeoPixel

#include <Encoder.h>              // Biblioteka do obsługi enkodera

Encoder myEnc(3, 4);              // Obiekt enkodera, dołączonego do pinów cyfrowych 3 i 4

#define PixelPin    2             // Moduł NeoPixel dołączony do pinu cyfrowego 2

#define ProgPin     5             // Przełącznike enkodera dołączony do pinu cyfrowego 5

#define ClkPin      6             // Zewnętrzny zegar dołączony do pinu cyfrowego 6

#define Trig1Pin   10             // Cztery sygnały wyzwalacza, dołączone do pinów 7-10

#define Trig2Pin    8

#define Trig3Pin    9

#define Trig4Pin    7

#define TapPin     11             // Przycisk tempa dołączony do pinu cyfrowego 11

#define ButtonPin  A0             // Wybór wyzalania (dielnika napięcia) do pinu analogowego A0

#define PotPin     A1             // Potencjometr czasu trwania impulsu do pinu analogowego A1

#define ExtClkPin  A2             // Zewnętrzne napięcie kontrolne zegtara do pinu analogowego A2

#define NumSteps   16             // Predefiniowana liczba kroków w sekwencji

int  Part = 1;                    // Aktualny wyzwalacz ustawiony początkowo na 1

Adafruit\_NeoPixel pixels(NumSteps, PixelPin, NEO\_GRB + NEO\_KHZ800);  // Konfiguracja modułu NeoPixel

unsigned int Euclid(int NumPulses) {

 unsigned int Number = 0;              // Number to liczba impulsów, które już zostaly zaalokowane

 for (int i=0; i<NumSteps; i++) {      // Przejście przez wszystkie kroki w sekwencji

   int bucket = bucket + NumPulses;    // Wypełnia bucket liczbą impulsów do alokacji

   if (bucket >= NumSteps) {           // Jeśli bucket > liczba kroków to opróźnia bucket

     bucket = bucket - NumSteps;       // ustawiający i-ty bit w sekwencju

     Number |= 1 << i;                 // i wpisując do bucket liczbę pozostałych impulsów.

   }

 }

 return(Number);

}

unsigned int RotateLeft(unsigned int Number) {

 int DROPPED\_MSB;

 DROPPED\_MSB = (Number >> NumSteps-1) & 1;

 Number = (Number << 1) | DROPPED\_MSB;

 return(Number);

}

unsigned int RotateRight(unsigned int Number) {

 int DROPPED\_LSB;

 DROPPED\_LSB = Number & 1;

 Number = (Number >> 1) & (~(1 << NumSteps-1));

 Number = Number | (DROPPED\_LSB << NumSteps-1);

 return(Number);

}

int CheckButtons() {

 int Buttons = analogRead(ButtonPin);         // Odczytuje napięcie z dzielnika napięciowego

 if ((Buttons>480)&&(Buttons<520)) Part = 1;  // Wybierz wyjście 1, gdy naciśnięto pierwszy przycisk

 if ((Buttons>580)&&(Buttons<620)) Part = 2;  // Wybierz wyjście 2, gdy naciśnięto pierwszy drugi

 if ((Buttons>680)&&(Buttons<720)) Part = 3;  // Wybierz wyjście 3, gdy naciśnięto pierwszy trzeci

 if (Buttons>800) Part = 4;                   // Wybierz wyjście 4, gdy naciśnięto pierwszy czwarty

}

void ClearPattern() {

 for (int i=0; i<NumSteps; i++)                   // Przejdź przez każdy piksel w pierścieniu LED

   pixels.setPixelColor(i, pixels.Color(0,0,0));  // Wyłącz każdy piksel

 pixels.show();                                   // Aktualizuj stan wyświetlacza

}

void BitPattern(int Chan, unsigned int Number) {

 int R;

 int G;

 int B;

 switch (Chan) {

   case 1: R = 20; G = 0;  B = 0;  break;  // Wzór 1 jest czerwony

   case 2: R = 0;  G = 20; B = 0;  break;  // Wzór 2 jest zielony

   case 3: R = 0;  G = 0;  B = 20; break;  // Wzór 3 jest niebieski

   case 4: R = 20; G = 20; B = 0;  break;  // Wzór 4 jest żółty

 }

 for (int i=0; i<NumSteps; i++)

   if (Number & (1<<i))

      pixels.setPixelColor(i, pixels.Color(R,G,B));

   else

      pixels.setPixelColor(i, pixels.Color(0,0,0));

 pixels.show();

}

void setup() {

**Serial**.begin(9600);                       // Uruchom konsolę szeregową

 pinMode(ProgPin,INPUT\_PULLUP);            // Przypisz piny enkodera i włącz podciągnięcie do zasilania

 pinMode(ClkPin,INPUT);                    // Przypisz pin selektroa zewnętrznego zegara

 pinMode(Trig1Pin,OUTPUT);                 // Przypisz wyjście 1

 pinMode(Trig2Pin,OUTPUT);                 // Przypisz wyjście 2

 pinMode(Trig3Pin,OUTPUT);                 // Przypisz wyjście 3

 pinMode(Trig4Pin,OUTPUT);                 // Przypisz wyjście 4

 pinMode(TapPin,INPUT);                    // Przypisz przycisk wyboru tępa

 pixels.begin();                           // Uruchom NeoPixel

 pixels.clear();                           // Wyczyść NeoPixel

}

void loop() {

 static unsigned int Ch1 = 0x8888;         // Domyślne wzory dla wyjść 1-4

 static unsigned int Ch2 = 0x4444;

 static unsigned int Ch3 = 0x2222;

 static unsigned int Ch4 = 0xEEEE;

 static int  Step        = 0;              // Aktywny krok jest zapisany w pamieci pomiędzy pętlami, ale startuje od 0

 static int  Delay       = 80;             // Czas pomiędzy krokami też jest zapisany w pamięci, startuje od 80 ms

 static int  Mode        = 0;              // Tryb zapisywany jest pomiędzy pętlami

 static long Mode1Pos    = 0;              // Ostatnia pozycja enkodera w trybie 1

 static int  PrevExtClk  = 0;              // Wartość zewnętrznego zegara do detekcji zbocza

 bool        Triggered   = false;          // Załóż, że nie będzie nowych kroków w tej iteracji pętli

 static unsigned long Time;                // Przygotowanie do pobrania danych z zegara Nano

 static unsigned long PrevTime = 0;        // Zlicza czas pomiędzy pętlami

 static long oldPosition;                  // Poprzednia wartość licznika enkodera

 static int  Pulses;                       // Liczba impulsów generowana dla aktywnego wyjścia

 static bool PrevProg;                     // Poprzedni stan przycisku enkodera

 static bool Prog;                         // Przygotowanie do odczytania obecnego stanu przycisku enkodera

 long newPosition = myEnc.read();          // Odczytaj licznik enkodera

 int  PotV        = analogRead(PotPin);    // Odczytaj wartość potencjometra długości impulsu

 int  ExtClkV     = analogRead(ExtClkPin); // Odczytaj wejście zewnętrznego zegara

 CheckButtons();                           // Odczytaj wartość przycisku wyjścia i aktualizuj aktywne wyjście

 PrevProg = Prog;                          // Zapis wartość przełącznika enkodera

 Prog     = digitalRead(ProgPin);          // Pobierz wartość przełącznika enkodera

 Time = millis();                          // Pobierz wartość czasu systemowego z Nano (w ms)

 if ((Prog==true) && (PrevProg==false)) {  // Jeśli naciśnięto przycisk enkodera trzeba zmienić tryb

    if (Mode==1) Mode1Pos = newPosition;                 // Zapisz pozycję enkodera, aby wrócić do trybu

    Mode++;                                                          // Przejdź do następnego trybu

    if (Mode>3) Mode=1;                                              // Przejdź do trybu 1 z trybu 3

    if (Mode==1) { myEnc.write(Mode1Pos); newPosition = Mode1Pos; }  // Odtwórz pozycję enkodera w tym trybie

    oldPosition = newPosition;

 }

 ClearPattern();

 switch(Mode) {

   case 1:                                    // Tryb 1 - Manualna regulacja tempa sekwencji

     if (newPosition!=oldPosition) {

        Delay = newPosition\*2;                // Wewnętrzne opóźnienie jest proporcjonalne do pozycji enkodera

        if (Delay<10) Delay = 10;             // Najmniejszy czas opóźnienia = 10 ms

        if (Delay>1000) Delay = 1000;         // Największy  czas opóźnienia = 1 s

        oldPosition = newPosition;

     }

     break;

   case 2:                                    // Tryb 2 - Zmiana liczby impulsów w trybie Euklidesowskim

     if (newPosition < oldPosition-3) {

       Pulses++;                              // Zwięsz liczbę impulsów

       if (Pulses>NumSteps) Pulses=NumSteps;  // Ograniczenie liczby impulsów <= NumSteps

       oldPosition = newPosition;

     }

     else if (newPosition > oldPosition+3) {

       Pulses--;                              // Zmniejsz liczbę impulsów

       if (Pulses<0) Pulses=0;                // Ograniczenie liczby impulsów => 0

       oldPosition = newPosition;

     }

**Serial**.println(Pulses);

     switch (Part) {                          // Obliczenia Rytmu Euklidesowskiego dla obecnego układu

       case 1: Ch1 = Euclid(Pulses); break;

       case 2: Ch2 = Euclid(Pulses); break;

       case 3: Ch3 = Euclid(Pulses); break;

       case 4: Ch4 = Euclid(Pulses); break;

     }

     break;

   case 3:                                    // Tryb 3 - Przekręcanie Rotate the Euclidean Rhythm of the active pattern

     if (newPosition > oldPosition+3) {

       oldPosition = newPosition;

       switch (Part) {

         case 1: Ch1 = RotateRight(Ch1); break;

         case 2: Ch2 = RotateRight(Ch2); break;

         case 3: Ch3 = RotateRight(Ch3); break;

         case 4: Ch4 = RotateRight(Ch4); break;

       }

     }

     else if (newPosition < oldPosition-3) {

       oldPosition = newPosition;

       switch (Part) {

         case 1: Ch1 = RotateLeft(Ch1); break;

         case 2: Ch2 = RotateLeft(Ch2); break;

         case 3: Ch3 = RotateLeft(Ch3); break;

         case 4: Ch4 = RotateLeft(Ch4); break;

       }

     }

     break;

 }

 switch(Part) {                               // Pokaż aktywny wzór wraz z aktualizacjami w obecnym trybie

   case 1: BitPattern(1,Ch1); break;

   case 2: BitPattern(2,Ch2); break;

   case 3: BitPattern(3,Ch3); break;

   case 4: BitPattern(4,Ch4); break;

 }

 if (digitalRead(ClkPin)==true) {             // Jeżeli układ korzysta z zewnętrznego zegara

    if ((ExtClkV>512) && (PrevExtClk<128)) {  // Szuka zbocza narastającego

       Delay     = Time-PrevTime;             // Moment wystąpienia zbocza

       Triggered = true;

    }

    PrevExtClk = ExtClkV;

 }

 else {                                       // Jeżeli układ korzysta z wewnętrznego zegara

   if ((Time-PrevTime)>Delay) {               // Sprawdź czy minęło dostatczni dużo, aby przejść do kolejnego kroku sekwencji

     Triggered = true;

   }

 }

 int TrigWidth = map(PotV, 0, 1023, Delay/2, Delay/50);  // Oblicz szerokość impulsu

 if ((Time-PrevTime)>TrigWidth) {                        // Wyłącz wyjścia sterujące, jeśli minął czas impulsu

   digitalWrite(Trig1Pin,LOW);

   digitalWrite(Trig2Pin,LOW);

   digitalWrite(Trig3Pin,LOW);

   digitalWrite(Trig4Pin,LOW);

 }

 if (Triggered) {                              // Gdy uruchomiono kolejny cykl

   PrevTime = Time;

   Step++;

   if (Step>NumSteps-1) Step = 0;              // Reset licznika kroków, jeśi sekwencja zakończyła się

   if (Mode==1) pixels.setPixelColor(Step, pixels.Color(40,40,40)); // Zapal obecny piksel na kolor wynikający z trybu

   if (Mode==2) pixels.setPixelColor(Step, pixels.Color(40,0,40));

   if (Mode==3) pixels.setPixelColor(Step, pixels.Color(20,20,0));

   pixels.show();

   if (Ch1 & (1<<Step)) digitalWrite(Trig1Pin, HIGH);

   if (Ch2 & (1<<Step)) digitalWrite(Trig2Pin, HIGH);

   if (Ch3 & (1<<Step)) digitalWrite(Trig3Pin, HIGH);

   if (Ch4 & (1<<Step)) digitalWrite(Trig4Pin, HIGH);

 }

}