
	Instytut Informatyki Politechniki Śląskiej Zespół Mikroinformatyki i Teorii Automatów Cyfrowych			
Rok akademicki:	Rodzaj studiów*: SSI/NSI/NSM	Przedmiot (Języki Asemblerowe/SMiW):	Grupa	Seksja
2019/2020	SSI	SmiW	3-4	1
Imię:	Hubert	Prowadzący: OA/JP/KT/GD/BSz/GB	JP	
Nazwisko:	Przegendza			
<h2><i>Raport końcowy</i></h2>				
<p>Temat projektu:</p> <p style="text-align: center;"> 1.0 Tablica wibrująca dla głuchych. 1.1 Wizualizacja dźwięku za pomocą algorytmu FFT na wyświetlaczu LED. </p>				
Data oddania: dd/mm/rr		5.2.2020		

1. Temat projektu i opis założeń

Tablica wibrująca dla głuchych – płyta z ułożonymi na niej silniczkami, które za pomocą wibracji będą wizualizowały dźwięk korzystając z algorytmu FFT(Fast Fourier Transform).

Ze względu na ograniczenia prędkości przełączania silniczków, zgodnie z prowadzącym zamieniono silniczki na LED i tym samym zmieniono temat na:

Wizualizacja dźwięku na wyświetlaczu skonstruowanym z LED, korzystając z algorytmu FFT.

Założenia:

- układ wczytuje próbkę dźwięku za pomocą dołączonego mikrofonu
- na pobranej próbce układ wykonuje algorytm FFT
- wynik tej operacji (tablica zawierająca dane o natężeniu kolejnych częstotliwości składowych próbki dźwięku) jest formatowany i przedstawiony na wyświetlaczu
- układ może być zasilany poprzez kabel USB podłączony do DevKita lub bateryjnie korzystając z koszyka na baterie podpiętego do gniazda typu jack

Funkcja urządzenia

Osoby głuche poprzez odbiór dźwięku w formie składowych częstotliwości, mogliby potencjalnie zyskać większą świadomość otaczającego ich świata, np. podczas oglądania filmu, osoby te nie wiedzą, że na tle rozmawiających na obrazie osób występuje jakiś inny hałas np. tupanie. Korzystając z tego urządzenia, osoby głuche będą wiedziały z jakich częstotliwości składa się dźwięk, którego nie słyszą i tym samym będą mogli np. bardziej wczuć się w oglądany film albo rozpoznać, kiedy będąc przy ulicy w mieście, podjeżdża do nich auto.

Po zamienieniu silniczków na wyświetlacz LED, funkcje te zostały trochę ograniczone, ale układ zyskał też nową funkcję - wizualizacja słuchanej muzyki wygląda bardzo estetycznie.

3. Analiza zadania, wybór elementów i użyte narzędzia

Analiza zadania

Algorytm FFT polega na wyciągnięciu z próbki dźwięku składowych częstotliwości i ich głośności. Jest to seria operacji matematycznych wykonywanych rekurencyjnie na próbce dźwięku o wielkości 2^n , gdzie n to liczba naturalna. Najlepsze efekty są dla próbek o wielkości 512 – 16348.

Wczytanie dźwięku może odbywać się na wiele sposobów, jednak z powodu ograniczonej ilości czasu oraz wysokiej komplikacji innych rozwiązań (jak na przykład gniazdo mini-jack), zdecydowałem się użyć tylko modułu mikrofonu.

Przedstawienie wyniku tego algorytmu początkowo miało być na silniczkach wibracyjnych rozłożonych na specjalnej tablicy. Przetestowałem ich działanie i zdecydowałem się z nich nie korzystać już po skonstruowaniu większości projektu, więc jedyne co mi zostało, to zamienić je na LED. Nie wymagają one tyle prądu ile układ jest w stanie im zapewnić, ale działają poprawnie.

Wybór elementów

Do realizacji algorytmu i sterowaniem całym układem wybrałem procesor **ESP32** w devkicie **WROOM V1**, ponieważ pracowałem z jego dokumentacją w projekcie z poprzedniego semestru i czułem się z nim stosunkowo pewnie. Poza tym nie jest drogi oraz oferuje raczej wysoką moc obliczeniową. Dodatkowo devkit posiada wbudowaną przetwornicę $5V \rightarrow 3.3V$ oraz gwarantuje poprawną współpracę procesora z komputerem. Przy wszystkich testach nie udało mi się jednak użyć większej próbki dźwięku od 1024 i zachować płynności obrazu (dla większych próbek obraz był mocno opóźniony).

Do wczytania dźwięku wybrałem tani moduł mikrofonowy z wzmacniaczem LM393.

Do wyświetlenia wyniku wykorzystuję świecące diody sterowane rejestrami przesuwными **TPIC6C595**, dlatego że te rejestry potrafią przyjąć na pin 100mA (na cały rejestr maksymalnie 250mA), co pozwala na podłączenie do nich elementów wymagających niemałego natężenia prądu, jak np. silniczki wibracyjne lub trochę mocniejsze LED.

Układ może być zasilany poprzez kabel USB podłączony do dewkita, który zawiera przetwornicę 5V na 3.3V. Drugą opcją jest zasilanie bateryjne (6 baterii AAA), które wymaga przetwornicy step down <9V na 5V. Na tą przetwornicę zdecydowałem się wybrać **Pololu S9V11F3S5**, ponieważ dla napięcia tych baterii ma stosunkowo wysoką wydajność.

Użyte narzędzia

Schemat elektryczny oraz płytkę PCB zostały wykonane w programie *Eagle* z licencją studencką.

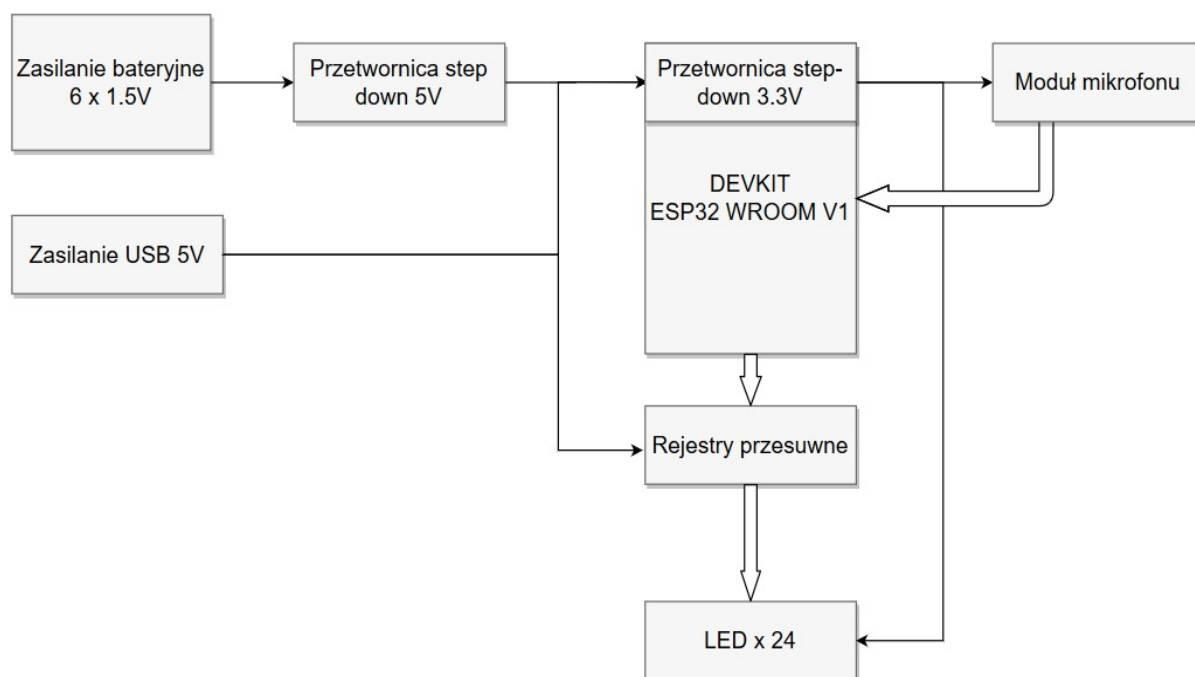
Program pisany był w *Arduino IDE*, darmowym środowisku programistycznym, ponieważ jest łatwy w użyciu, bardzo popularny, przez co ma aktywne fora oraz wiele bibliotek do obsługi różnych elementów, oraz dlatego że mam doświadczenie w korzystaniu z niego.

Płytkę PCB została wykonana w Chinach przez firmę JLCPCB.

Do lutowania wykorzystano tanią stację do lutowania o nieznanym marce.

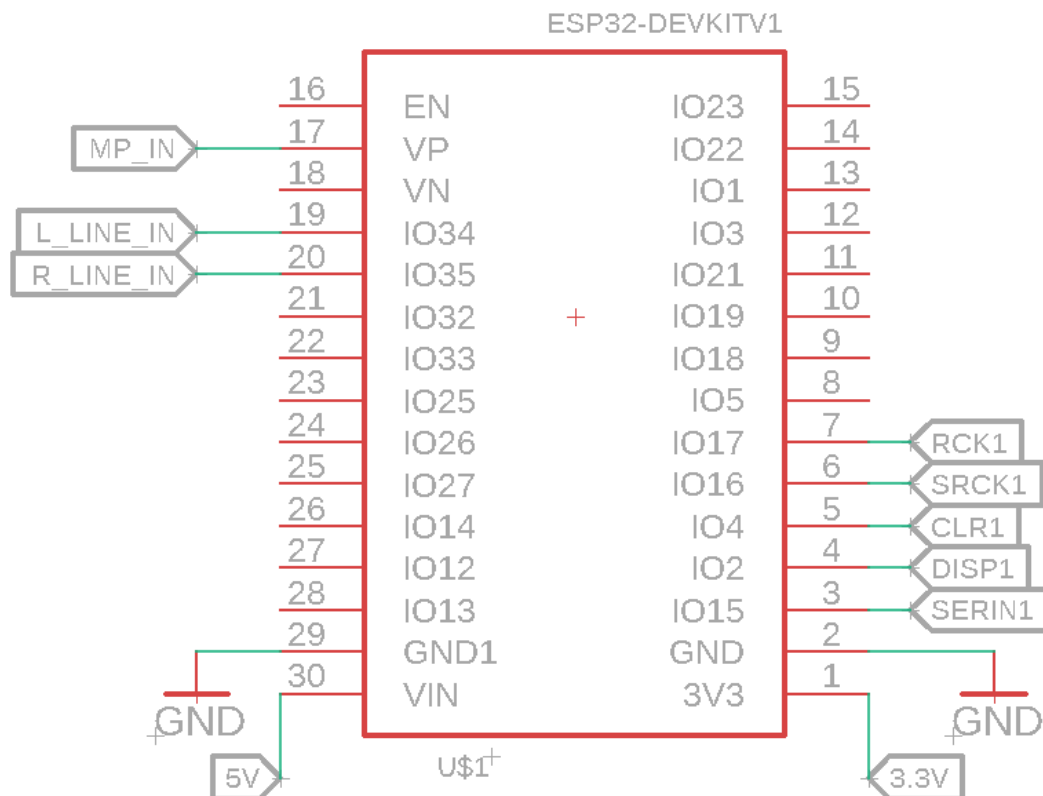
3. Specyfikacja wewnętrzna

a) Schemat blokowy i ideowy



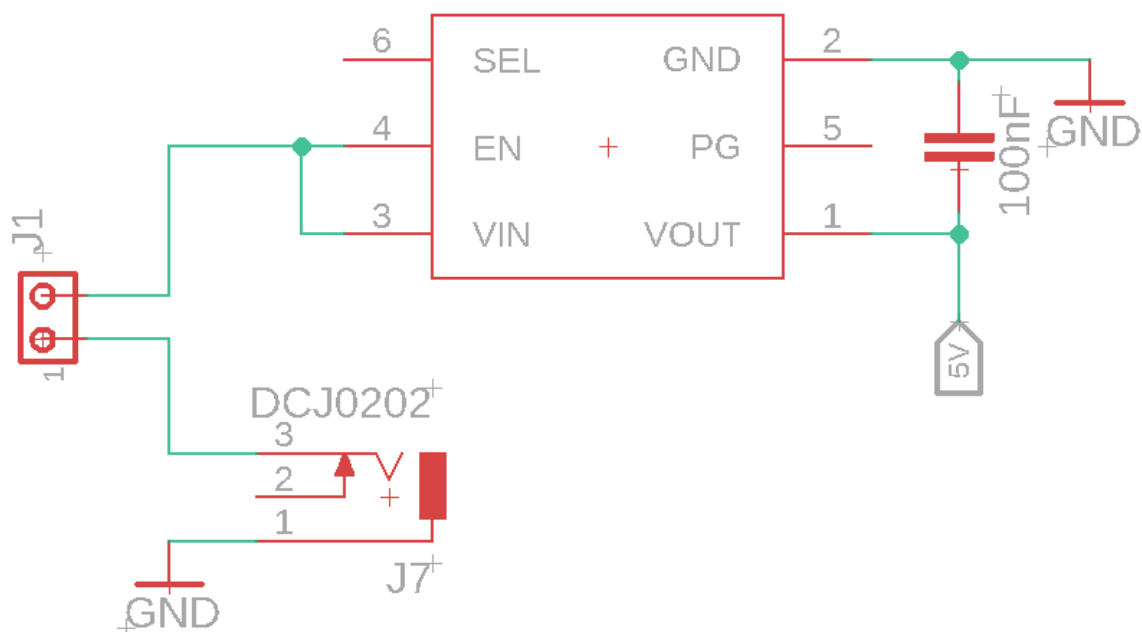
Schemat ideowy

- Procesor

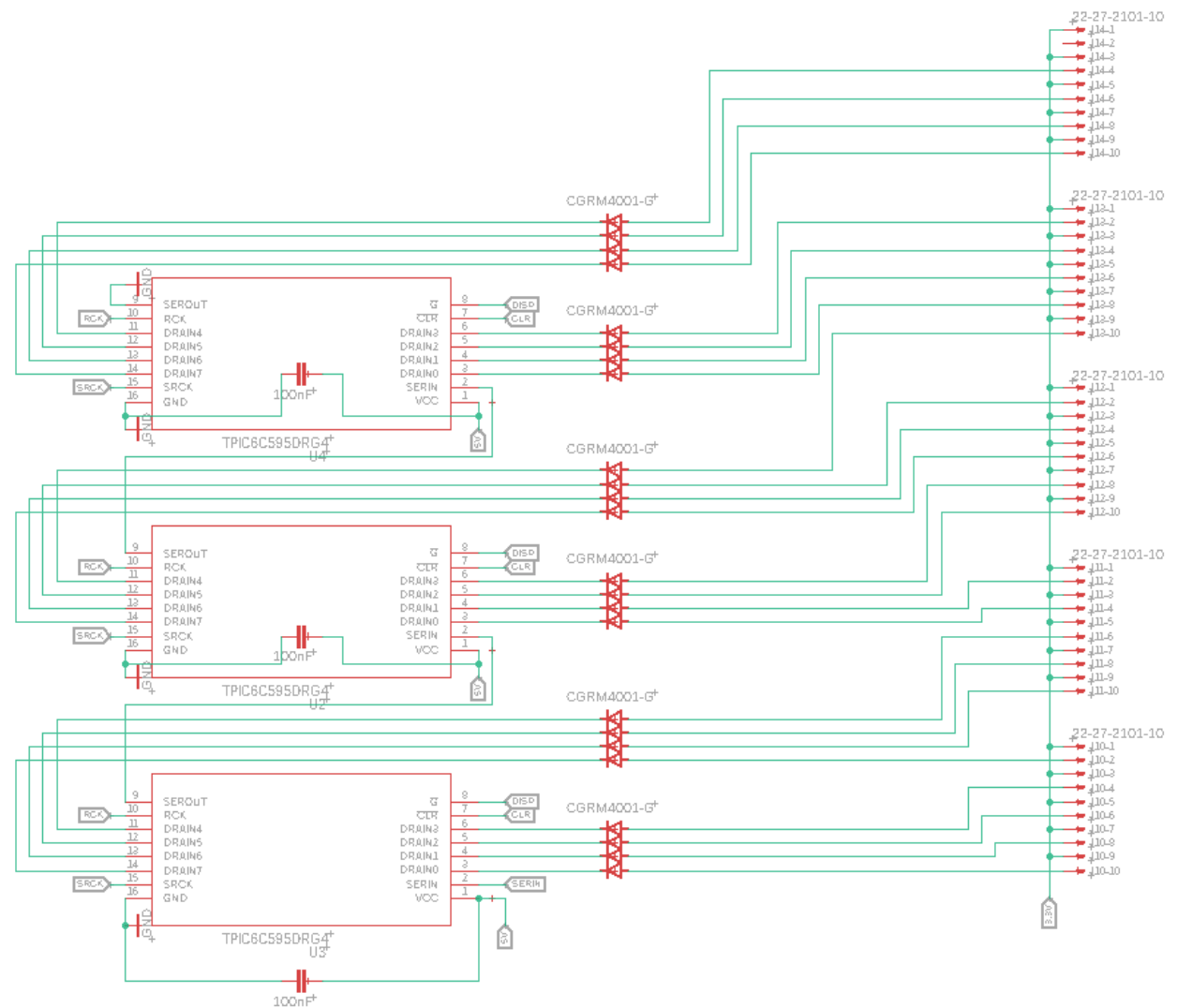


Za zadanie ma sterowanie całym układem, w tym wczytywanie próbki dźwięku z mikrofonu, wykonywanie algorytmu FFT, przetwarzanie wyniku i wysłanie go do wyświetlacza LED.

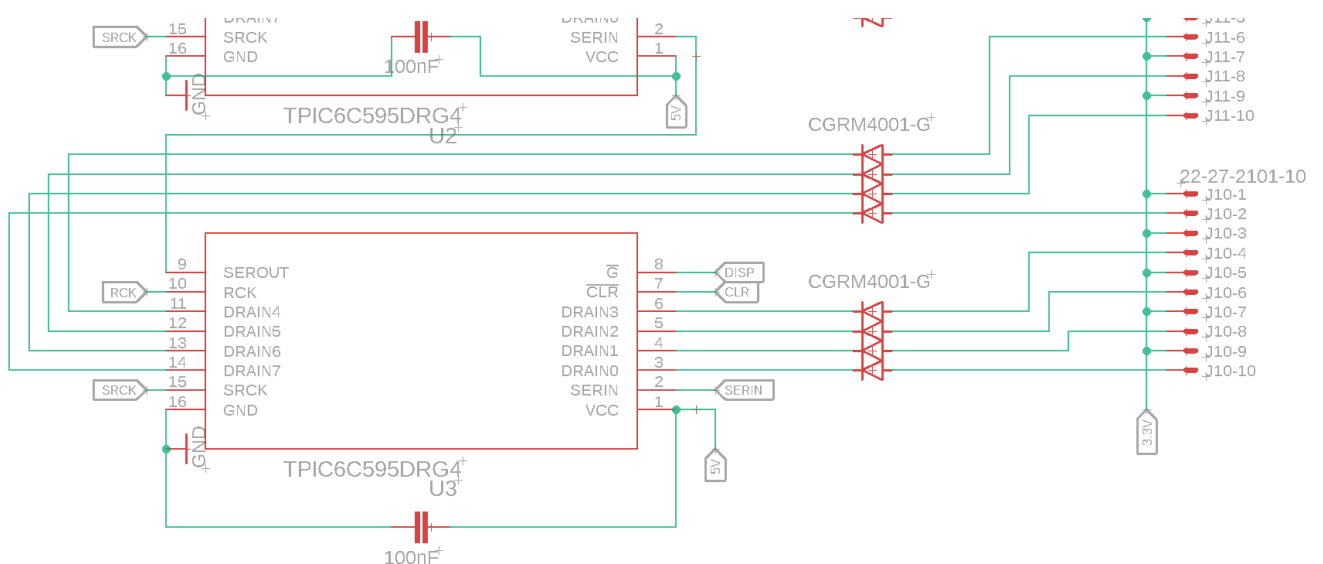
- Zasilanie bateryjne – gniazdo oraz przetwornica



Za zadanie ma stabilizację napięcia z koszyka baterii 6x1.5V na 5V. Dodatkowo znajduje się tu złącze J1 dla ewentualnego przełącznika, który włączał / wyłączał zasilanie bateryjne. Pin SEL jest na module specjalnie zlutowany z innym elementem, co wymusza na przetwornicy tryb step-down (w przeciwnym wypadku działałaby jako step-up). Pin PG wysyła informacje o jakości napięcia, której nie potrzebuję, więc zgodnie z instrukcją, zostawiam nóżkę niepodłączoną.



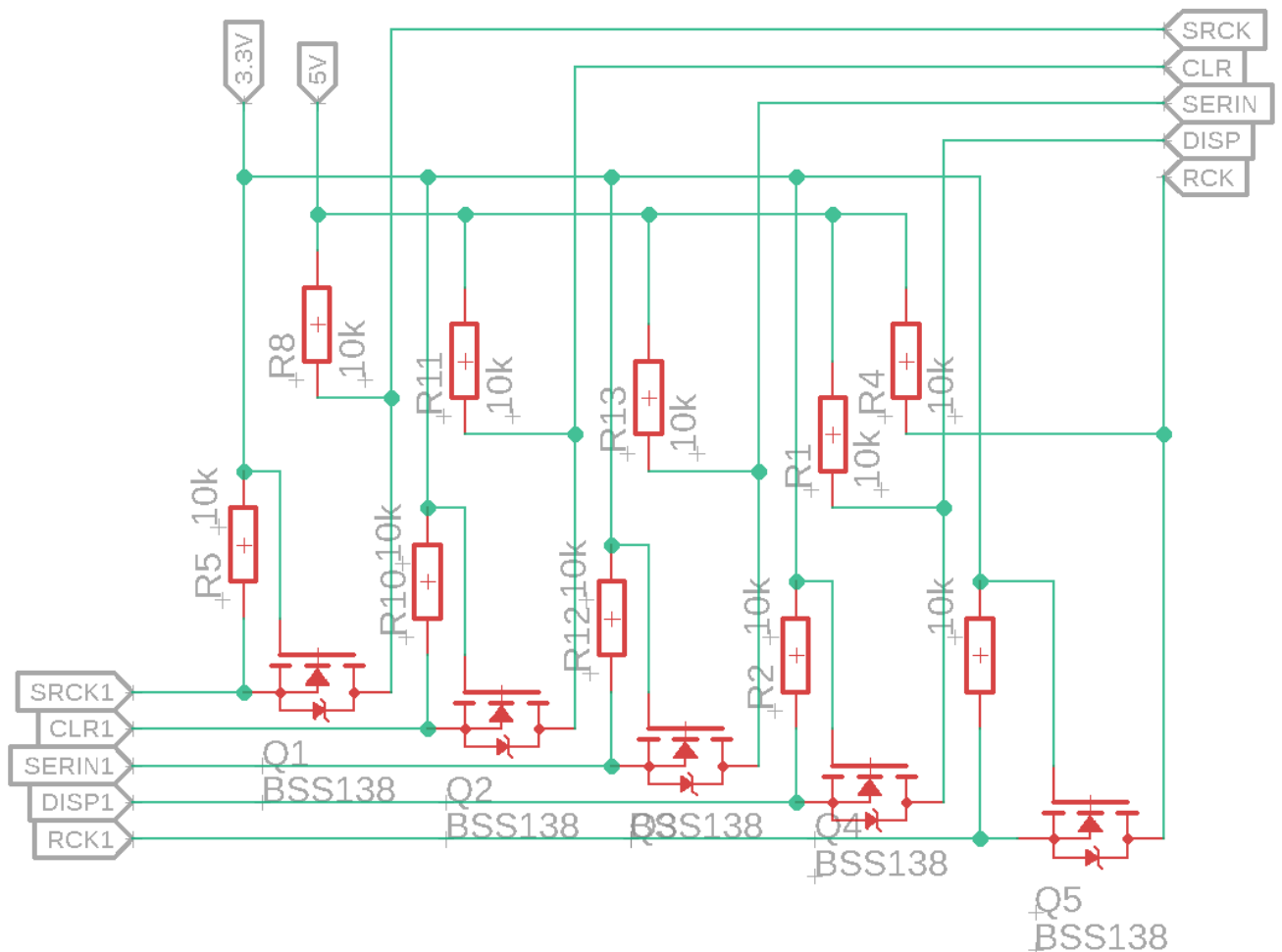
- Pojedynczy rejestr



Podpięte do rejestrów urządzenie (w tym wypadku LED) może pobierać do ~30mA, więc można operować nawet małymi silniczkami wibracyjnymi.

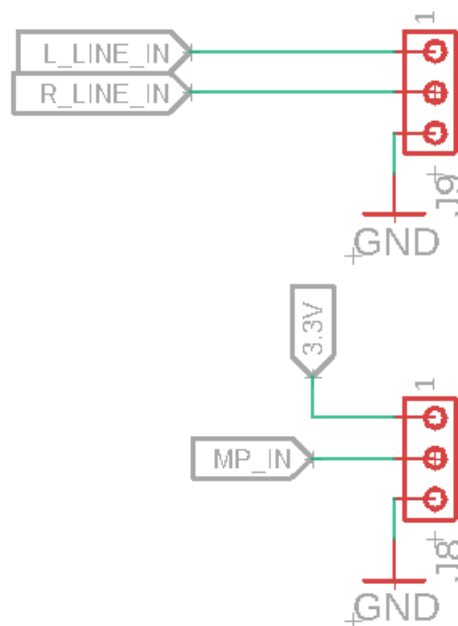
Szeregowo połączone 3 rejestry mogą sterować 24 urządzeniami.

- Pull-up sygnałów do rejestrów



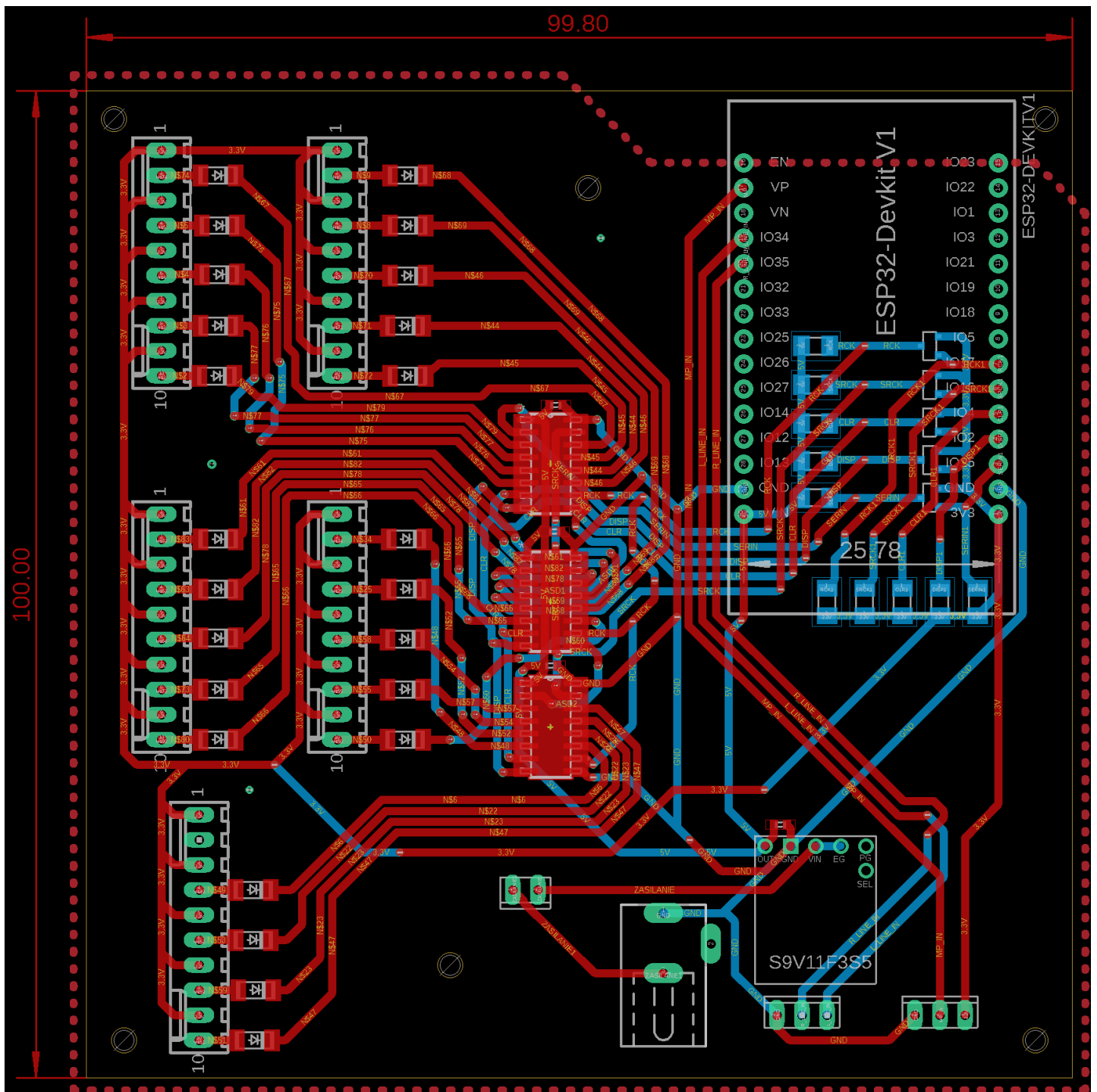
Rejestry są sterowane sygnałami 5V, a procesor nadaje do 3.3V. Pull-up logiczne zamieniają sygnał 3.3V na 5V.

- Podłączenie modułu mikrofonu i modułu mini jack

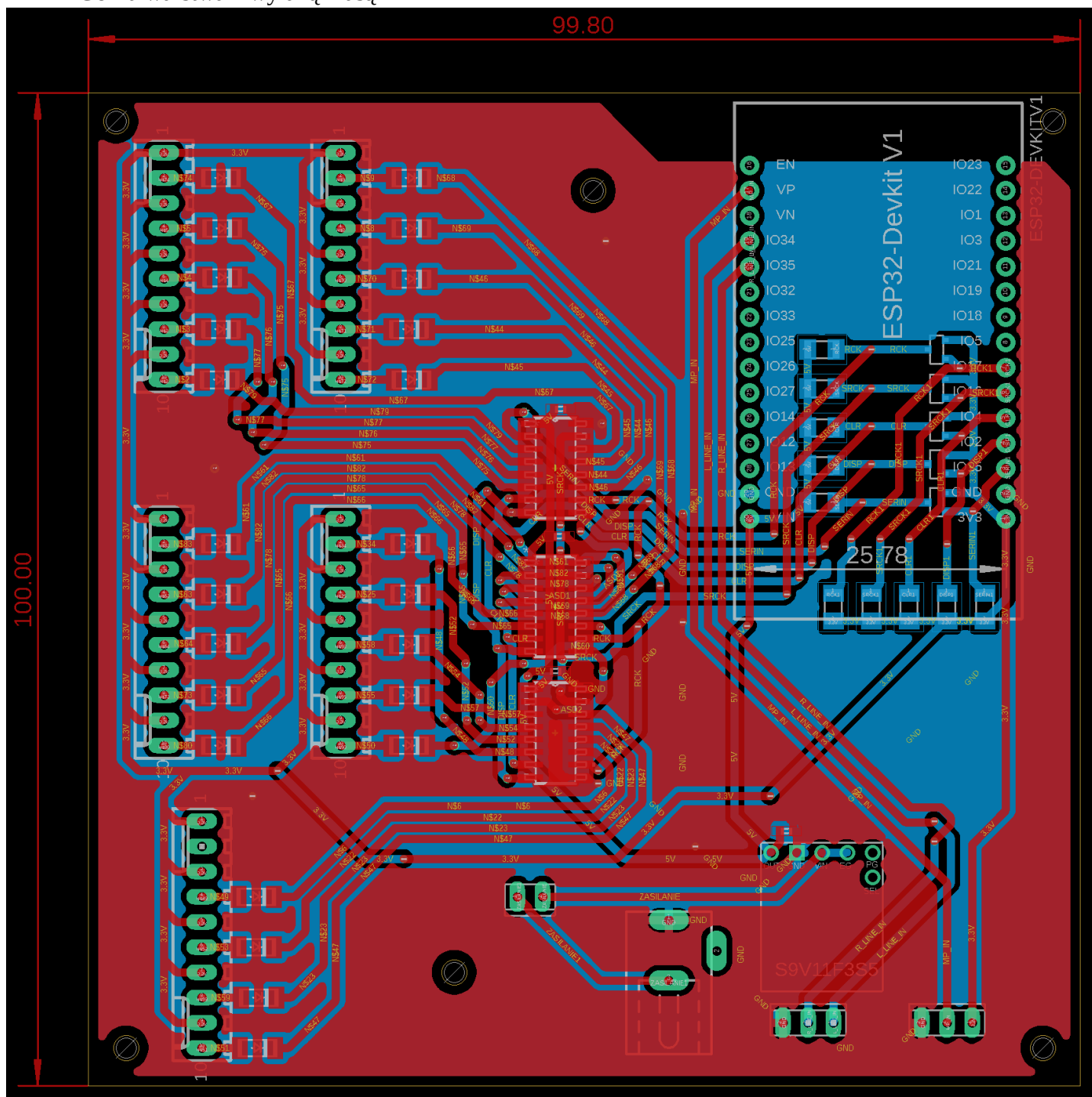


Moduł mikrofonu wymaga napięcia 3.3V, a nadaje sygnał analogowy linią MP_IN. Modułu mini jack nie udało mi się ani znaleźć gotowego do kupna, ani zrobić swojego. Gniazdo zostaje niepodłączone.

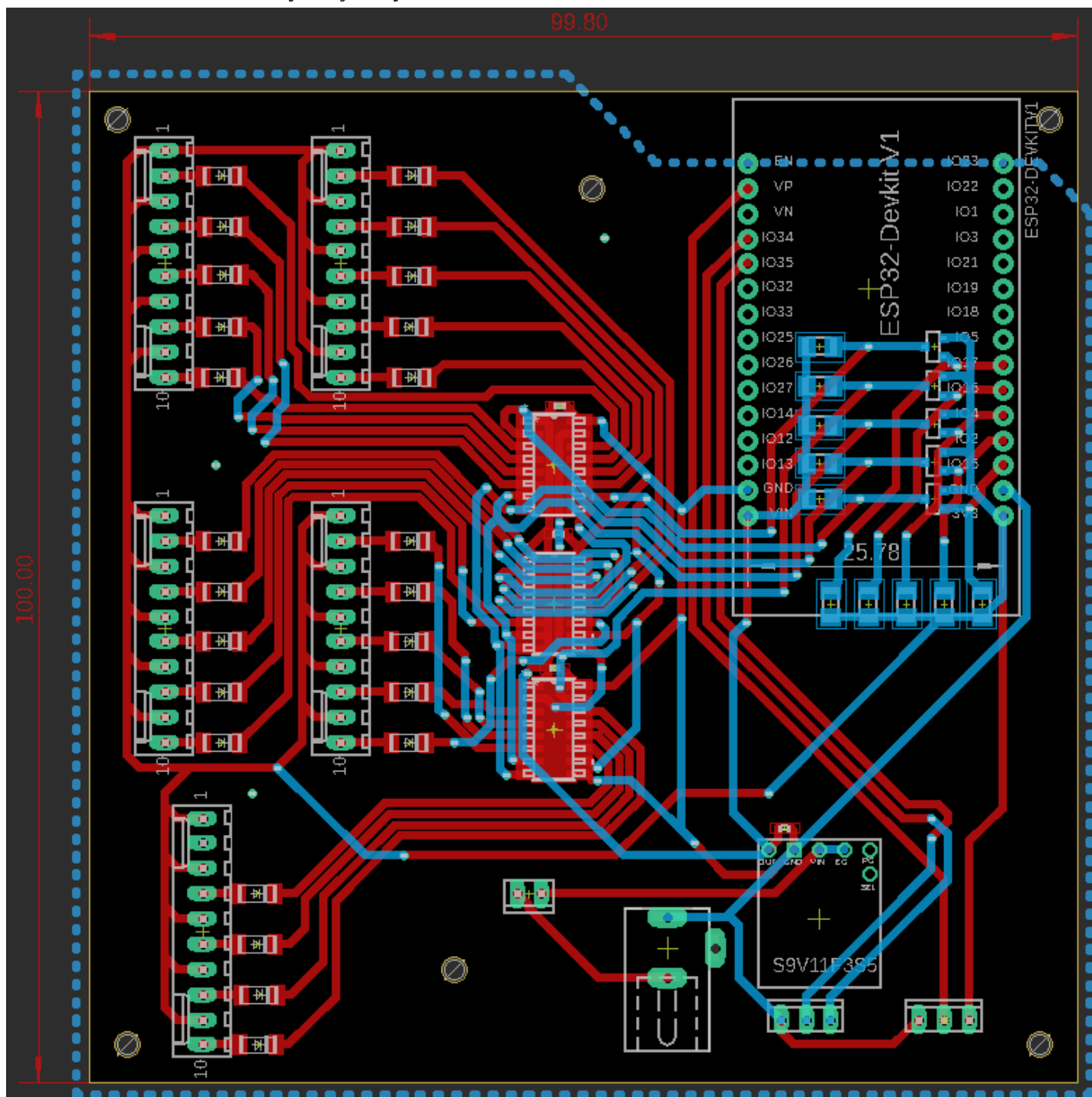
Górna warstwa bez wylanej masy



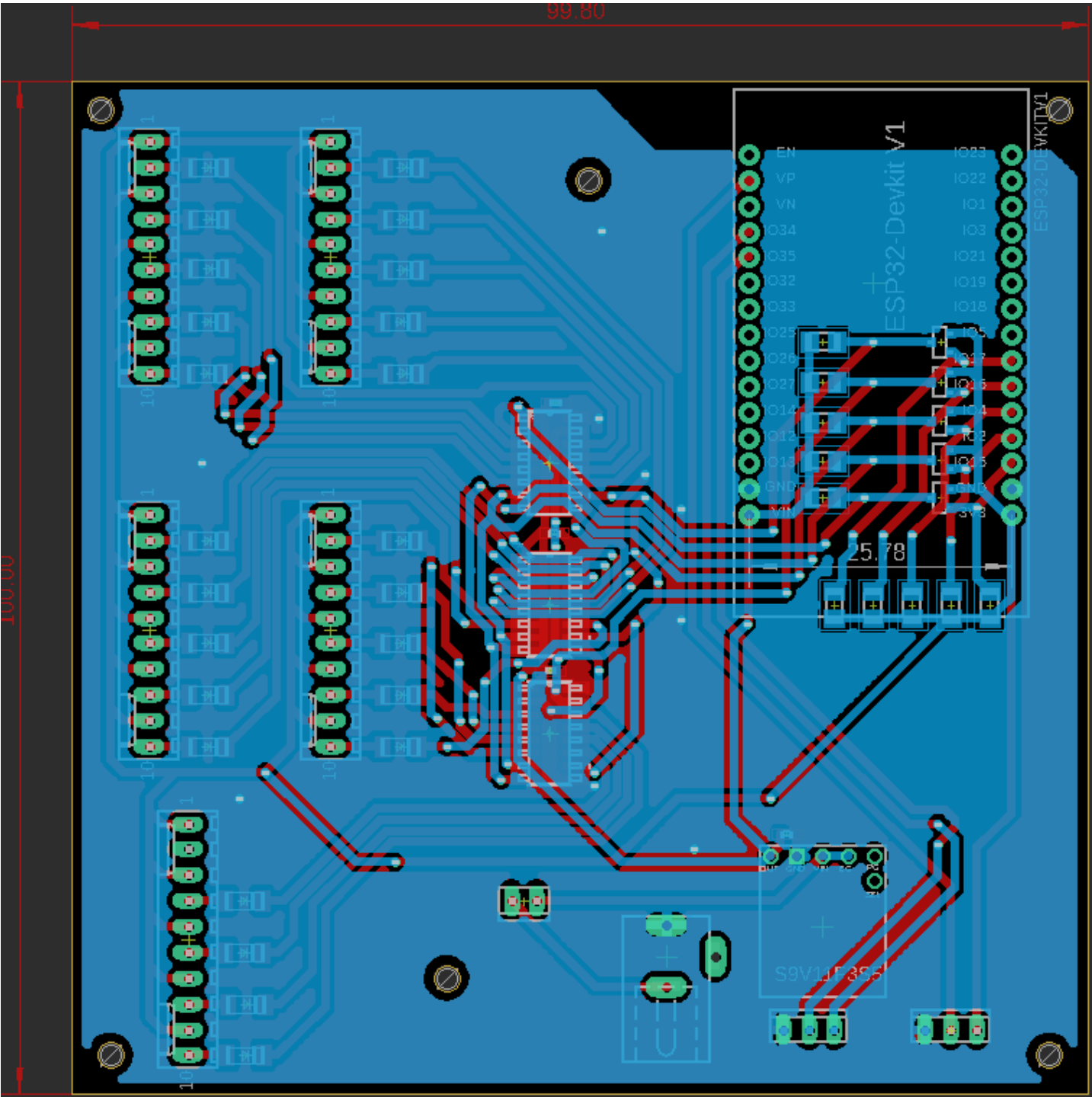
Górna warstwa z wylaną masą



Dolna warstwa bez wylanej masy



Dolna warstwa z wylaną masą



d) Algorytm oprogramowania urządzenia

Przebieg działania programu

- Wczytaj dźwięk w postaci próbki
- Wykonaj FFT na zebranej próbce
- Na podstawie wyniku FFT odczytaj natężenie konkretnych zakresów częstotliwości
- Wynik FFT przetransformuj tak, aby dało się go przedstawić na urządzeniach wyjściowych
- Powtarzaj ad inf

e) Opis wszystkich zmiennych.

f) Opis funkcji wszystkich procedur.

Znajdują się w kodzie w formie komentarzy

kod

```
#include <Wire.h>
#include "arduinoFFT.h" // Standardowa biblioteka FFT dla arduino
arduinoFFT FFT = arduinoFFT();
#define SAMPLES 1024 // wielkość próbki
#define SAMPLING_FREQUENCY 40000 // częstotliwość pobierania próbki
#define amplitude 200 // amplituda dopasowana do mikrofonu

unsigned int sampling_period_us; // jak długo czekać przed pobraniem kolejnej wartości
próbkki

double vReal[SAMPLES]; // tablica przechowująca najpierw wartości próbki dźwięku, a
// następnie realną część wyniku FFT
double vImag[SAMPLES]; // tablica przechowująca urojoną część wyniku FFT

unsigned long newTime, oldTime; // zmienne służące do odmierzenia czasu

byte peak[] = { 0,0,0,0,0,0,0,0 }; // przechowuje wysokość kolumny danego zakresu
//częstotliwości

int displayTab[8][3]; // tablica pośrednia przy translacji
// peak → translatedDisplayTab

bool translatedDisplayTab[24]; // tablica gotowa do wysłania do urządzeń wyjściowych

//piny odpowiedzialne za sterowanie rejestrami przesuwnymi
int SRCK_pin = 25;
int SERIN_pin = 14;
int DISP_pin = 27;
int CLR_pin = 26;
int RCK_pin = 33;

////////////////////////////////////
void setup() {
  Serial.begin(115200);
  sampling_period_us = round(1000000 * (1.0 / SAMPLING_FREQUENCY));

  pinMode(SRCK_pin, OUTPUT);
  pinMode(DISP_pin, OUTPUT);
  pinMode(SERIN_pin, OUTPUT);
  pinMode(CLR_pin, OUTPUT);
  pinMode(RCK_pin, OUTPUT);
  peak[0] = 0;
  peak[1] = 0;
  peak[2] = 0;
  peak[3] = 0;
  peak[4] = 0;
  peak[5] = 0;
  peak[6] = 0;
  peak[7] = 0;
```

```

digitalWrite(CLR_pin, HIGH);
digitalWrite(DISP_pin, HIGH);
digitalWrite(SRCK_pin, LOW);
digitalWrite(RCK_pin, LOW);
}

void loop() {
    for (int i = 0; i < SAMPLES; i++) {
        newTime = micros() - oldTime;
        oldTime = newTime;
        vReal[i] = analogRead(A0); // Konwersja ADC trwa około 1uS na ESP32
        vImag[i] = 0;
        while (micros() < (newTime + sampling_period_us)) {} // poczekaj przed wczytaniem
                                                                // kolejnej próbki
    }

    //przemnożenie przez maskę (w tym wypadku okna Hamminga)
    FFT.Windowing(vReal, SAMPLES, FFT_WIN_TYP_HAMMING, FFT_FORWARD);
    //wykonaj FFT
    FFT.Compute(vReal, vImag, SAMPLES, FFT_FORWARD);
    //oblicz wartość absolutną liczby złożonej (vReal[i], vImag[i]) i zapisz do vReal[i]
    FFT.ComplexToMagnitude(vReal, vImag, SAMPLES);

    for (int i = 2; i < (SAMPLES / 2); i++) {
        if (vReal[i] > 2000) { // prymitywny filtr szumu
            // Ręcznie dopasowane zakresy częstotliwości
            if (i <= 5 && i > 3) checkPeak(0, (int)vReal[i] / amplitude); // ~125Hz
            if (i > 5 && i <= 12) checkPeak(1, (int)vReal[i] / amplitude); // ~250Hz
            if (i > 12 && i <= 14) checkPeak(2, (int)vReal[i] / amplitude); // ~500Hz
            if (i > 14 && i <= 16) checkPeak(3, (int)vReal[i] / amplitude); // ~1000Hz
            if (i > 16 && i <= 20) checkPeak(4, (int)vReal[i] / amplitude); // ~2000Hz
            if (i > 20 && i <= 24) checkPeak(5, (int)vReal[i] / amplitude); // ~4000Hz
            if (i > 24 && i <= 28) checkPeak(6, (int)vReal[i] / amplitude); // ~8000Hz
            if (i > 28 && i <= 46) checkPeak(7, (int)vReal[i] / amplitude); // ~16000Hz
        }
    }

    translate(); // zamień tablicę peak na ciąg zer i jedynek odpowiadających
                // urządzeniom wyjściowym

    displayTable(); // prześlij wynik na urządzenia wyjściowe

    // zresetuj wierzchołki
    peak[0] = 0;
    peak[1] = 0;
    peak[2] = 0;
    peak[3] = 0;
    peak[4] = 0;
    peak[5] = 0;
    peak[6] = 0;
    peak[7] = 0;
}
//szuka maksymalnego wierzchołka w danym zakresie częstotliwości
void checkPeak(int band, int size) {
    int dmax = 3;
    if (size > dmax) size = dmax;
    if (size > peak[band]) { peak[band] = size; }
}

//zamienia wierzchołki zakresów częstotliwości na tablicę zer i jedynek
//dopasowaną do wyświetlacza
void translate() {
    for (int i = 0; i < 8; i++) {
        displayTab[i][0] = false;
        displayTab[i][1] = false;
        displayTab[i][2] = false;
        switch ((int)(peak[i] * 3 / 50)) {
            case 3: displayTab[i][2] = true;
            case 2: displayTab[i][1] = true;
            case 1: displayTab[i][0] = true;
            case 0:
                break;
        }
    }
}

```

```

        default:
            displayTab[i][0] = false;
            displayTab[i][1] = true;
            displayTab[i][2] = false;
            break;
    }
}

//urządzenia wyjściowe nie są idealnie poukładane, więc trzeba wykonać kolejną
//transformację
translatedDisplayTab[13] = displayTab[0][0];
translatedDisplayTab[24] = displayTab[0][1];
translatedDisplayTab[23] = displayTab[0][2];
translatedDisplayTab[17] = displayTab[1][0];
translatedDisplayTab[19] = displayTab[1][1];
translatedDisplayTab[22] = displayTab[1][2];
translatedDisplayTab[18] = displayTab[2][0];
translatedDisplayTab[20] = displayTab[2][1];
translatedDisplayTab[21] = displayTab[2][2];
translatedDisplayTab[7] = displayTab[3][0];
translatedDisplayTab[1] = displayTab[3][1];
translatedDisplayTab[11] = displayTab[3][2];
translatedDisplayTab[6] = displayTab[4][0];
translatedDisplayTab[2] = displayTab[4][1];
translatedDisplayTab[12] = displayTab[4][2];
translatedDisplayTab[5] = displayTab[5][0];
translatedDisplayTab[3] = displayTab[5][1];
translatedDisplayTab[16] = displayTab[5][2];
translatedDisplayTab[9] = displayTab[6][0];
translatedDisplayTab[4] = displayTab[6][1];
translatedDisplayTab[15] = displayTab[6][2];
translatedDisplayTab[10] = displayTab[7][0];
translatedDisplayTab[8] = displayTab[7][1];
translatedDisplayTab[14] = displayTab[7][2];
}

// wysyła wynik do urządzeń wyjściowych
void displayTable() {
    digitalWrite(DISPLAY_pin, HIGH); //wyłącz output

    for (int i = 0; i < 24; i++) {
        if (translatedDisplayTab[24 - i]) {
            digitalWrite(SERIN_pin, HIGH); // '1' - SERIN = HIGH
        }
        else {
            digitalWrite(SERIN_pin, LOW); // '0' - SERIN = LOW
        }

        digitalWrite(SRCK_pin, HIGH); // rejestr zapisuje SERIN
        digitalWrite(SRCK_pin, LOW); //
    }
    digitalWrite(RCK_pin, HIGH); // zapisane z SERIN wyślij na rejestr
    digitalWrite(RCK_pin, LOW); //

    digitalWrite(DISPLAY_pin, LOW); // włącz output
}

```

g) Opis interakcji oprogramowania z układem elektronicznym

Układ pobiera dane z mikrofonu za pomocą funkcji **analogRead(A0)**. Konwertuje ona aktualne napięcie na pinie analogowym A0 na wartość cyfrową.

Wysyłanie sygnału odbywa się za pomocą funkcji

digitalWrite(numer_pinu, sygnał_logiczny(HIGH/LOW)).

Wystawia ona na konkretnym pinie napięcie 0V lub 3.3V odpowiednio dla LOW i HIGH.

h) Szczegółowy opis ważniejszych procedur

FFT.Windowing(vReal, SAMPLES, FFT_WIN_TYP_HAMMING, FFT_FORWARD);

Funkcja okna czasowego – zwiększa jakość wykonanego później algorytmu FFT poprzez przemnożenie wartości pojemnika z próbką dźwięku przez maskę w tym wypadku okna Hamminga.

FFT.Compute(vReal, vImag, SAMPLES, FFT_FORWARD);

Wykonuje algorytm FFT. Jest to procedura rekurencyjna, która wymaga dość dużo pamięci. Dla płynności działania programu, wielkość próbki została ustawiona na 1024. wartości pojemnika z próbką dźwięku przez maskę w tym wypadku okna Hamminga.

FFT.ComplexToMagnitude(vReal, vImag, SAMPLES);

Oblicza wartość absolutną liczby złożonej (vReal[i], vImag[i]) i zapisuje ją do vReal[i].

4. Specyfikacja zewnętrzna

a) Opis funkcji elementów sterujących urządzeniem

Urządzenie sterowane jest wyłącznie źródłem dźwięku, w tym wypadku mikrofonem.

b) Opis funkcji elementów wykonawczych

Wynik algorytmu FFT wyświetlany jest na wyświetlaczu 8x3 skonstruowanym z 24 LED. Jest to 8 kolumn po 3 rzędy, każda kolumna odpowiadająca konkretnym zakresowi częstotliwości (np. 200 - 300HZ).

Im więcej diód w kolumnie jest zaświeconych, tym większe natężenie danego zakresu częstotliwości w próbce dźwięku.

c) Opis reakcji oprogramowania na zdarzenia zewnętrzne.

Oprogramowanie nie przewiduje żadnego specjalnego zachowania zależnego od zdarzeń zewnętrznych.

d) Skrócona instrukcja obsługi urządzenia.

1. Podłącz zasilanie do urządzenia poprzez USB lub gniazdo zasilania jack.
2. Skieruj mikrofon na źródło dźwięku, które chce się poddać analizie.
3. Obserwuj wynik na wyświetlaczu.

6. Opis montażu i uruchamiania.

a) Jakie problemy wystąpiły podczas montażu i uruchamiania i jak zostały rozwiązane

Podczas montażu okazało się, że schemat zawierał błąd. Jedna z nóżek rejestrów przesuwanych nie została wzięta pod uwagę i obsłużona. Jako że ta nóżka sterowana jest napięciem 5V, zmuszony byłem stworzyć nowy układ logicznego pull-up i dolutować go cienkimi kablami do PCB.

Kiedy testowałem urządzenie woltomierzem w poszukiwaniu błędu (który później okazał się być niepoprawnie przylutowanym rezystorem SMD) zwałem 2 nóżki na devkicie, co spowodowało jego spalenie. Resztę projektu kontynuowałem na pożyczonym devkicie z tym samym procesorem.

b) Jakie przeprowadzono testy poprawności działania urządzenia

Urządzenie testowano głównie na dźwięku tworzonym przez generator częstotliwości oraz „sweep” częstotliwości, czyli płynne przejście częstotliwości dźwięku z niskiej do wysokiej w czasie.

7. Wnioski z uruchamiania i testowania.

Projekt okazał się być o wiele bardziej skomplikowany niż na początku zakładałem. Stworzenie modułu dla gniazda mini-jack, żeby np. pobierać dźwięk bezpośrednio z komputera / telefonu za pośrednictwem kabla, okazało się być trudne ze względu na ograniczenie czasowe oraz małą ilość materiałów znalezionych przeze mnie w internecie (nie potrafiłem też znaleźć gotowego modułu).

Dowiedziałem się również przez testowanie, że chcąc pracować na silniczkach wibracyjnych, muszę wziąć pod uwagę m.in. jak szybko potrafią się przełączać, czego nie łatwo dowiedzieć się ze strony producenta.

Wyświetlacz 8x3 okazał się być ledwo wystarczający do czytelnej reprezentacji wyniku, więc następnym razem chciałbym zaimplementować wyświetlacz minimum 10x10.

Praca nad tym projektem była bardzo satysfakcjonująca, ale presja czasowa wynikająca z mojego wyjazdu na projekt Erasmus+ jeszcze zanim w Polsce zacznie się sesja spowodowała, że nie mogłem skupić się na ulepszaniu projektu, a jedynie na jego zaliczeniu.

Ukończenie tego projektu uświadomiło mnie, jak wiele różnych urządzeń może zrobić człowiek, jeśli poświęci trochę czasu i zasobów.