KoNaR

KOŁO NAUKOWE ROBOTYKÓW

STEROWNIK LED RGB – OŚWIETLENIE DOHYO

(Sprawozdanie końcowe)

Adrian Ciż,

Marek Gulanowski,

Maciej Trojnar

Wrocław 2009

Spis treści

Streszczenie

Poniżej zaprezentowano projekt realizowany w ramach Koła Naukowego Robotyków „KoNaR", polegający na wykonaniu sterownika do 24 diod RGB. Celem projektu była możliwość generowania atrakcyjnych efektów świetlnych podczas zawodów robotów minisumo. Sprawozdanie zawiera opis prac nad projektem, zastosowanych rozwiązań, schematy układów elektronicznych oraz zdjęcia wykonanych płytek drukowanych. Opisano także napotkane problemy wraz ze znalezionymi rozwiązaniami.

1 Wstęp

Celem realizacji projektu była budowa urządzenia zapewniającego efekty świetlne w czasie zawodów robotów minisumo.

Projekt realizowany był od listopada 2008 do marca 2009 przez trzech studentów II roku Automatyki i Robotyki na Wydziale Elektroniki: Adriana Ciża, Marka Gulanowskiego i Macieja Trojnara.

Wykonano układ elektroniczny, do którego podłączono 24 diody RGB, oraz stworzono oprogramowanie sterujące efektami świetlnymi.

Postęp prac nie był zgodny z harmonogramem, ponieważ projekt początkowo miał zostać ukończony w grudniu – wystąpiły zatem 3 miesiące opóźnienia.

2 Opis realizacji projektu

2.1 Punkt wyjścia

Był to dla nas pierwszy projekt realizowany w Kole Naukowym Robotyków KoNaR. Pomysł stworzenia oświetlenia dohyo na potrzeby zawodów minisumo został przedstawiony przez zarząd Koła na zebraniu rekrutacyjnym, na którym zapisaliśmy się do Koła. Punktem wyjścia dla projektu był zatem przede wszystkim nasz brak doświadczenia, który w sposób zdecydowany, a często dość zabawny, wpłynął na przebieg jego realizacji.

Sensownym wydaje się podanie w tym miejscu naszego zasobu umiejętności, z którym rozpoczynaliśmy prace nad projektem.

- Projektowanie układów elektronicznych: niewielkie doświadczenie.
- Produkcja płytek drukowanych: brak jakichkolwiek umiejętności i doświadczenia.
- Programowanie mikrokontrolerów AVR: niewielkie doświadczenie.

Realizacja projektu dała nam zatem przede wszystkim możliwość poznania (często na zasadzie pierwszego kontaktu) metod radzenia sobie z powyższymi problemami.

2.2 Projektowanie układu elektronicznego

Zalążek koncepcji projektu układu sterownika wynikł bezpośrednio z podpowiedzianej nam przez pomysłodawców projektu idei działania urządzenia: diody sterowane sygnałem PWM realizowanym sprzętowo w mikrokontrolerze, przekazywanym do diod za pomocą demultiplekserów. Słysząc te wskazówki, nie wiedziałem ani czym jest PWM, ani co robi demultiplekser – co chyba trafnie oddaje naszą początkową sytuację.

Pierwszym wyzwaniem było podjęcie decyzji, jakie zastosować elementy. Jako mikrokontroler wybraliśmy AVR ATmega8, ponieważ na nim realizowaliśmy już kiedyś

proste układy. Trochę czasu zajęło nam zastanowienie się, z jakich skorzystać demultiplekserów (ile mają mieć kanałów). Ostatecznie ze względu na dostępność wybraliśmy model 74HC138 – 8-kanałowy. Należało zastosować ich 9, jako że potrzebowaliśmy łącznie 72 kanałów.

2.2.1 Ogólny opis działania układu

Mikrokontroler ATmega8 posiada 3 kanały PWM. Umożliwiają one wysyłanie sygnału prostokątnego o modulowanej szerokości impulsu. Sygnał PWM o odpowiednio wysokiej częstotliwości umożliwia regulowanie jasności świecenia diody. W przypadku diod RGB sterowanie jasnością 3 barw podstawowych umożliwia mieszanie ich w dowolnych proporcjach, a zatem uzyskanie dowolnej barwy światła. Do jednej diody RGB należy zatem doprowadzić 3 kanały PWM.

Aby obsłużyć 24 diody RGB, należało sygnał każdej barwy podstawowej wysłać kolejno do każdej z diod. Przy odpowiednio dużej częstotliwości przełączania ludzkie oko nie jest w stanie dostrzec, że w danym momencie świeci tylko jedna dioda i występuje iluzja ciągłej pracy wszystkich diod.

Istotnym elementem koncepcji działania układu była reakcja na dźwięk – efekty miały być powiązane z odtwarzaną muzyką.

2.2.2 Podział układu

Układ został podzielony na następujące części:

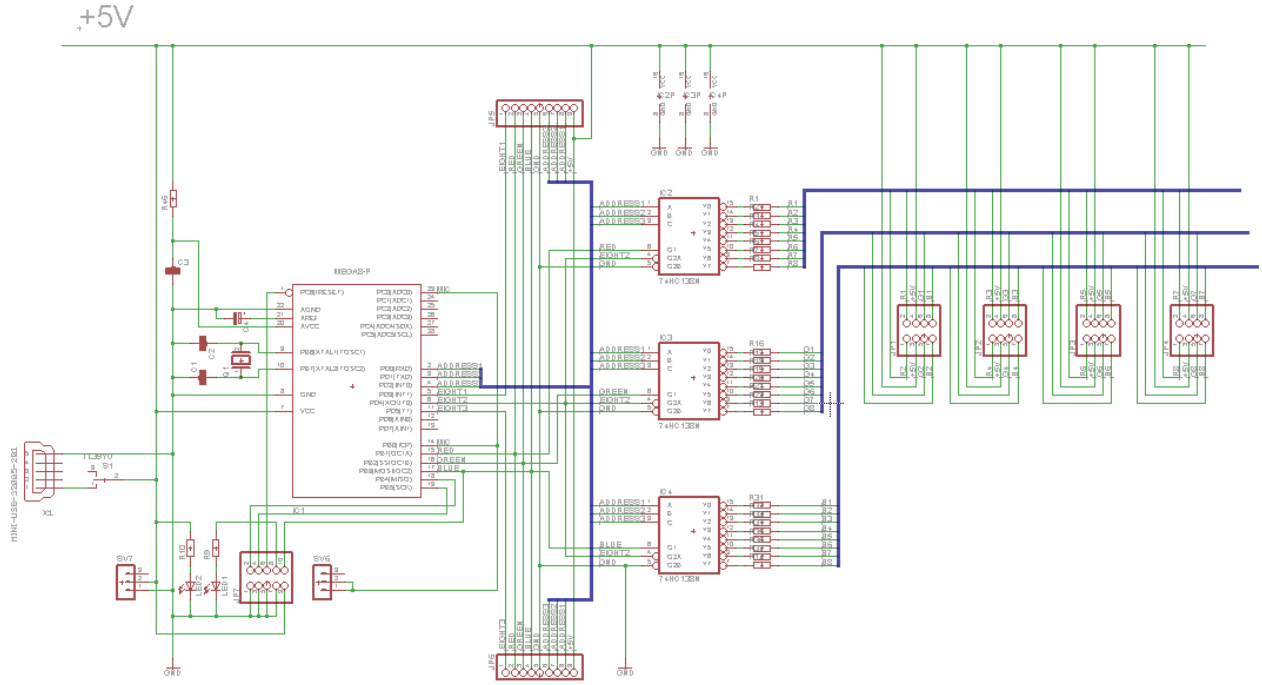
- moduł główny sterujący 8 diodami, zawierający mikrokontroler;
- dwa identyczne moduły boczne, sterujące 8 diodami każdy;
- moduł dźwiękowy.

2.2.3 Schematy i opis działania poszczególnych modułów

Poniżej przedstawiono schematy układu elektronicznego, utworzone w programie EAGLE.

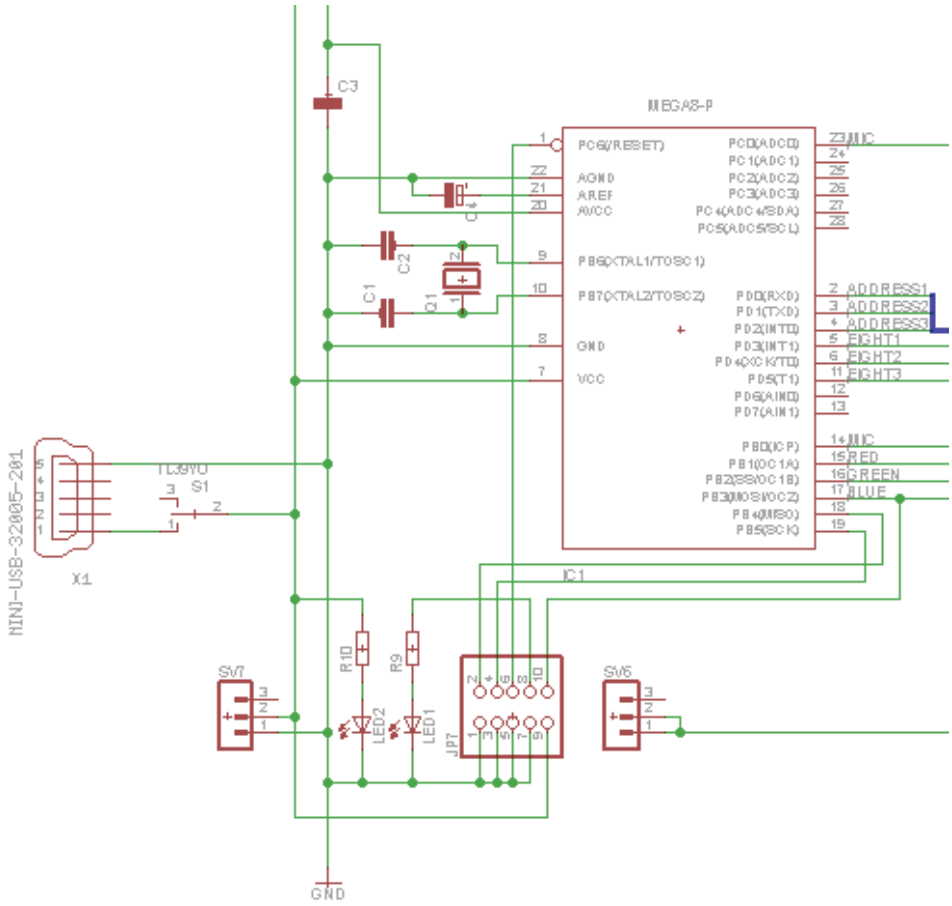
2.2.3.1 Moduł główny

Pierwszy schemat ukazuje moduł główny: mikrokontroler oraz demultipleksery z sygnałami wyjściowymi podłączonymi do zewnętrznych wyprowadzeń oraz złącza do modułów bocznych.



Schemat 1: Układ elektroniczny: moduł główny

Poniżej ukazano powiększenie fragmentu modułu głównego: mikrokontroler ATmega8 wraz z otoczeniem: złącze mini USB (zasilanie), złącza do modułu dźwiękowego, rezonator kwarcowy, kondensatory filtrujące zasilanie.

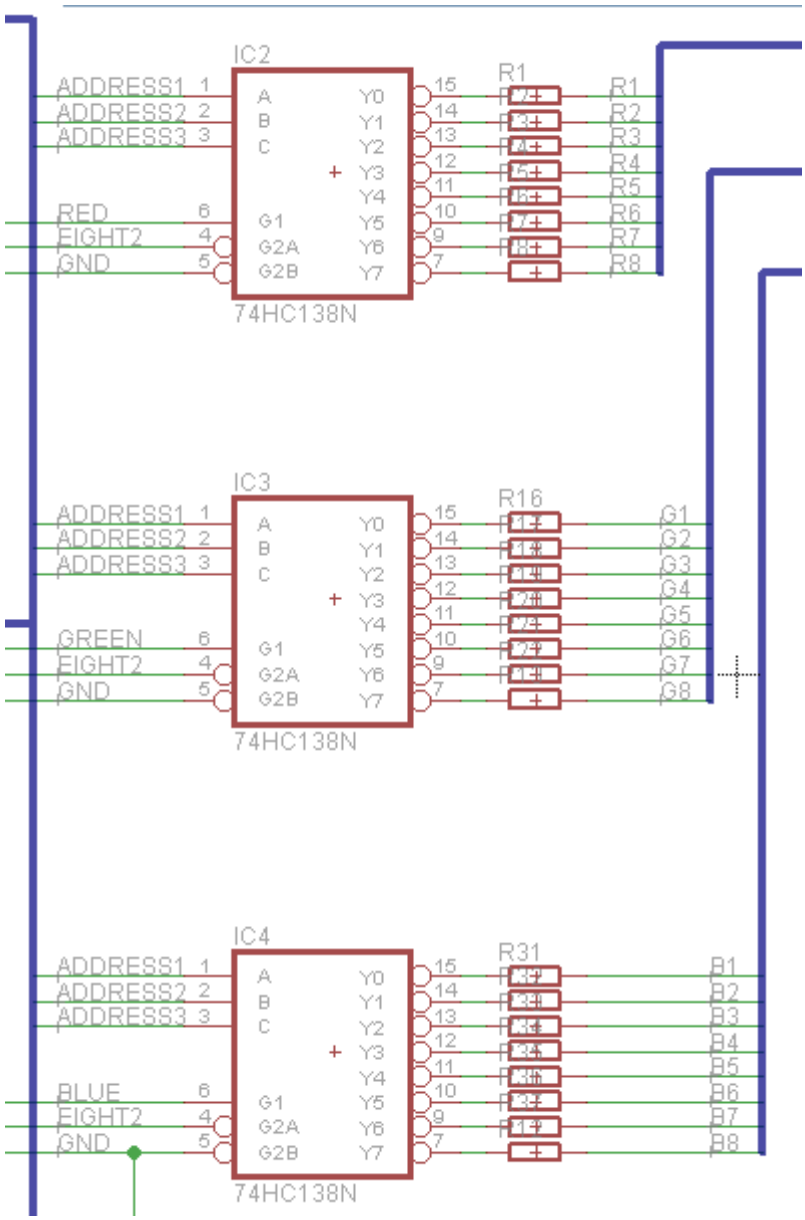


Schemat 2: Układ elektroniczny: Mikrokontroler i otoczenie (moduł główny)

Kolejna część schematu przedstawia 3 z 9 demultiplekserów, obsługujące 8 diod. Do każdego demultipleksera doprowadzone są następujące sygnały:

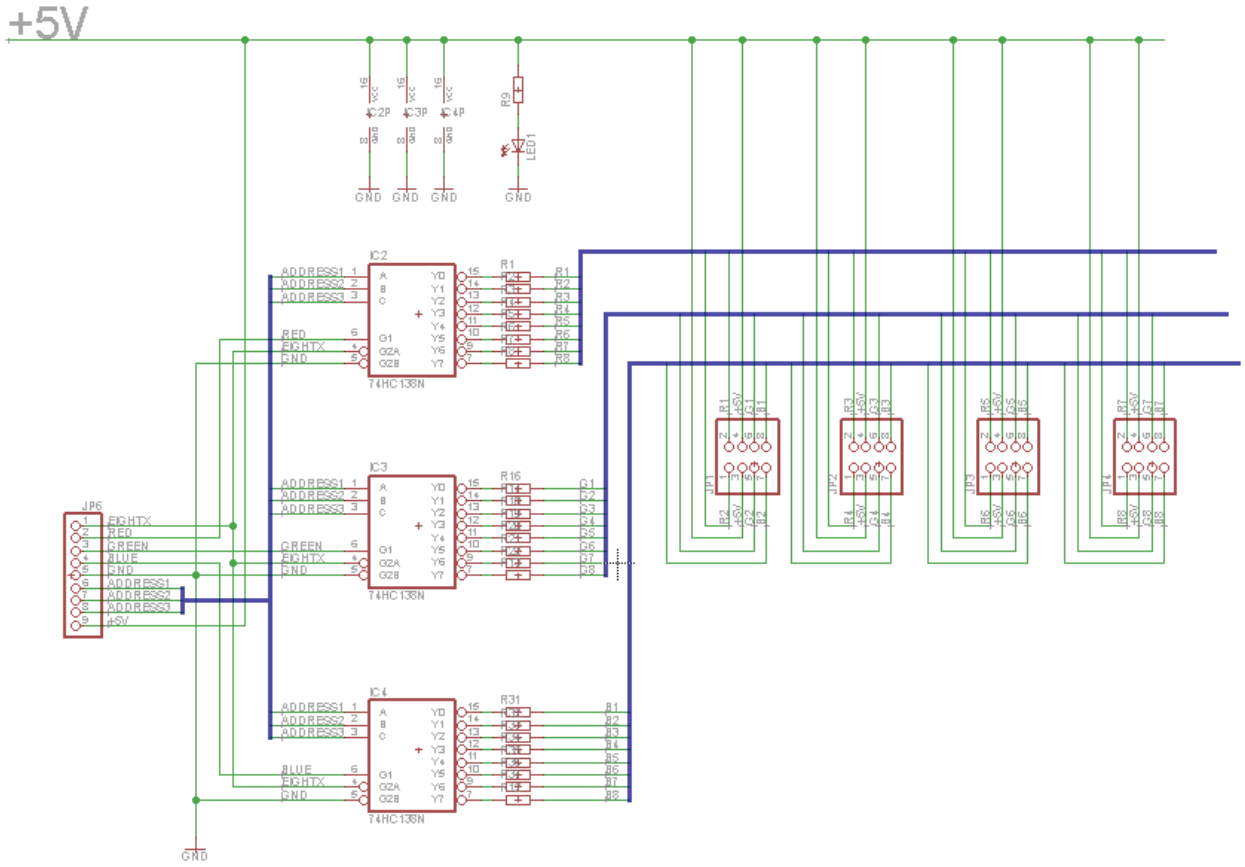
- adresowe A,B,C – służące do wyboru adresu diody (0-7), do której wysyłany jest sygnał PWM;
- sygnał PWM – RED, GREEN, BLUE – dla poszczególnych kolorów;
- EIGHTn - sygnał służący do wyboru, której (n-tej) „ósemki" diod dotyczy aktualny adres.

Wszystkie trzy barwy są zatem wysyłane jednocześnie do tej samej diody.



Schemat 3: Układ elektroniczny: demultipleksery (moduł główny)

2.2.3.2 *Moduły boczne*



Schemat 4: Układ elektroniczny: moduł boczny

2.2.3.3 *Moduł dźwiękowy*

Moduł ten ma za zadanie odebrać sygnał z mikrofonu, następnie odfiltrować i wzmocnić występujące w nim pasmo niskich częstotliwości.

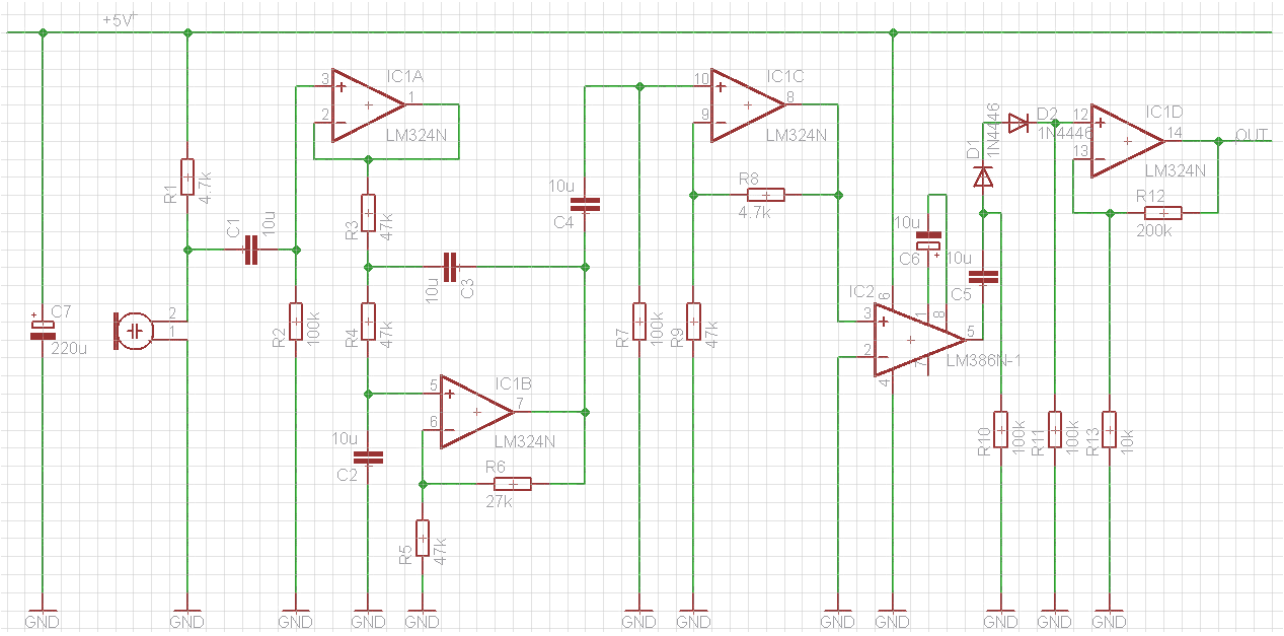
Filtrowanie dolnoprzepustowe, które jest najważniejsze w tym układzie, jest realizowane przez filtr aktywny typu ZNSN o charakterystyce Butterwortha i częstotliwości granicznej $f_g = 350\ Hz$.

Główne wzmocnienie sygnału przeprowadzane jest przez układ LM386, czyli wzmacniacz operacyjny do zastosowań audio. Inne wzmocnienia służą odpowiedniemu wyskalowaniu sygnału.

Na koniec sygnał przepuszczany jest przez dwie diody, co zmienia go w serię impulsów (wydruk z oscyloskopu na Rys. 3) o ustalonym progu i amplitudzie zależnej od natężenia dźwięku, dzięki czemu uzyskany efekt jest lepszy.

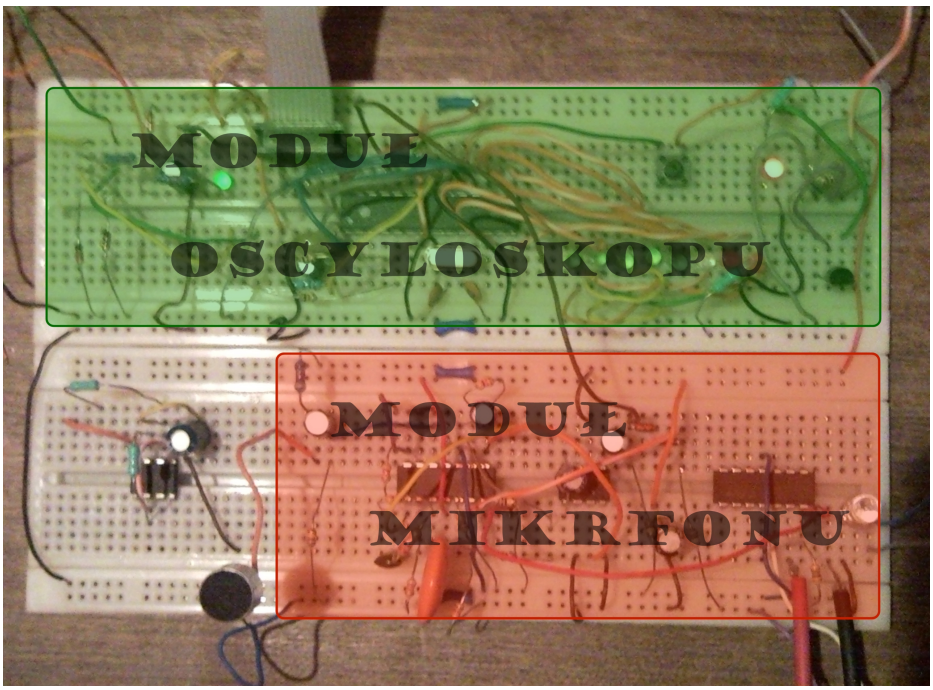
Ostateczne pasmo przenoszenia układu zostało zbadane doświadczalnie i mieści się

w zakresie od 40Hz do 500 Hz.

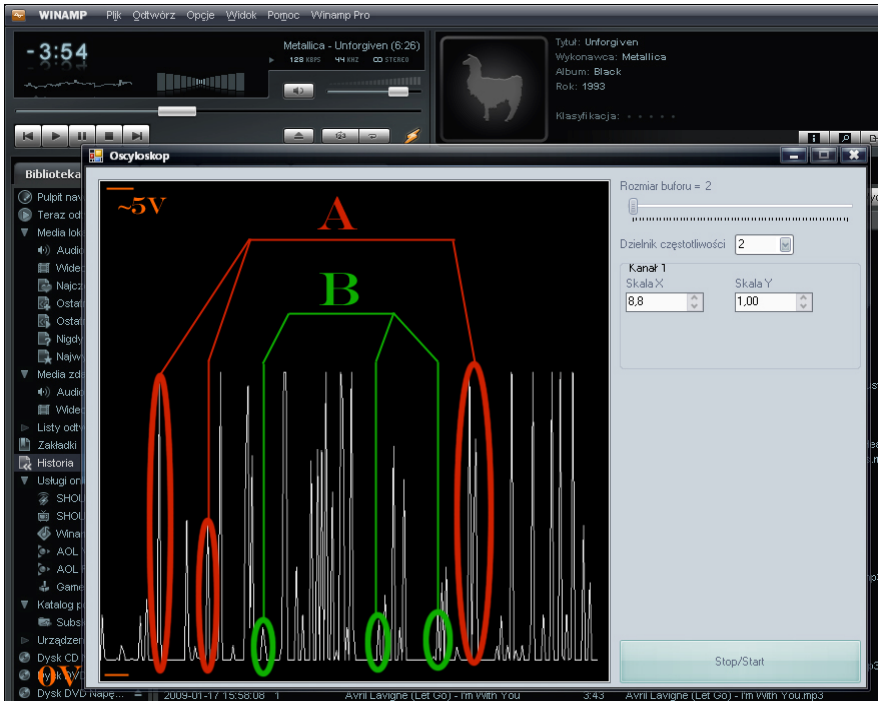


Schemat 5: Układ elektroniczny: moduł dźwiękowy

Jak widać na schemacie (przesuwając się od lewej strony) początkiem układu jest dzielnik napięcia z mikrofonem elektretowym. Sygnał z niego zostaje przepuszczony przez filtr górnoprzepustowy o dolnej częstotliwości granicznej poniżej 1Hz, dzięki czemu zostaje w całości przepuszczony, a jedynie jego składowa stała jest odfiltrowana. Kolejnym segmentem jest wtórnik na wzmacniaczu IC1A, który odciąża poprzednie układy, sygnał z niego jest następnie filtrowany przez wspomniany wcześniej dolnoprzepustowy filtr aktywny o charakterystyce Butterwortha, w którego skład wchodzą IC1B, rezystory R3-R6 oraz kondensatory C2 i C3. Następnym segmentem jest identyczny jak poprzednio filtr górnoprzepustowy z którego sygnał trafia na wzmacniacz nieodwracający o wzmocnieniu Ku=1.1 (IC1C wraz z R8 i R9) i wzmacniacz audio LM386 w konfiguracji, zaczerpniętej z „Typical Applications" z dokumentacji, dającej wzmocnienie Ku=200. Po ponownym przefiltrowaniu górnoprzepustowym i przejściu przez diody ustalające próg ok. 1.2V sygnał trafia na ostatni stopień wzmocnienia o Ku=21, które w ostatecznym projekcie można regulować dzięki zastąpieniu R12 potencjometrem.



Rysunek 1: Zdjęcie układów (w wersji prototypowej) modułu przetwarzania dźwięku i oscyloskopu, który posłużył do wykonania wydruków



Rysunek 2: Wydruk z oscyloskopu A) impulsy o napięciu wystarczającym do zapalenia diody B) impulsy zbyt małe by zaświecić diodę

2.3 Wykonanie płytek drukowanych

2.3.1 Wybór technologii produkcji

Spośród możliwych do zastosowania w warunkach amatorskich technologii produkcji płytek drukowanych wybraliśmy metodę fotolitografii. Jest ona dobrze opisana np. w dostępnych oficjalnie w internecie artykułach z czasopisma *Elektronika praktyczna*.

Technika ta polega na wykonaniu maski poprzez wydrukowanie na przezroczystej folii na czarno ścieżek, które mają pozostać na płytce. Następnie spryskuje się płytkę sprayem światłoczułym np. *Positiv 20*, suszy, przykrywa maską i naświetla. Po naświetleniu należy obraz na płytce wywołać w odpowiednim roztworze. Wtedy płytkę można wytrawić.

2.3.2 Projektowanie i wykonanie płytek

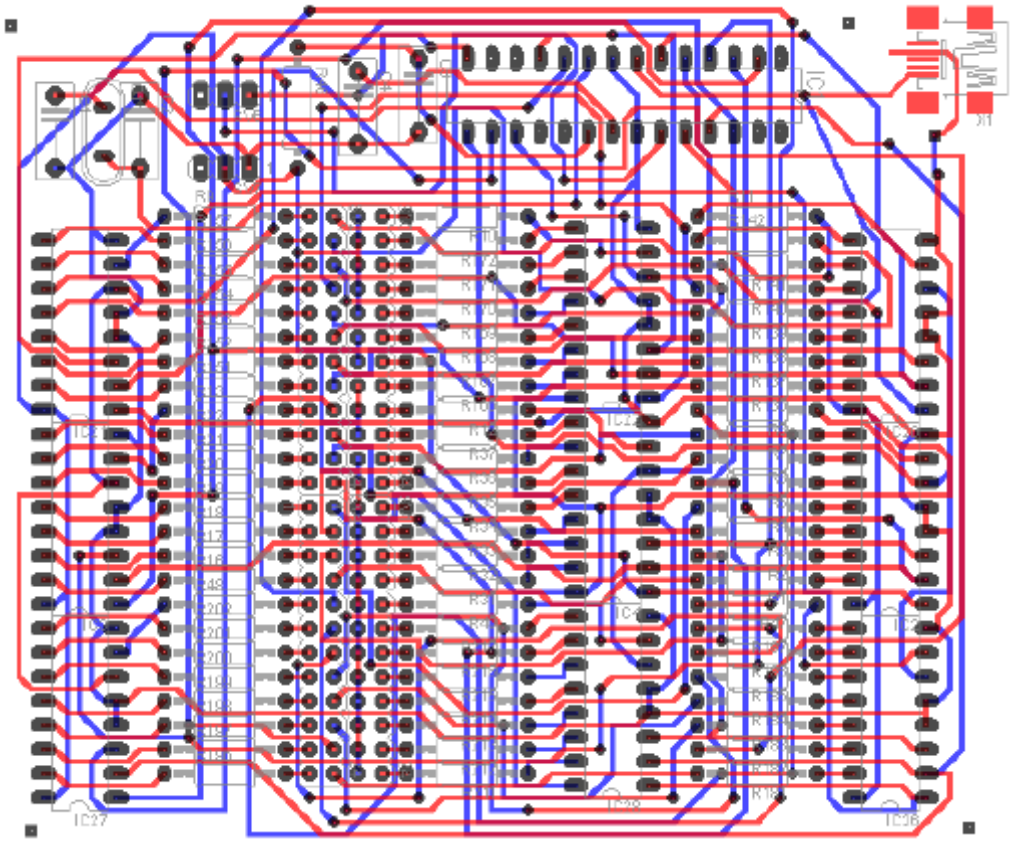
Płytki projektowaliśmy razem ze schematami układów w programie EAGLE.

2.3.2.1 Pierwsze podejście

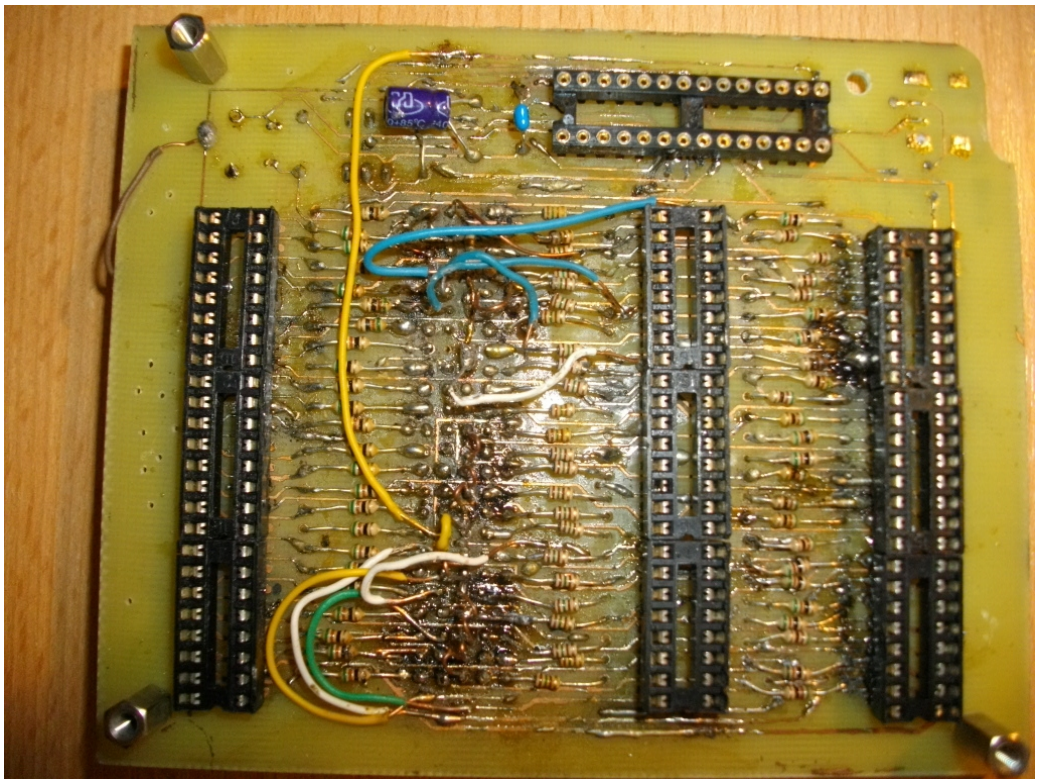
Była to pierwsza zaprojektowana przez nas płytka. **Nigdy w pełni nie zadziałała**. Spędziliśmy wiele dni i nocy na jej projektowaniu trawieniu i próbie lutowania. Oto lista błędów, które spowodowały, że płytka ta nie mogła zadziałać:

- zbyt cienkie ścieżki ze zbyt małymi odstępami – bardzo trudno je wytrawić tak, żeby ani się nie pozwierały, ani nie nadtrawiły, a potem nie odpadły przy lutowaniu;
- pola lutownicze umieszczone po niewłaściwej stronie (błąd który kosztował nas – dosłownie – wiele dni daremnej pracy) – tzn. umieszczone po tej samej stronie co elementy, uniemożliwiając właściwe przylutowanie podstawek pod układy scalone oraz pinów.

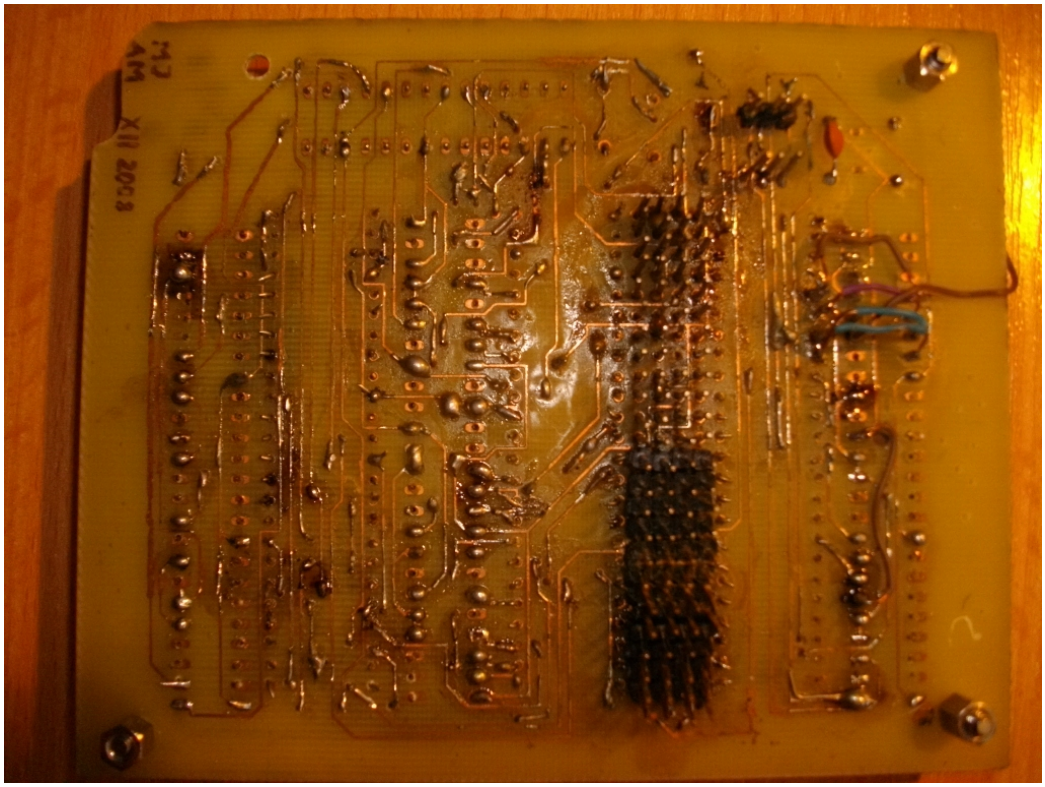
Jak widać, błędy ujawniły się dopiero na etapie lutowania, czyli wtedy, kiedy nie można ich już było naprawić. Staraliśmy się zatem „jakoś” zlutować cały układ, np. tam, gdzie zbyt cienkie ścieżki urywały się, wstawialiśmy kabelki. Gdy wszystko było już teoretycznie gotowe, okazało się, że płytka nie działa. Zamieszczone poniżej zdjęcia nie pozostawiają wątpliwości, dlaczego tak się stało.



Schemat 6: Pierwsza wersja płytki



Zdjęcie 1: Pierwsza płytka: góra

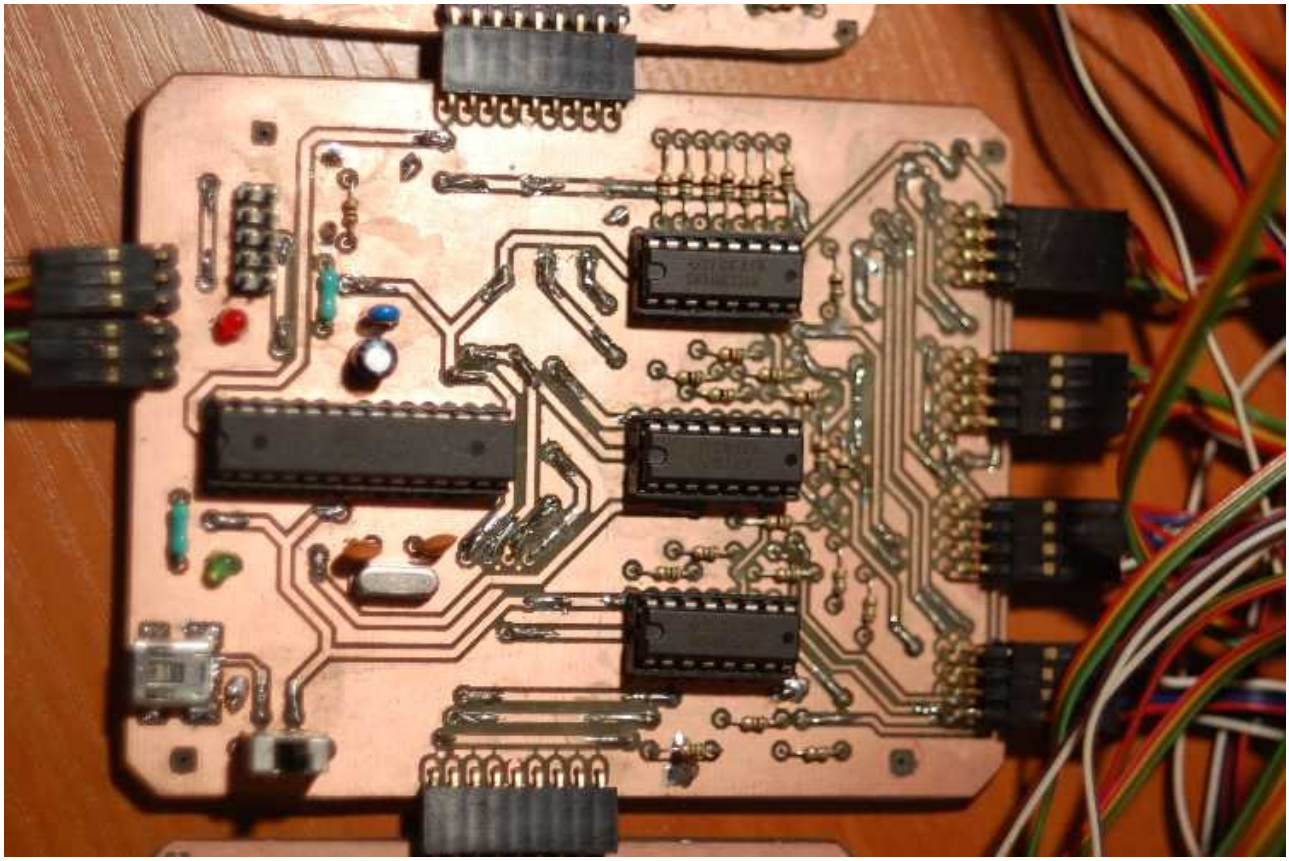


Zdjęcie 2: Pierwsza płytka: dół

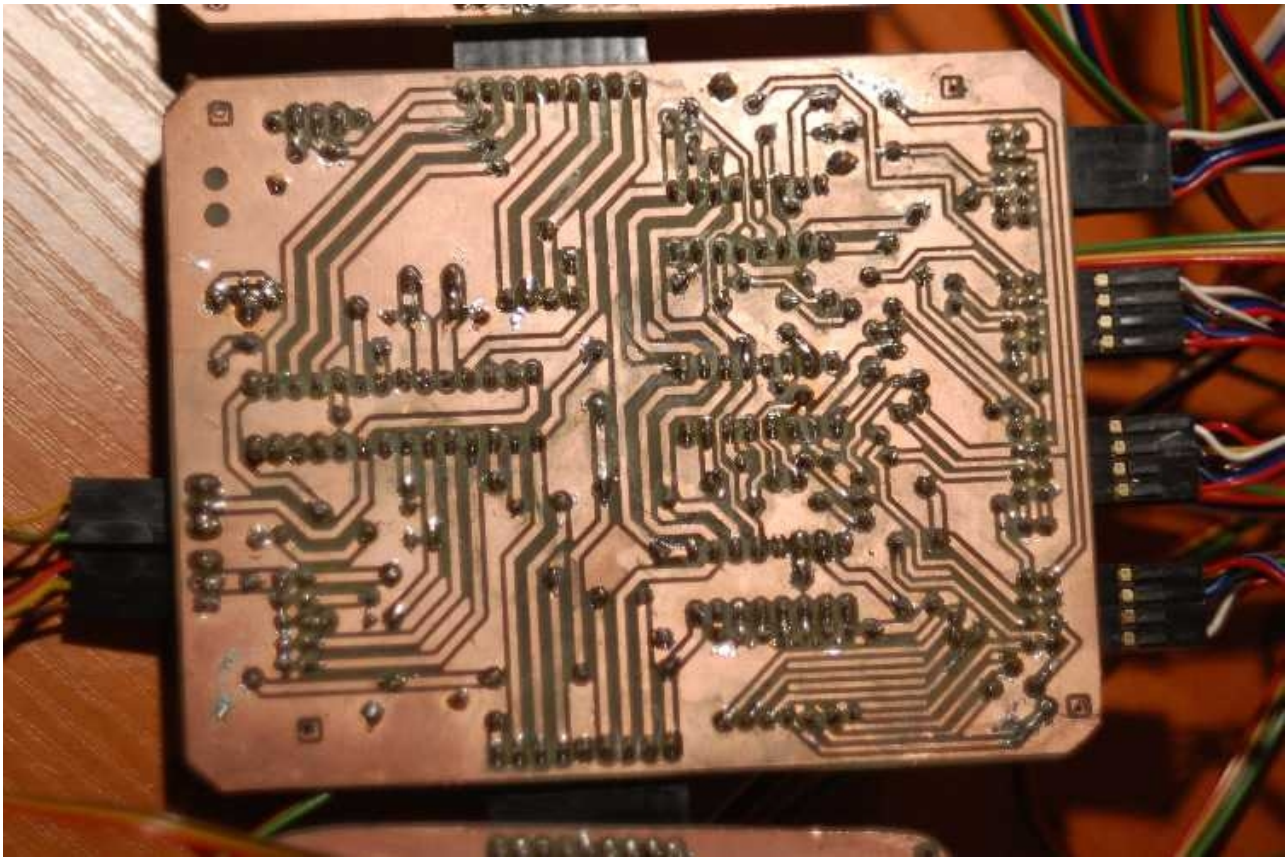
2.3.2.2 Drugie podejście

Podjęliśmy zatem trudną decyzję (byliśmy już ponad miesiąc po terminie), że należy stworzyć nową płytkę, biorąc pod uwagę nasze zdobyte właśnie w bólach doświadczenie. Oto, jakie przyjęliśmy założenia:

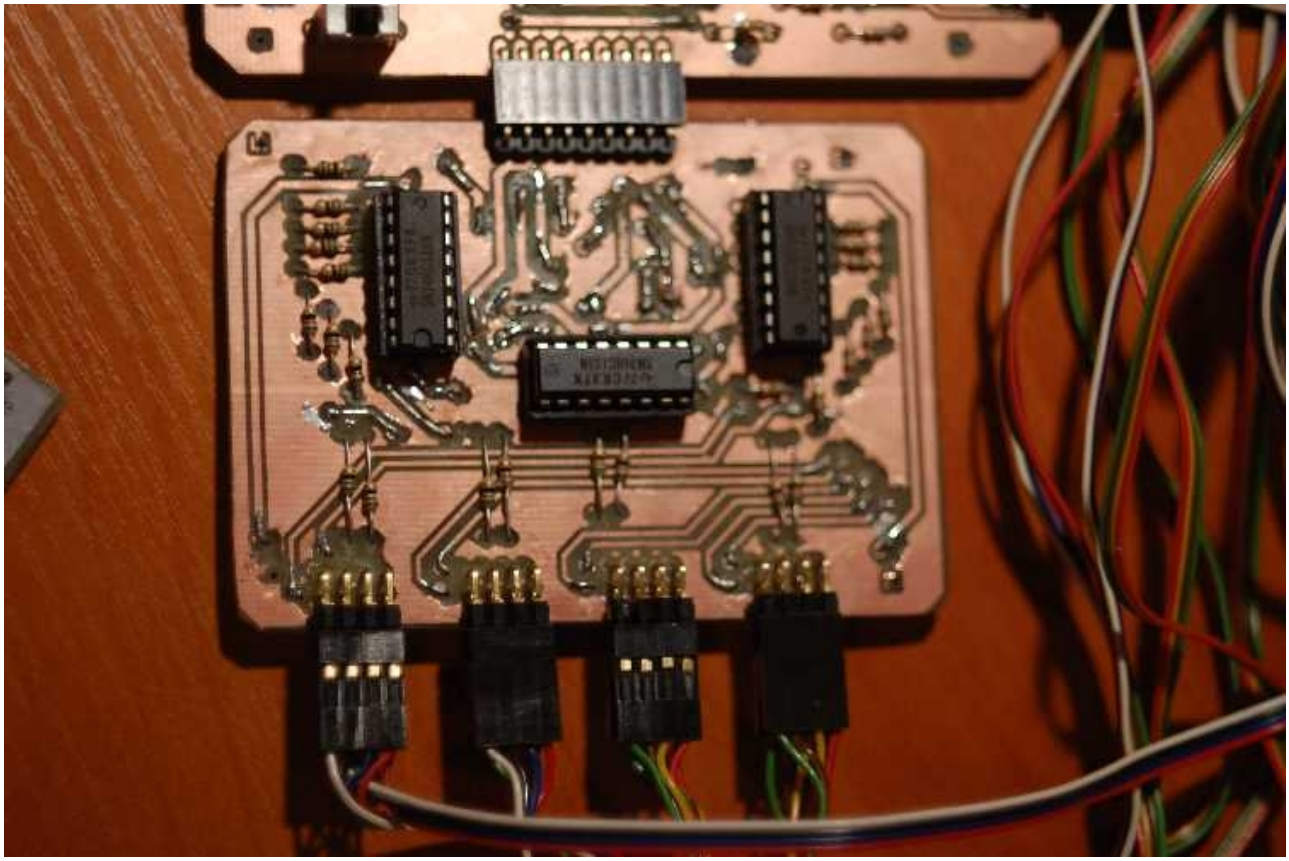
- Modułowa budowa układu. Zalety: kilka osób może jednocześnie lutować oddzielne moduły, w przypadku błędu zniszczona zostaje tylko część układu, płytki są mniej skomplikowane, a więc czytelniejsze.
- Szerokie, odpowiednio oddalone od siebie ścieżki, szczególnie te dostarczające zasilanie do układów scalonych. „Rozlana masa" - mniej zakłóceń, a także oszczędność wytrawiacza i krótszy czas trawienia, zapobiegający nadtrawieniu ścieżek.
- Pola lutownicze tylko po jednej stronie – przeciwnej do elementów. Zapewniło to łatwość lutowania (choć kilka pól przez pomyłkę znalazło się po niewłaściwej stronie).



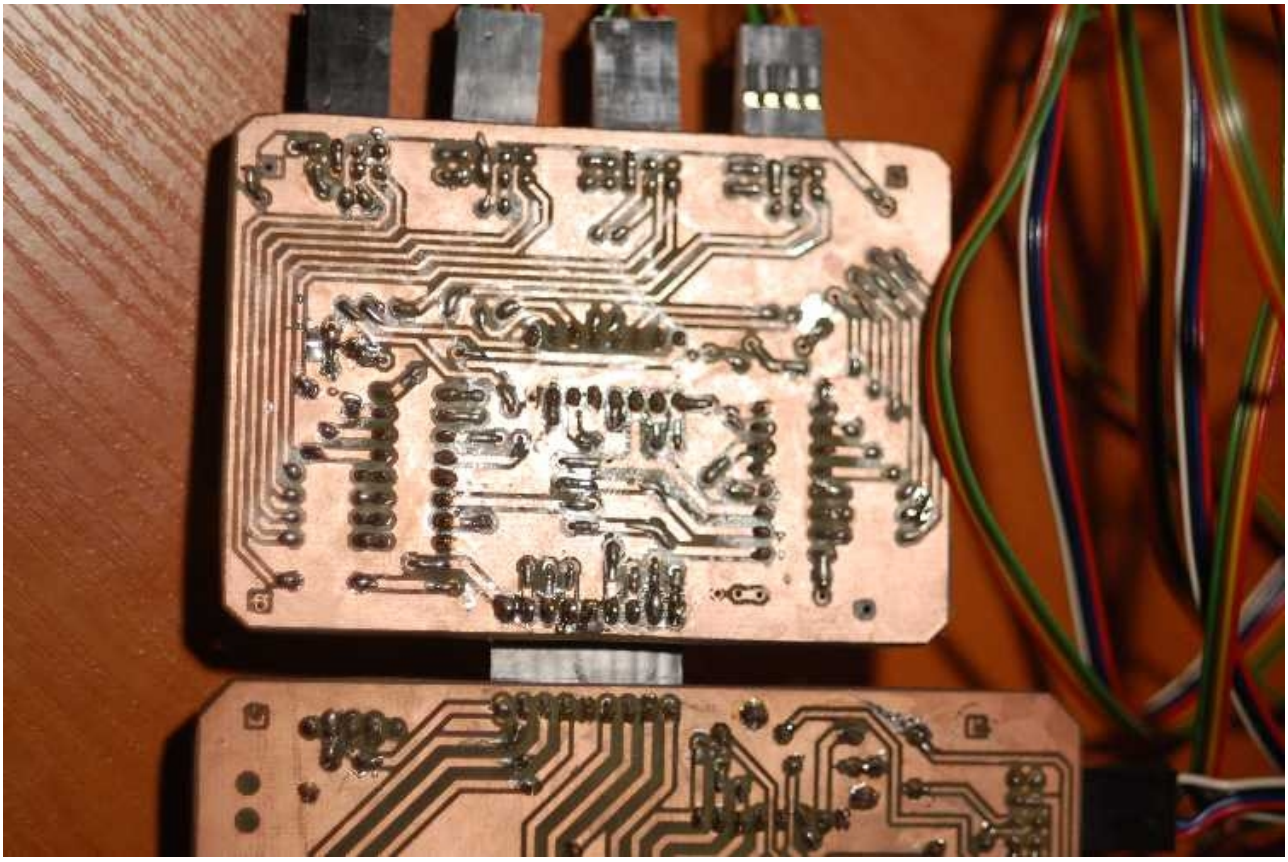
Zdjęcie 3: Moduł główny - góra



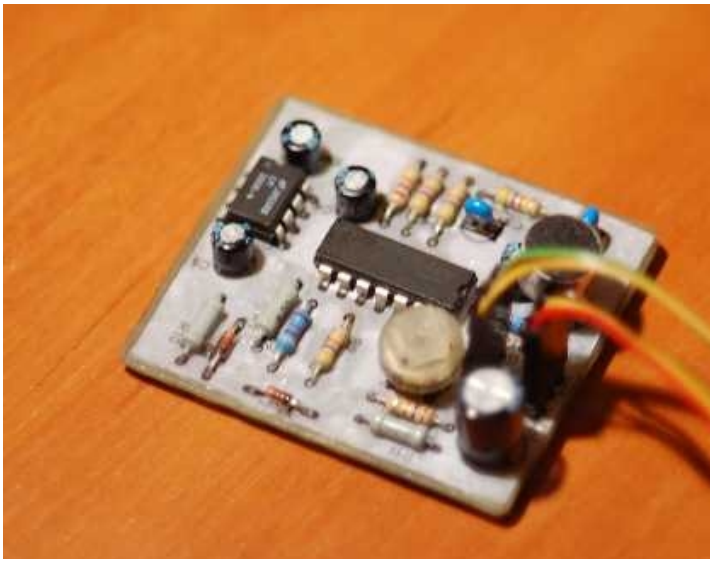
Zdjęcie 4: Moduł główny - dół



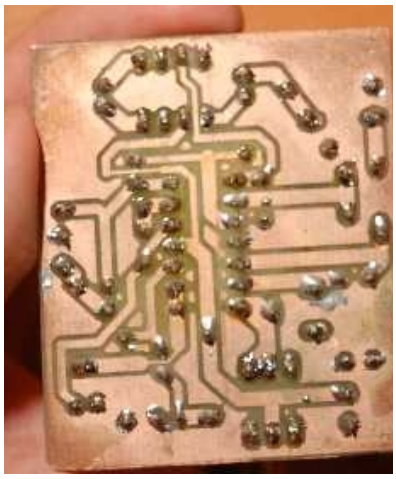
Zdjęcie 5: Moduł boczny - góra



Zdjęcie 6: Moduł boczny - dół



Zdjęcie 7: Moduł dźwiękowy - góra



Zdjęcie 8: Moduł dźwiękowy - dół

Po zlutowaniu układ zadziałał poprawnie, zatem przyjęte przez nas w drugim podejściu założenia okazały się właściwe.

2.4 Oprogramowanie

Program obsługujący diody został napisany w języku C w programie AVR Studio 4.

2.4.1 Opis działania programu

Podstawą działania programu jest wywoływane regularnie przerwanie, które zmienia cyklicznie numer obsługiwanej diody od 0 do 23. W przypadku braku wykrycia sygnału dźwiękowego, układ wyświetla płynnie zmieniające się barwy. Jest to zrealizowane następująco: w danym momencie istnieje „obowiązująca barwa", która dotyczy wszystkich diod, jednak dla każdej z nich wartości natężenia barw podstawowych przeliczane są przez ustalone w kodzie wagi. Wybrana zostaje losowa barwa oraz losowy czas przejścia do niej. Barwa zostaje płynnie zmieniona. Wtedy wagi zostają przesunięte (wagi dla każdej barwy podstawowej przesuwane są inaczej, żeby poszczególne ich ustawienia nie powtarzały się zbyt często), a następnie wylosowana zostaje następna barwa i cykl się zamyka.

W momencie, kiedy zostaje wykryty sygnał dźwiękowy, układ zmienia swoje działanie. Dla wartości sygnału mieszczących się w pewnym przedziale od wartości minimalnej (wykrycie sygnału) do pewnego progu, realizowany jest efekt podobny do opisanego powyżej z tą różnicą, że wartości natężenia barw na wszystkich diodach w danym momencie skalowane są wartością sygnału dźwiękowego. Jeśli natomiast wartość sygnału przekroczy ustalony próg, pojawia się jeszcze jeden efekt: przełączanie diod zostaje chwilowo spowolnione i przyjęta zostaje maksymalna wartość jasności diod, co daje wrażenie „odpalenia" jasnego impulsu, przebiegającego kolejno przez wszystkie diody.

2.4.2 Kod programu

```
/*****************************************************************
 * Oswietlenie dohyo LED RGB 1.0.0                               *
 * Program realizuje efekty swietlne na podstawie dostarczonego  *
 * sygnalu analogowego.                                          *
 *                                                               *
 * Autorzy:                                                      *
 * Adrian Ciż                                                    *
 * Marek Gulanowski                                              *
 * Maciej Trojnar                                                *
 * (Koło Naukowe Robotyków "KoNaR" - Politechnika Wrocławska)    *
 *                                                               *
 * Utworzono: 2008 - 2009                                        *
 *****************************************************************/

#include <avr/io.h>
#include <stdlib.h>
#include <avr/interrupt.h>

// zmienna j jest licznikiem, wskazujacym numer
// obslugiwanej w danym momencie diody
volatile int j;

// globalne zmienne okreslajace kolor
volatile short int r,g,b;
// zmienne wykorzystywane w funkcji SetColours
volatile short int red, green, blue;
```

```
// wagi, przez ktore mnozone sa natezenia poszczegolnych
// barw podstawowych na poszczegolnych diodach
volatile short int weight[3][24] = {
                  {10, 30, 50, 70, 90, 120, 150, 180,
                        230, 240, 255, 255, 255, 250, 240, 230,
                        220, 210, 180, 150, 130, 100, 70, 40},
                  {220, 210, 180, 150, 130, 100, 70, 40,
                        10, 30, 50, 70, 90, 120, 150, 180,
                        230, 240, 255, 255, 255, 250, 240, 230},
                  {230, 240, 255, 255, 255, 250, 240, 230,
                        220, 210, 180, 150, 130, 100, 70, 40,
                        10, 30, 50, 70, 90, 120, 150, 180},
                  };

// zmienne umozliwiajace pamietanie informacji
// na temat poprzednich stanow ukladu
short int memory = 0;
int negative_memory = 0;

/* Funkcja SetColours
 *
 * Ustawia aktualna barwe na podstawie wartosci
 * zmiennych globalnych oraz (jesli zostanie to wskazane)
 * sygnalu analogowego.
 *
 * param sound - flaga okreslajaca, czy ma byc brany pod
 *    uwage sygnal analogowy
 *
 */
void SetColours(int sound)
{
      if(sound)
      {
            if(ADCH>190)
            {
            // puszczony zostaje "impuls" przez wszystkie diody
                  TCCR0 |= (1 << CS00)
                        | (1 << CS02);
                  red = r;
                  green = g;
                  blue = b;
                  negative_memory = 0;
                  if(memory<=0)
                        memory = 5000;
            }

            else if(ADCH>20 || memory>0)
            {
                  red = r*ADCH/255;
                  green = g*ADCH/255;
                  blue = b*ADCH/255;
                  negative_memory = 0;
                  if(memory>0)
                        memory--;
            }
            else
            {
            // "impuls" zostaje zakonczony
                  TCCR0 &= (0 << CS02);
                  TCCR0 |= (1 << CS00);
                  red = 0;
                  green = 0;
                  blue = 0;
                  negative_memory++;
                  if(memory>0)
                        memory--;
            }
      }
      else //if(!sound)
```

```
	{
		TCCR0 &= (0 << CS02);
		TCCR0 |= (1 << CS00);
		red = r;
		green = g;
		blue = b;
		if(ADCH>20)
			negative_memory = 0;
	}
}

/* Funkcja MoveWeights
 *
 * Przesuwa wagi, przez ktore mnozone jest natezenie
 * barw podstawowych dla poszczegolnych diod.
 *
 */
void MoveWeights()
{
	static short int x,tmp1,tmp2;
	tmp1 = weight[0][0];

	if(ADCH>20)
		SetColours(1);

	for(x=1; x<24; x++)
		weight[0][x-1] = weight[0][x];

	if(ADCH>20)
		SetColours(1);

	weight[0][23] = tmp1;
	tmp1 = weight[1][23];

	if(ADCH>20)
		SetColours(1);

	for(x=23; x>0; x--)
		weight[1][x] = weight[0][x-1];

	if(ADCH>20)
		SetColours(1);

	weight[1][0] = tmp1;
	tmp1 = weight[2][0];
	tmp2 = weight[2][1];

	if(ADCH>20)
		SetColours(1);

	for(x=2; x<24; x++)
		weight[2][x-2] = weight[2][x];

	if(ADCH>20)
		SetColours(1);

	weight[2][22] = tmp1;
	weight[2][23] = tmp2;

	if(ADCH>20)
		SetColours(1);
}

/* Funkcja init_ports
 *
 * Inicjalizuje porty tak, zeby dzialal PWM.
 *
 */

void init_ports()
```

```
{
	DDRB = _BV(DDB1)   // OC1A
		| _BV(DDB2)   // OC1B
		| _BV(DDB3);  // OC2

	TCCR1A = _BV(WGM10)    // uruchomienie PWM na Timer1 (daje 2 kanaly PWM)
		| _BV(COM1A1)    // set OC1A/B on compare match, clear them at top
(sposob dzialania)
		| _BV(COM1B1);
	TCCR1B = _BV(CS10)     // no prescale
		| _BV(WGM12) ;  // wybor trybu fast PWM

	TCCR2 = _BV(WGM20)
		| _BV(WGM21)     // timer 2 fast PWM
		| _BV(COM21)       // clear OC2 on compare match, set OC2 at TOP
(sposob dzialania)
		| _BV(CS20);     // no prescale
}

/* Funkcja ChangeColour
 *
 * Zapewnia stopniowa zmiane aktualnej barwy na inna.
 *
 * param sound - flaga okreslajaca, czy sygnal analogowy
 *    ma byc brany pod uwage
 *
 */
void ChangeColour(int new_r, int new_g, int new_b,
                                int sound)
{

	while(r != new_r || g != new_g || b != new_b)
	{

		SetColours(sound);

		if( r < new_r)
			r++;
		else if( r > new_r)
			r--;

		SetColours(sound);

		if( g < new_g)
			g++;
		else if( g > new_g)
			g--;

		SetColours(sound);

		if( b < new_b)
			b++;
		else if( b > new_b)
			b--;

		SetColours(sound);
	}
}

int main (void)
{

	init_ports();

	DDRD = 0b00111111; // adresowanie

	TCCR0 |= (1 << CS00);
	TIMSK |= (1 << TOIE0); //allow interrupt
	SREG |= (1 << 7); //global interrupt enable
```

```
	ADCSRA = (1<<ADEN)|     //enable AD
	            (1<<ADFR)|  // free run-constant conversion
	            (1<<ADSC);  // start conversion

	ADMUX = 0x60;

	j = 0;

	r = 255;
	g = 255;
	b = 255;

	while(1)
	{
		MoveWeights();
		static short int rand1, rand2, rand3;
		if(ADCH>20)
			SetColours(1);
		rand1 = rand()%256;
		if(ADCH>20)
			SetColours(1);
		rand2 = rand()%256;
		if(ADCH>20)
			SetColours(1);
		rand3 = rand()%256;
		if(ADCH>20)
			SetColours(1);
		if(negative_memory>1000)
			ChangeColour(rand1, rand2, rand3,0);
		else
			ChangeColour(rand1, rand2, rand3,1);
	}
}


/*
 * Przerwanie, ktore wybiera kolejna diode
 * i przelicza wartosc natezenia barw podstawowych
 * zgodnie z obowiazujacymi wagami.
 *
 */
ISR(TIMER0_OVF_vect)
{
	j = (j+1)%24;
	PORTD = j%8 + (0b00111000 - (1 << (3+j/8)));

	OCR1A = red*weight[0][j]/255;
	OCR1B = green*weight[1][j]/255;
	OCR2 = blue*weight[2][j]/255;
}
```

2.5 Podsumowanie projektu i dalsze możliwości rozwoju

2.5.1 Działanie

Diody podłączyliśmy do sterownika i umieściliśmy w jednym rzędzie na tekturowej podstawie. Poniższe zdjęcia ukazują działanie układu.

2.5.2 Umieszczenie układu i diod pod dohyo

Urządzenie wykonane w ramach projektu jest funkcjonalnym sterownikiem diod RGB. Jednak aby mogło zostać wykorzystane na najbliższych zawodach minisumo, należy opracować sposób praktycznego umieszczenia go razem z diodami pod dohyo oraz odpowiedniego rozproszenia światła, aby uzyskać atrakcyjny wygląd całości.

Przykładowo można zamontować układ na płycie wiórowej, wyciętej w kształt dohyo, do której zamocowano by diody – umożliwiłoby to przy okazji zwiększenie ich jasności poprzez zastosowanie przełączników tranzystorowych (montaż diod na małych płytkach). Ze względu na małą liczbę diod efektownym (koniecznym) rozwiązaniem byłoby rozproszenie światła za pomocą generatora pary lub kawałków kleju silikonowego.

2.5.3 Udoskonalenie oprogramowania

Opublikowany powyżej kod implementuje jedynie bardzo proste efekty świetlne. Na tyle na ile pozwolą możliwości mikrokontrolera ATmega8, możliwe jest opracowanie większej ilości efektów, które mogłyby zmieniać się cyklicznie.

Istotna jest też możliwość optymalizacji oprogramowania, np. poprzez zastosowanie wstawek asemblerowych. Dotyczy to szczególnie przerwania, które wywoływane jest bardzo często i zajmuje dużą część dostępnego czasu obliczeniowego. Zoptymalizowanie tego przerwania pozwoliłoby na implementację bardziej złożonych efektów.

3 Zakończenie

Realizacja projektu umożliwiła nam zdobycie doświadczenia w obszarach wymienionych na początku sprawozdania: projektowaniu układów elektronicznych, produkcji płytek drukowanych, programowaniu mikrokontrolera.

Uświadomiliśmy sobie wiele problemów, które spotyka się przy realizacji tego typu zadań. Wyciągnęliśmy następujące wnioski:

- Korzystne jest podzielenie problemu na moduły, jak na przykład przy projektowaniu układu, co zarazem ułatwia podział prac pomiędzy osoby.
- W trakcie prac nad każdym etapem projektu należy brać pod uwagę możliwe konsekwencje w kolejnych etapach, np. projektować układ z myślą o łatwości lutowania.
- Tworzenie harmonogramu prac jest złudne w przypadku, gdy nie ma się doświadczenia w podobnych projektach, gdyż nie sposób przewidzieć możliwych problemów ani ocenić poziomu trudności poszczególnych zadań.