Politechnika Śląska
Wydział Elektryczny
Kierunek Mechatronika

Projekt inżynierski

Projekt pojazdu zdalnie sterowanego

Autor: **Zbigniew Sroczyński**
Nr albumu: 282025

Kierujący pracą: **dr inż. Krzysztof Konopka**
Recenzent: **dr. Inż. Artur Skórkowski**

Gliwice, styczeń 2022

Pojazd zdalnie sterowany

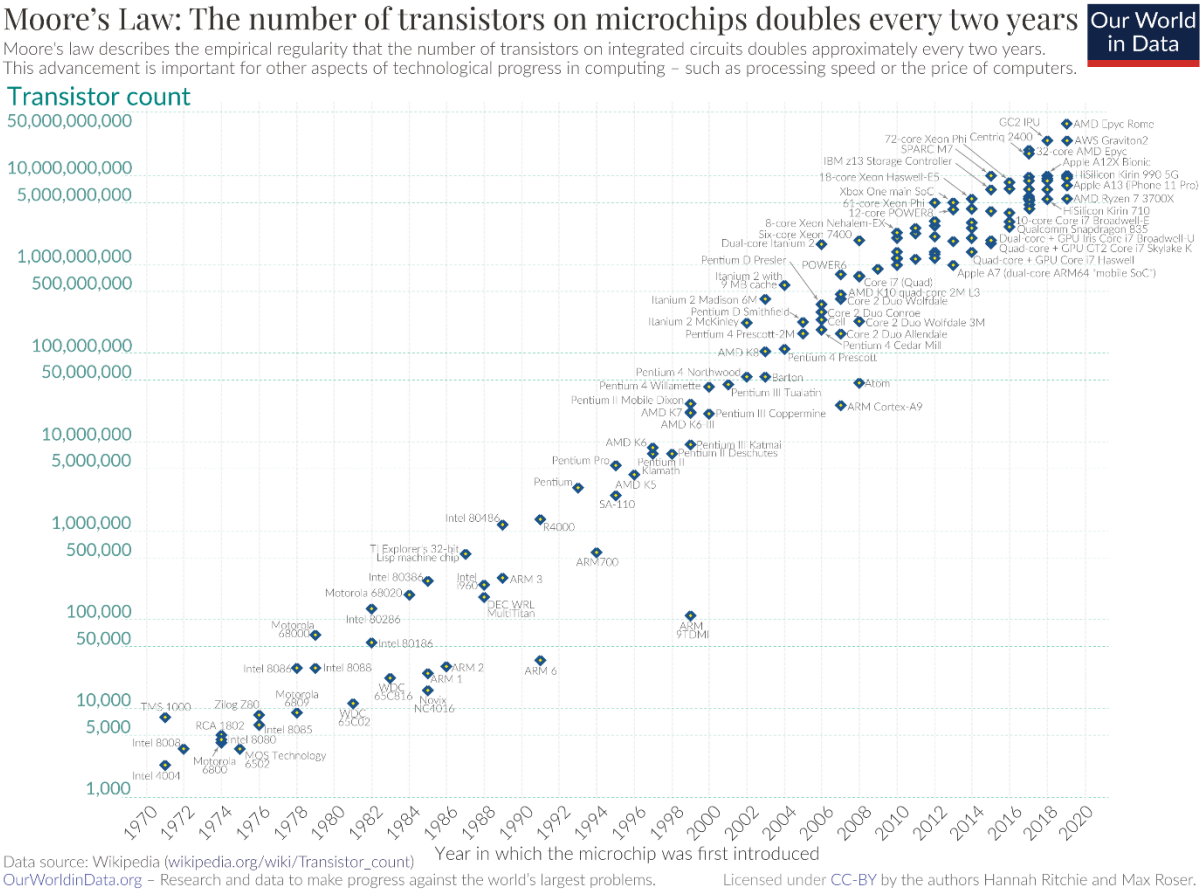
Spis treści

1. ZARYS PROJEKTU

1. Wstęp teoretyczny

W dzisiejszych czasach mikrokontrolery wykorzystywane są coraz częściej w różnych dziedzinach naszego życia. Coraz częściej zastępują one pracę człowieka. Mikrokontrolery często wykonują obliczenia wielokrotnie szybciej od człowieka, są niezawodne oraz wymagają niskiego poboru prądu.

Obecne mikrokontrolery osiągają coraz to większe moce obliczeniowe. Ich wewnętrzna struktura tranzystorowa ulega miniaturyzacji zgodnie z prawem Moore'a. Jeszcze w 2000 roku liczba tranzystorów na centymetr kwadratowy wynosiła 5 milionów tranzystorów na centymetr kwadratowy. Obecnie wartość ta sięga prawie do 50 miliardów tranzystorów na centymetr kwadratowy. Liczba tranzystorów w procesorze wzrasta wykładniczo, do tego spostrzeżenia doszedł amerykański fizyk i inżynier Gordon Moore. [1]



Rys. 1 – Prawo Moore'a [2]

Obecne mikrokontrolery aby zapewnić jak największą moc obliczeniową posiadają architekturę Harwardzką Oznacza to że wewnętrzne magistrale danych i rozkazów są od siebie rozdzielone. Prostsza budowa procesora (w odróżnieniu do architektury von Neumanna) przekłada się na większą szybkość działania. Architektura harwardzka znalazła zastosowanie w procesorach sygnałowych, które obecnie są najczęściej wykorzystywane. [3]

2. Cel projektu

Celem projektu inżynierskiego jest skonstruowanie zdalnie sterowanego pojazdu. Pojazd będzie wykonany jako fizyczny obiekt.

Pojazd będzie sterowany przez pilot na podczerwień, wykorzystujący protokół kodowania RC5. Wykonany pojazd będzie małogabarytowy. Koła pojazdu będą miały średnicę 60 mm. Wykorzystane zostaną koła mecanum, które będą poruszać się niezależnie od siebie. Koła umożliwią jazdę pojazdu: do przodu, do tyłu, w prawo, w lewo, jazdę po skosie oraz obrót w miejscu. Rozmiary podwozia to: 170 mm x 140 mm. Wszystkie elementy zostały zakupione przez konstruktora projektu.

Wykorzystane podwozie i pilot na podczerwień będą zakupione jako gotowy produkt. Wykonana praca podczas projektu jest następująca:

- Wykonanie płytki drukowanej, która będzie służyć jako sterownik pojazdu. Płytka drukowana będzie wykonana od podstaw przez konstruktora projektu. Płytka nie była inspirowana przez inne źródła. Konstruktor wykorzystał swoją wiedzę i zainteresowania aby wykonać płytkę PCB. Płytka będzie wykonana w programie Eagle i wykonana metodą termotransferu na laminacie pokrytym warstwą miedzi.
- Lutowanie elementów elektronicznych do wcześniej wykonanej płytki drukowanej. Elementy które zostaną przylutowane to m.in.: mikrokontroler Atmega8, drivery silników L293D, odbiornik podczerwieni, listwy goldpinów, rezystory, kondensatory, przetwornice.
- Napisanie programu w języku C na mikrokontroler Atmega8. Program został napisany od podstaw, samodzielnie przez konstruktora projektu. Napisany program obejmuje następujące funkcjonalności:
    - odbiór oraz analiza odebranych danych z czujnika podczerwieni. Dane są przesyłane protokołem RC5,
    - sterowanie driverami silników (każdym niezależnie od siebie), wytworzenie sygnału PWM tak aby zmiana prędkości silników była możliwa,
    - analiza napięcia podawanego na silniki DC. Napięcie będzie analizowane przez przetwornik analogowo-cyfrowy które wcześniej będzie obniżone przez dzielnik napięcia.

3. Założenia projektowe

Aby zrealizować zadanie projektowe przyjąłem poniższe kryteria:

- **Podwozie** – aluminiowa rama wykorzystujące 4 koła mecanum 60mm.
- **Jednostka sterująca** - mikrokontroler Atmega8 w obudowie DIP.
- **Napięcie zasilania** - 14,4V z akumulatorów litowo-jonowych.
- **Interfejs** – RC5 standard przesyłu danych między nadajnikiem a odbiornikiem za pomocą podczerwieni.
- **Nadajnik** – Pilot na podczerwień wykorzystujący kodowanie RC5.
- **Odbiornik** – Zdalnie sterowany pojazd.

4. Budowa układu

Nadajnik:

- Pilot na podczerwień z kodowaniem na podczerwień RC-5

Odbiornik:

- Odbiornik podczerwieni TSOP3123
- Mikrokontroler Atmega8
- Driver silników L249D
- Silnik DC

5. Wybór platformy

Aby zrealizować zadanie projektowe, postanowiłem rozważyć dostępne platformy lub mikrokontrolery które miały posłużyć jako jednostka sterująca.

- **Raspberry Pi**

  Platforma sprzętowa charakteryzująca się wydajnym procesorem który jest w stanie utrzymać pracę systemu operacyjnego Linux Raspbian. Układ wykorzystywany jako mobilny komputer małej mocy. Posiada szereg interfejsów, wyjścia/wejścia GPIO.

  Jednakże Raspberry Pi nie będzie najlepszym wyborem do tego projektu. Platforma ta posiada wiele elementów i mocy obliczeniowej które w niewielkim stopniu byłyby wykorzystane przy realizacji projektu. Zastosowanie tej platformy jest także nieopłacalne ze względu na koszt.

- **Arduino**

  Platforma ciesząca się wielkim zainteresowaniem ze strony hobbystycznych programistów-elektroników. Płytki tej rodziny można łatwo programować dzięki środowisku które posiada wiele wbudowanych bibliotek. Płytki są wykonane najczęściej w oparciu o mikrokontroler AVR, który posiada dodatkowy programator

  Nie wybrałem Arduino dlatego że te rozwiązanie wydaje się być nie profesjonalne. Ta platforma wykorzystuje język Arduino który jest dedykowany dla początkujących programistów. Program chcę realizować w oparciu o język C.

- **STM32**

  Mikrokontrolery 32-bitowe posiadające rdzeń Cortex zbudowany w oparciu o architekturę ARM. Mikrokontrolery najczęściej posiadające jeden rdzeń, częstotliwość taktowania może wynosić maksymalnie 72MHz.

  Nie zdecydowałem się na ten rodzaj mikrokontrolera, ponieważ aby napisać w nim program to należy wygenerować dużą ilość kodu w IDE. Program jest mało czytelny, wymaga instalacji wielu bibliotek i środowisk graficznych.

- **AVR**

  Mikrokontrolery 8-bitowe produkowane przez firmę Atmel. Nie są to skomplikowane mikrokontrolery. Maksymalna częstotliwość taktowania wynosi 16MHz. Posiadają podstawowe wbudowane układy peryferyjne, takie jak: przetworniki analogowo-cyfrowe, timery, sprzętową obsługę interfejsów I2C, RS-232.

  Zdecydowałem się na wybór tych mikrokontrolerów przy realizacji projektu. Program napisany na te układy jest bardzo przejrzysty, poszczególne rejestry i ich funkcje są bardzo przystępnie opisane w nocie katalogowej. Programowanie wymaga od twórcy znajomości języka C. Koszt tych mikrokontrolerów nie jest wysoki.

  Opisane zalety jak i wady pozwalają stwierdzić że mikrokontrolery z rodziny AVR będą doskonałym wyborem do realizacji projektu.

2. REALIZACJA PROJEKTU

1. Wykorzystane elementy

a) Mikrokontroler Atmega8

Mikrokontroler 32-bitowy, niskiego poboru mocy.
Całkowita ilość wyprowadzeń to 28 „nóżek”.
Posiada 23 programowalne wyprowadzenia I/O. [5]

Rys. 2 – Wykorzystany mikrokontroler Atmega8. [4]

Posiada następujące rodzaje i ilości pamięci:

- **8kB pamięci Flash** – rodzaj nieulotnej pamięci w której zapisywany jest skompilowany program.
- **1kB pamięci SRAM** – statyczna pamięć RAM, jest to pamięć ulotna w której znajdują się rejestry konfiguracyjne, przeprowadzane są w niej operacje, na końcu pamięci znajduje się stos.
- **512B pamięci EEPROM** – elektrycznie kasowalna i programowalna pamięć tylko do odczytu. Dane które są w niej zapisane nie są tracone w przypadku restartu urządzenia.

Układy peryferyjne mikrokontrolera Atmega8:

- Dwa 8-bitowe timery
- Jeden 16-bitowy timer
- Układ RTC – zegar czasu rzeczywistego
- Trzy kanały dla wyjścia sygnału PWM
- 8-kanałowy przetwornik analogowo-cyfrowy
- Sprzętowa obsługa interfejsu I2C
- Sprzętowa obsługa interfejsu RS-232, z wykorzystaniem modułu UART
- Sprzętowa obsługa interfejsu SPI
- Programowalny timer Watchdoga
- Pięć rodzajów trybów niskiego poboru prądu

Możliwości taktowania mikrokontrolera Atmega8:

- Wykorzystanie zewnętrznego rezonatora kwarcowego, osiągane częstotliwości z przedziału 0 – 16MHz
- Wykorzystanie wewnętrznego oscylatora RC, osiągane częstotliwości zawierają się w przedziale 0 – 8MHz
- Zewnętrzne taktowanie sygnałem TTL

b) Driver silnika L293D

Omawiany układ posiadał w swojej obudowie dwa niezależne mostki H. Każdy mostek H posiada dwa wejścia i dwa wyjścia. Sterowanie odbywa się poprzez podanie na jedno z wejść cyfrowego stanu wysokiego (cyfrowy stan wysoki; 5V zasilanie części cyfrowej).
W rezultacie na dwóch wyjściach mocy uzyskaliśmy polaryzację zależną od tego na którym wejściu panuje stan cyfrowy wysoki. [6]

Napięcie dla sekcji zasilania silników posiadały oddzielne zasilanie, ze względu na możliwość regulacji napięcia zasilania silników (napięcie na wyjściach mocy mieściło się w zakresie 5-8 V). Oddzielne zasilanie zrealizowano za pomocą dodatkowej przetwornicy.

Take rozwiązanie jest korzystne ze względu na odseparowanie części cyfrowej/sterowania z częścią zasilania silników. Do części cyfrowej nie przedostają się zakłócenia spowodowane ruchem silników.

Podstawowe parametry drivera L293D:

- Maksymalny prąd ciągły na kanał: 600mA
- Maksymalny prąd chwilowy na kanał: 1,2A
- Zabezpieczenie termiczne
- Zastosowanie diod niwelujących zakłócenia z silnika

c) Przetwornica Step-down

Jest to przetwornica typu DC-DC, obniżająca napięcie [7]. Podstawowe parametry przetwornicy:

- Napięcie wejściowe: 3-35 V
- Napięcie wyjściowe regulowane: 1,5-30 V
- Układ przetwornicy: LM2596
- Prąd wyjściowy: 2A stały, Max 3A (z radiatorem)
- Wydajność: Max 92%

d) Ogniwa litowo-polimerowe

Jako zasilanie całego układu wykorzystano cztery ogniwa litowo-polimerowe. Każde ogniwo posiada następujące parametry:

- Napięcie znamionowe: 3,6 V
- Pojemność: 2900 mAh

Cztery ogniwa połączone są szeregowo, co daje nam:

$$4 * 3{,}6V = 14{,}4V$$

Napięcie o takiej wartości podawane jest na przetwornice.

e) Silniki DC

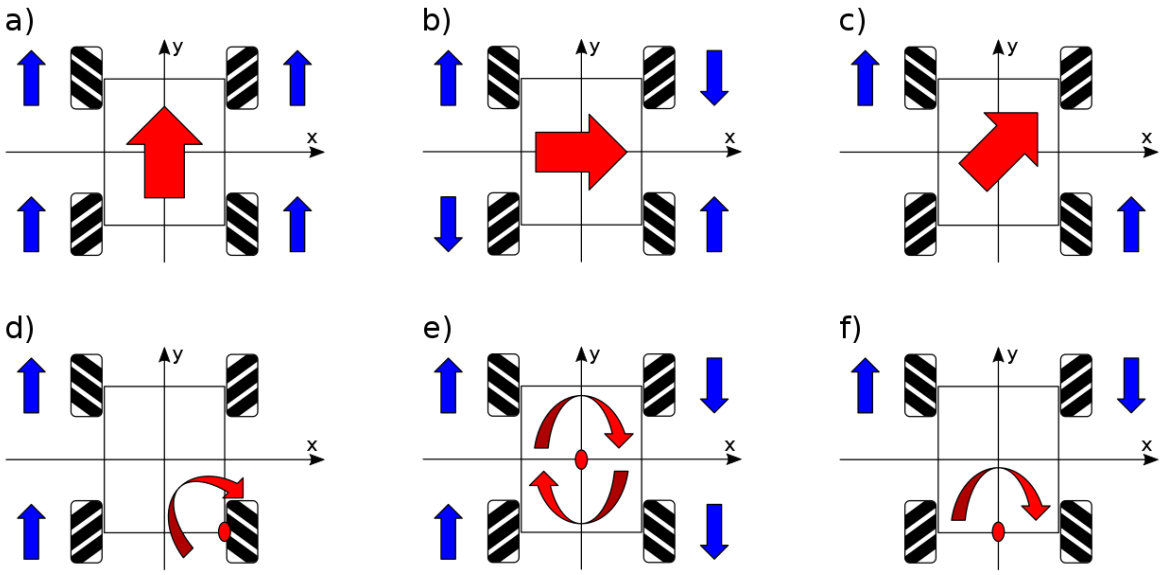
Silniki szczotkowe komutatorowe. Podstawowe parametry:

- Napięcie zasilania: 5V
- Pobór prądu: 180mA

W przeprowadzonym projekcie napięcie zasilania postanowiłem zmieniać w zakresie 5-8 V. Wiązało się to ze zwiększeniem prędkości obrotowej silnika oraz zwiększonym poborem prądu.

f) Koła typu macenum

Jest to rodzaj koła omnikierunkowego. Koło składa się z piasty na której zamontowane są obracające się rolki. W zależności od kierunku obrotów poszczególnych czterech kół w pojeździe możemy osiągnąć jazdę: przód, tył, lewo, prawo, jazdę po skosie oraz możliwość obrotu pojazdu w miejscu. [9]



Rys. 3 – Kierunki jazdy całego pojazdu w zależności od kierunków jazdy poszczególnych kół [10].

2. Wykorzystane magistrale i protokoły

a) Magistrala SPI

SPI to szeregowy interfejs urządzeń peryferyjnych. Występuje aby zapewnić komunikację między mikrokontrolerem i układami typu: przetworniki ADC/DAC, pamięci EEPROM, układy RTC [11].

Zalety magistrali SPI:

- Zapewnia komunikację typu full-duplex
- Prędkość przesyłania informacji większa niż w I2C

- Prosty interfejs sprzętowy
- Brak arbitrażu oraz konfliktów związanych ze zderzeniem się pakietów danych
- Układy slave wykorzystują taktowanie układu master i nie wymagają precyzyjnych oscylatorów
- Łatwa implementacja

Wady magistrali SPI:

- Sygnały są jednokierunkowe i pozwalają na izolację galwaniczną
- Brak adresowania w paśmie, wybór układu slave odbywa się przez linię SS
- Interfejs można stosować na niewielkie odległości, w porównaniu z RS-232
- Brak zdefiniowanego protokołu sprawdzającego błędy

Komunikacja wykorzystująca interfejs SPI odbywa sią za pomocą następujących linii:

- **MOSI** (ang. *master output slave input*) – dane z układu nadrzędnego dla układu peryferyjnego,
- **MISO** (ang. *master input slave output*) – dane z układu peryferyjnego dla układu nadrzędnego,
- **SCLK** (ang. *serial clock*) – sygnał zegarowy nadawany przez urządzenie nadrzędne (taktujący).
- **SS**(ang. *Slave select*) – linia umożliwiająca wybór układu podrzędnego. Linia nie jest wykorzystywana w przypadku gdy posiadamy jedno urządzenie peryferyjne.

W projekcie, omawiana magistrala jest wykorzystywana aby nawiązać połączenie komputer-mikrokontroler. Do portu USB w komputerze podłączony jest programator który programuje mikrokontroler Atmega8 wykorzystując interfejs SPI.

b) Protokół RC5

Protokół opracowany przez firmę Phillips na początku lat 80 XX. wieku. Protokół wykorzystywany w komunikacji na podczerwień. RC5 jest szeroko wykorzystywany w elektronice użytkowej. Jako nadajnik najczęściej jest wykorzystywany pilot na podczerwień. Odbiornikami są urządzenia audio i wideo [12].

Pojedynczy rozkaz wysłany przez protokół RC5 zawiera 14 bitów:

- **2 bity startu** – zawsze logiczne 1
- **1 bit toggle** – bit zmieniający wartość po kolejnym wciśnięciu przycisku
- **5 bitów adresu urządzenia** – możliwych jest 32 adresów urządzeń do których będą wysyłane dane
- **6 bitów kodu rozkazu** – kod wysłanego rozkazu, możliwych 64 kombinacji rozkazu

Sygnał w standardzie RC5 jest sygnałem bifazowym. Oznacza to tyle że stan bitu jest określany przez fazę sygnału. Rozpoznawanie odebranych bitów odbywa się w następujący sposób:

- Bit o wartości zero to sygnał RC5 który przechodzi ze stanu wysokiego do niskiego,
- Bit o wartości jeden to sygnał RC5 który przechodzi ze stanu niskiego do wysokiego

3. Sygnał PWM

Sygnał PWM to sygnał prostokątnym o zmiennym wypełnieniu, który może być wykorzystywany do zasilania lub przenoszenia informacji. Modulacja szerokości impulsu wykorzystuje przebieg prostokątny, którego szerokość impulsu jest modulowana, co powoduje zmianę średniej wartości przebiegu [13]. Średnia wartość przebiegu PWM wyrażona jest wzorem:

$$U_{śr} = \frac{1}{T}\int_0^T f(t)dt = \frac{1}{T}\left(\int_0^{DT} U_{max}dt + \int_{DT}^{T} U_{min}dt\right)$$

Gdzie:
Uśr – średnia wartość sygnału PWM
T – czas w którym dokonujemy pomiaru wartości średniej napięcia
f(t) – funkcja określająca przebieg sygnału
Uśr – średnia wartość sygnału PWM
Umax – stan wysoki przebiegu prostokątnego
Umin – stan niski przebiegu prostokątnego, dla sygnału PWM najczęściej 0V

4. Nadajnik sygnału RC5

Jako nadajnik sygnału PWM wykorzystano pilot na podczerwień generujący sygnał zgodny z RC5. Jest to pilot od telewizora zasilany dwoma ogniwami AAA 1,5V. Posiada 49 przycisków dostępnych dla użytkownika

5. Odbiornik sygnału RC5

Odbiornik w postaci układu scalonego TSOP3123. Odbiornik znajduje się na wykonanej przeze mnie płytce drukowanej. Podstawowe parametru odbiornika:

- Częstotliwość nośna: 31kHz
- Napięcie zasilania: 2,5-5,5 V
- Pobór prądu: 0,4mA

6. Wykorzystane środowisko programistyczne

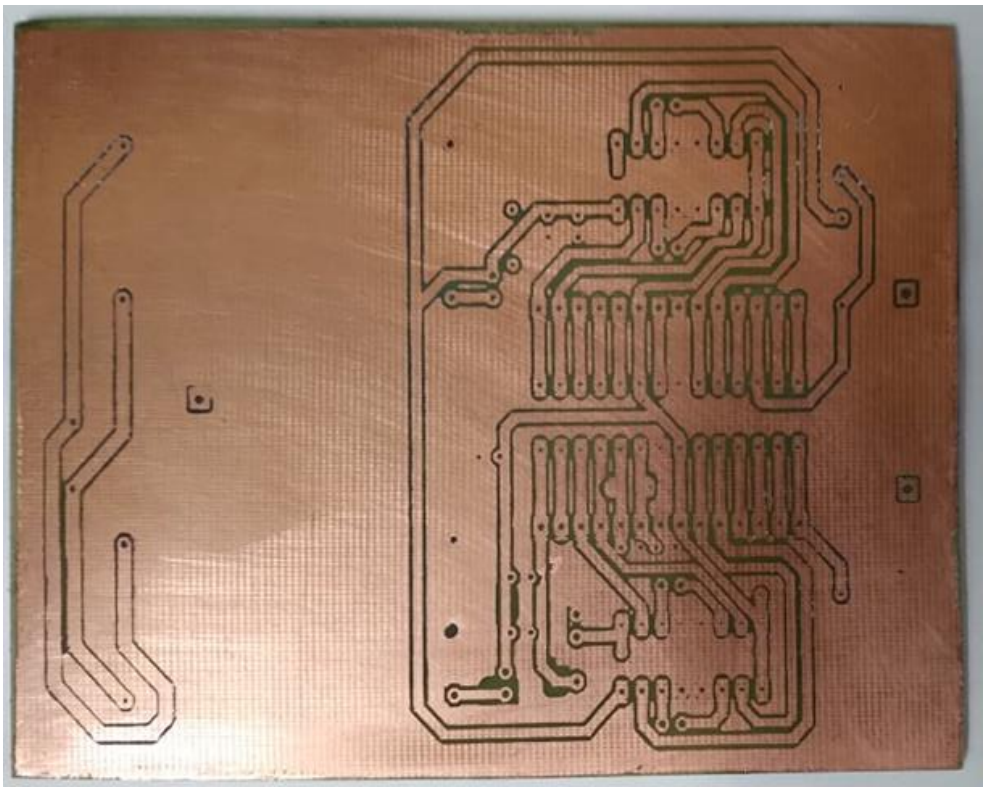
Jako środowisko programistyczne (IDE) wykorzystałem Eclipse Luna. Jest to IDE które posiada następujące właściwości:

- Możliwość pisania kodu w języku C
- Kolorowanie składni
- Tworzenie projektów, katalogów oraz plików źródłowych .c i nagłówkowych .h
- Możliwość zainstalowania kompilatora AVR-GCC
- Możliwość zainstalowania AVRDUDE – programu umożliwiającego wgranie skompilowanego programu do mikrokontrolera

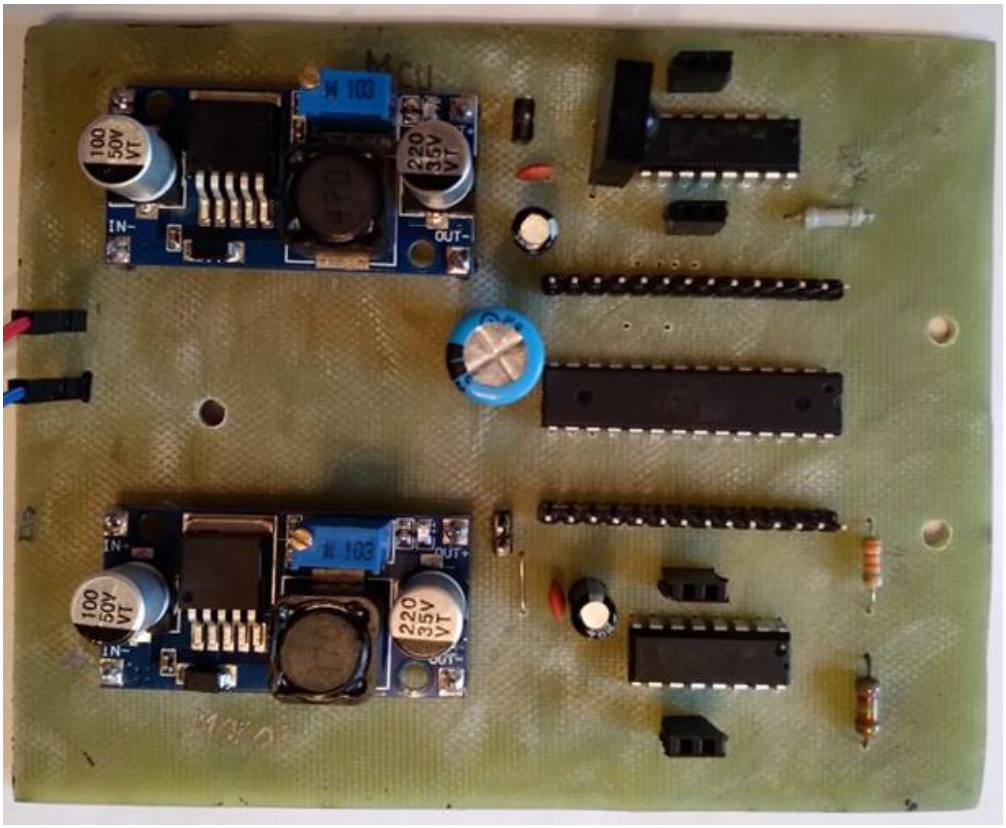
3. WYKONANIE PROJEKTU

1. Prototyp urządzenia

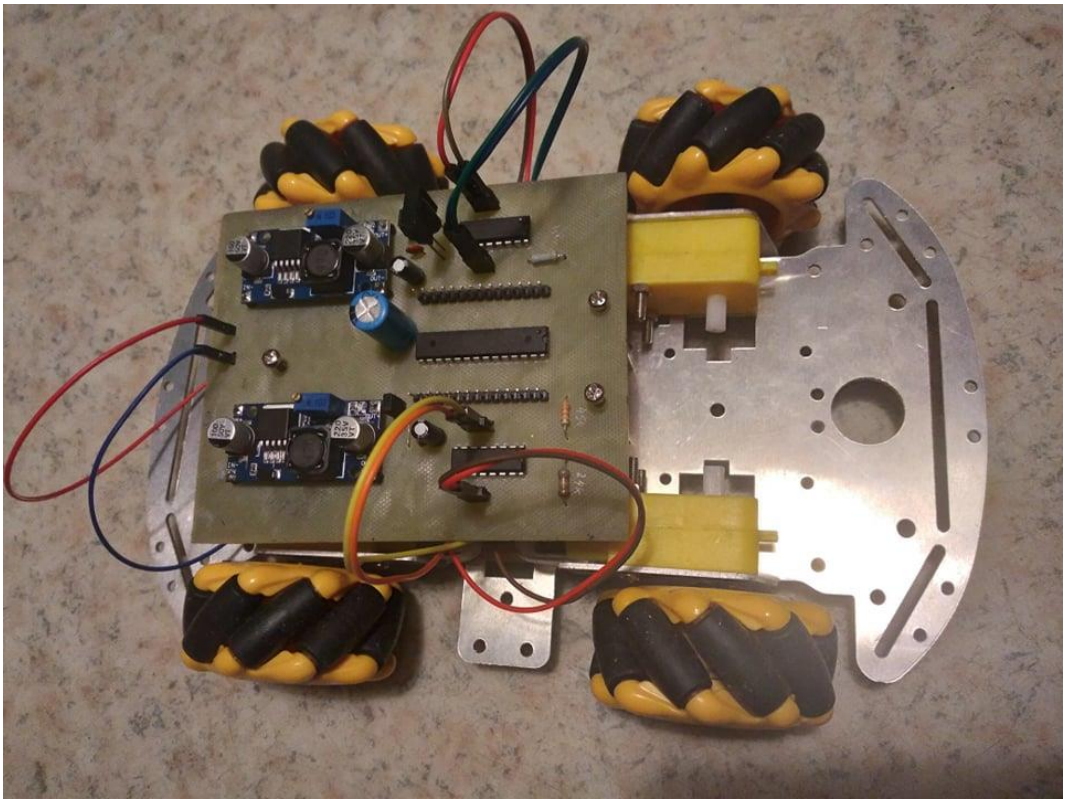
Projekt rozpoczął się od wykonania płytki drukowanej. Płytkę wykonano metodą termotransferu na laminacie pokrytym miedzią. Następnie przystąpiono do lutowania elementów elektronicznych. Wywiercono otwory, poprzez tuleje dystansowe połączono płytkę drukowaną z ramą podwozia. Podłączono zasilanie oraz silniki DC do wykonanej płytki.



Rys. 4 – Płytka drukowana po wytrawieniu.



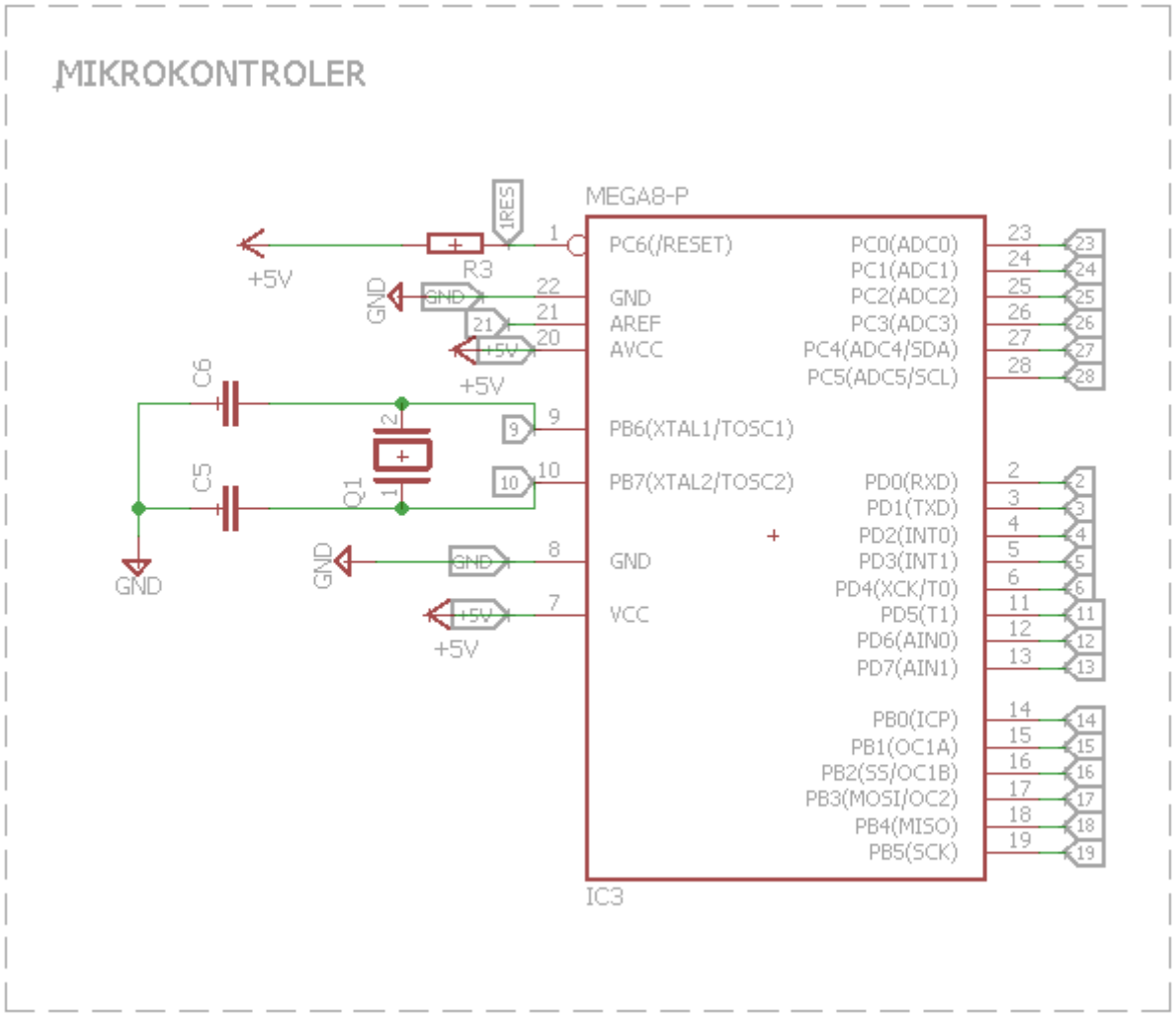
Rys. 5 – Płytka drukowana z przylutowanymi elementami.



Rys. 6 – Gotowy projekt inżynierski.

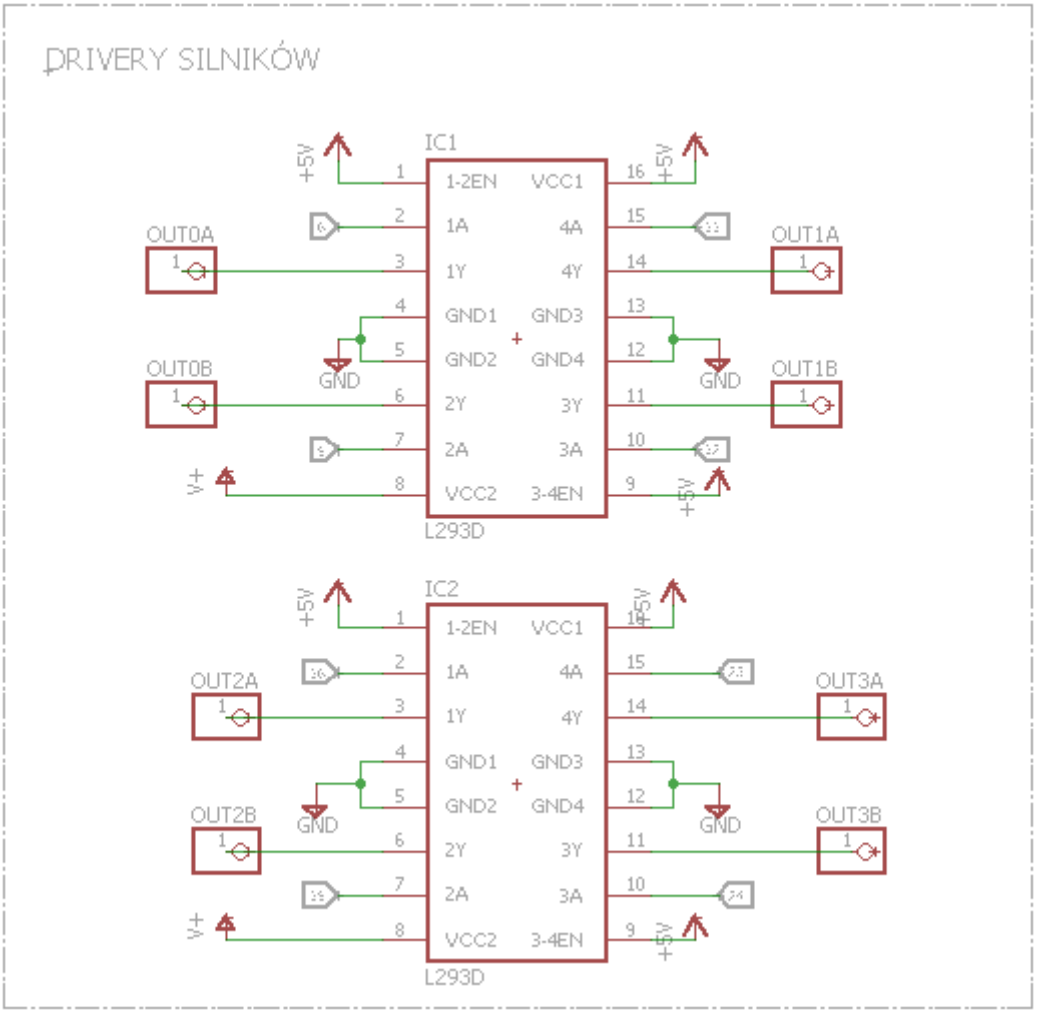
2. Projekt płytki drukowanej

Realizację wykonania płytki drukowanej rozpocząłem od sporządzenia schematu elektronicznego w programie Eagle.



Rys. 7 – Mikrokontroler

Powyższy schemat przedstawia mikrokontroler Atmega8. W lewej części widać podłączoną część zasilania uC oraz możliwość dołączenia rezonatora kwarcowego wraz z kondensatorami 22pF. Po prawej części mikrokontrolera wyeksportowano porty I/O do etykiet. W dalszej części tworzenia projektu, łatwiej będzie się odnieść do poszczególnych wyprowadzeń mikrokontrolera.

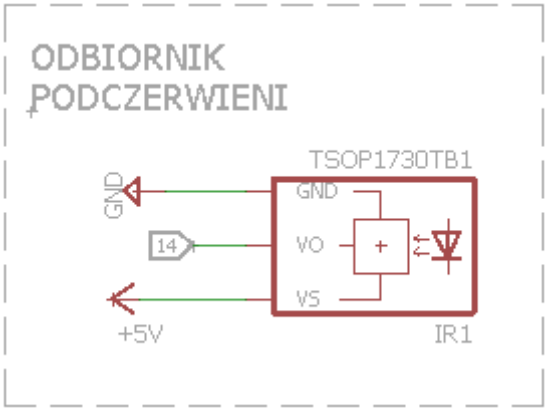


Rys. 8 – Drivery silników

Dalsza część schematu przedstawia dwa drivery L293D. Zasilanie części logicznej VCC1 wynosi 5V. Zasilanie części mocy VCC2 wynosi regulowane napięcie 5-8 V. Mostki H są włączone na stałe. Oznacza to że wyprowadzenia EN, na stałe są podłączone do napięcia 5V.

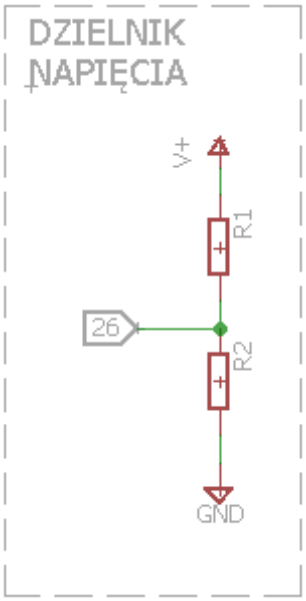
Wejścia driverów to: 1A, 2A, 3A, 4A. Do wejść driverów podłączone są wyjścia mikrokontrolera Atmega8. Połączenie odbywa się przez zdefiniowane wcześniej etykiety wyprowadzeń uC.

Wyjścia driverów to: 1Y, 2Y, 3Y, 4Y. Wyjścia wyprowadzone są na goldpiny z których następnie przewodami podłączone są silniki DC.



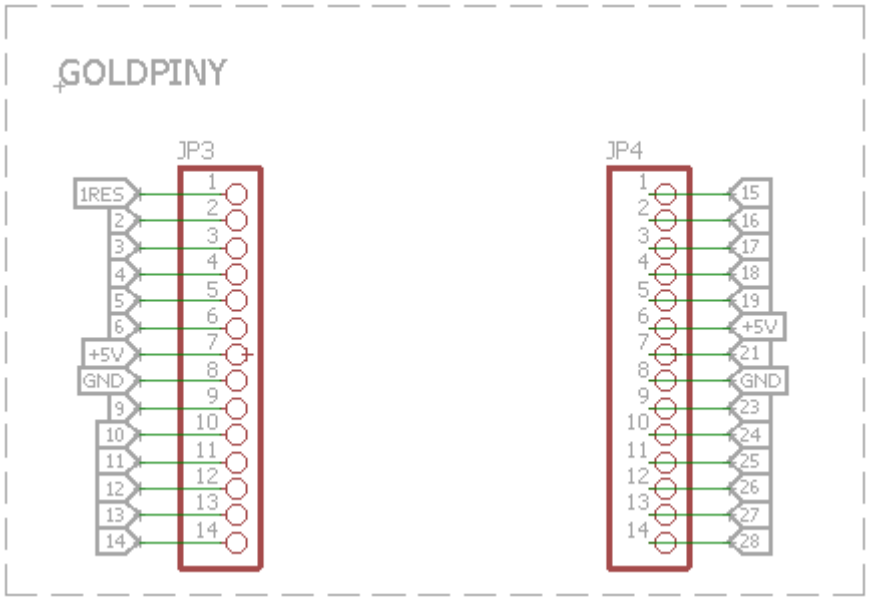
Rys. 9 – Odbiornik podczerwieni

Dalsza część schematu przedstawia odbiornik podczerwieni. Odbiornik podłączony jest do napięcia zasilania 5V, masy GND oraz do 14 wyprowadzenia mikrokontrolera. Wyprowadzenie VO odbiornika TSOP3123 to wyjście sygnału RC5.



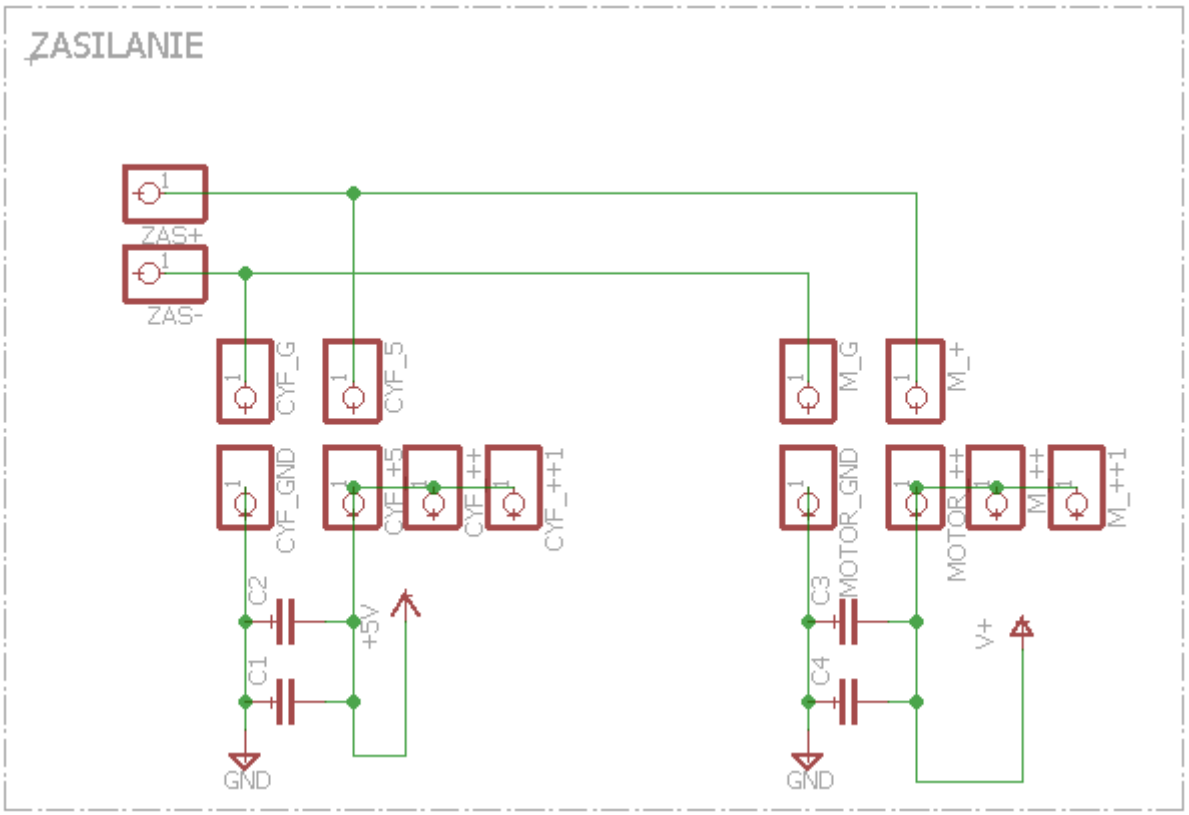
Rys. 10 – Dzielnik napięcia

Następnie wykonano dzielnik napięcia. Na wejście dzielnika podano napięcie zasilania silników z przedziału 5-8V. Na wyjściu uzyskamy sygnał nieprzekraczający 5V. Napięcie wyjściowe nie może przekroczyć 5V aby nie uszkodzić części cyfrowej. Badanie napięcia zasilania silników jest dodatkową informacją dla uC przy obliczaniu wartości PWM sterowania silnikami.



Rys. 11 – Goldpiny

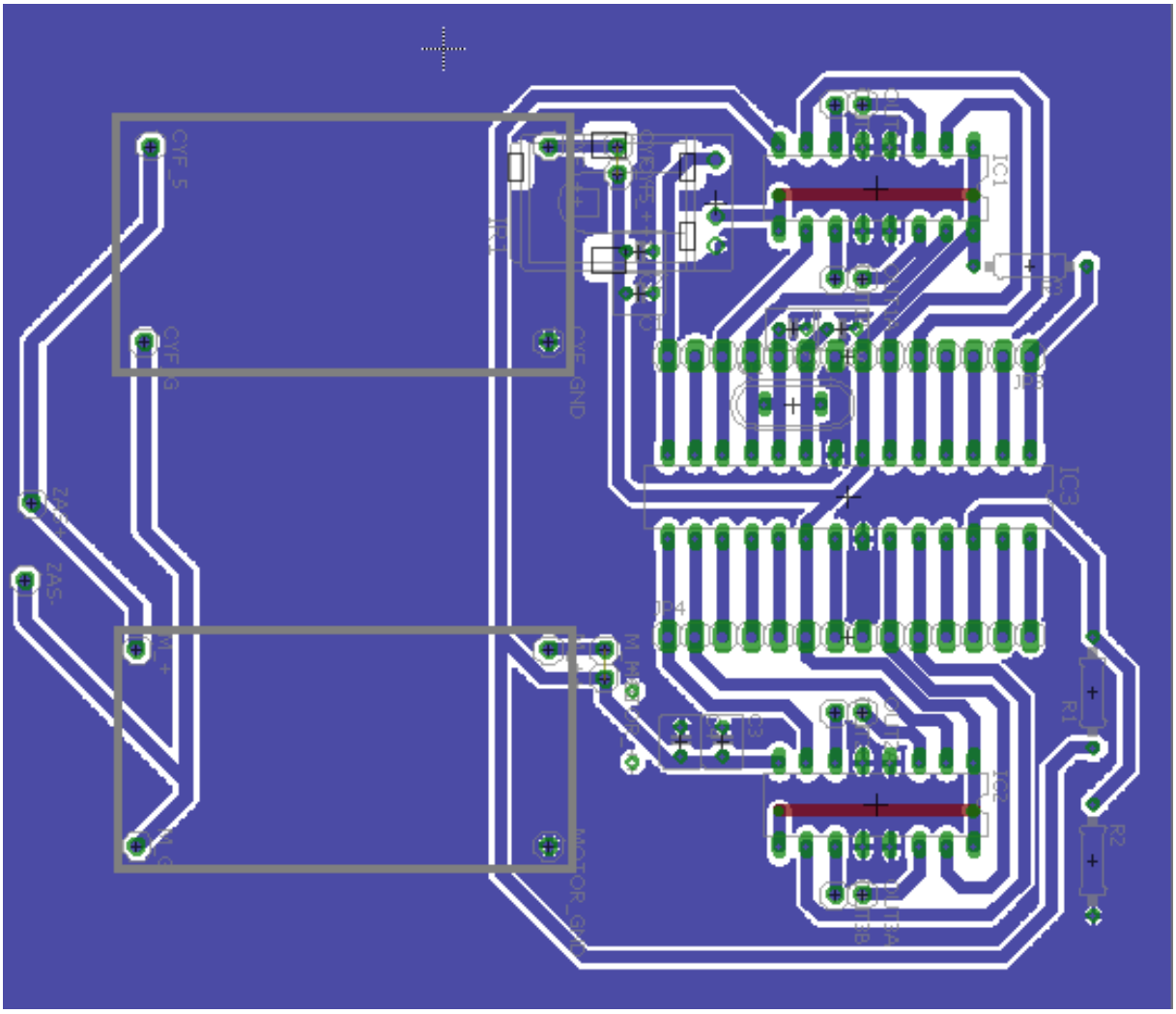
Na płytce drukowanej umieściłem dwie listwy goldpinów. Litwy te to przedłużenie wyprowadzeń mikrokontrolera. Umożliwiają podłączenie programatora do płytki oraz podpięcie dodatkowych elementów gdy będzie taka potrzeba.



Rys. 12 – Zasilanie

Schemat przedstawia część odpowiedzialną za zasilanie całego układu oraz filtrację zasilania. Do wyprowadzeń ZAS+ i ZAS- podłączone jest zasilanie z czterech ogniw litowo-

jonowych. Do pozostałych wyprowadzeń przylutowane są przetwornice oraz zwory umożliwiające odłączenie zasilania.



Rys. 13 – Rozmieszczenie elementów oraz ścieżek na płytce drukowanej

W środkowej prawej części powyższego rysunku znajduje się mikrokontroler Atmega8. Wyżej oraz poniżej uC znajdują się listwy goldpinów. Nad listwami i poniżej nich znajdują się drivery silników. W lewej części płytki znajdują się część odpowiedzialna za zasilanie płytki.

3. Implementacja programu na mikrokontroler

Dalsza część projektu została zrealizowana w programie Eclipse. Poniższe rozdziały zawierają kod napisany w języku C. Cały program jest rozdzielony na składowe pliki programowe i nagłówkowe..

a) Program główny – main.c

```
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>    //import bibliotek z toolchaina

#include "IR_ODBIORNIK/ir_odbiornik.h"
#include "motor/motor.h"      //import pozostałych plików programu
```

```
volatile uint8_t pwm = 100;

//funkcja inicjalizująca timer2
void Timer2_init(void)
{
	TCCR2|= (1<<WGM21);	//CTC
	TCCR2|= (1<<CS22);	//prescaler 64
	TIMSK|= OCIE2;		//zezwolenie przerwania od Timer0

	OCR2 = 1;		//f_przerwania_timera2 = 100kHz
}

int main(void)
{
	cli();			//zablokój przerwania
	ir_init();		//inicjalizacja modolu odczyttu RC5
	lcd_init();
	Timer2_init();	//Inicjalizacja Timera2
	sei();			//odblokój przerwania

	while(1)
	{
		if(IR_flaga)	//sprawdź czy odebrano dane RC5
		{
			ir2motors(IR_komenda);	//wysterój pojazd
			IR_flaga = 0;		//wyczyśc flage odbioru danch RC5
			_delay_ms(200);		//czekaj 200ms
		}
		else	//jeśli nie odebrano żadnych danych w czasie 200ms
		{
			ruch_pojazdu(stop);	//zatrzymaj pojazd
		}
	}

}


//---------------------------------------- Przerwanie timer2
ISR(TIMER2_COMP_vect)
{
	static uint8_t cnt = 0;
	cnt++;
	if(cnt >= 101) cnt = 0;			//licznik zliczajacy w góre 0-100

	if(cnt < pwm) ruch_pojazdu(stop);
	else ir2motors(IR_komenda);		//realizacja programowa PWM
}
```

b) Odbiór kodów w podczerwieni – ir.c

```
#include <avr/io.h>
#include <avr/interrupt.h>	//import bibliotek z toolchaina

#include "ir.h"
#include "../makra.h"		//import pozostałych plików programu

volatile uint8_t IR_flaga;	//flaga informująca że zostały odebrane dane

//odebrane wartości z pilota
volatile uint8_t IR_adres;	//adres urządzenia
```

```
volatile uint8_t IR_komenda;  //komenda z urządenia
volatile uint8_t IR_toggle;   //znak toggle

volatile uint8_t irCnt;       //licznik odebranych bitów

//inicjalizacja peryferiow odbierania danch przez RC5
void ir_init()
{
      DDR(IR_PORT) &= ~IR_IN;       // pin jako wejście
      PORT(IR_PORT) |= IR_IN;       // podciągnięcie pinu do VCC
      TCCR1B |= (1<<CS11)|(1<<CS10); //prescaler 64

      TCCR1B &= ~(1<<ICES1);        //ICR zbocze opadające
      irCnt = 0;                    //licznik wystapien zboczy

      TIMSK |= (1<<TICIE1);         //wlaczenie przerwan od ICR
      IR_flaga = 0;                 //flaga, czy dane odebrano
}



// procedura obsługi przerwania ICR1
ISR(TIMER1_CAPT_vect)
{
      //makra okreslające status odebranej ramki danych
      #define IR_RESTART 0
      #define IR_OK 1
      #define IR_KONIEC 2
      #define IR_BLAD 3

      //zmienne dotyczace dlogosci impulsu, wartosci i statusu ramki
      static uint16_t ostatniaWar;
      uint16_t szerokoscIr;
      static uint16_t statusIr;
      static uint16_t IrSzerCnt;
      static uint16_t odebranaIr;


      //szerokosc impulsu, obliczenie
      szerokoscIr = ICR1 - ostatniaWar;
      ostatniaWar = ICR1;

      TCCR1B ^= (1<<ICES1);   //zmiana zbocza wyzwalającego przerwanie ICR

      if (szerokoscIr > MAX_BIT) irCnt = 0;  //jeśli blednie odczytany bit

      if (irCnt > 0) statusIr = IR_OK;    //licznik wiekszy od zero => OK

      if (irCnt == 0)   //pomyslnie odebrana ramka danych
      {
            TCCR1B |= (1<<ICES1);
            IrSzerCnt = 0;
            odebranaIr = 0;
            irCnt++;
            statusIr = IR_KONIEC;
      }

      //jesli status jest gotowy do odbioru danych ...
      if (statusIr == IR_OK)
      {
            // restart odbioru jeśli czas na jeden pol bit jest zbyt krotki
```

```
            if(szerokoscIr < MIN_POL_BITU) statusIr = IR_RESTART;

            // restart odbioru jeśli czas na jeden pol bit zostanie
            //przekroczony
            if( szerokoscIr > MAX_BIT ) statusIr = IR_RESTART;

            //jeśli szerokośc pol bitu miesci sie w ramach czasowych to OK
            if (statusIr == IR_OK)
            {
                  //inkrementacja licznika pol bitow
                  if (szerokoscIr > MAX_POL_BITU) irCnt++;

                  //dekodowanie bitow
                  if (irCnt > 1)
                  if ((irCnt % 2) == 0)
                  {
                        //zapisywanie w zmiennej odebranaIr
                        odebranaIr = odebranaIr << 1;
                        if((TCCR1B & (1<<ICES1))) odebranaIr |= 1;
                        IrSzerCnt++;

                        //jeśli odebrano poprawnie całą ramke danych
                        if (IrSzerCnt > 12)
                        {
                        //jesli wczesniejsze dane zostaly odczytane w
                        //programie
                        if (IR_flaga == 0)
                        {
                        //rozdzielenie odebranaIR na składowe
                        IR_komenda = odebranaIr & 0b0000000000111111;
                        IR_adres = (odebranaIr & 0b0000011111000000)>>6;
                        IR_toggle = (odebranaIr & 0b0000100000000000)>>11;
                        }
                        statusIr = IR_RESTART;
                        IR_flaga = 1;
                        }
                  }
                  irCnt++;
            }
      }

      //zerowanie licznika przy restarcie
      if (statusIr == IR_RESTART)   {
            irCnt = 0;
            TCCR1B &= ~(1<<ICES1);
      }
}
```

c) Odbiór kodów w podczerwieni – ir.h

```
#ifndef IR_H_
#define IR_H_

#define IR_PORT B             //określenie portu na którym się znajduje ICR
#define IR_PIN 0              //
#define IR_IN (1<<IR_PIN)     //


//konwercja taktowania przerwania na milisekundy
```

```
#define ir_micro_s(dec) ((dec)*(F_CPU/1000000)/TIMER1_PRESCALER)

//określenie min/max dlugosci pol bitu z tolerancja
#define TOLERANCJA 200
#define MIN_POL_BITU    ir_micro_s(889    - TOLERANCJA)
#define MAX_POL_BITU    ir_micro_s(889 + TOLERANCJA)
#define MAX_BIT ir_micro_s((889+889) + TOLERANCJA)

//flaga informująca ze dane sa gotowe do odbioru
extern volatile uint8_t IR_flaga;

//składowe odebrane przez RC5
extern volatile uint8_t IR_toggle;        // bit TOGGLE
extern volatile uint8_t IR_adres;         // adres
extern volatile uint8_t IR_komenda;       // komenda

//inicjalizacja modolu
void ir_init();         //funkcja inicjalizujaca


#endif /* IR_H_ */
```

d) Sterowanie ruchem silników – motor.c

```
#include <avr/io.h>
#include "motor.h"
#include "../makra.h"   //import pozostałych plików programu

// położenie silników oraz przyjęty kierunek
// o - kropka narysowana markerem na ramie
//          o[0]  [1]
//
//
//
//       [2]      [3]

volatile uint8_t pwm;   //wartośc sygnału PWM

//ruch silnika M0 kierunek lewo/prawo
void M0_ruch(ruch_silnika kierunek)
{
      if(kierunek == motorPrawo)
      {
            PORT(M0_prawo_port) |= (1<<M0_prawo);
            PORT(M0_lewo_port) &= ~(1<<M0_lewo);
      }
      else if(kierunek == motorLewo)
      {
            PORT(M0_lewo_port) |= (1<<M0_lewo);
            PORT(M0_prawo_port) &= ~(1<<M0_prawo);
      }
      else
      {
            PORT(M0_lewo_port) &= ~(1<<M0_lewo);
            PORT(M0_prawo_port) &= ~(1<<M0_prawo);
      }
}

//ruch silnika M1 kierunek lewo/prawo
void M1_ruch(ruch_silnika kierunek)
{
```

```
	if(kierunek == motorPrawo)
	{
		PORT(M1_prawo_port) |= (1<<M1_prawo);
		PORT(M1_lewo_port) &= ~(1<<M1_lewo);
	}
	else if(kierunek == motorLewo)
	{
		PORT(M1_lewo_port) |= (1<<M1_lewo);
		PORT(M1_prawo_port) &= ~(1<<M1_prawo);
	}
	else
	{
		PORT(M1_lewo_port) &= ~(1<<M1_lewo);
		PORT(M1_prawo_port) &= ~(1<<M1_prawo);
	}
}

//ruch silnika M2 kierunek lewo/prawo
void M2_ruch(ruch_silnika kierunek)
{
	if(kierunek == motorPrawo)
	{
		PORT(M2_prawo_port) |= (1<<M2_prawo);
		PORT(M2_lewo_port) &= ~(1<<M2_lewo);
	}
	else if(kierunek == motorLewo)
	{
		PORT(M2_lewo_port) |= (1<<M2_lewo);
		PORT(M2_prawo_port) &= ~(1<<M2_prawo);
	}
	else
	{
		PORT(M2_lewo_port) &= ~(1<<M2_lewo);
		PORT(M2_prawo_port) &= ~(1<<M2_prawo);
	}
}

//ruch silnika M3 kierunek lewo/prawo
void M3_ruch(ruch_silnika kierunek)
{
	if(kierunek == motorPrawo)
	{
		PORT(M3_prawo_port) |= (1<<M3_prawo);
		PORT(M3_lewo_port) &= ~(1<<M3_lewo);
	}
	else if(kierunek == motorLewo)
	{
		PORT(M3_lewo_port) |= (1<<M3_lewo);
		PORT(M3_prawo_port) &= ~(1<<M3_prawo);
	}
	else
	{
		PORT(M3_lewo_port) &= ~(1<<M3_lewo);
		PORT(M3_prawo_port) &= ~(1<<M3_prawo);
	}
}

//ruch całego pojazdu, funkcja steruje czterema silnikami
void ruch_pojazdu(ruch_silnikow kierunek)
{
	switch(kierunek)
```

```
{
case przod:
      M0_ruch(motorLewo);
      M1_ruch(motorLewo);
      M2_ruch(motorLewo);
      M3_ruch(motorLewo);
      break;
case tyl:
      M0_ruch(motorPrawo);
      M1_ruch(motorPrawo);
      M2_ruch(motorPrawo);
      M3_ruch(motorPrawo);
      break;
case lewo:
      M0_ruch(motorPrawo);
      M1_ruch(motorLewo);
      M2_ruch(motorLewo);
      M3_ruch(motorPrawo);
      break;
case prawo:
      M0_ruch(motorLewo);
      M1_ruch(motorPrawo);
      M2_ruch(motorPrawo);
      M3_ruch(motorLewo);
      break;
case przodPrawo:
      M0_ruch(motorLewo);
      M1_ruch(motorStop);
      M2_ruch(motorStop);
      M3_ruch(motorLewo);
      break;
case przodLewo:
      M0_ruch(motorPrawo);
      M1_ruch(motorStop);
      M2_ruch(motorStop);
      M3_ruch(motorPrawo);
      break;
case tylPrawo:
      M0_ruch(motorStop);
      M1_ruch(motorLewo);
      M2_ruch(motorLewo);
      M3_ruch(motorStop);
      break;
case tylLewo:
      M0_ruch(motorStop);
      M1_ruch(motorPrawo);
      M2_ruch(motorPrawo);
      M3_ruch(motorStop);
      break;
case obrotPrawo:
      M0_ruch(motorLewo);
      M1_ruch(motorPrawo);
      M2_ruch(motorLewo);
      M3_ruch(motorPrawo);
      break;
case obrotLewo:
      M0_ruch(motorPrawo);
      M1_ruch(motorLewo);
      M2_ruch(motorPrawo);
      M3_ruch(motorLewo);
      break;
```

```
	case stop:
		M0_ruch(motorStop);
		M1_ruch(motorStop);
		M2_ruch(motorStop);
		M3_ruch(motorStop);
		break;
	}
}

//przepisanie odczytanego kodu podczerwieni do ruchu pojazdu
void ir2motors(uint8_t war_ir)
{
	switch(war_ir)
	{
	case 32: ruch_pojazdu(przod); break;
	case 33: ruch_pojazdu(tyl); break;
	case 17: ruch_pojazdu(obrotLewo); break;
	case 16: ruch_pojazdu(obrotPrawo); break;

	case 2: ruch_pojazdu(przod); break;
	case 8: ruch_pojazdu(tyl); break;
	case 4: ruch_pojazdu(lewo); break;
	case 6: ruch_pojazdu(prawo); break;

	case 1: ruch_pojazdu(przodLewo); break;
	case 3: ruch_pojazdu(przodPrawo); break;
	case 7: ruch_pojazdu(tylLewo); break;
	case 9: ruch_pojazdu(tylPrawo); break;

	case 38: pwm += 10; if(pwm > 100) pwm = 0; break;
	case 15: pwm -= 10; if(pwm > 240) pwm = 100; break;

	case 59: ruch_pojazdu(stop); break;
	}
}
```

e) Sterowanie ruchem silników – motor.h

```
#ifndef MOTOR_MOTOR_H_
#define MOTOR_MOTOR_H_

//położenie silników wzglendem ramy
//           [0]  [2]
//
//           [1]   o[3]

//zdefiniowanie wyprowadzeń MCU dla sterowania silnikiem M0
#define M0_lewo_port D
#define M0_lewo 4
#define M0_prawo_port D
#define M0_prawo 3

//zdefiniowanie wyprowadzeń MCU dla sterowania silnikiem M1
#define M1_lewo_port D
#define M1_lewo 5
#define M1_prawo_port D
#define M1_prawo 6

//zdefiniowanie wyprowadzeń MCU dla sterowania silnikiem M2
#define M2_lewo_port B
#define M2_lewo 1
```

```
#define M2_prawo_port B
#define M2_prawo 2

//zdefiniowanie wyprowadzeń MCU dla sterowania silnikiem M3
#define M3_lewo_port C
#define M3_lewo 0
#define M3_prawo_port C
#define M3_prawo 1

//definicja enumeratora dla ruchu pojedynczym silnikiem
typedef enum
{
	motorPrawo, motorLewo, motorStop
} ruch_silnika;

//definicja enumeratora dla ruchu całego pojazdu
typedef enum
{
	przod, tyl, lewo, prawo, przodLewo, przodPrawo, tylLewo, tylPrawo,
	obrotLewo, obrotPrawo, stop
} ruch_silnikow;

extern volatile uint8_t pwm;  //wartośc sygnału PWM

void M0_ruch(ruch_silnika kierunek);//ruch silnika M0 kierunek lewo/prawo
void M1_ruch(ruch_silnika kierunek);//ruch silnika M1 kierunek lewo/prawo
void M2_ruch(ruch_silnika kierunek);//ruch silnika M2 kierunek lewo/prawo
void M3_ruch(ruch_silnika kierunek);//ruch silnika M3 kierunek lewo/prawo

void ruch_pojazdu(ruch_silnikow kierunek);//ruch całego pojazdu

//przepisanie odczytanego kodu podczerwieni do ruchu pojazdu
void ir2motors(uint8_t war_ir);

#endif /* MOTOR_MOTOR_H_ */
```

f) Makra upraszczające dostęp do portów – make.h

```
#ifndef MAKRA_H_
#define MAKRA_H_

//Makra upraszczające dostęp do rejestru PORTx
#define PORT(x) ZPORT(x)
#define ZPORT(x) (PORT##x)

//Makra upraszczające dostęp do rejestru PINx
#define PIN(x) ZPIN(x)
#define ZPIN(x) (PIN##x)

//Makra upraszczające dostęp do rejestru DDRx
#define DDR(x) ZDDR(x)
#define ZDDR(x) (DDR##x)

#endif /* MAKRA_H_ */
```

4. PODSUMOWANIE

Podczas realizacji projektu inżynierskiego, dokonano następujących czynności:

- Wykonano samodzielnie program na mikrokontroler Atmega8 w języku C w programie Eclipse. Program spełnia założenia projektowe opisane w rozdziale 1.2. Cel projektu. Nie zrealizowano obsługi przetwornika analogowo-cyfrowego na mikrokontrolerze. Ta funkcjonalność okazała się zbędna. Generowany przebieg PWM nie zależał w żadnym stopniu od napięcia zasilania silników.
- Wykonano samodzielnie płytkę drukowaną która ma służyć jako sterownik pojazdu. Płytka była projektowana w programie Eagle. Wykonano ją na laminacie pokrytym miedzią.
- Dokonano lutowania elementów elektronicznych na płytkę drukowaną. Lutowania dokonano własnoręcznie.

Wykonana praca przyczyniła się do skonstruowania obiektu jakim jest zdalnie sterowany samochód. Zrealizowano wszystkie elementy projektowe opisane w rozdziale 1.2. Cel projektu (z wyjątkiem programowej obsługi przetwornika cyfrowo-analogowego która okazała się zbędna).

5. WNIOSKI

Do realizacji projektu inżynierskiego wykorzystano wiedzę z następujących dziedzin:

- **Elektronika** – należało wykorzystać wiedzę na temat mikrokontrolerów. Należało wiedzieć w jaki sposób zasilić mikrokontroler, w jaki sposób zapewnić odpowiednią filtrację napięcia, należało znać metody taktowania MCU – wykorzystano wewnętrzny oscylator RC, znać budową mikrokontrolera – znać budową procesora (ALU) oraz pozostałych układów peryferyjnych (ADC, timery itd.). Należało umieć się posługiwać dokumentacją mikrokontrolera Atmega8. Należało wiedzieć w jaki sposób działa dzielnik napięcia. Należało wiedzieć w jaki sposób wykorzystać sygnał PWM do sterowania silnikami i jak go wytworzyć. Wykorzystano wiedzę na temat przetwornic DC-DC. Dokonano wyboru odpowiednich przetwornic. Wykorzystano drivery silników DC. Należało tego dokonać, gdyż silniki DC generują dużo zakłóceń. Drivery silników niwelują te zakłócenia i dodatkowo posiadają mostki H które umożliwiają sterowanie silnikami lewo-prawo. Wykorzystano znajomość programu komputerowego Eagle do projektowania oraz wykonano płytkę drukowaną. Dokonano lutowania elementów elektronicznych
- **Informatyka** – należało wykorzystać znajomość języka C. Wykorzystać jego składnię. Znać typy zmiennych. Znać podstawową budowę programów na mikrokontrolery: funkcja main() z pętlą nieskończoną while(1). Należało potrafić programowo włączyć i skonfigurować przerwania. Znać typy plików: programowych (.c) i nagłówkowych (.h). Znać zagadnienia dotyczące: kompilacji, asemblacji i konsolidacji. Należało potrafić skonfigurować środowisko programistyczne w programie Eclipse.
- **Mechanika** – należało rozpatrzyć jaki napęd wykorzystać w pojeździe. Wykorzystane koła typu Mecanum pozwalają na jazdę pojazdu w różnych kierunkach. Zapewniają jazdę pojazdu: do przodu, do tyłu, w lewo, w prawo, obrót lewo/prawo, jazdę po skosie.

Wykorzystując trzy powyższe dziedziny można śmiało stwierdzić że projekt spełnia zagadnienia mechatroniki.

6. ZAWARTOŚĆ PŁYTY CD

Na płycie CD dostarczonej do pracy inżynierskiej znajdują się następujące foldery:

- Program Atmega8 – wykonany program na mikrokontroler
- Płytka PCB – pliki dotyczące wykonania płytki drukowanej

7. WYKORZYSTANE ŹRÓDŁA

[1] Życiorys Gordona Moora:
https://en.wikipedia.org/wiki/Gordon_Moore

[2] Prawo Moora – wykres:
https://pl.wikipedia.org/wiki/Prawo_Moore%E2%80%99a#/media/Plik:Moore's_Law_Transistor_Count_1970-2020.png

[3] Opis architektury Harwardzkiej mikrokontrolerów:
https://pl.wikipedia.org/wiki/Architektura_harwardzka

[4] Atmega8 – grafika:
https://cdn.sos.sk/productdata/6e/6a/e03f2d0f/atmega8-16pu.jpg

[5] Atmega8 – datasheet:
https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-2486-8-bit-AVR-microcontroller-ATmega8_L_datasheet.pdf

[6] L239D – datasheet:
https://pdf1.alldatasheet.com/datasheet-pdf/view/22432/STMICROELECTRONICS/L293D.html

[7] Przetwornica 3A – informacje:
https://abc-rc.pl/product-pol-5375-Przetwornica-3A-3-35V-na-1-5-30V-DC-DC-step-down-LM2596-do-FPV.html

[8] TSOP3123 – datasheet:
https://pdf1.alldatasheet.com/datasheet-pdf/view/914139/VISHAY/TSOP31233.html

[9] Opis koła typu Mecanum:
http://www.kms.polsl.pl/mi/pelne_26/01_26_57.pdf

[10] Koła typu Mecanum – grafika:
https://en.wikipedia.org/wiki/Mecanum_wheel

[11] Opis magistrali SPI:
https://pl.wikipedia.org/wiki/Serial_Peripheral_Interface
https://en.wikipedia.org/wiki/Serial_Peripheral_Interface

[12] Opis protokołu RC5:
https://pl.wikipedia.org/wiki/RC5_(RTV)
https://en.wikipedia.org/wiki/RC-5
https://www.turjasuzworld.in/2017/10/19/decode-rc5-protocol-using-tm4c12x-series-mcu/

[13] Opis PWM:
https://en.wikipedia.org/wiki/Pulse-width_modulation