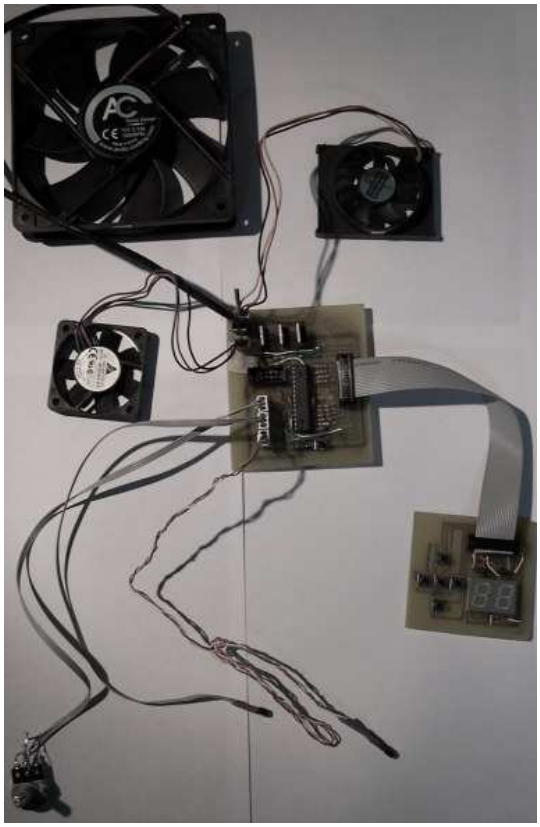
Spis treści: ..... 2

1. Założenia projektowe ..... 3
   1.1. Temat projektu ..... 3
   1.2. Wymagania projektowe ..... 3
2. Realizacja projektu - hardware ..... 4
   2.1. Wybór mikroprocesora oraz urządzeń peryferyjnych ..... 4
   2.2. Schemat ideowy ..... 5
   2.3. Płytka PCB ..... 6
   2.4. Wizualizacja płytki 3D ..... 7
   2.5. Wykonana płytka PCB ..... 8
   2.6. Wykaz elementów oraz kosztorys ..... 9
3. Algorytm sterownika, kod źródłowy C ..... 10
   3.1. Schemat blokowy algorytmu sterownika ..... 10
   3.2. Opis ważniejszych fragmentów kodu ..... 11
4. Możliwe zmiany ..... 13
   4.1. Rozbudowa sterownika ..... 13
   4.2. Optymalizacja kodu ..... 13
5. Załącznik ..... 14
   5.1. Zawartość ..... 14
6. Wnioski ..... 14



# 1. ZAŁOŻENIA PROJEKTOWE

Celem projektu jest wykonanie urządzenia sterowanego mikroprocesorem. Cel projektu obejmuje także wykonanie prototypu oraz jego zaprogramowanie.

## 1.1. Temat projektu

Tematem projektu jest: **system kontroli temperatury.** Sterownik ma za zadanie kontrolować temperaturę urządzeń peryferyjnych różnych układów elektronicznych, np. komputera PC.

Sterownik dedykowany jest dla tzw. „overclocker-ów„ – czyli osób, które zajmują się zmianą ustawień fabrycznych podzespołów w celu uzyskania lepszej wydajności sprzęttowej. Efektem ubocznym takich działań jest nadmierne wydzielanie się ciepła oraz problem z jego odprowadzeniem.

## 1.2. Wymagania projektowe

- Pomiar temperatury w 3-ech miejscach
- Płynna regulacja obrotów 3 wentylatorów – modulacja PWM
- Wyświetlenie temperatur oraz wartości PWM (procentowo)
- Tryb pracy automatycznej – obroty wentylatora zależne od temperatury
- Tryb pracy ręcznej – sterowanie niezależne od temperatury
- Uniwersalność – praca z różnymi urządzeniami, nie tylko PC
- Możliwość podłączenia różnych wentylatorów
- Możliwość szybkiego przeprogramowania układu
- Praca w zakresie temperatur
- Ograniczenie maksymalnych i minimalnych obrotów wentylatorów

## 2. Realizacja projektu – hardware:

Realizacja projektu od strony fizycznej.

### 2.1. Wybór mikroprocesora oraz urządzeń peryferyjnych

Zgodnie z wymaganiami w pkt. 1.2. po przejrzeniu ofert różnych firm elektronicznych wybrane zostały następujące elementy:

➔ Mikrokontroler:

**ATmega8 16-PU** w obudowie dip28:

- 3 kanały PWM
- Przetwornik analogowo-cyfrowy 6 wejść
- Programowanie w układzie – ISP

➔ Czujnik temperatury:
**LM 35 DZ**

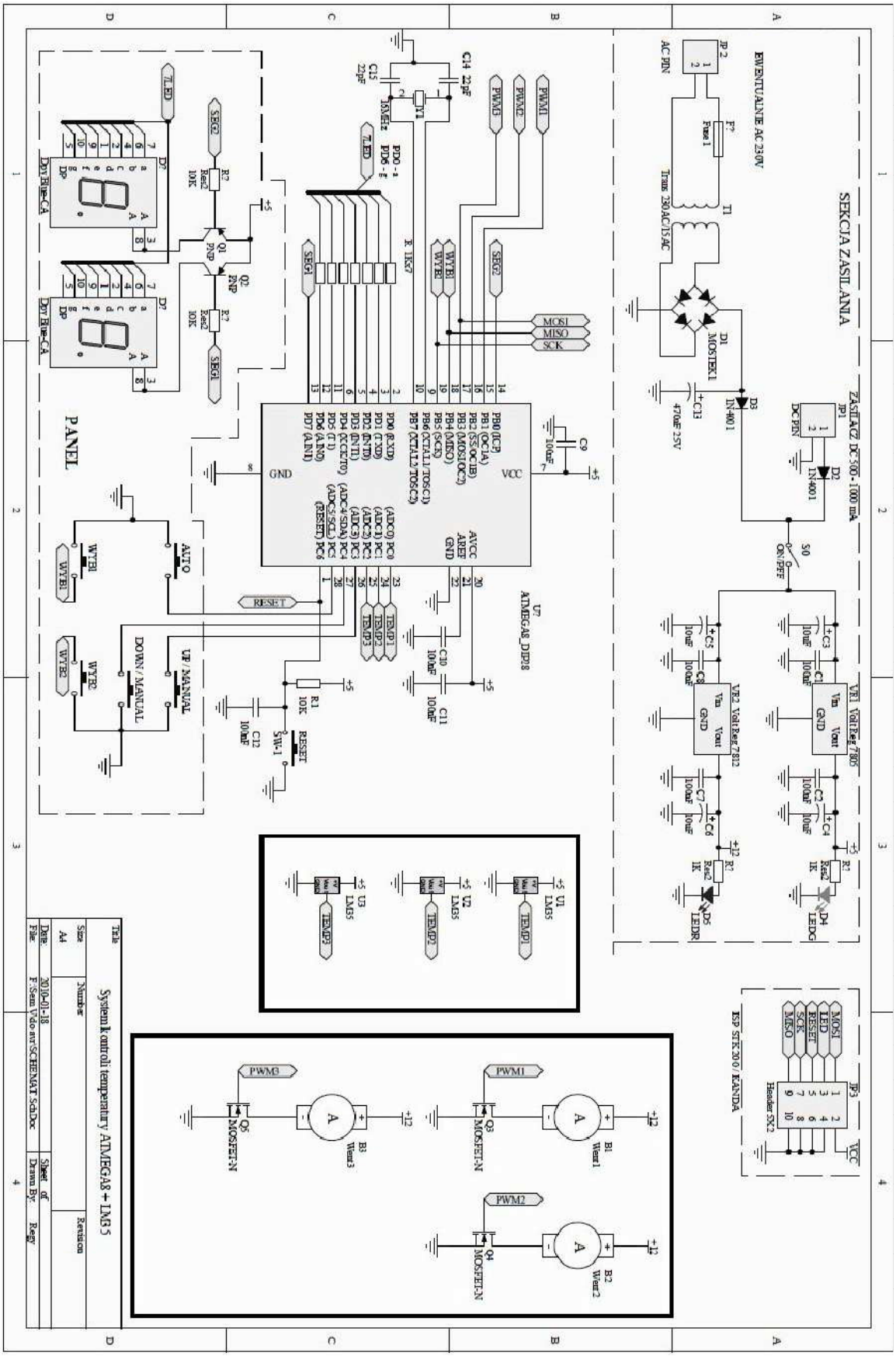
- Zakres 2 ÷ 150 ˚C
- Wyjście analogowe 10 mV/˚C

➔ Pozostałe elementy:

- Tranzystory sterujące MOSFET-N – BUZ 10
- Rezystory 0,25 W
- Wyświetlacz LED – 7 segmentów
- Przyciski – switch-e
- Kondensatory ceramiczne
- Złącza molex oraz złącza taśmowe

## 2.2. Schemat ideowy

Schemat ideowy przedstawia połączenia peryferii do mikrokontrolera, sekcja zasilania jest w tym układzie tylko opcją, sam układ wymaga zasilania 12V oraz 5V.

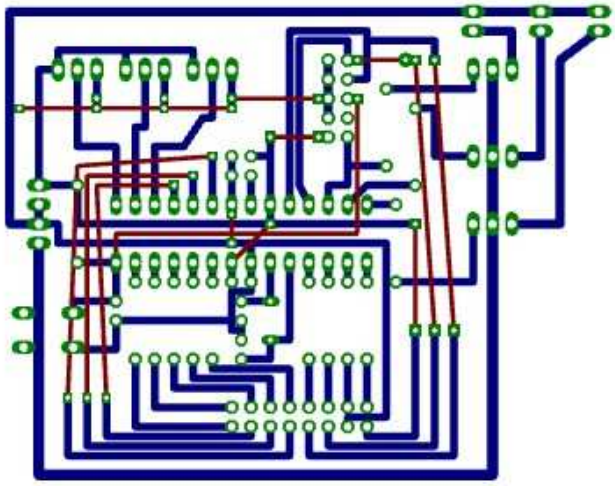


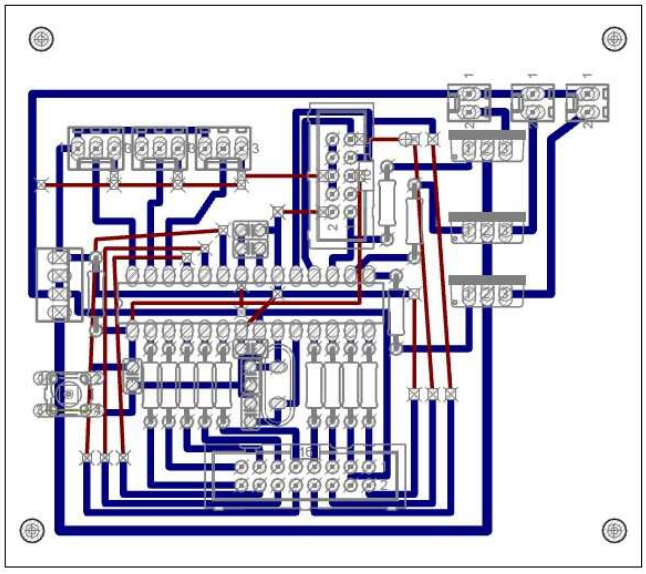
### 2.3. Płytka PCB

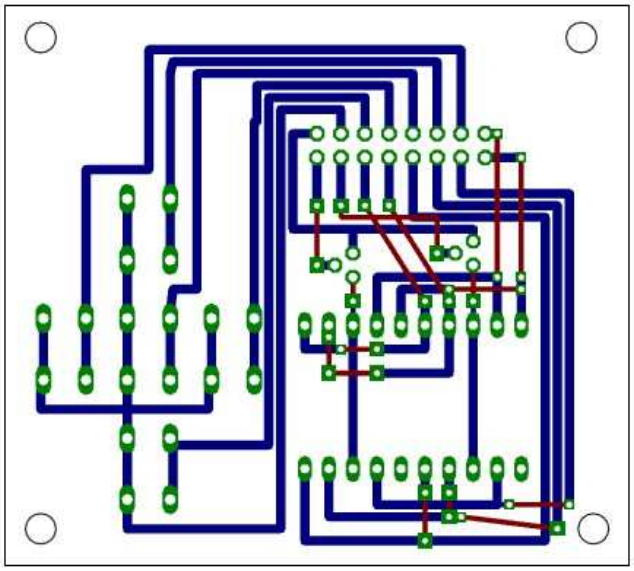
Układ został zaprojektowany na dwóch płytkach PCB połączonych złączem taśmowym. Zdecydowałem się na takie rozwiązanie gdyż układ jest podzielony na panel z wyświetlaczem i przyciskami oraz na układ wykonawczy. Płytki są jednostronne wykonane metodą termo transferu we własnym zakresie. Brak wylanej masy to ułatwienie przy lutowania w celu uniknięcie przypadkowego zwarcia.

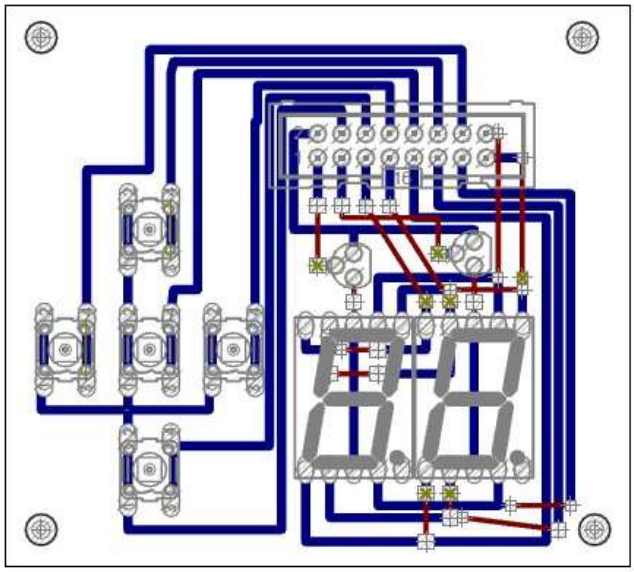
Zworki prowadzone na płytce są wynikiem braku możliwości wykonania płytki dwustronnej. Jest to układ prototypowy, w przyszłości rozwijany, więc płytki jednostronne są tu wystarczające.

Kolor niebieski warstwa BOTTOM
Kolor czerwony warstwa TOP – zworki

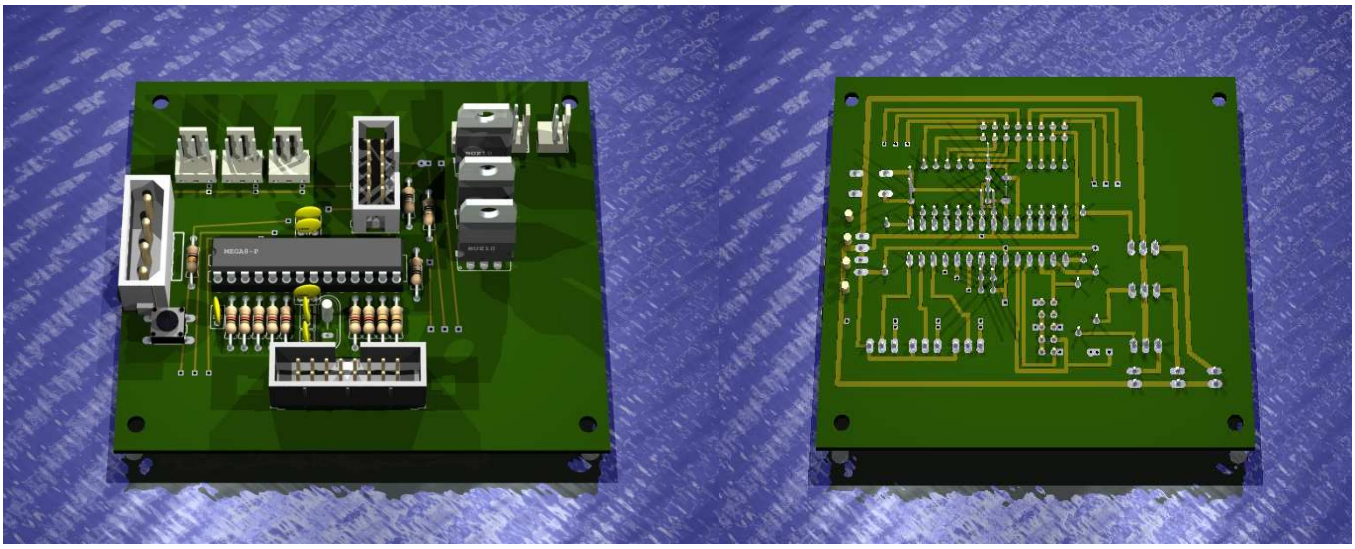


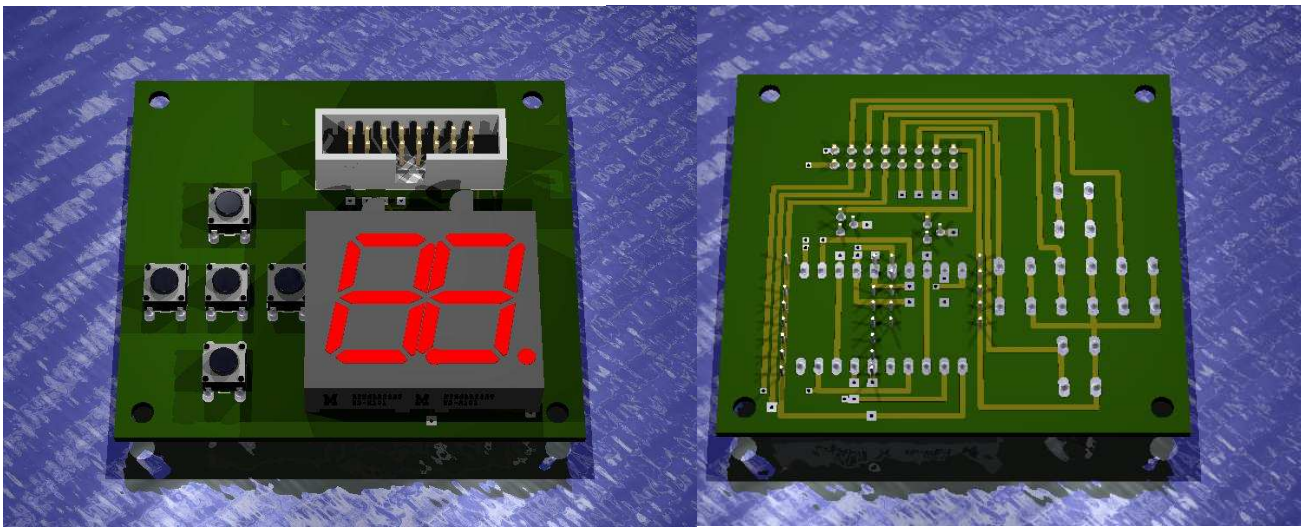






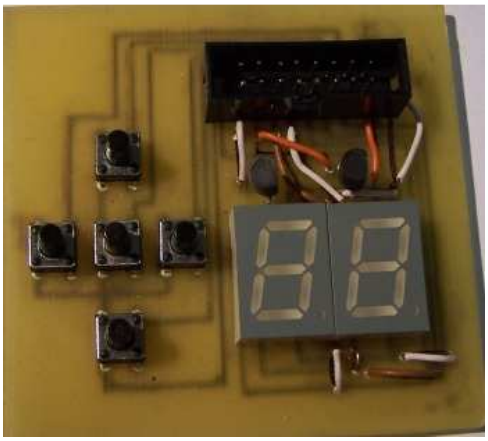
## 2.4.Wizualizacja płytki 3D

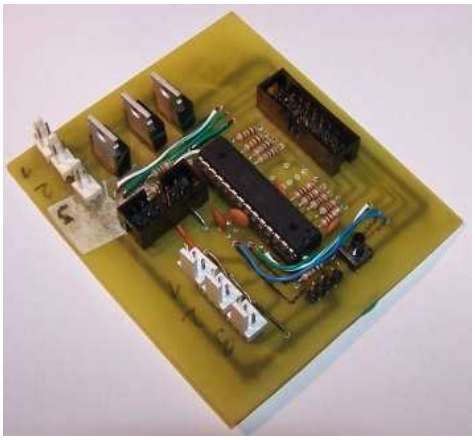


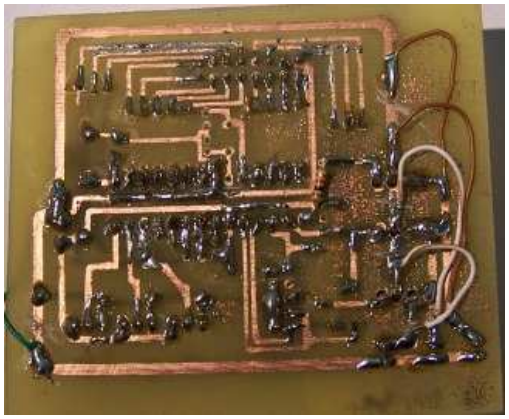


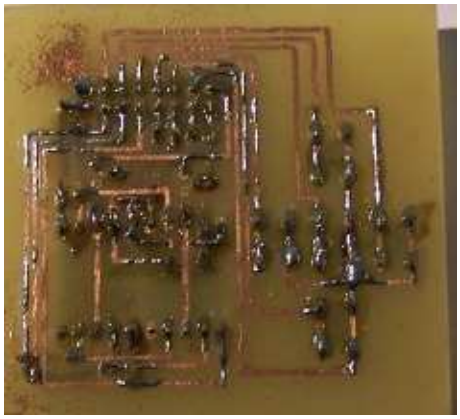
Wizualizacje wykonane za pomocą wtyczki do programu Eagle. Wtyczka dysponuję ograniczoną baza elementów jednak pozwala w pełni zobrazować naszą płytkę przed wykonaniem. Taka wizualizacja pozwala przemyśleć projekt oraz zauważyć błędy projektowe.

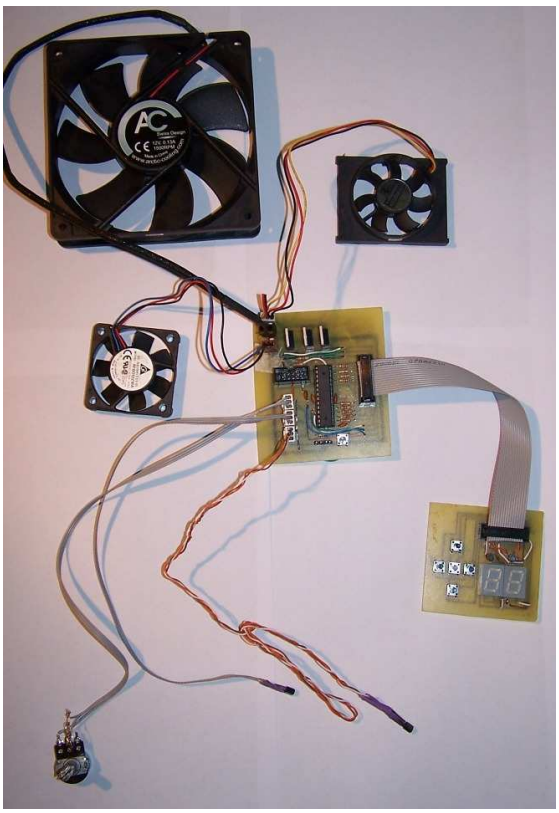
## 2.5. Wykonana płytka PCB











### 2.6.Wykaz elementów i kosztorys

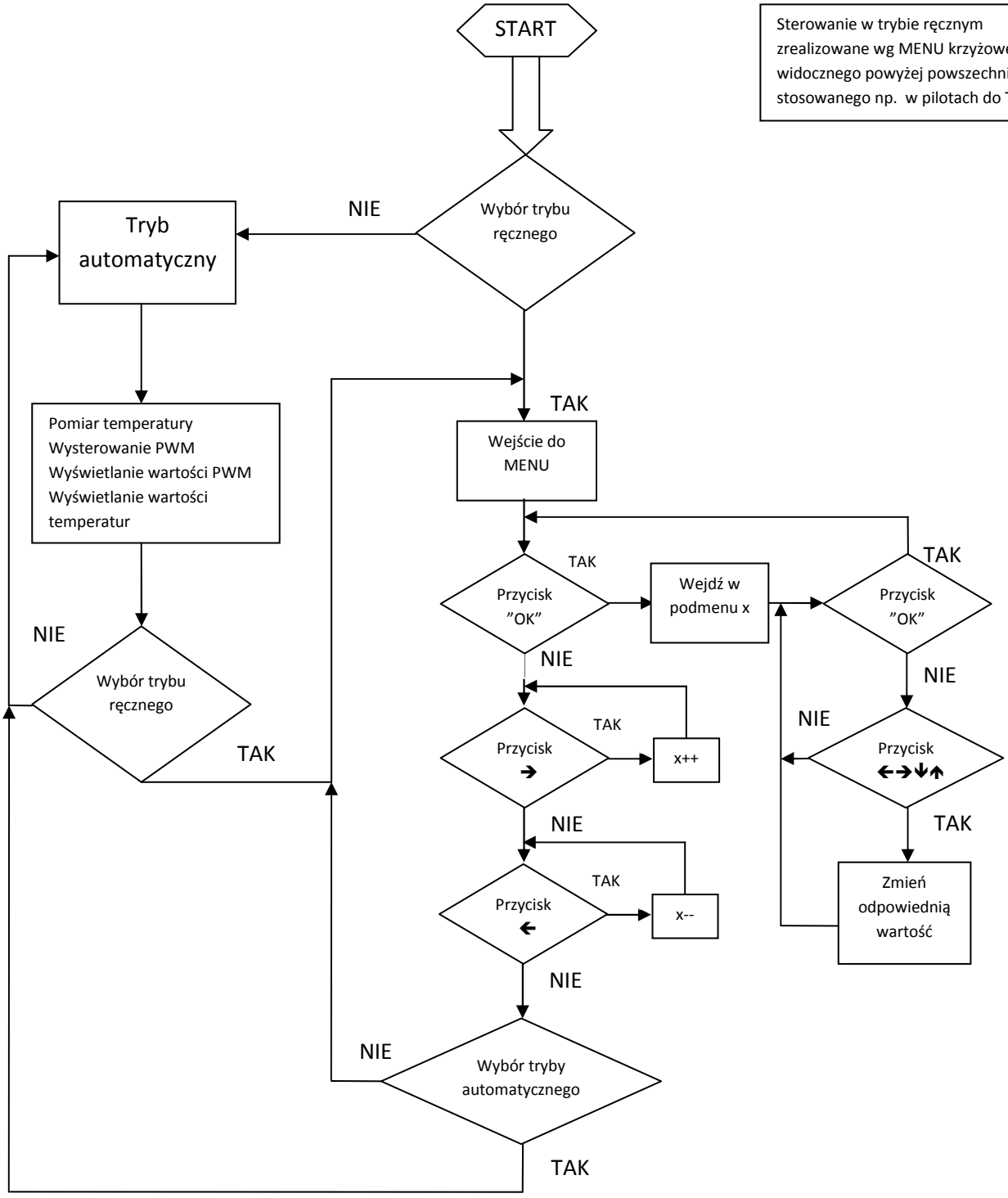
| Nazwa | Ilość | Cena jednostkowa | Cena | Uwagi |
|---|---|---|---|---|
| Atmega 8 16-PU | 1 | 5,50 zł | 5,50 zł | |
| Rezystor 1k | 7 | 0,10 zł | 0,70 zł | |
| Rezystor 10k | 3 | 0,10 zł | 0,30 zł | |
| Kondensator 100n | 4 | 0,10 zł | 0,40 zł | |
| Kondensator 22pf | 2 | 0,10 zł | 0,20 zł | niezamontowany |
| Rezonator | 1 | 1,00 zł | 1,00 zł | niezamontowany |
| BUZ 10 | 3 | 1,50 zł | 4,50 zł | |
| Złącza Molex 2 pin | 3 | 0,25 zł | 0,75 zł | |
| Złącza Molex 3 pin | 3 | 0,25 zł | 0,75 zł | |
| Switch 6mm | 6 | 0,15 zł | 0,90 zł | |
| LED 7 segment. | 2 | 1,00 zł | 2,00 zł | |
| Złącze taśmowe ML10 | 1 | 0,50 zł | 0,50 zł | |
| Złącze taśmowe ML16 | 2 | 0,50 zł | 1,00 zł | |
| Taśma 16 żył | 1 | 0,50 zł | 0,50 zł | |
| Wtyk 16 pin | 2 | 0,50 zł | 1,00 zł | |
| BC 557B | 2 | 0,20 zł | 0,40 zł | |
| Goldpin | 4 | 0,05 zł | 0,20 zł | |
| Płytka PCB | 2 | 2,50 zł | 5,00 zł | |
| | | **Razem:** | **25,60 zł** | |

## 3. Algorytm sterownika, kod źródłowy C

Algorytm zakłada dwa tryby sterowania, ręczny i automatyczny. Program został napisany w języku C w programie AVR Studio z wykorzystaniem środowiska Win AVR.

### 3.1. Schemat blokowy algorytmu


Sterowanie w trybie ręcznym zrealizowane wg MENU krzyżowego widocznego powyżej powszechnie stosowanego np. w pilotach do TV



### 3.2. Opis ważniejszych fragmentów kodu

Program został napisany w języku C. Kod źródłowy w całości znajduje się w załączniku.

➔ Pomiar ADC z trzech wejść:

- 2 funkcje: do inicjalizacji przetwornika oraz to rozpoczęcia pomiaru
- Zrealizowany w przerwaniu generowanym po zakończeniu konwersji
- Zmienna pomocnicza flaga informuje o przypisaniu wartości ADC do zmiennej
- W trybie auto po zakończeniu pomiaru z jednego czujnika zostaje wywołany pomiar z drugiego
- Przetwornik pracuje w trybie „Free running" z częstotliwością próbkowania 125 kHz
- Ograniczenie rozdzielczości do 8 bitów, wyrównanie do lewej oraz zapis najbardziej znaczących bitów

```
if((flaga & _BV(0))&&(we==0x04))
{
        temp2 = pomiar;
        we=0x05;
        flaga &= ~_BV(0);    //wyzerowanie flagi
        ADC_start(we);
 }
```

Powyższy warunek sprawdza czy został wybrany czujnik na wejściu ADC 4 oraz stan flagi która informuje o końcu przerwania. Spełnienie tych warunków powoduje przypisanie wartości ADC pod zmienna pomiar do zmiennej temp2, która odpowiada czujnikowi nr 2. Dalej zostaje wybrane następne wejście – ADC 5, wyzerowanie flagi, oraz uruchomienie następnego pomiaru.

```
ISR(ADC_vect)   //przerwanie od ADC
{
pomiar = ADCH;
cbi(ADCSRA,ADSC); // zatrzymanie konwersji
flaga |= _BV(0);
}
```

Powyższy fragment to przerwanie od przetwornika ADC, następuje zapis wartości rejestru ADCH – 8 najstarszych bitów do zmiennej pomiar. Następnie zatrzymujemy konwersję i ustawiamy flagę w stan wysoki, która jest informacją o zakończeniu przerwania.

➔ Multipleksowanie wyświetlaczy LED:

- Zrealizowane w przerwaniu z wykorzystanie TIMER-a.
- Możliwość zmiany częstotliwości multipleksowania w programie
- Łatwa rozbudowa dla multipleksowania większej ilości

Fragment kodu źródłowego przedstawia przerwanie od przepełnienia licznika. Multipleksowanie zrealizowane jest na prostej instrukcji *switch()*, która co przerwanie zapala inny wyświetlacz. Dla dwóch segmentów wystarczą dwa warunki *case* łatwo więc zauważyć, że aby zwiększyć ilość multipleksowanych wyświetlaczy wystarczy dodać następny warunek.

Funkcja *led7()* odpowiada za zapalenie odpowiednich segmentów wyświetlacza. Najpierw zapalony jest wyświetlacz nr 1 z odpowiednią dla niego wartością, a w kolejnym przerwaniu zapalony jest wyświetlacz nr 2 z następną wartością.

```
ISR(TIMER0_OVF_vect)
{
 TCNT0=timer;
  switch (w)
        {
                case 0:
                                PORTD =0xFF;    // wyzerowanie portu D - zgaszenie wszystkich lini
                                sbi(PORTB,0); //zgaszenie segmentu 2
                                led7(x);
                                cbi(PORTD,7); // zapalenie segment 1
                                w++;
                break;

                case 1:
                                PORTD =0xFF;
                                sbi(PORTD,7); ///zgaszenie segmentu 1
                                led7(y);
                                cbi(PORTB,0);    // zapalenie segment 2
                                w=0;
                break;
        }
}
```

➔ Praca sterownika w zakresie temperatur oraz przy dolnej i górnej granicy obrotów:

- Możliwość zmiany zakresu
- Dostosowanie do różnego rodzaju wentylatorów (różne napięcia pracy)

Przykład: Chcemy aby nasz sterownik kontrolował obroty wentylatora w zakresie 20 – 30 ˚C, oraz wiemy, że wentylator pracuje w zakresie napięć 7-12 V. Poniżej 20˚C wentylator jest wyłączony, a powyżej 30˚C pracuje na maksymalnych obrotach, w zakresie 20 – 30 stopni następuje zmiana obrotów proporcjonalnie do temperatury.

pwmax – wartość odpowiadająca maksymalnym obrotom przy danym napięciu (12V)
pwmmin - wartość odpowiadająca minimalnym obrotom przy danym napięciu (7V)
tempmin - wartość odpowiadająca dolnej granicy pracy (20˚C)
tempmax - wartość odpowiadająca górnej granicy pracy (30˚C)
temp – zmierzona wartość temperatury

Wartość wypełnienia PWM określona jest wzorem:

$$\mathbf{PWM} = \frac{\mathbf{pwmmax} - \mathbf{pwmmin}}{\mathbf{tempmax} - \mathbf{tempmin}} * (\mathbf{temp} - \mathbf{tempmin}) + \mathbf{pwmmin}$$

Np. pwmax = 255; pwmmin = 200; zmierzona temperatura 25˚C

$$PWM = \frac{255 - 200}{30 - 20} * (25 - 20) + 200 = 227{,}5$$

Dla temp 26˚C PWM będzie już równe 233.

Kod źródłowy odpowiadający zapisowi powyższego równania.

temp3 – zmierzona wartość z czujnika

```
if(temp3<20){pwm3=0;}                    // wyłączenie wentylatora
else if((temp3>=20)&&(temp3<=30))
{
pwmobl=0;                                // zmienne pomocnicze
pwmobl2=0;

pwmobl=pwmmax2-pwmmin2;
pwmobl=pwmobl/10;
pwmobl2=temp3-20;
pwmobl=pwmobl*pwmobl2;
pwmobl=pwmobl+pwmmin2;
pwm3=pwmobl;
}
else if(temp3>30){pwm3=255;}             // maksymalne obroty
```

# 4. Możliwe zmiany

Sterownik został wykonany z zamiarem jego dalszego udoskonalania. Spełniane funkcję są dalekie od rozwiązań jakie proponują firmowe sterowniki, dlatego układ został wykonany na płytce prototypowej z możliwością programowania w układzie. Pozwala to na szybką zmianę kodu i przeprogramowanie urządzenia.

## 4.1.Rozbudowa sterownika

W następnych wersjach sterownik może zostać rozbudowany w celu usprawnienia jego funkcjonalności oraz zwiększenia możliwości sprzętowych.

Założenia rozbudowy:

- Zwiększenie liczby czujników pomiarowych
- Zwiększenie liczby sterowanych wentylatorów
- Zmiana mikroprocesora na posiadający interfejs JTAG
- Zmiana wyświetlacza LED na LCD
- Dodanie systemu alarmowego, buzzer, diody sygnalizacyjne
- Możliwość pomiaru obrotów poszczególnych wentylatorów – sprzężenie zwrotne
- Minimalizacja ilości połączeń
- Układ wykonany na jednej dwustronnej płytce
- Użycie interfejsu 1-wire dla czujników pomiarowych
- Możliwość programowania przyciskami i zapisu danych do pamięci EEprom

## 4.2.Optymalizacja kodu

Założenia:

- Skrócenie kodu źródłowego
- Stworzenie własnych bibliotek
- Stworzenie funkcji dla powtarzających się elementów programowych
- Lepsze wykorzystanie możliwości programowych języka C