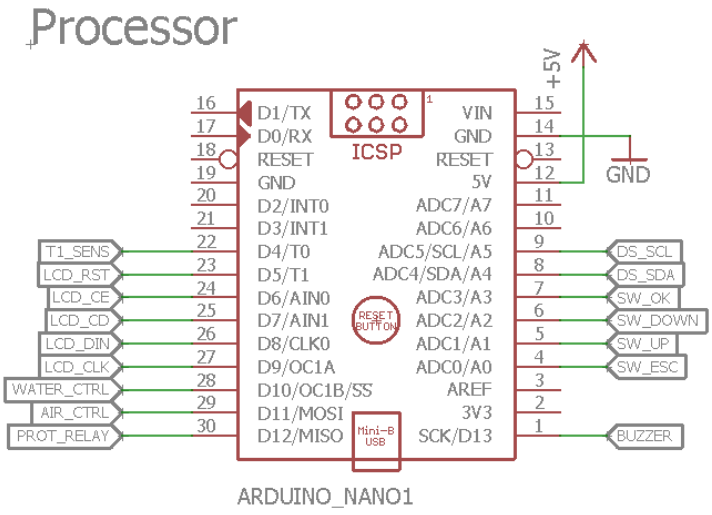
6. Projekt sprzętowy regulatora

6.1 Założenia projektowe

Głównym założeniem projektowym jest zbudowanie prostego regulatora sterującego pracą wodnego kotła centralnego ogrzewania opalanego węglem. Projekt opiera się o wykorzystanie mikrokontrolera jako głównego układu sterującego oraz implementacji jednego z przedstawionych wcześniej algorytmów. Urządzenie ma mieć możliwość zadawania temperatury, oraz możliwość podglądu aktualnej temperatury. Sterownik ma współpracować z dmuchawą zasilaną napięciem sieciowym i sterować jej pracą. Ma ono również przewidywać możliwość sterowania pompą wody znajdującej się w obiegu systemu grzewczego. Dodatkowym zadaniem jest zabezpieczenie przed nadmiernym wzrostem temperatury znajdującej się w obiegu. Wszelkie parametry pracy mają być dostępne dla użytkownika w czasie rzeczywistym. Pełny schemat sterownika znajduje się w załączniku pierwszym.

6.2 Dobór mikrokontrolera

Jako główny układ sterujący użyty został ośmiobitowy mikrokontroler firmy Atmel z serii ATmega o oznaczeniu Atmega328P. Posiada on 23 programowalne porty wejścia/wyjścia, trzy sprzętowe liczniki, w tym jeden 16-bitowy, pozwalające na zrealizowanie sterowania mocą poprzez modulację szerokości impulsu, co umożliwia regulację mocy dmuchawy kotła oraz może być taktowany z maksymalną częstotliwością 16MHz. Pamięć flash dostępna w rozmiarze 32 KB oraz 2KB pamięci RAM w zupełności wystarcza na implementację algorytmów sterowania, interfejsu użytkownika oraz zabezpieczeń. Dodatkowa pamięć EEPROM pozwala na zapamiętywanie ustawień sterownika po wyłączeniu zasilania i ponowne ich wczytanie przy włączaniu. Aby mieć łatwą możliwość programowania mikrokontrolera użyty został cały moduł Arduino Nano, który posiada wbudowany konwerter magistrali USB do RS232, oraz dostępne są na nim elementy niezbędne do prawidłowej pracy mikrokontrolera takie jak na przykład zewnętrzny rezonator kwarcowy 16MHz. Wbudowany w moduł konwerter pozwala na bezpośrednie programowanie ze złącza USB, które jest dostępne na samym module. Dodatkowo daje możliwość debugowania programu poprzez terminal połączony z portem COM komputera. Schemat połączeń przedstawiony został na rysunku 6.1.

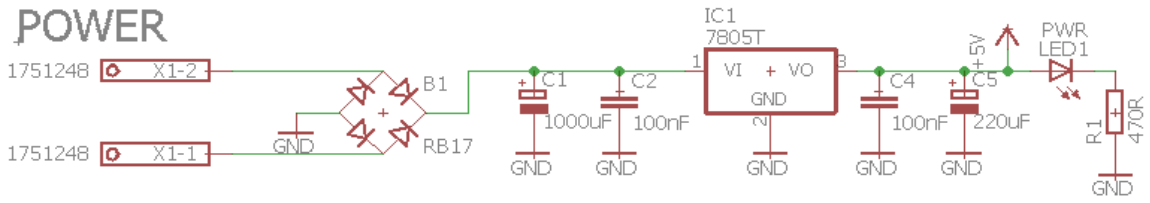


Rysunek 6.1. Schemat wyprowadzeń mikrokontrolera.

Na schemacie połączeń widoczne jest, że do obsługi peryferii została wykorzystana większość linii wejścia/wyjścia. Do wejść A0-A3 dołączone zostały 4 przyciski, które odpowiadają za komunikację pomiędzy użytkownikiem, a sterownikiem i pozwalają na wprowadzanie nastaw. Magistrala I2C połączona jest z układem RTC służącym do automatycznego odmierzania i zapamiętywania aktualnego czasu. Linie D10-D12, są wyjściami służącymi do sterowania dmuchawą oraz pompą wody. Komunikacja z wyświetlaczem odbywa się poprzez wyjścia D5-D9, a realizacja protokołu 1Wire do obsługi czujnika temperatury została zaimplementowana z użyciem portu D4. W module pozostały wolne 2 wejścia analogowe oraz 4 cyfrowe, które pozwalają na dodanie kolejnych funkcjonalności w przyszłości jeżeli zaistnieje taka potrzeba.

6.3 Układ zasilania

Schemat układu zasilania przedstawia rys. 6.2. Na wejście sterownika, do złącza X1 podawany jest sinusoidalny sygnał z transformatora o przekładni 230V/9V. Sygnał ten trafia następnie do mostka prostowniczego B1 o oznaczeniu RB17, którego maksymalna obciążalność prądowa wynosi 1,5A. Kondensatory C1, oraz C2 służą do wygładzenia napięcia wychodzącego z prostownika, tak aby uzyskać napięcie stałe. Układ IC1 jest stabilizatorem liniowym, którego napięcie wyjściowe wynosi 5V. Kondensatory C4 oraz C5 zostały dobrane w oparciu o notę aplikacyjną układu LM7805. Dioda LED1 służy do sygnalizacji pracy sterownika, a rezystor R1 ogranicza płynący przez nią prąd.

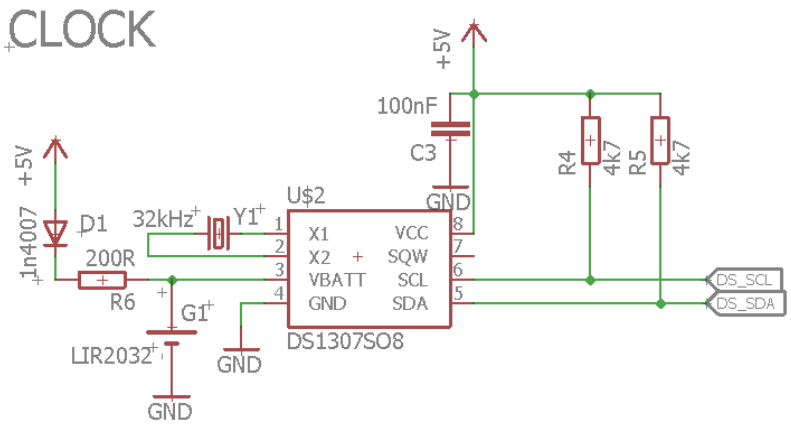


Rysunek 6.2. Schemat układu zasilania sterownika.

6.4 Układ odmierzania czasu

Do odmierzania czasu rzeczywistego został zastosowany dedykowany układ RTC o oznaczeniu DS1307. Pozwala on na odczyt czasu w postaci godziny, minuty i sekundy, oraz miesiąca, dnia i roku. Napięcie zasilania modułu musi zawierać się w zakresie 4,5-5,5V, a maksymalny pobór prądu w stanie czuwania wynosi 200µA. Komunikacja z modułem odbywa się poprzez magistralę I2C, gdzie maksymalna prędkość sygnału zegarowego wynosi 100kHz. Schemat połączeń układy wraz z niezbędnymi elementami zewnętrznymi przedstawia rys. 6.3. Do linii komunikacyjnych SCL, oraz SDA zegara dołączone zostały rezystory R4 oraz R5 o wartościach 4,7KΩ. Istnieje konieczność ich stosowania, ponieważ zgodnie ze specyfikacją przesyłania danych przy pomocy magistrali I2C, wyjścia tych linii są typu otwarty dren, co uniemożliwia wymuszenia stanu wysokiego na magistrali, który według standardu jest stanem dominującym. Element oznaczony na schemacie jako G1, jest to litowo-jonowe ogniwo zasilające o napięciu 3,7V. Dołączone zostało do wejścia VBAT, układu RTC, co umożliwi zachowywanie godziny oraz odmierzanie czasu nawet po odłączeniu zasilania. Dioda D1 oraz rezystor R6 to prosty układ ładowania ogniwa

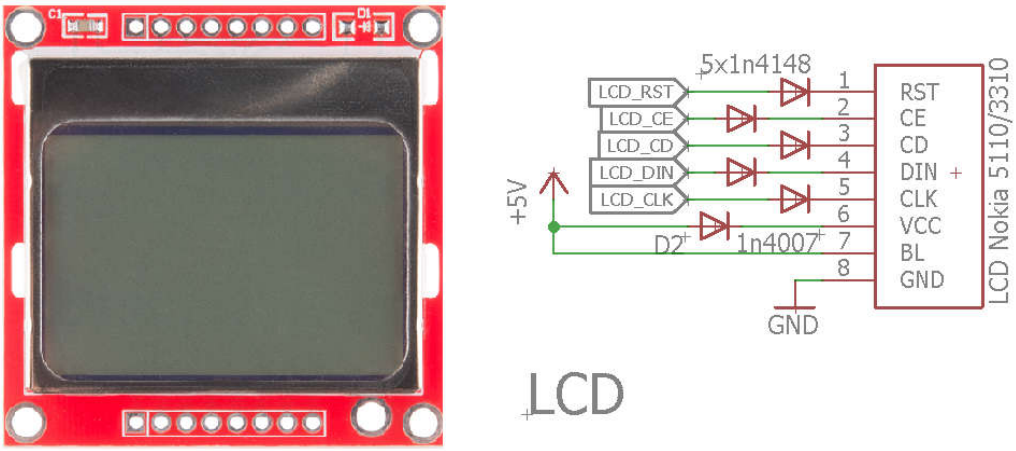
podtrzymującego odliczanie czasu. Dodatkowo do wejść X1 oraz X2, zgodnie z notą aplikacyjną układu DS1307, dołączony jest rezonator kwarcowy o częstotliwości 32768Hz.



Rysunek 6.3. Schemat modułu RTC.

6.5 Wyświetlacz

Do łatwego odczytu parametrów pracy sterownika wykorzystany został moduł monochromatycznego wyświetlacza ze sterownikiem Philips PCD8544 o rozdzielczości 84x48 pikseli. Posiada on wbudowaną pamięć DDRAM, która działa jako bufor wyświetlanego obrazu, i po wpisaniu do niej wartości są zachowywane, aż do kolejnego resetu sterownika wyświetlacza. Maksymalna częstotliwość pracy magistrali danych wynosi 4MHz, co pozwala na teoretyczne osiągnięcie wyświetlania ponad 120 klatek na sekundę. Wygląd modułu oraz schemat połączeń przedstawia rys. 6.4.



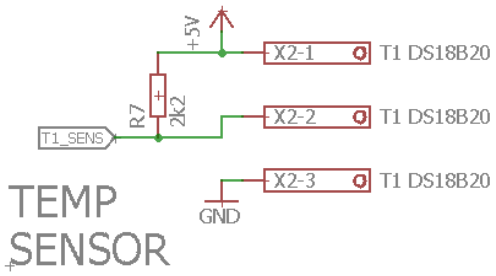
Rysunek 6.4. Wygląd wyświetlacza oraz schemat połączeń.

Do komunikacji mikrokontrolera z wyświetlaczem wykorzystywana jest magistrala SPI, oraz kilka dodatkowych sygnałów sterujących. Na liniach danych zastosowane zostały diody 1N4148 po to, aby obniżyć napięcie dochodzące do wyświetlacza. Rozwiązanie takie jest możliwe, gdyż komunikacja pomiędzy mikrokontrolerem, a wyświetlaczem odbywa się tylko w jedną stronę. Mikrokontroler po wysłaniu danych

nie ma możliwości ich późniejszego odczytu. Podobne rozwiązanie zastosowano przy doprowadzeniu napięcia zasilającego, gdzie aby obniżyć napięcie, w szeregu dodana została dioda 1N4007.

6.5 Pomiar temperatury

Jako termometr użyty został układ o oznaczeniu DS18B20, który pozwala na pomiar temperatury w zakresie -55 ˚C, do 125˚C z rozdzielczością 12 bitów, oraz dokładnością ±0,5˚C. Komunikacja odbywa się poprzez magistralę 1W, która podobnie jak w przypadku I2C wymaga rezystora podciągającego do napięcia zasilania, aby wymusić stan dominujący na linii. Specyfikacja magistrali nie określa maksymalnej liczby czujników dołączonych do magistrali, a odczyt temperatury z konkretnego czujnika, wywoływany jest przy pomocy unikalnego numery seryjnego przypisanego do każdego z nich. Schemat połączeń przedstawia rys. 6.5.

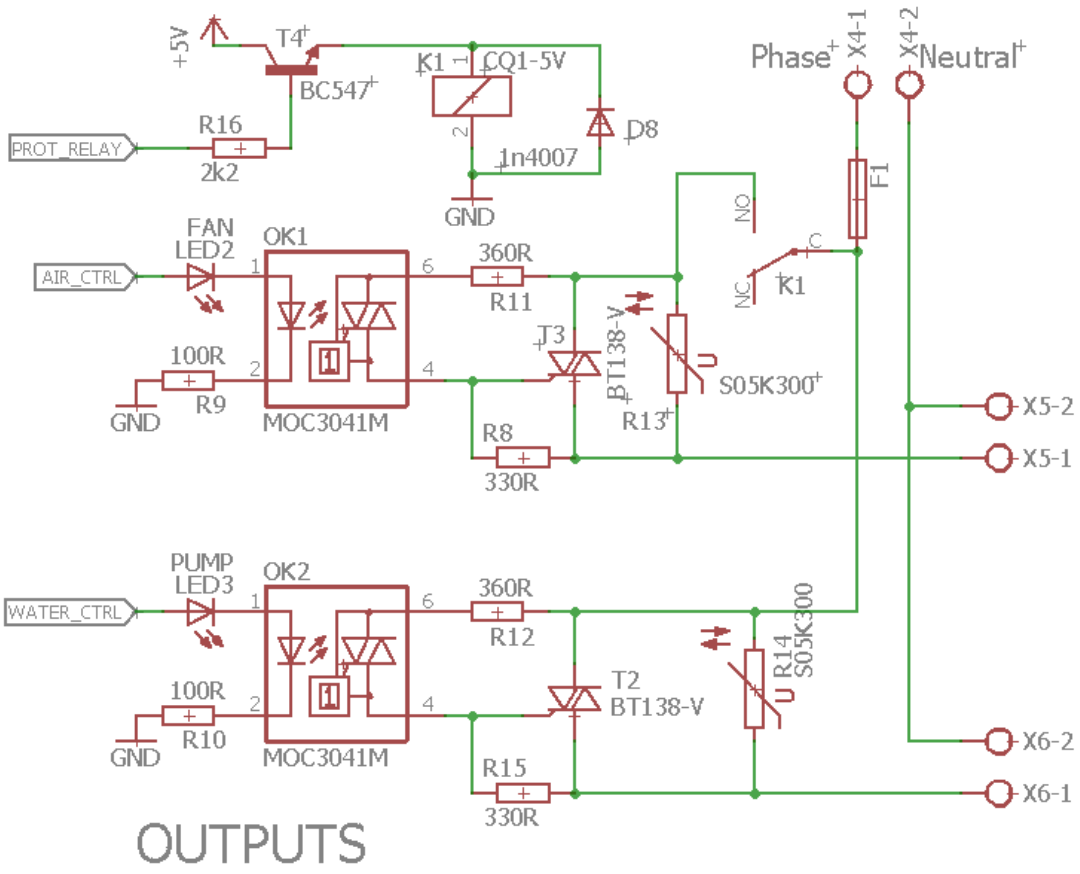


Rysunek 6.5. Schemat połączeń czujników temperatury.

W sterowniku użyte zostały dwa czujniki temperatury, jeden współpracujący z wentylatorem, oraz jeden z pompą wody. Dołączone są one do jednego wejścia oznaczonego na schemacie X1. Do podciągnięcia linii danych służy rezystor R7.

6.6 Wyjścia mocy

Do sterowania urządzeniami zasilanymi napięciem sieciowym konieczne było zastosowanie układów separujących, tak aby w chwili uszkodzenia układu wykonawczego, napięcie sieciowe nie dostało się do układu mikrokontrolera. Schemat takiego rozwiązania wraz z realizacją dodatkowych zabezpieczeń przedstawia rys. 6.6. Do sterowania pracą pompy wody oraz wentylatora zastosowano dwa takie same układy połączeń elementów. Na wejście układu OK1, w momencie załączania podawany jest stan wysoki, co powoduje zaświecenie diody sygnalizującej pracę LED2. Dioda zawarta w strukturze układu MOC3041 również zostanie załączona w tym samym momencie, gdyż połączona jest w szeregu z diodą sygnalizującą pracę. Prąd diod ograniczony jest poprzez rezystor R9. W chwili czasowej, gdy napięcie sieciowe przejdzie przez zero załączeniu ulegnie optotriak umieszczony w strukturze układu OK1, a następnie załączy on triak T3 i poda napięcie sieciowe na złącze X5, gdzie podłączony jest wentylator. Takie samo zadanie realizuje grupa elementów LED3, R10, OK2 oraz T2. Bezpiecznik F1 służy do zabezpieczenia przed zwarciem, któregoś z urządzeń dołączonych do wyjść sterownika.

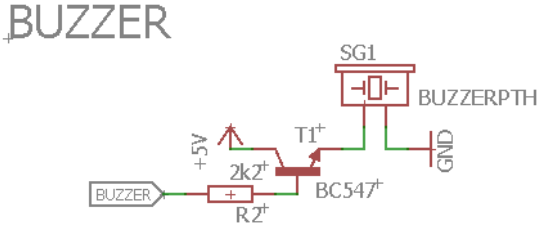


Rysunek 6.6. Schemat połączeń układów wyjść mocy.

Z racji, że urządzenia dołączone do sterownika mają charakter indukcyjny jako dodatkowe zabezpieczenie przed przepięciami mogącymi uszkodzić triaki zastosowane zostały warystory R13 oraz R14, których napięcie przewodzenia wynosi 300V napięcia skutecznego. Dodatkowym zabezpieczeniem, przed pożarem jest przekaźnik K1 pozwalający odłączyć zasilanie wentylatora w momencie, gdy triak ulegnie uszkodzeniu i będzie cały czas załączony, co jest częsty uszkodzeniem tego typu elementów.

6.7 Sygnalizacja dźwiękowa

W przypadku wystąpienia poważnego błędu sterownika istnieje konieczność dźwiękowego powiadamiania o tym użytkownika, tak aby miał on świadomość o takowym błędzie. Schemat układu powiadamiania przedstawia rys 6.7.

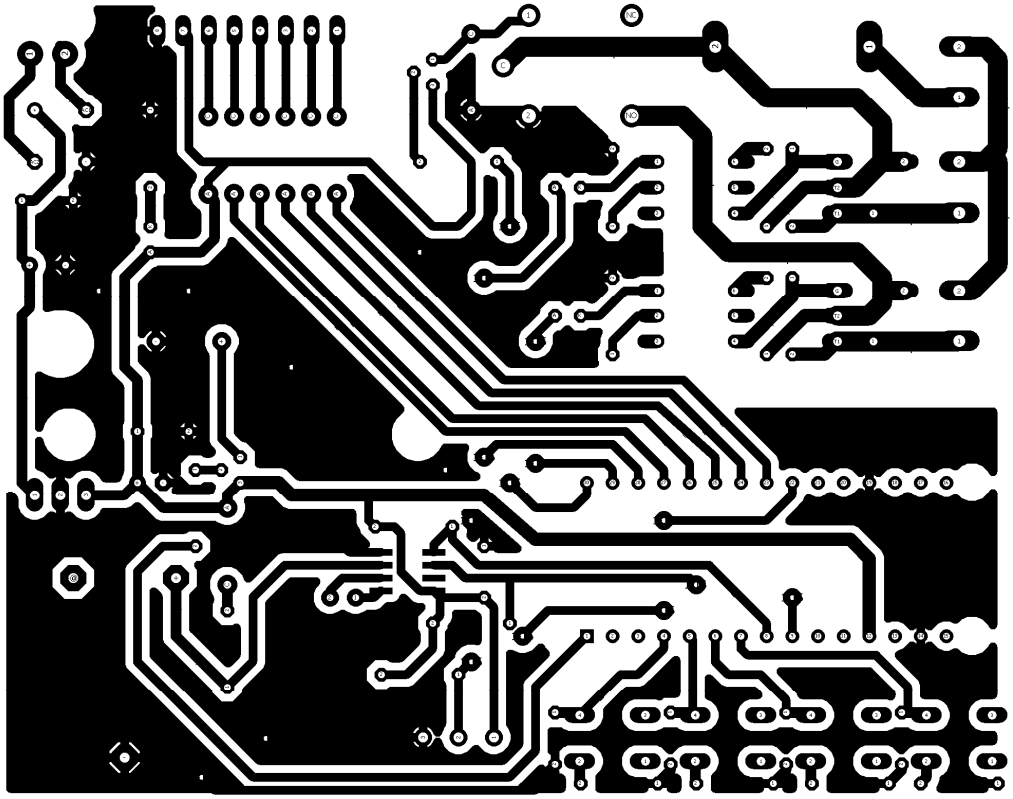


Rysunek 6.7. Schemat połączeń sygnalizatora dźwiękowego.

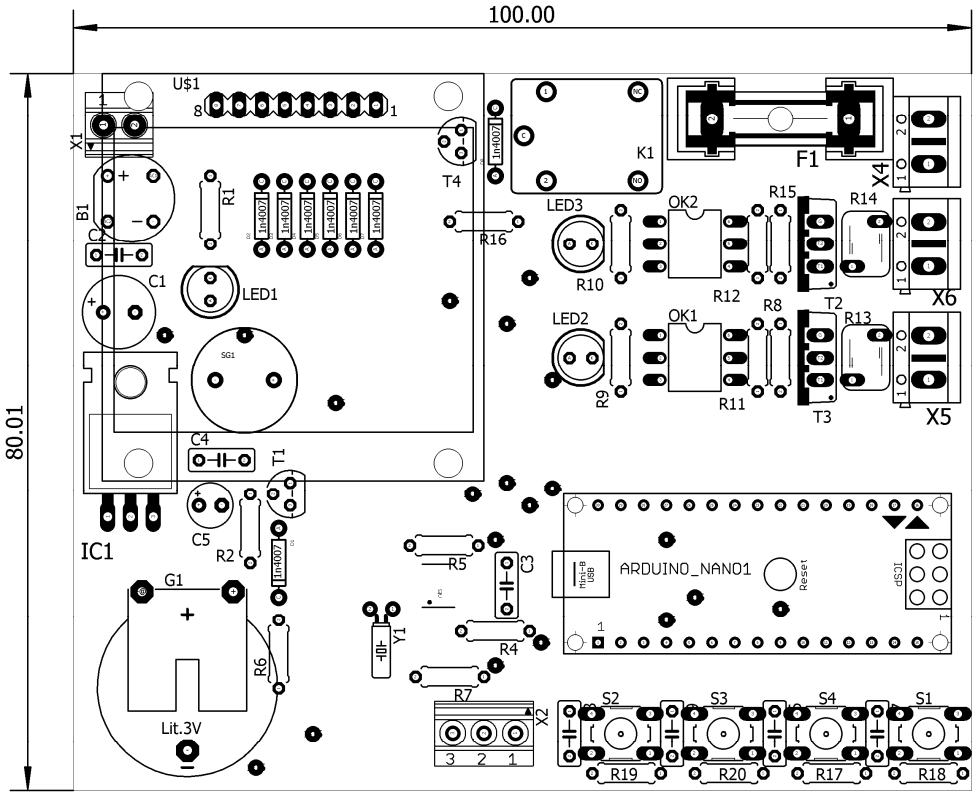
Do sygnalizacji dźwiękowej wykorzystany został najprostszy buzzer z generatorem o oznaczeniu HCM1206 zasilany z napięcia 5V. Jako że pobierany przez niego maksymalny prąd wynosi 30mA konieczne było zastosowanie tranzystora T1 wraz z rezystorem R2 do sterowania jego pracą.

6.8 Obwód drukowany

Na potrzeby sterownika zaprojektowany został obwód drukowany. Wykonano go w oprogramowaniu Eagle w wersji 7.7.0. Sama płytka posiada wymiary 100mm x 80mm i została zaprojektowana tak, aby wszystkie ścieżki poprowadzone były po jednej warstwie. Wygląd obwodu drukowanego oraz rozmieszczenia elementów przedstawiają rysunki 6.8 oraz 6.9.



Rysunek 6.8. Mozaika ścieżek obwodu drukowanego.



Rysunek 6.8 Widok rozmieszczenia elementów na obwodzie drukowanym.

7. Oprogramowanie sterownika

7.1 Środowisko programistyczne

Program został napisany w środowisku programistycznym Arduino IDE w wersji 1.8.5. Wybranie tego środowiska spowodowane jest wcześniejszym użyciem platformy sprzętowej Arduino, oraz szerokim dostępem do bibliotek współpracującymi z peryferiami. Skompilowany kod zajmuje 58% pamięci ROM oraz 57% pamięci RAM, co pozwala na dopisanie dodatkowych funkcjonalności sterownika w przyszłości. Kod programu został umieszczony w pracy jako załącznik drugi.

7.2 Funkcja inicjalizująca

Funkcja inicjalizująca pracę programu przedstawiona została na listingu 7.1. Na początku sprawdzany jest warunek, czy pamięć EEPROM mikrokontrolera jest pusta. Jeżeli tak, to oznacza to, że sterownik jest uruchamiany po raz pierwszy i należy wprowadzić do pamięci ustawienia początkowe. Po sprawdzeniu tego warunku do struktury opisującej aktualny stan parametrów sterownika wczytywana są wartość z pamięci EEPROM. Następnie następuje inicjalizacja sensorów temperatury, wyświetlacza oraz modułu zegara oraz zostaje wysłany rozkaz konwersji temperatury do czujników. Podczas inicjalizacji zegara ustawiana jest zmienna *flag_get_time* przekazująca do przerwania od licznika informację o tym, że rozpoczęła się transmisja na magistrali I2C. W momencie, gdy mikrokontroler przez sekundę nie otrzyma odpowiedzi od układu wywoływana jest funkcja obsługi błędu, a sterownik przechodzi w tryb awaryjny. Po poprawnej inicjalizacji peryferii, pobrane są początkowe wartości temperatury oraz uruchamiany zostaje PWM z maksymalną wartością, przekaźnik zabezpieczający załącza dmuchawę, a sterownik przechodzi w tryb rozpalania.

Listing 7.1. Funkcja inicjalizująca.

```
//------- SETUP FUNCTION --------------//
void setup() {
  int a = 0;
  EEPROM.get(0, a);
  if (a == 255) { // if new uC
    WriteInitValeuToEeprom(); // write initial values
  }
  ReadParamFromEeprom();
  DisplayInit(); // lcd init
  sensors.begin();// temp sensors init
  CheckSensors();
  sensors.requestTemperatures();
  delay(750);
  PinsInit();
  PwmInit();
  flag_get_time = 10;
  clock.begin(); // clock init
  flag_get_time = 0;
  fur.pump_temp = sensors.getTempC(ds18b20_pump);
  fur.fan_temp = sensors.getTempC(ds18b20_fan);
  fur.last_fan_temp = fur.fan_temp;
  fur.last_pwm_state = 10;
  PwmSet(fur.last_pwm_state);
  ProtRelayOn();
}
```

7.2 Funkcja główna programu

Funkcja główna programu służy do obsługi zdarzeń związanych z pracą sterownika. Przy użyciu sprzętowego licznika zaimplementowany został prosty scheduler, i na wzór pracy systemów RTOS, wywołuje zdarzenia zawarte w głównej funkcji programu. Rozwiązanie takie powoduje zwolnienie pracy czasu mikrokontrolera, gdyż nie wykonuje on w koło tych samych czynności z każdym obiegiem funkcji głównej, a jedynie w momencie, gdy scheduler wyśle mu informację o potrzebie wykonania pewnych działań. Dodatkowo pozwala też na wyeliminowanie funkcji oczekiwania na wykonanie jakiegoś zdarzenia, takiego jak na przykład czas potrzebny na konwersję temperatury dla modułu DS18B20. Implementację obsługi zdarzeń przedstawia listing 7.2.

Listing 7.2. Funkcja główna programu.

```
//------- MAIN FUNCTION ---------------//
void loop() {
  if (menu_mode == 0) {
    // meas temp event
    if (flag_get_temp) {
      fur.pump_temp = sensors.getTempC(ds18b20_pump);
      fur.fan_temp =  sensors.getTempC(ds18b20_fan);
      sensors.requestTemperatures();
      flag_get_time = 10;
      dt = clock.getDateTime();
      flag_get_time = 0;
      flag_get_temp = 0;
      DisplayDrawMainScreen();// draw screen event
    }
    if (flag_calc_pwr == 1) {
      CalcAndSetFanPwr();
      flag_calc_pwr = 0;
    }
    // check sensors are broken
    if (flag_check_sensors) {
      CheckSensors();
      flag_check_sensors = 0;
    }
    if (eeprom_update_flag == 1) {
      EEPROM.update(EEPROM_SET_TEMP_ADDR, (int)fur.set_temp);
      eeprom_update_flag == 0;

    }
  }
  if (menu_mode == 1) {
    DrawMenu();
    DisplayDrawMainScreen();// draw screen event
  }
}
```

Początkowo sprawdzane jest, czy sterownik znajduje się w trybie edycji parametrów i jeżeli warunek zostanie spełniony praca sterownika zostaje zatrzymana oraz wywoływana jest funkcja obsługująca zmianę parametrów pracy. Gdy sterownik znajduje się w trybie pracy wykonywany jest szereg warunków sprawdzających stan flag zdarzeń wywoływanych przez scheduler. W momencie zgłoszenia flagi pomiaru temperatury oznaczonej jako *flag_get_temp*, wywoływane są funkcje, które odczytują wartości z sensorów, odczytują aktualny czas oraz wyświetlają dane na wyświetlaczu LCD. W chwili zakończenia powyższych czynności flaga jest zwalniana, a program przechodzi do kolejnych warunków. Jako następna sprawdzana jest flaga obliczania mocy wentylatora, która steruje jego prędkością. Jako zabezpieczenie przed uszkodzeniem któregoś z czujników, co 10 sekund wywoływane jest zdarzenie mające na celu sprawdzić poprawność podłączenia czujników i jeżeli funkcja *CheckSensors()* nie uzyska odpowiedzi od obu czujników dołączonych do sterownika przejdzie on w tryb awaryjny, wyświetlając jednocześnie stosowną informacje o błędzie na wyświetlaczu. Użycie takie rozwiązania ma na celu ochrony przed uszkodzeniem instalacji

ogrzewania lub pożarem spowodowanym ciągłą pracą wentylatora wtłaczającego powietrze do kotła. W momencie, gdy sterownik znajduje się w trybie pracy możliwa jest regulacja zadanej temperatury poprzez przyciski góra/dół dołączone do sterownika. Sama obsługa przycisków umieszczona jest w przerwaniu sprzętowego licznika, a flaga *eeprom_update_flag* służy do zapisu ostatnio zadanej temperatury, tak aby po ponownym włączeniu była ona taka sama jak ustawiona wcześniej.

7.3 Obsługa przerwania od licznika

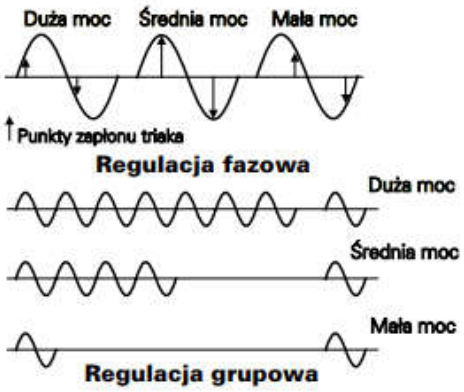
Głównym zarządcą sterującym pracą programu jest szesnastobitowy sprzętowy licznik wbudowany w strukturę mikrokontrolera. Ustawiony został w taki sposób, że przepełnienie licznika realizowane jest z częstotliwością 5Hz. Każde przepełnienie wywołuje obsługę przerwania, którego część znajduje się w listingu 7.3.

Listing 7.3. Obsługa przerwania od licznika.

```
// timers interrupths
ISR(TIMER1_COMPA_vect)
{
  if (!menu_mode) {
    if (fur.fan_temp > OVERHEAT_TEMP) { // hard fault
      noInterrupts();// disable interrupts
      halt(OVERHEATING); // overheat error
    }
    if (flag_get_time > 9) flag_get_time++; // hard clock fault
    if (flag_get_time > 29) halt(CLOCK_ERROR);

    if ((fur.fan_temp - fur.pump_temp) > fur.pump_temp_diff) WaterPumpOn();
    else WaterPumpOff(); // pump driver

    digitalWrite(AIR_CTRL, LOW);  // pwm for fan
    event_counter++;              // scheduler ++
    if (fur.start_flag > 0) fur.start_flag++; // start mode flag

    if ((event_counter % 5) / 4) flag_get_temp = 1;; // call temp event every 800ms
    if ((event_counter % 51) / 50) flag_check_sensors = 1; // call check sensor event every 10sec
    if ((event_counter % 601) / 600) flag_calc_pwr = 1; //call calculate power event every 120 sec
    // check key status
    noInterrupts();
    keys = CheckButtons();     // check key status
    interrupts();
    ...
}
```

Podczas wywołania obsługi przerwania w pierwszej kolejności sprawdzane jest w jakim trybie znajduje się sterownik i jeżeli nie jest on w trybie pracy wykonywane jest natychmiastowe wyjście z obsługi przerwania. Jeżeli natomiast jest on w trybie pracy sprawdzany jest warunek zabezpieczający, czy temperatura wody znajdującej się w obiegu nie przekracza maksymalnej dopuszczalnej temperatury zdefiniowanej w programie. Jeżeli warunek się potwierdzi program jest natychmiastowo przerywany, przekaźnik zabezpieczający odłącza dmuchawę, a pompa wody ulega załączeniu. Następnie zaimplementowane jest sprawdzanie, czy układ zegara nie uległ uszkodzeniu, i jeżeli mikrokontroler nie uzyska odpowiedzi od niego w określonym czasie, program również jest zatrzymywany. W dalszej kolejności realizowane jest załączanie pompy w obiegu wody. Jeżeli temperatura powrotu wody będzie większa niż maksymalna różnica temperatur podana w parametrach pracy sterownika ulegnie ona załączeniu. Jako następne zrealizowany jest sprzętowy PWM do obsługi mocy dmuchawy. Jako, że w sterowniku nie przewidziano detekcji przejścia sygnału przez zero, niemożliwe było zrealizowanie sterowania fazowego. Użyty został sposób sterowania grupowego[13], którego idea przedstawiona została na rys. 7.1. Realizacja takiego sposobu sterowania mocą niesie za sobą zmniejszenie zakłóceń oddawanych do sieci. Pozwala także na pracę silników z mocą, a co za tym idzie prędkością obrotową, mniejszą niż udałoby się uzyskać przy pomocy sterowania fazowego. Kolejne trzy

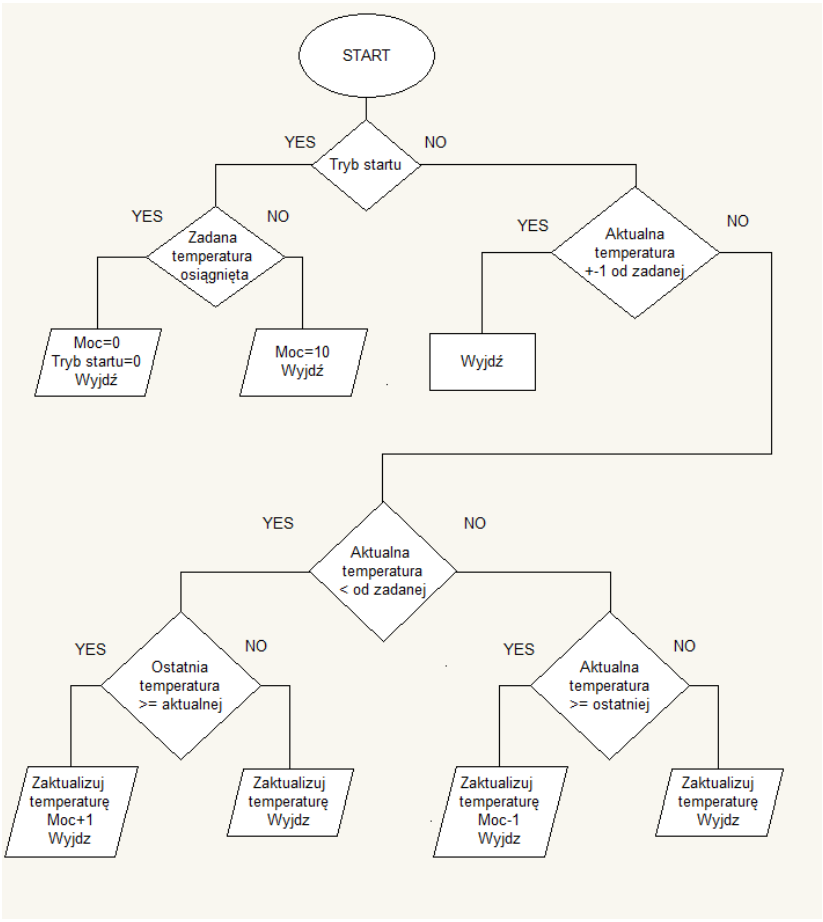
sprawdzenia warunków dotyczą obsługi wykonywania zdarzeń funkcji głównej programu. Zdarzenie pomiaru temperatury wywoływane jest co 800ms, obecności czujników temperatury co 10 sekund, a przeliczenie mocy dmuchawy co 2 minuty. Jako ostatnia zrealizowana jest obsługa wciśnięcia klawiszy podłączonych do sterownika. Podczas wciśnięcia przycisku up aktualnie zadana temperatura jest zwiększana, down zmniejszana, a wciśnięcia klawisza ok powoduje wejście w tryb edycji parametrów sterownika.



Rysunek 7.1. Porównanie sterowania grupowego oraz fazowego [13].

7.4 Algorytm sterowania mocą dmuchawy

Na potrzeby obliczania mocy dmuchawy w zależności od uchybu temperatury opracowany został prosty algorytm przedstawiony na rysunku 7.2.



Rysunek 7.2. Diagram algorytmu sterowania mocą dmuchawy [13].

Jest to w rzeczywistości uproszczona implementacja regulatora typu tupu PI z autoadaptacją członu proporcjonalnego. Z każdym wywołaniem funkcji wyliczającej aktualną nastawę regulatora początkowo sprawdzane jest, czy sterownik znajduje się w trybie startu. Jeżeli tak, moc dmuchawy ustawiana jest na wartość maksymalną, a funkcja jest opuszczana. Jeżeli sterownik jest w trybie zwykłej pracy w dalszej kolejności sprawdzane jest, czy zadana temperatura jest w zakresie ±1˚C w stosunku do zadanej temperatury. W momencie, gdy warunek zwróci wartość prawdziwą, aktualna moc pracy dmuchawy pozostaje niezmieniona, a funkcja kończy swoją pracę. Jeśli natomiast temperatura nie mieści się w podanym zakresie sprawdzane jest, czy aktualna temperatura jest większa, czy mniejsza od zadanej, a następnie kolejne dwa zagnieżdżone warunki, które ustalają, czy temperatura od ostatniego wywołania wzrosła, czy zmalała. Jeżeli nastawa jest mniejsza od wartości zmierzonej, a ostatnia wartość pomiaru była mniejsza lub równa obecnej, moc dmuchawy jest zwiększana o jeden stopień. Gdy temperatura zmierzona w poprzednim wywołaniu funkcji nastawy była mniejsza niż obecna, funkcja aktualizuje jedynie wartość ostatniego pomiaru na obecny, po czym zostaje zakończona. Taki sam model zastosowany został w przypadku, gdy nastawa jest mniejsza niż wartość zmierzona, tylko wtedy, gdy temperatura rośnie moc jest zmniejszana o jeden stopień. Ciało funkcji przedstawia listing 7.4.

Listing 7.4. Algorytm sterowania mocą dmuchawy

```
void CalcAndSetFanPwr() {
  // star mode
  if (fur.start_flag > 0) { // if start mode
    if ((fur.fan_temp < fur.set_temp)) PwmSet(10);  // if not to reach set temp
    else {  // if its reached                       // stay wtih max power
      fur.start_flag = 0;     // of start timer
      fur.last_pwm_state = 0; // disable fan
      PwmSet(0);              // one more time
      fur.last_fan_temp = fur.fan_temp; // update last temp
    }
    if (fur.start_flag > fur.max_start_time) halt(START_ERROR); // if start time has
  }                                                             //expired throw error
  // normal mode
  if (fur.start_flag == 0)  // if controller is in normal mode
  {
    // between set_temp-1&&set_temp+1
    if (((fur.fan_temp - fur.set_temp) < 1) && ((fur.set_temp - fur.fan_temp) < 1)) {
      fur.last_fan_temp = fur.fan_temp;
      return;
    }

    if (fur.fan_temp < fur.set_temp) {  // if curre nt temp<set temp
      if (fur.last_fan_temp >= fur.fan_temp) { // if last temp>current temp
        fur.last_fan_temp = fur.fan_temp;     // update temp
        fur.last_pwm_state++;                 // power++
        if (fur.last_pwm_state > 10) fur.last_pwm_state = 10;
        PwmSet(fur.last_pwm_state);         // set power
        return;                             // end function
      }
      fur.last_fan_temp = fur.fan_temp;
    }
    if (fur.fan_temp >= fur.set_temp) {   // same as upper if
      if ((fur.fan_temp >= fur.last_fan_temp)) {
        fur.last_fan_temp = fur.fan_temp;
        fur.last_pwm_state--;               // but opposite direction
        if (fur.last_pwm_state < 0) fur.last_pwm_state = 0;
        PwmSet(fur.last_pwm_state);
        return;
      }
      fur.last_fan_temp = fur.fan_temp;
    }
  }
}
```

7.5 Struktura menu użytkownika

Informacje wyświetlane na wyświetlaczu LCD zostały podzielone na dwa tryby wyświetlania. Pierwszym trybem jest tryb pracy, gdzie wyświetlane są informacje odnośnie bieżących parametrów sterownika takich jak temperatury, godzina, tryb pracy oraz moc dmuchawy. Wciśnięcie przycisku ok spowoduje wejście w tryb ustawiania parametrów. Pierwszym parametrem jest maksymalny czas rozpalania, po jakim sterownik zgłosi błąd rozpalania. Edycja parametru odbywa się poprzez zmianę wartości przyciskami up/down. Po ustawianiu żądanej wartości są dwie opcje do wyboru. Pierwsza z nich to przejście do ustawienia kolejnego parametru przyciskiem ok lub wyjście z trybu ustawiania parametrów przyciskiem esc. Przy ustawianiu następnych parametrów występują te same zasady. Cała struktura menu ustawień oraz ekranu głównego przedstawiona została na rysunku 7.3.

Rysunek 7.3. Struktura menu sterownika.