POLITECHNIKA WARSZAWSKA

WYDZIAŁ

MECHANICZNY ENERGETYKI I LOTNICTWA

ZAKŁAD MASZYN I URZĄDZEŃ ENERGETYCZNYCH

PRACA DYPLOMOWA INŻYNIERSKA

Piotr Klukowski

Rozbudowa układu pomiarowego turbiny gazowej

Expansion of the measurement system of the gas turbine

Nr albumu: 252662

Energetyka

Systemy i urządzenia energetyczne

Promotor: dr inż. Konrad Wojdan

Warszawa, luty 2016

Oświadczenie autora (autorów) pracy

Świadom odpowiedzialności prawnej oświadczam, że przedstawiona praca dyplomowa:

- została napisana przeze mnie samodzielnie i nie zawiera treści uzyskanych w sposób niezgodny z obowiązującymi przepisami,
- nie była wcześniej przedmiotem procedur związanych z uzyskaniem tytułu zawodowego lub stopnia naukowego w wyższej uczelni.

Oświadczam ponadto, że niniejsza wersja pracy jest identyczna z załączoną wersją elektroniczną.

....
data podpis autora (autorów) pracy

Oświadczenie

Wyrażam zgodę / ~~nie wyrażam zgody~~ *) na udostępnianie osobom zainteresowanym mojej pracy dyplomowej. Praca może być udostępniana w pomieszczeniach biblioteki wydziałowej. Zgoda na udostępnienie pracy dyplomowej nie oznacza wyrażenia zgody na jej kopiowanie w całości lub w części.

Brak zgody nie oznacza ograniczenia dostępu do pracy dyplomowej osób:
- reprezentujących władze Politechniki Warszawskiej,
- członków Komisji Akredytacyjnych,
- funkcjonariuszy służb państwowych i innych osób uprawnionych, na mocy odpowiednich przepisów prawnych obowiązujących na terenie Rzeczypospolitej Polskiej, do swobodnego dostępu do materiałów chronionych międzynarodowymi przepisami o prawach autorskich.

Brak zgody nie wyklucza także kontroli tekstu pracy dyplomowej w systemie antyplagiatowym.

....
data podpis autora (autorów) pracy

*) niepotrzebne skreślić

SŁOWA KLUCZOWE: turbina gazowa, instalacja automatyki i opomiarowania, stanowisko laboratoryjne

STRESZCZENIE

W pracy zaprezentowany został opis układu pomiarowego mikro turbiny gazowej, zbudowanej jako projekt Koła Naukowego Energetyków (KNE), oraz opis modyfikacji tego układu. Głównym celem pracy było stworzenie nomenklatury pomiarów i wdrożenie jej w istniejący kod mikrokontrolera zarządzającego pomiarami. Co więcej mikrokontroler ten został opisany, jak również oprogramowanie do jego programowania. Częściowo został przedstawiony kod programu wgranego w mikrokontrolerze pomiarowym oraz mikrokontrolerze sterującym turbiną. Przedstawione zostały również zmiany i modyfikacje czujników temperatury (termopary) i czujnika obrotów wału alternatora. Dodatkowo została napisana instrukcja dodawania nowych pomiarów temperatury, jak również problemy jakie mogą wystąpić w przypadku dodawania pomiarów innego typu. Ostatnią częścią tej pracy jest dokończenie sekwencji autostartu mikro turbiny.

ABSTRACT

In this thesis a description of the measurement system of a gas micro turbine, built as a project of “Koło Naukowe Energetyków (KNE)”, is presented with its alteration. The main purpose of work was creation of a map of measurements and implementation of aforementioned in the code of the measurement microcontroller which is described alongside with software for its programming. Also the code uploaded in the measurement microcontroller as well as the control one is partially presented. Moreover, changes and modifications of temperature sensors (thermocouples) and a rotation sensor of alternator shaft are specified. Additionally, the instruction of adding new temperature measurements is described, as well as possible problems with other types of measurements. Completing the sequence of automatic startup was the last part of this work.

SPIS TREŚCI

1. WSTĘP

Tematem mojej pracy jest rozbudowa układu pomiarowego turbiny gazowej, która została zbudowana jako projekt Koła Naukowego Energetyków (KNE). Budowa układu zakończyła się w grudniu 2014 roku, pod koordynacją Jana Pleszyńskiego. Jego praca inżynierska [1] stanowiła bazę startową do mojej pracy.

Głównym celem pracy jest stworzenie komunikacji pomiędzy mikrokontrolerem Arduino, znajdującym się w szafie sterującej mikro turbiny, a systemem SCADA. System ten jest tworzony przez Krzysztofa Setlaka, w ramach pracy inżynierskiej [2], na bazie oprogramowania firmy Transition Technologies - EDS. Fizycznie granica naszych prac będzie na porcie szeregowym USB.

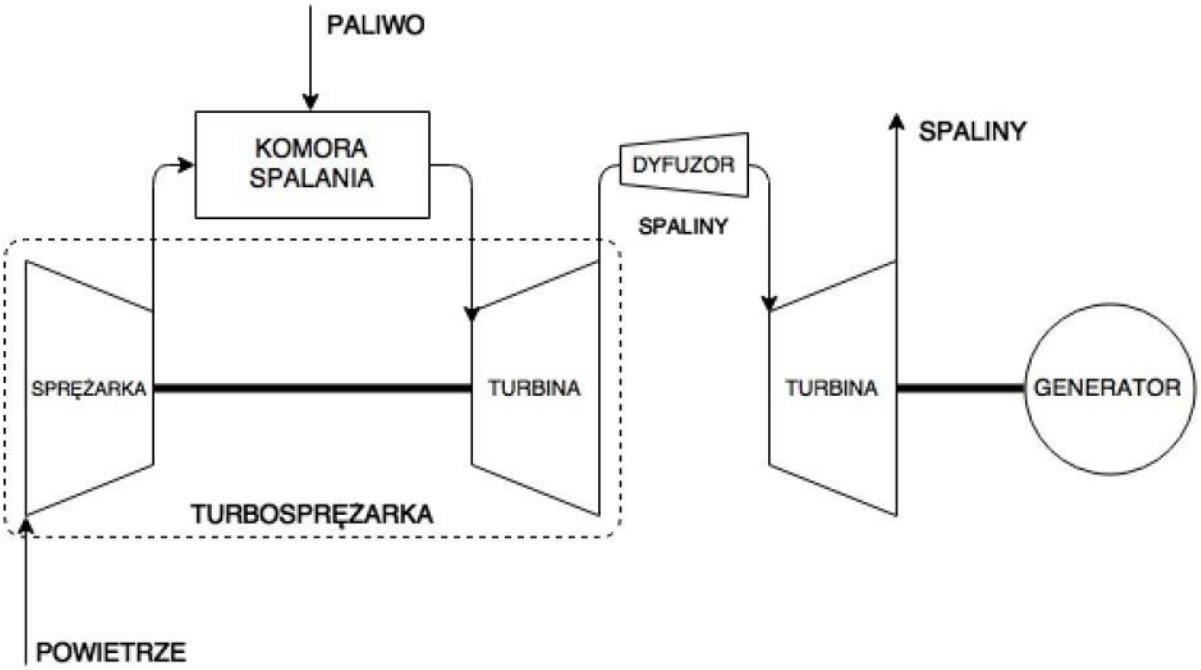
Dodatkowym, lecz koniecznym celem jest techniczna rozbudowa układu pomiarowego. Ma ona polegać na zmianie czujnika obrotów alternatora z optycznego na magnetyczny, ze względu na duży błąd pomiaru, oraz dodanie trzech pomiarów z termopar, poprzez specjalne moduły wzmacniające sygnał analogowy. Wcześniej pomiary wysokich temperatur były realizowane przez urządzenie automatyki LabJack.

Ostatnim celem jest dopracowanie sekwencji autostartu turbiny (kod Arduino), którego głównym problemem jest brak sprawdzenia płomienia, co jest bardzo niebezpieczne, ponieważ grozi zniszczeniem turbiny, gdyby do zapłonu doszło poza komorą spalania.

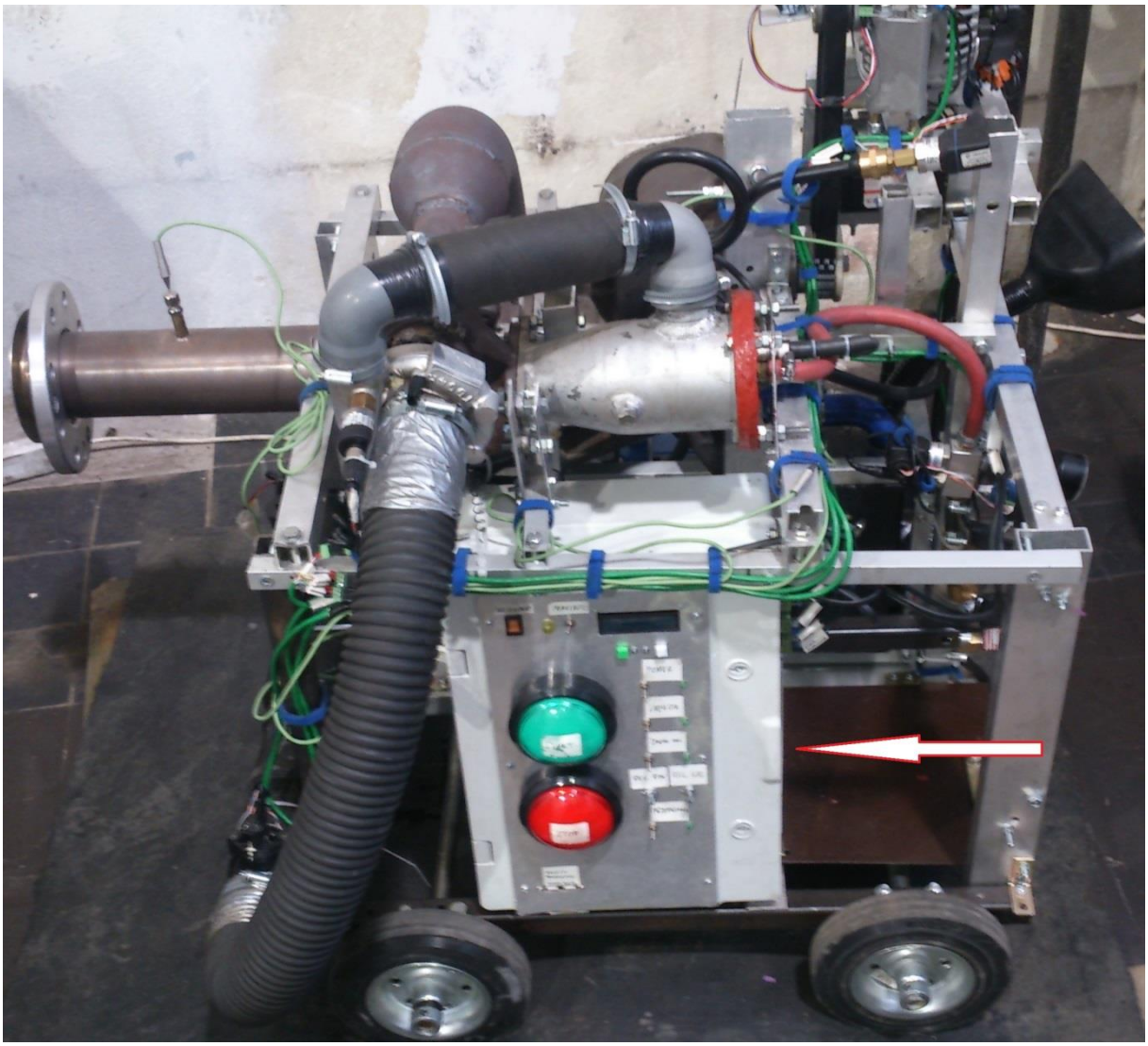
2. MIKROTURBINA

Zbudowany układ to dwustopniowa turbina, zbudowana z turbosprężarki samochodowej oraz turbiny turbocompund o mocy maksymalnej ok. 6kW, oraz sprawności ok. 6,5%, przy tej mocy. Schemat blokowy został przedstawiony na Rysunek 1. Na Rysunek 2 zdjęcie zrealizowanego projektu, bez części z komorą natryskową. Więcej na temat budowy mikro turbiny znajduje się w pracy dyplomowej inżynierskiej Jana Pleszyńskiego [1].

Moja praca skupia się na mikrokontrolerach znajdującej się w skrzynce sterującej. Skrzynka ta została przedstawiona wraz z turbiną i zaznaczona na Rysunek 2. Dokładny opis budowy skrzynki znajduje się w [1]. Natomiast nomenklatura pomiarów oraz aktualny opis podpięcia złącz do mikrokontrolera odpowiadającego za pomiar znajduje się w Mapowanie.



Rysunek 1. Schemat blokowy mikro turbiny gazowej (źródło: [4])



Rysunek 2. Zdjęcie mikro turbiny gazowej

3. ARDUINO

3.1. Mikrokontroler

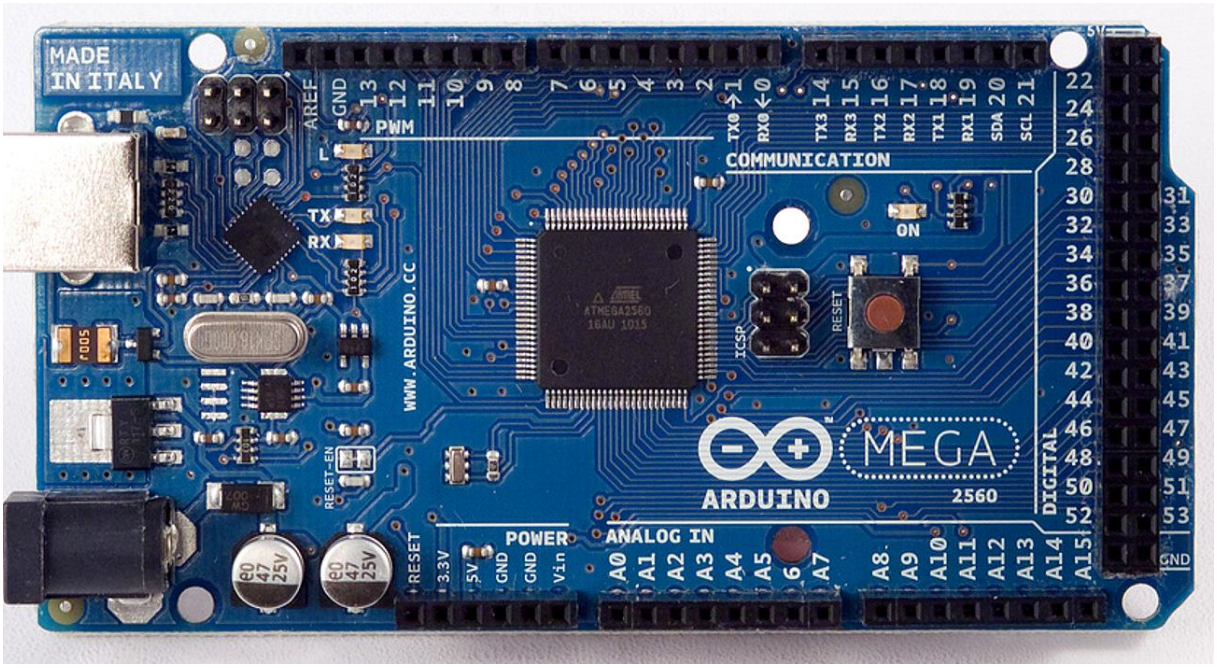
Arduino z punktu widzenia hardware'u jest mikrokontrolerem mającym postać płytki wyposażonej w złącze uniwersalnej magistrali szeregowej USB, służące do komunikacji z komputerem, a także inne złącza służące do podłączania zewnętrznych elementów elektronicznych takich jak diody, czy tranzystory . Język programowania Arduino jest oparty na środowisku Wiring i zasadniczo na języku C/C++. Celem projektu Arduino jest przygotowanie narzędzi ogólnodostępnych, tanich, nie wymagających dużych nakładów finansowych, elastycznych i łatwych w użyciu. Arduino stanowi również alternatywę dla osób, które nie mają dostępu do bardziej zaawansowanych kontrolerów.

W przypadku naszej mikro turbiny gazowej zostały użyte dwie płytki Arduino Mega, jedno do sterowania, drugie do zarządzania pomiarami. Na Rysunek 3 znajduje się tabela z parametrami technicznymi tych płytek.

Technical specs

Microcontroller	ATmega2560
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limit)	6-20V
Digital I/O Pins	54 (of which 15 provide PWM output)
Analog Input Pins	16
DC Current per I/O Pin	20 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	256 KB of which 8 KB used by bootloader
SRAM	8 KB
EEPROM	4 KB
Clock Speed	16 MHz
Length	101.52 mm
Width	53.3 mm
Weight	37 g

Rysunek 3. Tabela z danymi technicznymi Arduino Mega 2560 (źródło: [3])



Rysunek 4. Arduino Mega 2560 (źródło: hifiduino.wordpress.com)

Można zauważyć, że na płytce znajduje się 16 złącz analogowych (ang. analog pins) oraz 54 złącza cyfrowe (ang. digital pins), w tym 15 typu PWM (Pulse Width Modulation). Złącze typu PWM to takie złącze, na którym możemy nadawać sygnał okresowo zmienny - prostokątny, z częstotliwością ok. 500Hz (czas okresu = 2ms), którego szerokość możemy regulować. Szerokość sygnału jest to czas, w którym potencjał jest dodatni, czyli przyjmuje napięcie ok. 5V. W mikrokontrolerach Arduino modulacja taka jest 8-bitowa, czyli każdy okres 2ms, możemy wypełnić z rozdzielczością 1/256. Dzięki temu rozwiązaniu tzn. wypełniając część okresu, z napięcia na jakim pracuje mikrokontroler Arduino możemy uzyskać efekt jak przy napięciu wprost proporcjonalnemu do iloczynu napięcia operacyjnego (5V) z nadaną szerokością podzieloną przez 256.

Dodatkowo ważnym elementem jest port szeregowy (ang. serial port). Służy on do komunikacji, najczęściej z komputerem. Więcej na jego temat znajduje się przy opisie komunikacji z systemem SCADA (Komunikacja przez port szeregowy USB). Wracając do złącz cyfrowych ważnym jest, że o ile używamy któregokolwiek z portów szeregowych (wersja Arduino Mega posiada ich 4), to nie możemy używać poszczególnych złącz cyfrowych (po 2 na każdy port), ponieważ służą one do transmisji/odbioru danych. W przypadku płytki Arduino Mega serial (domyślnie nr. 0) korzysta ze złącz cyfrowych 0(RX) i 1(TX), serial1 – 19(RX) i 18(TX), serial2 – 17(RX) i 16(TX), serial3 – 15(RX) i 14(TX). RX – oznacza złącze realizujące odbiór danych (ang. receive), zaś TX – przesył (ang. transmit).

Złącza analogowe służą do pomiaru wartości napięć sygnałów analogowych, których odczyt umożliwia 10-bitowy dzielnik napięcia (dokładność: 1/1024), który jest wbudowany w układ mikroprocesora. Dzielnik napięcia daje dokładność ok. 4,9mV przy napięciu referencyjnym 5V, które jest napięciem domyślnym dla dzielnika napięcia. Możliwa jest również zmiana tego napięcia na inne. Zrealizować to można na trzy sposoby: skorzystać z innego napięcia domyślnego – 3,3V, skorzystanie z napięcia wewnętrznego – 1,1V lub 2,56V, czy też doprowadzenie innego napięcia z zewnętrznego źródła. Górna granica tego napięcia wynosi 5V, zaś dolna 0V. Zmiana napięcia referencyjnego dotyczy wszystkich złącz analogowych. Jeżeli zajdzie taka potrzeba złącza analogowe mogą działać jako piny cyfrowe (z ominięciem dzielnika napięcia).

Warto również dodać, że do Arduino można dodawać tak zwane „shieldy", są to nakładki na Arduino, które mogą wykorzystywać część złącz, np. do komunikacji WiFi. W tym przypadku na Arduino sterujące został nałożony „Proto Shield" – czyli shield do własnego zaprojektowania (przewody się wlutowuje). Do pomiarowego zaś nałożono „Screw shield", w celu skręcenia przewodów do wejść Arduino. Skręcane złącza są na wszystkie złącza mikrokontrolera z wyjątkiem cyfrowych I/O o numerach 22-54. Połączenie z nimi wymaga wlutowania przewodów.

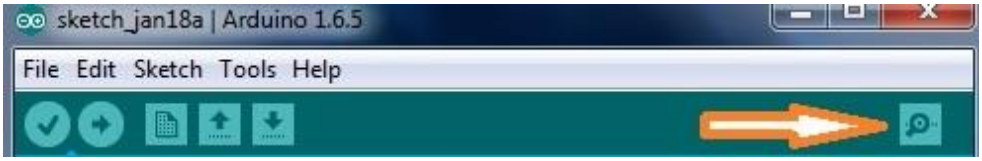
Na sam koniec chciałbym dodać, że najważniejszym na płytce jest mikrokontroler ATmega, który znajduje się na środku i może on działać samodzielnie po usunięciu z płytki. Więcej informacji na ten temat (programowania mikroprocesorów ATmega328 za pomocą Arduino) znajduje się w instrukcji dodawania nowych pomiarów (Inne pomiary), oraz na stronie producenta [3].

3.2. Oprogramowanie

Arduino IDE to środowisko programistyczne, które przeznaczone jest do pisania kodu dla mikrokontrolerów Otwartego Projektu Arduino, oraz urządzeń kompatybilnych. Mimo prostej formy program cechuje się bardzo rozbudowanymi dodatkami i bogatą dokumentacją. Język programowania Arduino jest oparty na środowisku Wiring.

Zaraz po uruchomieniu środowiska, ukazuje się pusty projekt. Od razu jest on gotowy do pracy i użytkownik może przejść do pisania kodu. Jeśli jednak ktoś zaczyna swoją pracę z elektroniką, może skorzystać z licznych opcji, które pomagają poprawnie zacząć nowy projekt. Jest to przede wszystkim baza gotowych projektów na stronie producenta oraz na wielu stronach poświęconym Otwartym Projekcie Arduino.

Środowisko posiada także wbudowany moduł przykładów dla wykorzystania konkretnych funkcjonalności oferowanych przez urządzenie. Stanowi on alternatywę dla gotowych projektów, udostępniając jedynie zarys kodu niezbędnego do skorzystania z danej funkcji (file -> examples).



Rysunek 5. Pasek narzędzi programu ArduinoIDE

Oprogramowanie to posiada również tak zwany *Serial Monitor*. Umożliwia on podgląd danych przesyłanych z Arduino na port szeregowy USB komputera. Może posłużyć również do wysyłania danych do Arduino. Serial Monitor uruchamiamy przez naciśnięcie przycisku zaznaczonego czerwoną strzałką na Rysunek 5, lub poprzez skrót klawiszowy Ctrl+Shift+M. Więcej na temat przesyłu danych w KOMUNIKACJA Z SYSTEMEM SCADA, zaś otwarty *Serial Monitor* znajduje się na Rysunek 16. Zrzut ekranu serial monitora ArduinoIDE, po podłączeniu do Arduino pomiarowego mikro turbiny gazowejRysunek 16 (Zmiana systemu SCADA). Widoczne jest tam pole tekstowe, pod nazwą okna dialogowego. Służy ono do wpisywania danych przekazywanych do mikrokontrolera po naciśnięciu przycisku „Send" znajdującego się po prawej stronie pola tekstowego.

4. MODERNIZACJA CZUJNIKÓW

Po zbudowaniu turbiny wystąpiły poważne problemy z niektórymi czujnikami. Były to: czujnik obrotów alternatora, oraz termopary. Wartości ich pomiaru znacząco odbiegały od rzeczywistości (można to było zweryfikować gołym okiem), w związku z czym konieczna była ich modyfikacja. Dodatkowym czynnikiem ograniczającym był czas wykonywania pętli pomiarowej, który nie mógł przekraczać 1 sekundy, ze względu na jakość wizualizacji pomiarów. W pierwotniej wersji pomiary były wykonywane w ok. 850ms. Resztę czasu mikrokontroler „czekał" aż minie pełna sekunda.

4.1. Czujnik obrotów alternatora

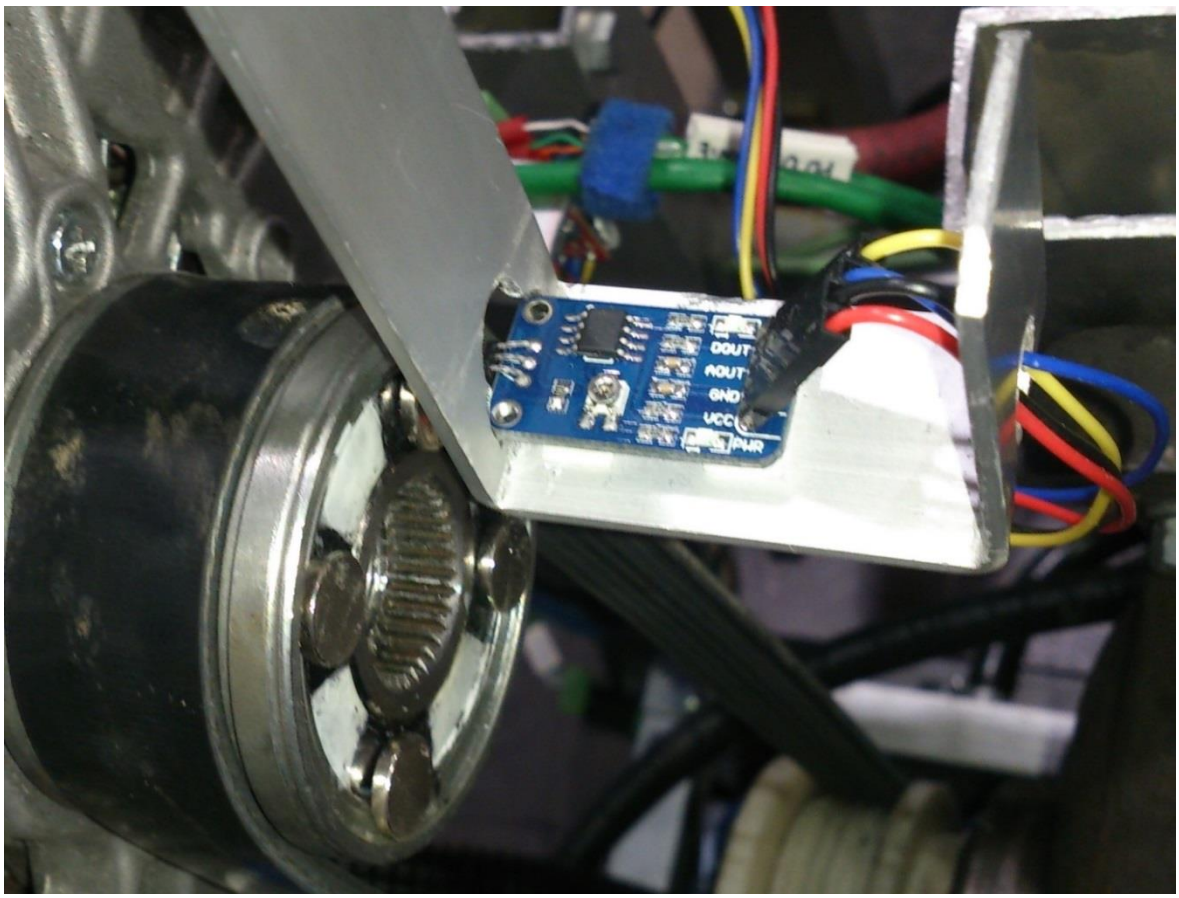
Pierwszą modyfikacją układu pomiarowego był demontaż istniejącego czujnika odbiciowego KTIR0711S (Rysunek 6), znajdującego się na ramie aluminiowej przymocowanej do alternatora. Konieczność jego wymiany wynikała z dużej niedokładności pomiaru w miejscach oświetlonych.

Rysunek 6. Czujnik odbiciowy (źródło: [1])

Następnie na miejsce poprzedniego czujnika został zamontowany *Czujnik Halla AH49E - moduł Waveshare*, oraz cztery magnesy neodymowe (Rysunek 7). Pod chipem można zauważyć białe podkładki, są one wykonane ze styropianu i zamocowane przez taśmę dwustronną w celu izolacji modułu od aluminiowej ramy.

Warto wspomnieć, że moduł AH49E może być bezpośrednio podpięty do mikrokontrolera pomiarowego, pod złącze cyfrowe, bez użycia komparatora ze skrzynki sterującej, ponieważ posiada go wbudowanego. Za pomocą potencjometru na chipie, można go wyregulować. Regulacja polega na ustawieniu napięcia referencyjnego dla czujnika.

Ostatecznie chip, został podłączony do mikrokontrolera Arduino poprzez swoje wyjście analogowe, w identyczny sposób co czujnik odbiciowy czyli przez komparator w skrzynce sterującej. A następnie wyregulowany poprzez odpowiedni potencjometr znajdujący się na tym komparatorze.



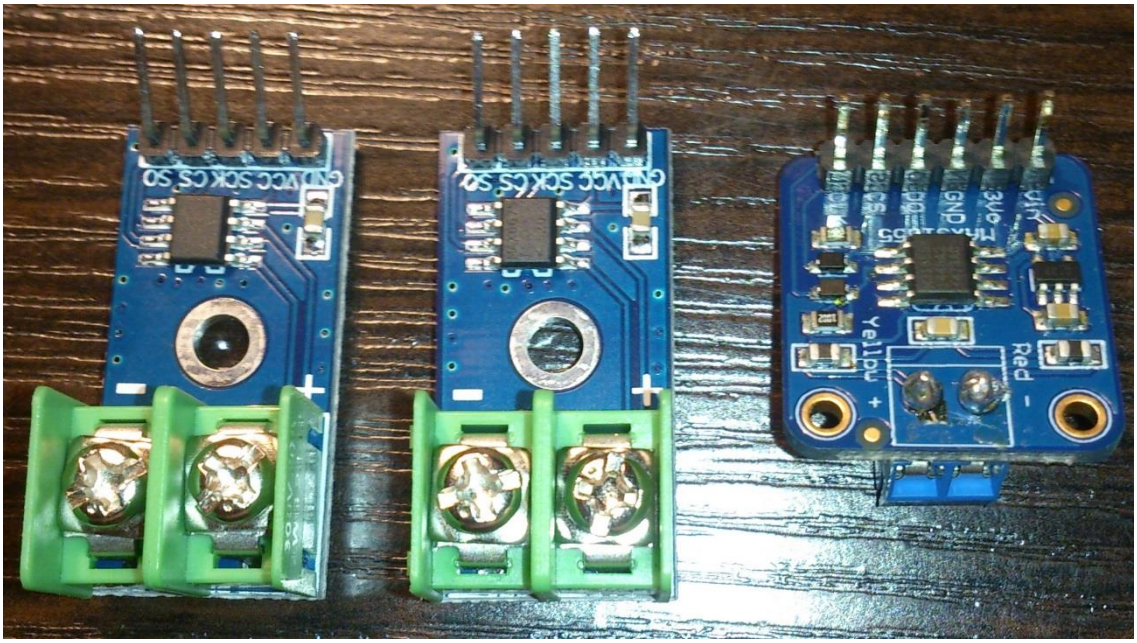
Rysunek 7. Zamontowany czujnik halla

4.2. Termopary

Jednymi z najważniejszych pomiarów w obiegu głównym turbiny gazowej jest pomiar temperatury w: komorze spalania, za pierwszym stopniem turbiny oraz za drugim stopniem turbiny. Problemem w podłączeniu termopar (w tym przypadku typu K) była wartość napięcia wyjściowego z tych termopar, które jest rzędu kilkudziesięciu mV. To uniemożliwiało sensowny pomiar napięcia, ponieważ napięcie referencyjne na płytce Arduino Mega wynosi 5V, co daje przy 10-bitowej dokładności mikrokontrolera (dokładność 4,8mV), czyli błąd pomiaru ponad $100^{O}C$.

Problem ten został rozwiązany poprzez zastosowane trzech modułów do termopar, realizujących pomiar i wysyłających jego wartość w postaci cyfrowej. Użyto dwóch chipów MAX6675 (wymiary: 32x16mm, maksymalna temperatura $1024^{O}C$, dokładność $0,25^{O}C$), do pomiaru temperatury między pierwszym, a drugim stopniem turbiny, oraz za jej drugim stopniem, na wylocie. Do pomiaru temperatury w komorze spalania został zastosowany moduł MAX31855 (wymiary: 20x20mm, maksymalna temperatura $1372^{O}C$, dokładność

0,25^{O}C). Tak duża temperatura oczywiście nie występuje przed samą turbiną. Górna granica technologiczna wynosi 1000^{O}C, w przeciwnym wypadku doszło by do uszkodzenia turbiny, lecz dzięki wcześniej przeprowadzonym pomiarom w [4] wiadome było, że temperatura wskazywana przez termoparę może wynosić nawet 1200^{O}C. Dzieje się tak prawdopodobnie przez nierównomierne spalanie w komorze spalania. Był to powód dla którego niezbędne było użycie czujnika droższego i o wyższej maksymalnej temperaturze pomiaru. Na Rysunek 8 widać użyte moduły, po prawej stronie znajduje się MAX31855.



Rysunek 8. Moduły do termopar

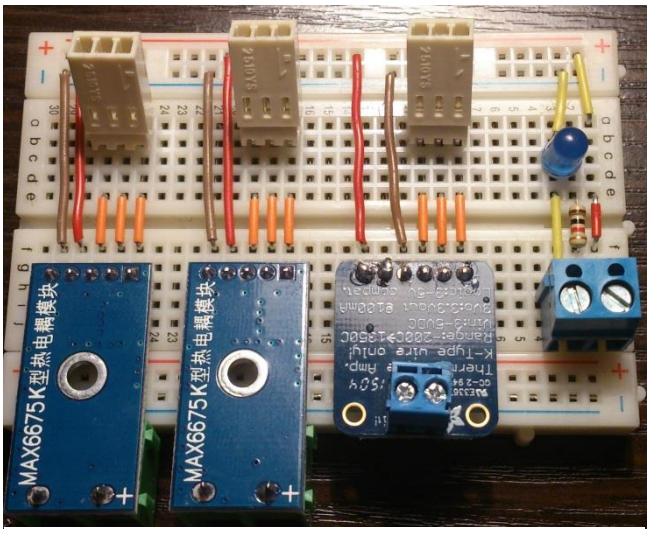
Obydwa moduły są typu 3-wire, czyli 3 złącza służą do odczytu danych. Z punktu widzenia mikrokontrolera nadrzędnego dwa z nich są sygnałami sterującymi, zaś trzeci służy do odbioru danych. Są to złącza o nazwach: CK, CLK, DO. Złącze CK mówi, czy rozpocząć konwersję i dalej przesłać dane. CLK jest to złącze taktowania zegarowego, zaś DO służy do odbioru wartości temperatury przez mikrokontroler. Wartości pomiarów są wysyłane w postaci cyfrowej.

Czas wykonania pomiaru wynosi dla modułu MAX6675 ok. 34ms, zaś dla modułu MAX31855 ok.68ms. Daje to w sumie 136ms czasu odczytu termopar co w sumie daje czas wykonywania pętli głównej prawie 990ms. Pomiary zostały przeprowadzone przez użycie własnej płytki mikrokontrolera Arduino UNO (Rysunek 10).

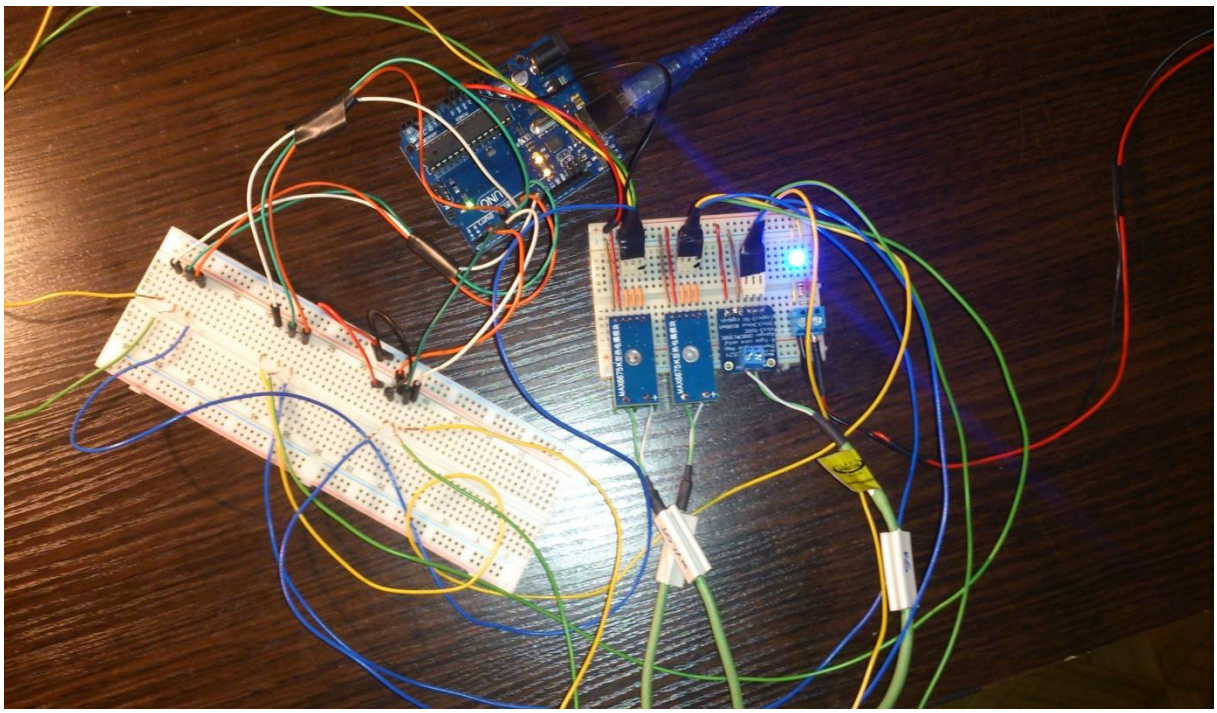
Problemem napotkanym podczas montażu była minimalna ilość wolnej przestrzeni w skrzynce sterującej. Problem ten został rozwiązany następująco. Moduły zostały zamontowane do płytki stykowej, do której zasilanie jest doprowadzone z komparatora (z powodu wolnych złączek z zasilaniem). Zbudowany układ został zamocowany w skrzynce

„do góry nogami”, do jej sufitu. Z lewej strony skrzynki zostały wywiercone trzy otwory 6mm na kable do termopar. Na Rysunek 9 znajduje się płytka ze zmontowanymi modułami. Niebieska dioda służy do sygnalizacji istnienia napięcia.

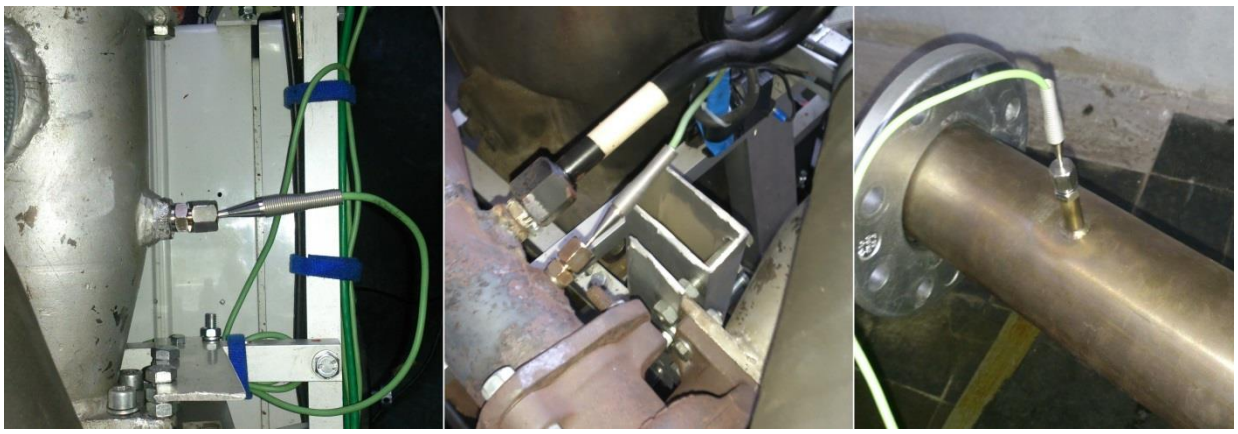
Ostatnim elementem montażu było podłączenie termopar do modułów znajdujących się w skrzynce sterującej, oraz zamocowanie ich w poszczególnych miejscach pomiarowych (Rysunek 11).



Rysunek 9. Płytka stykowa

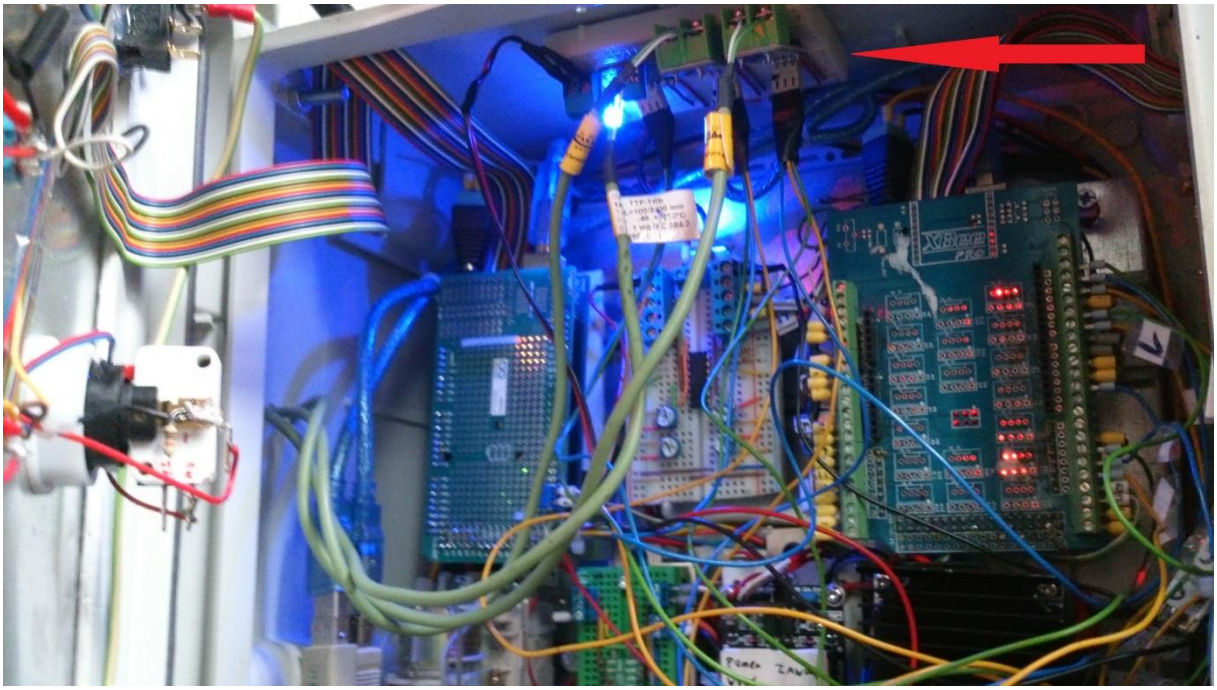


Rysunek 10. Realizacja testu płytki + pomiar czasu wykonywania pomiaru



Rysunek 11. Zainstalowane termopary - od lewej: komora spalania, międzystopniowa i wylotowa

Na Rysunek 12 widać zamontowaną płytkę stykową w skrzynce sterującej, wraz z modułami i złączami. Płytka została wskazana na rysunku czerwoną strzałką. Kable łączące moduły z mikrokontrolerem pomiarowym zostały oznaczone następująco: zółty – CLK, zielony – CS, niebieski – DO. Dodatkowo na każdy kabel została naklejona kartka do którego modułu termopary należy: KS – komora spalania, 1 – międzystopniowa, 2 – wylotowa.



Rysunek 12. Wnętrze szafy po montażu

5. KOD

Język programowania mikrokontrolera Arduino jest oparty na środowisku Wiring, czyli jest bardzo podobny do języka C/C++. Jest to dużo prostszy język w porównania do klasycznych języków programowania mikrokontrolerów. Osoby które opanowały podstawy języka C/C++ nie powinny mieć problemów z opanowaniem nowego języka. Realizacja dodatkowych funkcji, pętli i instrukcji warunkowych niczym się między nimi nie różni. Dzięki tej prostocie Arduino staje się coraz bardziej popularne wśród mikrokontrolerów. Istotną różnica języka Arduino w porównaniu z C/C++ są funkcję obsługi wejść i wyjść (I/O), oraz portu szeregowego USB. W następnym rozdziale został przedstawiony przykładowy kod w celu pokazania w praktyce obsługi wejść, wyjść, oraz portu szeregowego.

5.1. Przykładowy kod

Na Rysunek 13 znajduje się zrzut ekranu przykładowego programu w języku Arduino. Został on pokazany w celu zapoznania się czytelnika z podstawowymi instrukcjami wejść, wyjść, oraz obsługi portu szeregowego. Bardzo ułatwi to czytanie kodu mikrokontrolerów mikro turbiny gazowej dla osób dopiero poznających możliwości mikrokontrolera Arduino.

Kod działa w następujący sposób. Co około 300ms następuje pomiar z przycisku (zwracana wartość: 0 lub 1) i potencjometru (zwraca wartość z przedziału: 0-1023), oraz komunikacja obustronna mikrokontrolera z komputerem. Wysyłając z komputera odpowiednie liczby rozdzielone dowolnym znakiem, innym niż cyfra, sterujemy diodą led (włącz/wyłącz) oraz jasnością drugiej diody (złącze PWM). Po odczycie danych przychodzących do mikrokontrolera (o ile były wysłane) następuje transmisja danych do komputera, z informacją o stanie przycisku i wskazaniu potencjometru.

W pierwszych 4 linijkach występuje dyrektywa „#define”. Definiuje ona łańcuch znaków, który w momencie programowania mikrokontrolera zostaje zamieniony na zdefiniowaną liczbę. W liniach 6-8 występuje deklaracja zmiennych globalnych. Przed funkcją setup() należy deklarować wszystkie zmienne globalne i biblioteki. Trzeba także deklarować funkcje do obsługi większości modułów (chipów) np. BMP180 (parametry otoczenia), czy też moduły termopar.

```
1  #define fadeLed 3
2  #define przycisk 8
3  #define led_pin 12
4  #define sensor A0
5
6  int button_state = 0;
7  int sensor_value = 0;
8  unsigned long previousMillis = 0;
9
10 void setup()
11 {
12   Serial.begin(115200);
13   pinMode(buzzer, OUTPUT);
14   pinMode(przycisk, INPUT);
15   pinMode(led_pin, OUTPUT);
16 }
17
18 void loop()
19 {
20   while (Serial.available() > 0)
21   {
22   int led_state = Serial.parseInt();
23   int fadeValue = Serial.parseInt();
24
25   if (Serial.read() == '\n')
26     {
27           if (led_state == 1)
28           {
29           digitalWrite(led_pin, HIGH);
30           }
31           else
32           {
33           digitalWrite(led_pin, LOW);
34           }
35           delay(100);
36           analogWrite(fadeLed,fadeValue);
37     }
38   }
39   Serial.println(analogRead(sensor));
40   Serial.println(digitalRead(przycisk));
41   Serial.print("\n");
42   delay(200);
43 }
```

Rysunek 13. Zrzut ekranu przykładowego kodu dla mikrokontrolera Arduino

W linii 10 rozpoczyna się funkcja setup(). Od niej zaczyna się realizacja programu mikrokontrolera i jest wykonywana tylko jeden raz. Jeżeli w pętli głównej jest wykorzystywana komunikacja przez port szeregowy należy ją zdeklarować (linia 12). Więcej na ten temat komunikacji w 26. W funkcji setup() należy też zadeklarować złącza, a dokładniej czy będą one używane jako wejścia lub wyjścia. W najnowszych wersjach oprogramowania wszystkie złącza domyślnie ustawione są jako wejścia. Złącz analogowych nie trzeba deklarować, ponieważ domyślnie są one złączami wejścia. W przypadku wykorzystywania złącza analogowego jako cyfrowego, należy go zadeklarować, może być wykorzystywany jako wyjście. Wtedy oznaczenie zmienionego złącza jest zależne od rodzaju płytki mikrokontrolera, np. dla Arduino UNO A0 to 14, A1 to 15 i analogicznie następne.

W linii 18 rozpoczyna się druga funkcja specjalna loop(). Jest to nieskończona pętla realizowana po funkcji setup(), wykonywana do resetu, utraty zasilania lub awarii. W celu pokazania operacji innych niż w języku C/C++ nie zostaną opisane pętle, ani instrukcje warunkowe w funkcji specjalnej loop(). W następnym akapicie znajdują się instrukcje obsługi portu szeregowego (ang. Serial port). Więcej na temat komunikacji znajduje się w 26.

W linii 20 występuje instrukcja „Serial.available() > 0”. Instrukcja ta zwraca 1, jeżeli na buforze portu szeregowego znajdują się jakiekolwiek dane do odbioru, a 0 gdy ich nie ma. W linii 22 i 23 znajduje się instrukcja „Serial.praseInt()” odczytująca pierwszą liczbę

całkowitą z bufora. W przypadku odczytu innego znaku niż cyfra omija go (jeżeli nie znalazło jeszcze liczby), lub przerywa odczyt liczby (jeżeli już znalazło pewien ciąg cyfr). W linii 25 występuje instrukcja „Serial.read()”. Odczytuje ona jeden bajt dostępny na buforze danych, czyli jeden znak. W liniach 29 i 33 występuje instrukcja „digitalWrite(nr_złącza, wartość_logiczna)”. Ustawia ona napięcie na danym złączu na 0V lub 5V. W linii 35 i 42 występuje instrukcja „delay(wartość)”. Jest to instrukcja „uśpienia” mikrokontrolera, w której nie wykonuje on żadnych operacji. Wartość w nawiasie to czas uśpienia podawany w milisekundach. W linii 36 jest instrukcja „analogWrite(złącze, wartośćPWM)”. Realizuje ona modulacje omówioną w 11. Warunkiem dla instrukcji jest to, że dane złącze jest typu PWM, oraz wartośćPWM musi być w przedziale 0-255. W liniach 39 i 40 znajdują się instrukcje „analogRead(złącze)” oraz „digitalRead(złącze)”. Służą one do odczytu napięcia na poszczególnych złączach. „digitalRead zwraca wartości 0 lub 1, „analogRead” zaś wartość z przedziału 0-1023 odpowiadający napięciu z zakresu 0-5V.

5.2. Opis zmian w kodzie mikrokontrolera pomiarowego

Większość kodu została napisana przez inż. Jana Pleszyńskiego. Dokonana została jego modernizacja. Głównie było to usunięcie zbędnych fragmentów kodu, oraz dodanie funkcji realizujących odczyt termopar.

Najważniejszym celem modernizacji kodu było umożliwienie dodania pomiaru z termopar, bez zmieniania czasu transmisji danych (1 sekundy) przez port szeregowy USB. Było to bardzo ważne ze względu na jakość wizualizacji danych. Co więcej zmniejszanie czasu wykonywania pomiarów w pętli głównej, w kodzie, umożliwia w przyszłości dodanie nowych pomiarów bez zmieniania czasu transmisji danych przez port szeregowy. Opis kodu dotyczącego pomiaru z termopar znajduje się w 35.

Z wcześniejszych rozważań na temat czasu realizacji pętli mikrokontrolera pomiaru po dodaniu modułów termopar (16), wynika, że czas wykonywania wszystkich pomiarów i komunikacji wynosi prawie 990 ms. Jest to mniej niż 1 sekunda, ale czas wykonywania pętli ten nie jest jednak sztywno określony, ponieważ komunikacja nie odbywa się w sposób ciągły. To znaczy, że mikrokontroler sterujący próbuje skomunikować się z mikrokontrolerem pomiarowym co jakiś czas. Sterownik robi to po wykonaniu określonych przez programistę funkcjach, które mogą zająć kilka, a nawet kilkadziesiąt milisekund. Oczekiwanie na sygnał od mikrokontrolera sterującego może spowalniać czas wykonywania pętli głównej na tyle, że przekroczy ona czas wykonywania 1 sekundy. Z tego powodu

konieczne było usunięcie wszystkich zbędnych linijek kodu, aby zyskać kilkanaście/kilkadziesiąt milisekund zapasu.

Największymi usuniętymi elementami było wykrywanie i usuwanie błędów (debugowanie), oraz przesył danych z mikrokontrolera sterującego do pomiarowego. Debugowanie zostało usunięte z powodu braku detekcji błędów zarówno pomiarowych, jak i komunikacyjnych. Komunikacja natomiast została usunięta z tego powodu, że dane odczytane przez mikrokontroler pomiarowy nie były nigdzie wykorzystywane. Były to informacje na temat stanu pompy, zapłonu, zaworu odcinającego i przepustnicy (czy włączone, czy wyłączone), oraz informacje na temat stanu otwarcia zaworu gazu (propanu), oraz mocy silnika rozruchowego.

Dodatkową godną uwagi korektą była zmiana paczki danych wysyłanych przez mikrokontrolera pomiarowego do sterującego. Przed zmianą dane dla mikrokontrolera sterującego wysyłane były w 5 paczkach (ponieważ wysyłanych jest 5 pomiarów). Obecnie dane przed wysłaniem do sterownika są zbierane, a następnie wysyłane w jednej paczce danych. Zabieg ten był ważny z punktu widzenia bezpieczeństwa pracy. Dzięki niemu znacząco zmniejszyło się prawdopodobieństwo wystąpienia błędu odczytu wartości przez mikrokontroler sterujący. Dokładniej jeżeli odbiór danych przez mikrokontroler sterujący wystąpił by minimalnie wcześniej niż wysyłanie danych z mikrokontrolera pomiarowego (różnica rzędu kilku mikrosekund), to przy wysokich wartościach (długich łańcuchach danych) mógł nastąpić zbyt wczesny odbiór wszystkich danych przez sterownik. Skutkować to mogło by niepełnym odczytem danych.

5.3. Kod mikrokontrolera pomiarowego

Z powodu, że kod ma ponad 300 linijek, uproszczony został jego opis do pętli głównej i jej omówienia. W kodzie (szkic w programie Arduino) numery linii mogą trochę różnić się od przedstawionych w tym podrozdziale. Spowodowane jest to ciągłymi kosmetycznymi poprawkami, oraz dodawaniem/usuwaniem komentarzy. Dokładny opis funkcji realizujących pomiar z termopar znajduje się w 35.

Na Rysunek 14 znajduje się fragment kodu realizowanego przez mikrokontroler pomiarowy. Fragment ten to funkcje specjalne setup() i loop(), omówione w 22. W funkcji setup() znajduje się deklaracja komunikacji z komputerem przez port szeregowy – „Serial.begin(prędkość_transmisji)”, oraz deklaracja komunikacji z Arduino pomiarowym – „Serial1.begin(prędkość_transmisji)”. Więcej informacji na ten temat znajduje się w 26. W

funkcji setup() w linii 127 znajduje się deklaracja czujnika BMP180, mierzącego ciśnienie i temperaturę otoczenia.

```
123 void setup()
124 {
125   Serial.begin(115200);
126   Serial1.begin(115200);
127   BMP180.begin();
128 }
129
130 void loop()
131 {
132   cycle_start = millis();
133   start_DS18B20_conversion();
134   read_BMP180();
135   read_AS();
136   convert_AS();
137   FS_timeout = conversion_time - (millis() -  cycle_start);
138   read_FS_freqs();
139   read_DS18B20();
140   thermocouples();
141   communicate_with_control();
142   vals_str += ("\n");
143   do
144   {
145   Serial.print(vals_str); vals_str = "";
146   }
147   while (millis()-cycle_start<cycle_time-1);
148   loop_counter++;
149 }
```

Rysunek 14. Zrzut ekranu głównej części kodu mikrokontrolera pomiarowego

Pętla loop() rozpoczyna się zmierzeniem czasu wewnętrznego mikrokontrolera instrukcją „millis()”, oraz zapisu tej wartości. Jest to czas mierzony w milisekundach od ostatniego restartu układu Arduino. Następnie w linii 133 znajduje się funkcja informująca cyfrowe czujniki temperatury, o nazwie DS18B20, aby rozpoczęły konwersję temperatury, która trwa 700ms. Konwersja odbywa się wewnątrz czujników, więc czas konwersji nie wpływa na czas wykonywania reszty kodu. W linii 134 znajduje się funkcja odczytująca czujnik parametrów otoczenia. Następnie w linii 135 następuje odczyt napięcia zwracanego przez wszystkie czujniki analogowe. Dokładniejszy opis złącz oraz typów czujników do nich podłączonych znajduje się w 28. W linii 136 jest funkcja konwertująca uzyskane wcześniej wartości napięć na odpowiednie jednostki. Następnie w linii 137 następuje pomiar czasu pozostałego do rozpoczęcia odbioru danych z czujników DS18B20. Czas ten definiuje czas do realizacji odczytu czujników częstotliwości, których funkcja odczytu znajduje się w linii 138. Po 700ms od rozpoczęcia konwersji czujników DS18B20, następuje ich odczyt (funkcja w

linii 139). Następnie zostaje realizowana funkcja odczytu termopar (linia 140). Ten fragment kodu zostanie dokładnie omówiony w 35. W następnej linii znajduje się funkcja realizująca komunikacje z mikrokontrolerem sterującym.

W linii 142 znajduje się instrukcja dodająca do łańcucha znaków „vals_str” znaku końca linii – „\n”. Łańcuch jest zbierany przez cały okres działania pętli i zawiera wartości wszystkich pomiarów. Opis formy przedstawienia tych pomiarów znajduje się w 28. W liniach 143-147 znajduje się pętla realizująca wysłanie łańcucha znaków zawierającego wartości pomiarów. Po wysłaniu danych, zmienna przechowująca łańcuch znaków zostaje wyczyszczona. Mikrokontroler wysyła później puste paczki danych, do momentu gdy czas wykonywania pętli nie będzie większy niż 999ms. W ostatniej linii pętli głównej (148) znajduje się zmienna licząca ilość wykonanych pętli.

6. KOMUNIKACJA Z SYSTEMEM SCADA

Każdy układ Arduino posiada przynajmniej jeden port szeregowy (ang. serial port), wersja mega posiada ich aż cztery. Umożliwia on proste przesyłanie danych do i z mikrokontrolera. Z pozycji programu ArduinoIDE możliwe jest uruchomienie terminala, za pomocą którego można odczytać dane wysyłane przez mikrokontroler, jak również wysłać dane do mikrokontrolera. Komunikacja może występować pomiędzy mikrokontrolerem arduino i komputerem, jak również pomiędzy dwoma mikrokontrolerami Arduino.

6.1. Komunikacja przez port szeregowy USB

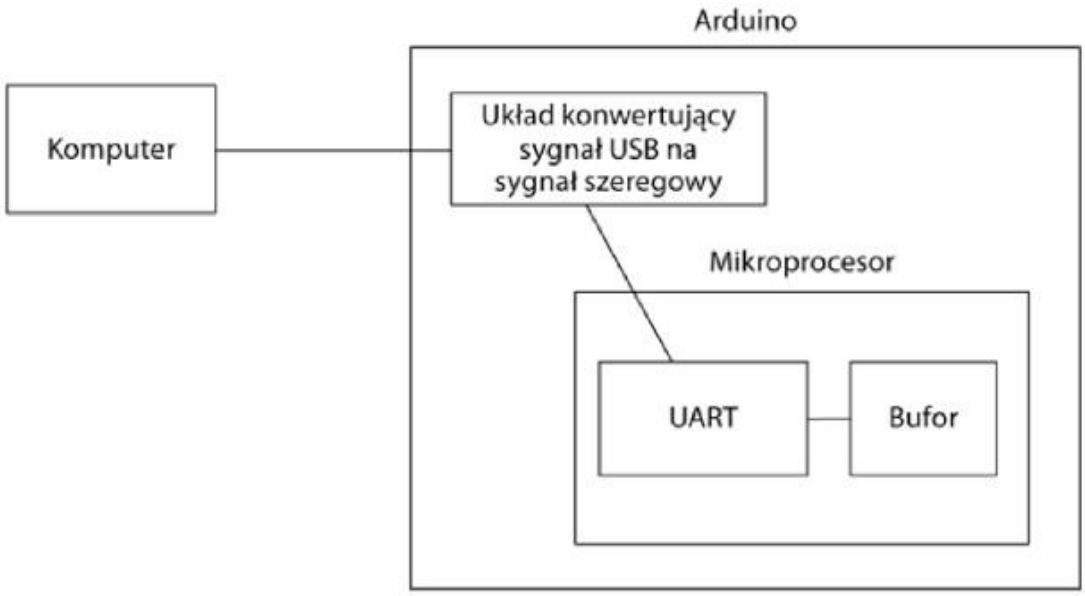
Aby używać poszczególnych portów szeregowych należy je wcześniej zadeklarować w funkcji specjalnej setup() poprzez komendę „Serial.begin(prędkość_transmisji, sposób_wysyłania_danych)”. Prędkość transmisji (ang. baud rate) podawana jest w bitach na sekundę. Jej typowe wartości to: 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, lub 115200. Na stronie producenta maksymalna podana standardowa prędkość to 115200, zaś w najnowszym oprogramowaniu dopuszczalna prędkość wynosi 250000. Możliwe jest również zdefiniowanie innej nietypowej prędkości transmisji (poprzez podanie jej wartości), ale producent nie gwarantuje udanej transmisji danych.

Sposób wysyłania danych mówi nam jak wygląda wysyłanie danych w paczce. Paczka danych składa się od 5 do 8 bitów, bitu kontrolnego oraz jednego lub dwóch bitów stop. Bit kontrolny obliczany jest przez kontrolę parzystości – oznaczenie E (eng. Even), lub nieparzystości – oznaczenie O (eng. Odd), czy też bez kontroli – oznaczenie N (non). Jeżeli nie podany zostanie sposób wysyłania danych domyślnie zostanie wybrany sposób *SERIAL_8N1*, czyli 8 bitów (bajt) danych, bez kontroli parzystości i 1 bit stop.

W mikrokontrolerach w turbinie gazowej komunikacja z komputerem oraz pomiędzy sobą jest realizowana standardowym sposobem przesyłu danych, oraz z prędkością 115200 bitów na sekundę. Instrukcje korzystania z portu szeregowego zostały przedstawione w przykładzie kodu (5.2).

Odbiór danych przez port szeregowy to nie tylko przesył danych z innego urządzenia, ale także konwersja sygnału USB na sygnał możliwy do odczytu przez mikrokontroler. Jest to także bufor danych, który przechowuje otrzymane dane, do momentu ich odczytu przez mikroprocesor. Dzięki temu nie jest konieczna synchronizacja dwóch mikrokontrolerów. Należy jedynie na tyle często odczytywać bufor, by go nie „zapchać”, ponieważ ma on

określoną pojemność. W układzie arduino pojemność ta wynosi 128 bajtów. W chwili odczytu danych z bufora są z one z niego usuwane. Na Rysunek 15 został przedstawiony schemat blokowy transferu informacji z komputera do mikrokontrolera Arduino.



Rysunek 15. Schemat blokowy odbioru danych przez mikrokontroler Arduino (źródło: [5])

6.2. Mapowanie

Najważniejszym elementem komunikacji, po obu stronach (mikrokontrolera i komputera) było mapowanie czyli stworzenie pewnej ogólnej nomenklatury pomiarów. Została ona stworzona w dwóch wersjach. Pierwsza z nich to wersja z punktu widzenia programisty Arduino, zaś druga z punktu widzenia programisty systemu SCADA.

Każdy punkt pomiarowy otrzymał unikalny numer (ID), który składa się z trzech cyfr. W Tabela 1cyfry te zostały oznaczone jako X, Y, Z. Dwie pierwsze odpowiadają za miejsce, zaś ostatnia za typ pomiaru. Tabela 1 to legenda oznaczeń punktów pomiarowych.

Tabela 1. Legenda nomenklatury pomiarów

XYZ: [wartość] - forma transmisji								
X (który obieg)		Y (miejsce)					Z (co?)	
		\	dla X = 0 (spal/pow)	dla X = 1 (olej)	dla X = 2 (woda KN)	dla X = 3 (mechanika)		
0	obieg główny turbiny	0	otoczenie	przed WC	przed KN	wał turbosprężarki	0	temperatura
1	olej	1	przed sprężarką	za WC	w KN	wał alternatora	1	ciśnienie
2	woda (komora natryskowa)	2	za sprężarką	za pompą	za KN (zbiornik)		2	przepływ
3	mechaniczne	3	gaz (dopływ)				3	obroty
		4	przed turbo (TS)				4	napięcie
		5	za turbo (TS)				5	Prąd
		6	za turbiną (TC)				6	poziom wody
		7	generator				7	
		8	za KN				8	
		9					9	

Została również ustalona forma przesyłania poszczególnych danych, czyli informacji o punkcie pomiarowym, oraz odczytanej i przekonwertowanej wartości z danego czujnika. Forma ta jest następująca „XYZ:wartość_pomiaru”, gdzie XYZ to wcześniej przedstawione ID punktu. Dodatkowo na końcu dodawany jest znak końca linii „\n”.

W Tabela 2 znajduje się opis podłączeń określonych urządzeń pomiarowych do poszczególnych złącz mikrokontrolera.

We wszystkich tabelach w tym podrozdziale użyto następujących kolorów czcionek w celu sygnalizacji nietypowości złącza lub jego niepełnego wyprowadzenia. Kolorem czerwonym oznaczono złącza używane we wcześniejszych wersjach automatyki, obecnie zostało po nim tylko wyprowadzenie na zewnątrz skrzynki. Kolorem żółtym oznaczono złącze wyprowadzone ze skrzynki, do punktu pomiarowego, lecz na chwilę obecną brak jest urządzenia pomiarowego. Kolorem zielonym oznaczono złącza wyprowadzone ze skrzynki, do punktu pomiarowego, jednakże brakuje dokładnych informacji o pomiarze.

Do złącza cyfrowego 10 jest podpięty kabel, lecz brakuje informacji na temat jego wyprowadzenia.

Tabela 2. Nomenklatura pomiarów dla programisty Arduino

typ pinu	nr pinu	ID	Opis	urządzenie pomiarowe
analog	0	031	ciśnienie gazu (propan)	Wika A-10
analog	1	032	przepływ gazu (propan)	Omron-50A
analog	2	121	ciśnienie za pompą oleju	Wika A-10
analog	3	021	ciśnienie za sprężarką	Wika A-10
analog	4	-	-	-
analog	5	-	-	-
analog	6	012	przepływ powietrza do sprężarki	BOSH hfm5
analog	7	010	temperatura przed sprężarką	BOSH hfm5
analog	8	051	ciśnienie za TS / przed TC	Wika A-10
analog	9	201	ciśnienie przed KN	Wika A-10
analog	10	075	prąd z generatora	Pololu ACS709
analog	11	074	napięcie na generatorze	dzielnik napięcia
analog	12	z	pierwotnie miały być używane jako wejście do termopar	brak
analog	13	z		
analog	14	z		
analog	15	z		

digital	0	Serial	możliwość komunikacji z komputerem (przez USB)	-
digital	1			
digital	2	-	-	-
digital	3	050	Termopara międzystopniowa - pin do odbioru danych (DO)	Moduł MAX6675 + termopara K
digital	4	122	pierwotnie mierzył przepływ oleju	Adafruit 833
digital	5	202	przepływ przed KN	???
digital	6	313	obroty wału alternatora	czujnik halla
digital	7	303	obroty wału turbosprężarki	czujnik optyczny
digital	8	050	Termopara międzystopniowa - piny kolejno CLK, CS	termopara K
digital	9			
digital	10	???	???	???
digital	11			
digital	12	040	Pomiar temperatury w komorze spalania piny kolejno CLK, DO, CS	Moduł MAX31855 + termopara K
digital	13			
digital	14	wiele	temperatury do 125°C	DS18B20
digital	15			
digital	16	060	Pomiar temperatury za drugim stopniem turbiny – piny kolejno CS, DO, CLK	Moduł MAX6675 + termopara K
digital	17			
digital	18	Serial1	Komunikacja z arduino sterującym	-
digital	19			
digital	20	000	parametry otoczenia (temperatura i ciśnienie)	BMP180
digital	21	001		

Pozostałe złącza o numerach 22 do 54 są wolne. W przypadku chęci zagospodarowania tych wolnych złącz należy wlutować przewody w odpowiednie miejsca na płytce Arduino, ponieważ nie posiadają one wejść dla tulejek, ani złącz skręcanych.

Następne tabele o numerach: Tabela 3, Tabela 4, Tabela 5 i Tabela 6 przedstawiają mapowanie z punktu widzenia programisty systemu SCADA, kolejno dla: obiegu głównego, obiegu olejowego, komory natryskowej i mechaniki.

Tabela 3. Nomenklatura pomiarów dla programisty systemu SCADA, część I

OBIEG GŁÓWNY

ID	jednostka	typ pinu	nr pinu	opis
000	^{O}C	digital	20/21	temperatura otoczenia
001	hPa	digital	20/21	ciśnienie otoczenia
010	^{O}C	analog	7	temperatura przed sprężarką
012	kg/h	analog	6	przepływ powietrza do sprężarki
020	^{O}C	digital	14	temperatura za sprężarką
021	bar	analog	3	ciśnienie za sprężarką
030	^{O}C	digital	14	temperatura gazu (propan)
031	bar	analog	0	ciśnienie gazu (propan)
032	l/min	analog	1	przepływ gazu (propan)
040	^{O}C	digital	11-13	temperatura KS
050	^{O}C	digital	3,8,9	temperatura za TS / przed TC
051	bar	analog	8	ciśnienie za TS / przed TC
060	^{O}C	digital	15-17	temperatura za TC
074	V	analog	11	napięcie na generatorze
075	A	analog	10	prąd z generatora
080	^{O}C	digital	14	temperatura za komorą natryskową

Tabela 4. Nomenklatura pomiarów dla programisty systemu SCADA, część II

OBIEG OLEJOWY

ID	jednostka	typ pinu	nr pinu	opis
100	^{O}C	digital	14	temperatura przed WC
110	^{O}C	digital	14	temperatura za WC
121	bar	analog	2	ciśnienie za pompą
122	l/min	digital	4	przepływ oleju

Tabela 5. Nomenklatura pomiarów dla programisty systemu SCADA, część III

KOMORA NATRYSKOWA

ID	jednostka	typ pinu	nr pinu	opis
200	^{O}C	digital	14	temperatura przed KN
201	bar	analog	9	ciśnienie przed KN
202	b/d	digital	5	przepływ przed KN
220	^{O}C	digital	14	temperatura za KN (zbiornik)

Tabela 6. Nomenklatura pomiarów dla programisty systemu SCADA, część IV

MECHANIKA

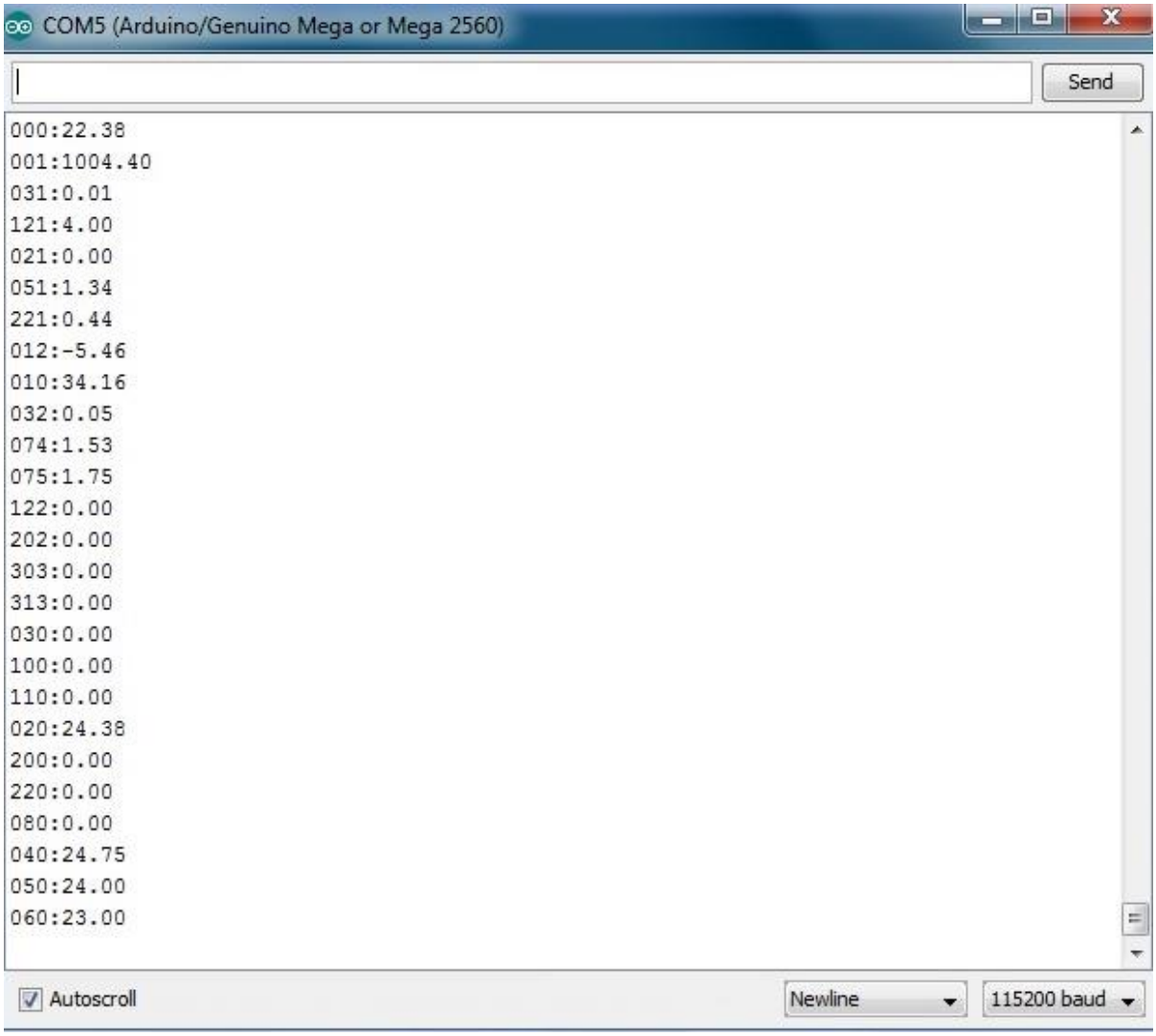
ID	jednostka	typ pinu	nr pinu	opis
303	Hz	digital	7	obroty wału turbosprężarki
313	obr/min	digital	6	obroty wału alternatora

6.3. Zmiana systemu SCADA

Zmiana systemu SCADA jest bardzo prosta w porównaniu ze stworzeniem takiego systemu. Należy wiedzieć jak wysyłane są dane przez mikrokontroler Arduino, oraz znać nazwę portu USB połączonego z danych mikrokontrolerem.

Wiedza na temat wysyłania danych składa się z dwóch elementów: jest to rodzaj transferu (prędkość i sposób – 26), oraz forma wysyłania danych (powyższa nomenklatura). W przypadku rodzaju transferu najczęściej manipulowana jest prędkość transmisji, zaś sposób jest standardowy, czyli jest 8 bitowy, bez kontroli parzystości, z 1 bitem stop (*SERIAL_8N1*). Jest tak, ponieważ jest to standardowy sposób transmisji dla znacznej większości środowisk programistycznych. Natomiast dane pomiarowe, są wysyłane według nomenklatury omówionej w poprzednim podrozdziale. Jest tych pomiarów 26. Między każdym pomiarem jest znak nowej linii „\n”. Na końcu paczki danych z pomiarami jest dodatkowy znak nowej linii w celu zasygnalizowania końca paczki. Na Rysunek 16 znajduje się zrzut ekranu z programu *ArduinoIDE*, który ilustruje dane przekazywane z mikrokontrolera pomiarowego i

odczytywane przez komputer. Na rysunku wyświetlona jest cała paczka omówionych wcześniej danych.



Rysunek 16. Zrzut ekranu serial monitora ArduinoIDE, po podłączeniu do Arduino pomiarowego mikro turbiny gazowej

Znając te dane możemy stworzyć dla tego układu nowy system SCADA, w dowolnym środowisku programistycznym. System może pobierać dane bezpośrednio z portu szeregowego, lub można napisać skrypt odczytujący dane od mikrokontrolera, konwertujący je i przekazujący dalej do systemu SCADA. Ten drugi sposób został zrealizowany przez Krzysztofa Setlaka, więcej na ten temat w [2]. Skrypt odczytujący dane został napisany przez niego w środowisku pythonowym.

7. INSTRUKCJA DODAWANIA NOWYCH POMIARÓW

Obecnie dodawanie nowych pomiarów wiąże się ze zmianą czasu wykonywania pętli głównej kodu. Jest to problematyczne ze względu na jakość odczytywanych pomiarów. Wyjątkiem jest prosty odczyt ze złącza analogowego, oraz czujnik temperatury DS18B20. Z tego powodu w tym rozdziale znajduje się instrukcja dodania pomiaru temperatury z czujnika DS18B20. W drugim podrozdziale znajduje się instrukcja dodawania nowych termopar. Skupia się ona na samym kodzie.

W chwili obecnej poważnym problemem jest dodanie nowego pomiaru wykonywanego z prędkością większą niż kilka milisekund, jakim jest np. pomiar z termopar poprzez moduły MAX6675 czy MAX31855. Powodem tego problemu jest zmniejszenie częstotliwości przesyłu danych z mikrokontrolera pomiarowego. Wiąże się to nie tylko ze spadkiem jakości pomiarów w systemie SCADA, ale również spowolnieniem reakcji mikrokontrolera sterującego, co skutkuje spadkiem poziomu bezpieczeństwa. Z powyższych powodów zostanie w podrozdziale trzecim omówiony inny sposób dodawania pomiarów. Dokładniej przeniesienia części z nich do podrzędnego mikrokontrolera, który będzie komunikował się ze swoim nadrzędnym kontrolerem przez port szeregowy. Opcja ta była rozważana przy montażu modułów termopar. Dzięki redukcji kodu (22) problem został usunięty.

7.1. Czujnik temperatury DS18B20

Komunikacja pomiędzy mikrokontrolerem, a czujnikiem wykorzystuje tylko bibliotekę „OneWire”, służy ona dla mikrokontrolera do komunikacji ze wszystkimi czujnikami za pomocą jednego złącza cyfrowego. Obecnie używa się najczęściej także bibliotekę „DallasTemperature”, ponieważ bardzo upraszcza ona kod i nie wymaga wcześniejszego sprawdzenia adresu czujnika.

Aby dodać pomiar z czujnika DS18B20, pierwszą rzeczą jaką należy wykonać to pobranie unikalnego adresu nowego czujnika. Można to zrobić za pomocą dowolnej płytki Arduino i wcześniej wspomnianej biblioteki „DallasTemperature”. Prosty instruktarz jak jej używać znajduje pod linkiem: *https://www.youtube.com/watch?v=fa6LYTZcv2s*. Następnym etapem jest modyfikacji deklaracji czujników. Na Rysunek 17 znajduje się zrzut ekranu

fragmentu kodu mikrokontrolera pomiarowego. Ten fragment kodu dotyczy czujników DS18B20. Jest to jedyny element kodu, który należy edytować, aby dodać nowy czujnik.

```
79 #define number_of_DS18B20  7   // ilosc czujnikow
80 byte DS18B20_pin = 14;       // zlacze
81 OneWire  ds(DS18B20_pin); // inicjalizacja magistrali OneWire obslugujacej czujniki temp ds18b20
82
83 byte DS18B20_ROMS[number_of_DS18B20 * 8] =
84                     {0x28,   0xFF,   0x47,   0x7,    0x11,   0x14,   0x0,    0x40, // Gaz przed KS
85                      0x28, 0xFF, 0xB1, 0x39, 0x10, 0x14, 0x0,  0x8C, // Olej przed WC
86                      0x28, 0xFF, 0xC4, 0xC,  0x11, 0x14, 0x0,  0x51, // Olej za WC
87                      0x28,   0xFF,   0xB4,   0x11,   0x11,   0x14,   0x0,  0x99, // Powietrze za sprezarka
88                      0x28, 0xFF,   0xC0,   0x33,   0x10,   0x14,   0x0,  0x37, // Woda KN
89                      0x28, 0xFF,   0x4C,   0x7,    0x11,   0x14,   0x0,    0x30, // Woda zbiornik
90                      0x28,   0xFF,   0xC0,   0x36,   0x11,   0x14, 0x0,  0x1D,}; // Spaliny za KN
91
92 String DS18B20_places[] = {"03",  "10", "11", "02", "20", "22", "08"}; // w tym miejscu znajdują sie adres miejsca
93
94 float DS18B20_temps[number_of_DS18B20];
```

Rysunek 17. Zrzut ekranu fragmentu kodu mikrokontrolera pomiarowego, dotyczącego czujnika DS18B20

Pierwszą czynnością jest zmiana ilości odczytywanych czujników w linii 79. W następnej linii znajduje się deklaracja złącza odczytującego czujniki. W linii 81 znajduje inicjalizacja magistrali OneWire obsługującej czujniki DS18B20. Następnie w liniach 83-90 znajduje się tablica zawierająca adresy używanych czujników. Należy tu wprowadzić na końcu tablicy adres nowego czujnika. Ostatnim elementem modyfikacji kodu jest dodanie do tablicy, znajdującej się w linii 92, adresu nowego pomiaru. Trzeba go nadać według nomenklatury znajdującej się w 28.

W pozostałych elementach kodu odpowiedzialnych za pomiar, funkcje same dostosowują się do zwiększonej liczby czujników. Czas konwersji temperatury na sygnał możliwy do odczytu przez mikrokontroler wynosi 700ms i nie zależy od ilości czujników.

7.2. Termopara

W tym podrozdziale zostanie dokładnie przedstawiona funkcja „thermocouples()” wykonująca pomiary z termopar typu K za pomocą modułów MAX6675 i MAX31855. Instrukcja dotyczy tylko kodu programu mikrokontrolera. Omówiona zostanie obsługa wyżej wymienionych modułów. Obsługa tych chipów jest bardzo podobna z punktu widzenia osoby nie zagłębiającej się w wykorzystywane biblioteki. Dodatkową różnicą jest fakt, że moduł MAX31855 posiada wbudowany termometr wewnętrzny do mierzenia temperatury chipu.

Pomiar tej temperatury przy wykorzystaniu standardowej biblioteki nie jest możliwy do ominięcia, przez co czas konwersji modułu MAX31855 jest dwa razy dłuższy niż MAX6675.

Na rysunkach Rysunek 18 i Rysunek 19 zaprezentowane zostały zrzuty ekranu kodu mikrokontrolera Arduino wraz z komentarzami. Pierwszy z rysunków dotyczy deklaracji użycia poszczególnych modułów. Fragment ten znajduje się przed funkcją specjalną setup(). Dodatkowy elementem, nie pokazanym na żadnym rysunku jest deklaracja bibliotek. Nazywają się one następująco: „Adafruit_MAX31855.h" i „max6675.h".

```
96  //Czujniki cyfrowe do termopar MAX6675 (x2) i MAX31855 (do KS)
97  float KS_temp;
98  float therm_1;
99  float therm_2;
100
101 Adafruit_MAX31855 thermocouple_KS(11,13,12);  // W kolejności: CLK, CS, DO
102 MAX6675 thermocouple_1(8,9,3);                // W kolejności: CLK, CS, DO
103 MAX6675 thermocouple_2(17,15,16);             // W kolejności: CLK, CS, DO
```

Rysunek 18. Zrzut ekranu fragmentu kodu mikrokontrolera pomiarowego, dotyczącego odczytu z termopar, część I

W liniach 97-99 występuje deklaracja zmiennych przechowujących wartości poszczególnych pomiarów. W liniach 101-103 następuje deklaracja obiektów poszczególnych pomiarów. Deklaracja odbywa się przez podanie nazwy klasy, nazwy obiektu wywołującego pomiar i numerów złącz obsługujących dany moduł. W tym przypadku klasy nazywają się „Adafruit_MAX31855", oraz „MAX6675". Nazwa obiektu jest dowolna, zaś złącza muszą być podawane w odpowiedniej kolejności: CLK (SCK), CK (CS), DO (SO). Oznaczenia w nawiasach pochodzą z nazewnictwa stosowanego w części azjatyckich fabryk.

Na następnym rysunku znajduje się fragment kodu wykonywanego przez funkcję „thermocouples()", wspomnianą w 23. W kodzie funkcja ta znajduje się w oddzielnej karcie, nazwanej „termopary".

W liniach 3-5 znajduje się odczyt temperatury przez termopary, oraz przyporządkowanie ich wartości do poszczególnych zmiennych. W następnych liniach następuje sprawdzenie czy odczyt został wykonany i dopisanie pomiaru do łańcuchu znaków wysyłanego przez port szeregowy, do komputera. W przypadku błędu w trakcie wykonywania pomiaru zostaje wysłana wartość -999. Linie 8-17 sprawdzają moduł MAX31855, 20-26 modułu MAX6675 mierzący temperaturę między stopniami turbinowi. Ostatni fragment (linie 29-35) dotyczą również modułu MAX6675, ale na wylocie z drugiego stopnia turbiny.

```
void thermocouples()
{
   KS_temp = thermocouple_KS.readCelsius(); // MAX31855 - KS
   therm_1 = thermocouple_1.readCelsius(); // MAX6675 - miedzystopniowe
   therm_2 = thermocouple_2.readCelsius(); // MAX6675 - za TC

   //MAX31855 - KS
   if (isnan(KS_temp))
   {
     vals_str += ("040:-999\n");
   }
   else
   {
     vals_str += ("040:");
     vals_str += (KS_temp);
     vals_str += ("\n");
   }

   // MAX6675 - miedzystopniowe
   if (isnan(therm_1)) {
     vals_str += ("050:-999\n");
   } else {
     vals_str += ("050:");
     vals_str += (therm_1);
     vals_str += ("\n");
   }

  // MAX6675 - za TC
   if (isnan(therm_2)) {
     vals_str += ("060:-999\n");
   } else {
     vals_str += ("060:");
     vals_str += (therm_2);
     vals_str += ("\n");
   }
}
```

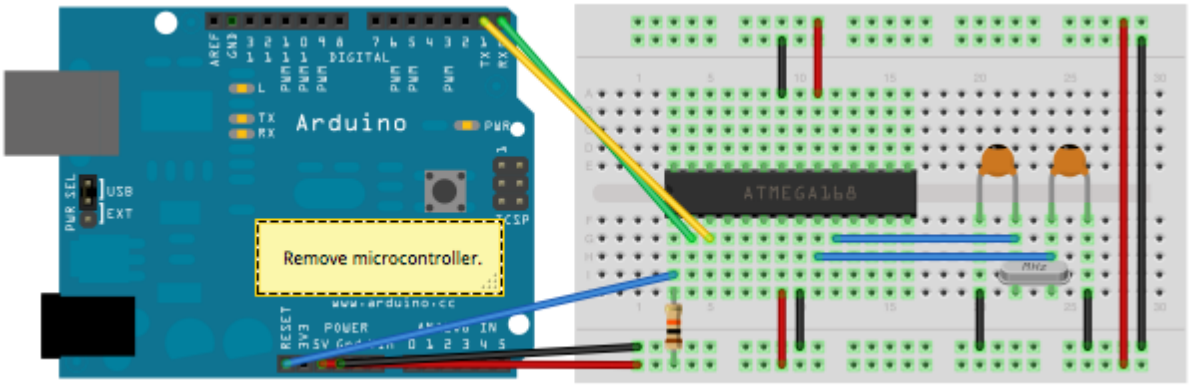
Rysunek 19. Zrzut ekranu fragmentu kodu mikrokontrolera pomiarowego, dotyczącego odczytu z termopar, część II

7.3. Inne pomiary

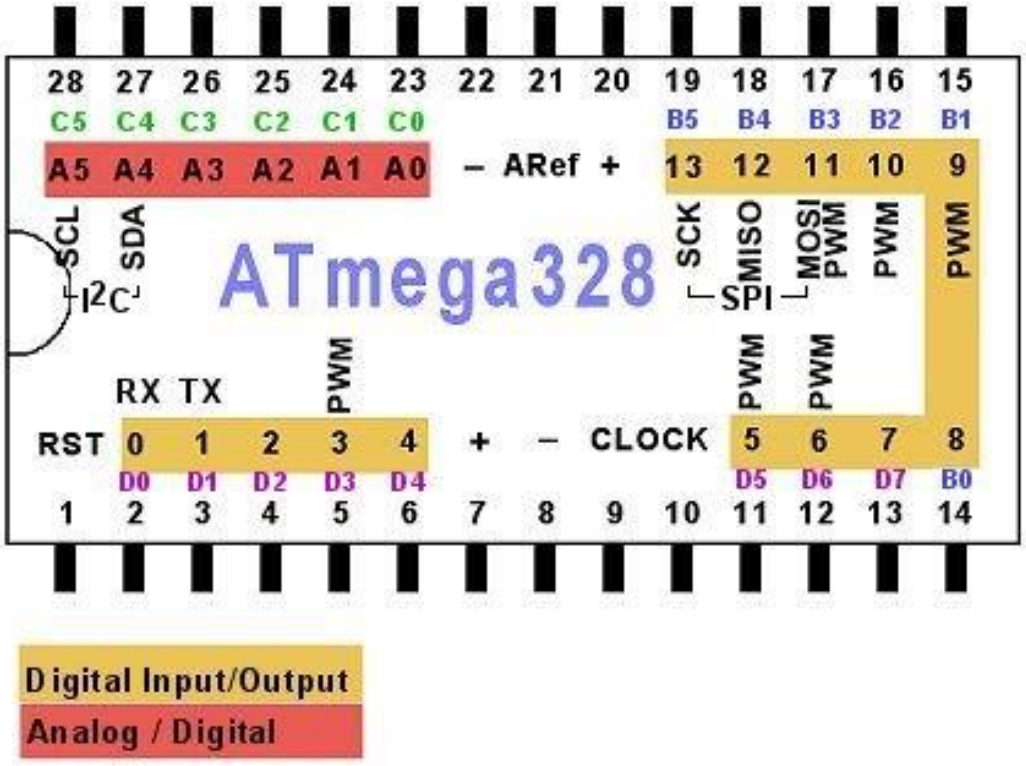
W przyszłości może zaistnieć potrzeba dodania nowych pomiarów, których czas konwersji przekracza kilka milisekund. W takiej sytuacji można zmniejszyć częstotliwość wysyłania paczek danych do komputera przez mikrokontroler Arduino. Jest to jednak nie wskazane z przyczyn omówionych we wcześniejszych etapach pracy. W takim przypadku jednym z możliwych rozwiązań jest przeniesienie nowych pomiarów, lub obecnych do nowej płytki Arduino, czy też do samego mikrokontrolera ATmega328, który jest mózgiem Arduino

UNO. Druga opcja jest tańszym i bardziej elastycznym pod względem ergonomii rozwiązaniem. Co więcej programowanie mikrokontrolera ATmega328 w sposób pokazany na Rysunek 20. Schemat do zaprogramowania mikrokontrolera ATmega328 (źródło: [4]) jest identyczny jak dowolnej płytki Arduino.

Z obecnego punktu widzenia najsensowniejszym krokiem modernizacji było by przeniesienie realizacji pomiarów częstotliwości na nowy mikrokontroler. Byłby on podrzędny w stosunku do obecnego mikrokontrolera pomiarowego. Z tego powodu na Rysunek 20 przedstawiony został schemat podłączenia mikrokontrolera, w celu jego programowania w standardowy sposób. Warunkiem jest posiadanie przez mikrokontroler



Rysunek 20. Schemat do zaprogramowania mikrokontrolera ATmega328 (źródło: [4])



Rysunek 21. Mapa złącz mikrokontrolera ATmega328 (źródło: www.hobbytronics.co.uk)

wgranego bootloadera. Więcej informacji na ten temat znajduje się na stronie producenta [4]. Warto wspomnieć, że w sieci znajduje się wiele poradników w języku polskim jak to dokładnie zrobić, oraz jak w razie potrzeby wgrać bootloader. Na Rysunek 21 została przedstawiona mapa złącz mikrokontrolera ATmega328. Jak można zauważyć ilość złącz cyfrowych i analogowych jest taka sama co w Arduino UNO.

Wracając do zagadnień technicznych w przypadku dostawienia jakiegokolwiek mikrokontrolera, należy zwolnić jeden z dwóch pozostałych portów szeregowych (pozostały o numerach 2 i 3). Należało by wtedy przepiąć dwa lub jedno złącze, w zależności od sposobu komunikacji między mikrokontrolerami (jedno czy dwustronna). Można to zrobić na dwa sposoby. Odpiąć złącze od poszczególnych modułów termopar (Serial2 – złącza 16 i 17), lub jednego od modułu termopary i od czujników DS18B20 (serial3 – złącza 15 i 14). Należało by następnie odpięte złącze przenieść do pozostałych wolnych złącz o numerach: 2,22-54. Na podanych złączach, z wyjątkiem 2 nie ma skręcanych śrubkami złączek, to znaczy, że należało by przewody wlutować do płytki Arduino.

7.4. Podsumowanie

Dodając każdy nowy pomiar należy pamiętać, aby nazwać go według nomenklatury z 28. Należy także uzupełnić informacje w pliku mapowanie.xml, który zostanie udostępniony dla osób zainteresowanych dalszym rozwojem projektu mikro turbiny gazowej.

Kolejnym skutkiem w momencie dołączenia do programu mikrokontrolera arduino nowego pomiaru jest konieczność uzupełnienia informacji w systemu SCADA. W przypadku braku takiej informacji, system nie będzie działał w ogóle. Więcej informacji na ten temat w [2].

8. AUTOSTART

Dodanie funkcji autostartu wiązało się z ingerencją w mikrokontroler sterujący. Wcześniej była tylko możliwość startu manualnego. Tryb manualny miał jednak swoje wady. Najważniejszymi z nich były: konieczna dobra znajomość turbiny przez użytkownika, oraz brak sprawdzenia stanu zapłonu w komorze spalania.

W rozdziale tym zostanie przedstawiony zarówno prosty schemat działania sekwencji autostartu, jak i kod dla mikrokontrolera Arduino do jego realizacji. Dodatkowo również zostanie przedstawiona zmiana funkcji w kodzie Arduino sterującego „turbine_work()", która odpowiada za pracę turbiny po rozruchu.

Co więcej została usunięta część komunikacji z Arduino pomiarowym, a dokładniej wysyłanie danych z mikrokontrolera sterownika do mikrokontrolera pomiarowego. Zostało to zrealizowane, ponieważ mikrokontroler pomiarowy nie pobiera już żadnych danych z mikrokontrolera sterującego.

8.1. Opis działania wraz z kodem.

Sekwencja autostartu została całkowicie przerobiona. Dodatkowo została napisana sekwencja pracy turbiny po autostarcie, opisana w następnym rozdziale. Kod trybu manualnego został w niezmienionej formie.

Schemat działania jest bardzo podobny do schematu podanego w [1]. Jednakże wszystkie wartości początkowe, wyznaczane wcześniej empirycznie są inne. Takimi wartościami są: poziom startowe mocy silnika rozruchowego, oraz zaworu gazu. Dodatkowo zostały zmienione progi wskazujące na start turbiny. Były to: temperatura w komorze spalania i prędkość obrotowa wału turbosprężarki, która jest podawana w Hz. Temperatura progowa została ustawiona na 400^{O}C. Może się wydawać niska w porównaniu do temperatur mogących występować w komorze spalania, ale empirycznie wyznaczono, że po maksymalnie 3 sekundach od zgaśnięcia płomienia, temperatura ta spada poniżej wyznaczonego progu. Wartość progowa prędkość obrotowej wału turbosprężarki ustawiona została empirycznie na 1000Hz.

Schemat działania jest stosunkowo prosty. Składa się z 7 kroków. Od początku procesu następuje ciągłe sprawdzanie czy nie został naciśnięty przycisk stop. W punktach 4 i 6 w każdym przejściu przez pętle jest sprawdzany stan zapłonu. Jest to element bezpieczeństwa. Oto następujące kroki, wraz z zrzutami ekranu kodów je realizującymi:

1. Sprawdzenie wejść panelu kontrolnego, czy są wyzerowane, czekanie na naciśnięcie przycisku start, na skrzynce sterującej. Następnie Włączenie: pompy, zapłonu i otwarcie zawór zamykającego gaz (Rysunek 22).

```
205 void start_auto()
206 {
207   tryb_auto = true;
208
209   // Sprawdz wyzerowanie wejsc panelu kontrolnego
210   lcd_print(0, "Tryb Auto");
211   lcd_print(1, "Wyzeruj wejscia");
212   while (!check_control_panel_I())
213   {communicate_with_measure();}
214   ////////////////////////////////////////////////////
215
216   //Tryb auto, czekaj na inicjalizacje
217   lcd_print(0, "Tryb Auto");
218   lcd_print(1, "Wcisnij start");
219   while(!digitalRead(SW_START))
220   {communicate_with_measure();}
221   ////////////////////////////////////////////////////
222
223   //Wlacz pompe
224   lcd_update[OIL] = HIGH;
225   lcd_update[REV] = HIGH;
226   lcd_print(0, "Wlaczam pompe   ");
227   lcd_print(1, "p:      n:");
228   time_delay(autostart_delay, false);
229   set_DO(PUMP_OIL, ON);
230   ////////////////////////////////////////////////////
231
232   //Wlacz zaplon
233   if (!digitalRead(SW_STOP)) {
234   lcd_print(0, "Wlaczam zaplon");
235   time_delay(autostart_delay, true);
236   set_DO(SPARK, ON);
237   } else {stop = true;}
238   ////////////////////////////////////////////////////
239
240   //Wlacz zawor odc
241   if (!stop) {
242     if (!digitalRead(SW_STOP)) {
243       lcd_print(0, "Wlaczam zawor od");
244       time_delay(autostart_delay, true);
245       set_DO(VALVE_GAS, ON);}
246       else {stop = true;} }
247   ////////////////////////////////////////////////////
```

Rysunek 22. Zrzut ekranu kodu autostartu, krok 1

2. Ustawienie początkowej wartości mocy silnika rozruchowego, która wynosi 30% i odczekanie 4 sekund. Otwarcie zaworu gazu na ok. 5% i co sekundę zwiększanie poziomu jego otwarcia o 5%. Stabilny zapłon występuje przy 20%. Poziom otwarcia zaworu gazu jest ustawiany na 30% (Rysunek 23).

```
//Ustawienia wartosci poczatkowej powietrza, oraz stopniowe zwiekszanie gazu do 30%
if (!stop) {
  lcd_print(0, "air:    gas:");
  lcd_update[GAS] = true;
  lcd_update[AIR] = true;
  if (!digitalRead(SW_STOP)) {
    set_PWM(AIR, air_start_level);
    air_auto_level = air_start_level;
    time_delay(autostart_delay*2, true);
    } else {stop = true;} }
gas_auto_level = 0;
while ((gas_auto_level < gas_start_level) && !stop) {
  gas_auto_level = gas_auto_level + 50;
  if (digitalRead(SW_STOP))
    {stop = true;} else {
    time_delay(autostart_delay/2, true);
    if (!digitalRead(SW_STOP))
      {set_PWM(GAS, gas_auto_level);}
      else {stop = true;} } }
////////////////////////////////////////////////////////////////////////
```

Rysunek 23. Zrzut ekranu kodu autostartu, krok 2

3. Sprawdzenie zapłonu. Jeżeli przez 15 sekund temperatura w danej chwili, w komorze spalania nie przekracza 600^{O}C, lub średnia z dwóch lub trzech pomiarów (możliwość ustawienia) nie przekracza 400^{O}C, włącza się sekwencja zatrzymywania turbiny (Rysunek 24).

```
 // Sprawdzenie zaplonu przez max 15 sekund
unsigned long timer = millis();
while (((historia_termopar[0] < ((exhaust_temp_ignition_treshold*3)/2)) ||
(exhoust_temp < exhaust_temp_ignition_treshold)) && !stop) {
  if (((millis() - timer)>autostart_abort_time) || digitalRead(SW_STOP))
    {stop = true;}
    else {time_delay(1, true);}}
////////////////////////////////////////////////////////////////////////
```

Rysunek 24. Zrzut ekranu kodu autostartu, krok 3

4. Pętla zwiększa moc silnika rozruchowego o 2% co sekundę, aż do 80%. W przypadku niedoboru gazu, co skutkuje skokami ciśnienia w obiegu, zostaje zwiększenie otwarcia zaworu gazu o ok. 6% (Rysunek 25).

```
289  //Petla zwiekszajaca moc powietrza do 80%
290  float before_air_press = air_press;
291  while ((air_auto_level <= 800) && !stop && !digitalRead(SW_STOP) && (exhoust_temp > exhaust_temp_ignition_treshold))
292  {
293    time_delay(autostart_delay/2, true);
294    air_auto_level += air_auto_incr;
295    if (before_air_press - air_press>air_press_drop)  // turbina zaczyna kaszlec. brakuje gazu
296    { gas_auto_level += gas_auto_incr*3; }           // to dodaj gazu
297    set_PWM(AIR, air_auto_level);
298    set_PWM(GAS, gas_auto_level);
299    before_air_press = air_press;
300  }
301  ////////////////////////////////
```

Rysunek 25. Zrzut ekranu kodu autostartu, krok 4

5. Punkt analogiczny do 3, czyli sprawdzenie zapłonu. Jeżeli przez 15 sekund temperatura w danej chwili, w komorze spalania nie przekracza 600^{O}C, lub średnia z dwóch lub trzech pomiarów (możliwość ustawienia) nie przekracza 400^{O}C, włącza się sekwencja zatrzymywania turbiny (Rysunek 26).

```
296  // Sprawdzenie zaplonu 2
297  timer = millis();
298  while (((historia_termopar[0] < ((exhaust_temp_ignition_treshold*3)/2)) ||
299  (exhoust_temp < exhaust_temp_ignition_treshold)) && !stop) {
300    if (((millis() - timer)>autostart_abort_time) || digitalRead(SW_STOP))
301      {stop = true;}
302      else {time_delay(1, true);}}
303  ////////////////////////////////////////////////////////////////////////////
```

Rysunek 26. Zrzut ekranu kodu autostartu, krok 5

6. Otwarcie zaworu gazu zostaje zwiększane co sekundę o 2%, aż do osiągnięcia wymaganych obrotów wału turbosprężarki, czyli 1000Hz. W przypadku otwarcia zaworu gazu wyższego niż 65% i nie osiągnięcia wymaganych obrotów następuje sekwencja zatrzymania turbiny. Po osiągnięciu wymaganych obrotów następuje otworzenie przepustnicy powietrza i wyłączenie zapłonu (Rysunek 27).

```
303  //Petla zwiekszajaca moc turbiny poprzez dodawanie gazu az do osiafnieca wymaganych obrotow
304  while ((crank_rev < crank_rev_start_treshold) && !stop) {
305    if (digitalRead(SW_STOP))
306      {stop = true;} else {
307        time_delay(autostart_delay/2, true);
308        if (crank_rev < crank_rev_start_treshold) {
309          gas_auto_level += gas_auto_incr;
310          if ((gas_auto_level > 650) || (exhoust_temp < exhaust_temp_ignition_treshold) || digitalRead(SW_STOP))
311            {stop = true;}
312            else {set_PWM(GAS, gas_auto_level);} } } }
313  ////////////////////////////////////////////////////////////////////////
314
315  // Po osiagnieciu obrotow
316  if (!digitalRead(SW_STOP) && !stop) {
317    time_delay(autostart_delay/2, true);
318    set_DO(VALVE_AIR, ON);
319    set_PWM(AIR, 0);}
320  else {stop = true;}
321  set_DO(SPARK, OFF);
322  ////////////////////////////////
```

Rysunek 27. Zrzut ekranu kodu autostartu, krok 6

7. Sprawdzenie czy turbina na pewno wystartowała, czyli sprawdzenie obrotów i temperatury w komorze spalania (Rysunek 28).

```
328  // Sprawdź czy turbina wystartowała
329  if ((crank_rev > crank_rev_start_treshold) && (exhoust_temp > exhaust_temp_ignition_treshold) && !stop && !digitalRead(SW_STOP))
330  {started = true; start = false;}
331  ////////////////////////////////
332 }
```

Rysunek 28. Zrzut ekranu kodu autostartu, krok 7

8.2. Tryb pracy turbiny po autostarcie

Gdyby turbina z autostartu przechodziła by do poprzedniej wersji funkcji „turbine_work”, to automatycznie zawór gazu zostałby zamknięty, a turbina zatrzymana. Działo się tak, ponieważ odczyt z potencjometru regulującego otwarcie zaworu gazu wynosił 0. Dlatego został napisany fragment kodu odpowiadający pracy turbiny po autostarcie.

```
void turbine_work()
{
  lcd_update[AIR] = false;
  lcd_update[GAS] = true;
  if (tryb_auto == true)
  {
    lcd_update[GAS] = false;
    lcd_print(0, "Ustaw regulator");
    lcd_print(1, "gazu");

    // Czekaj na ustawienie regulatora gazu
    while ((analogRead(POT_GAS) < gas_auto_level) && !digitalRead(SW_STOP) && !stop
     && (exhoust_temp > exhaust_temp_ignition_treshold) && (crank_rev > (crank_rev_start_treshold/2.))) {
      communicate_with_measure();
      check_oil_press();}
      lcd_print(0, "Praca   gaz:");
      lcd_print(1, "p:      n:");
      lcd_update[GAS] = true;
    ////////////////////////////////////////

    // Praca turbiny, sterowanie moca poziomem otwarcia zaworu gazu
    while (!digitalRead(SW_STOP) && !stop && (exhoust_temp > exhaust_temp_ignition_treshold)
     && (crank_rev > (crank_rev_start_treshold/2.))) {
      set_PWM(GAS, analogRead(POT_GAS));
      communicate_with_measure();
      update_lcd();
      check_oil_press();}
    ////////////////////////////////////////

    stop = true;
    tryb_auto = false;
    started = false;
  }
  else
  {
  lcd_update[AIR] = false;
  lcd_update[GAS] = true;
  lcd_print(0, "Praca   gas:");
  while (!digitalRead(SW_STOP) && !stop)
    {
    set_PWM(GAS, analogRead(POT_GAS));
    communicate_with_measure();
    update_lcd();
    check_oil_press();
    if (!check_start_tresholds()){ stop = true; }
    }
  started = false;
  }
}
```

Rysunek 29. Funkcja turbine_work() w programie mikrokontrolera sterującego

Działanie funkcji, pokazanej na Rysunek 29. Funkcja turbine_work() w programie mikrokontrolera sterującego , można krótko opisać w następujący sposób. W instrukcji warunkowej „if”, w linii 401 jest sprawdzane, czy rozruch wystąpił przez sekwencje autostartu. Jeżeli rozruch był manualny to program przechodzi do linii 430, gdzie instrukcje pochodzą z wersji programu przed modernizacją.

Jeżeli natomiast rozruch nastąpił przez tryb automatyczny to pierwszą rzeczą jest prośba sterownika o ustawienie potencjometru regulującego otwarcie zaworu gazu (linie 407-415). Należy go wtedy powoli przekręcać według wskazówek zegara i obserwować wyświetlacz na skrzynce sterującej. W chwili gdy pozycja na potencjometrze będzie odpowiadała fizycznemu otwarciu zaworu gazu, następuje przełączenie na bezpośrednie sterownie otwarciem zaworu gazu przez regulator (linie 417-424).

W trybie pracy, w zatrzymanie turbiny następuje w przypadku wystąpienia jednego z trzech zdarzeń. Jest to: spadek temperatury w komorze spalania poniżej $400^{O}C$, spadek ilości obrotów poniżej 500Hz, lub naciśnięcie przycisku stop.

9. ZAKOŃCZENIE

Zaprogramowanie i wykonanie automatycznego rozruchu zakończyło się pełnym sukcesem. Przeprowadzono również liczne testy systemów bezpieczeństwa zaimplementowanych w kodzie. Co więcej udało się sukcesywnie wmontować wszystkie nowe czujniki i zaimplementować ich pomiary.

Dzięki wykonanej nomenklaturze pomiarów, możliwe jest użycie systemu SCADA wykonanego przez Krzysztofa Setlaka [2]. Co więcej nomenklatura zawiera informacje na temat aktualnego stanu podłączonych czujników do mikrokontrolera pomiarowego. Dzięki temu łatwiejsze będzie dodawanie nowych czujników.

Uporządkowanie te, jak i kod realizujący działanie mikrokontrolerów zostaną udostępnione dla osób zainteresowanych dalszym rozwojem projektu mikro turbiny gazowej.

Dodatkowo planowana jest realizacja autostartu z poziomu systemu SCADA i/lub z aplikacji telefonu komórkowego. Dodatkowo będzie również możliwe zdalne sterowanie turbiną po autostarcie, co wyeliminuje niewygodę korzystania z regulatora otwarcia zaworu gazu.

Dodatek A: Kod mikrokontrolera pomiarowego

```
#include <OneWire.h>
#include <Wire.h>
#include <SFE_BMP180.h>
#include <SPI.h>
#include "Adafruit_MAX31855.h"
#include "max6675.h"

//Wej□cia analogowe
byte PRESS_1  = 0;  //Gaz przed KS
byte PRESS_2  = 2;  //Olej za WC
byte PRESS_3  = 3;  //Powietrze za sprezarka
byte PRESS_4  = 8;  //Spaliny za TS
byte PRESS_5  = 9;  //Woda w KN
byte FLOW_1   = 6;  //Powietrze przed sprezarka
byte TEMP_1   = 7;  //Powietrze przed sprezarka
byte FLOW_2   = 1;  //Gaz przed KS
byte VOLTAGE_1  = 11; //Elektrycznosc alternator
byte CURRENT_1  = 10; //Elektrycznosc alternator

// Opis czujnik□w analogowych - AS - Analog Sensor
#define number_of_AS 10 // zmiana z 14 na 10
byte AS_pins[number_of_AS]    = { PRESS_1, PRESS_2, PRESS_3, PRESS_4, PRESS_5, FLOW_1, TEMP_1, FLOW_2, VOLTAGE_1, CURRENT_1}; // Usuniete temp2-temp5
unsigned int AS_raw_values[number_of_AS] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}; // usuniete 4
float AS_values[number_of_AS] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}; // usuniete 4
float (*AS_conv_funs[number_of_AS])(float) = {  convert_press_10,
                        convert_press_4,
                        convert_press_4,
                        convert_press_4,
                        convert_press_1,
                        convert_bosh_flow,
                        convert_bosh_temp,
                        convert_omron_flow,
                        convert_voltage_div,
                        convert_current_sens};
String AS_quantities[number_of_AS] = {"1","1","1","1","1","1","2","0","2","4","5"};
String AS_places[number_of_AS]  = {"03","12","02","05","20","01","01","03","07","07"};   // w innych wersjach 20 może być 22

// Wej□cia cyfrowe czunik□ww cz□stotliwo□ci
byte FLOW_3    = 4;  //Olej przed WC
byte FLOW_4   = 5;  //Woda przed KN
byte REV_1    = 7;  //Wal TS
byte REV_2    = 6;  //Wal alternator

// Opis czujnik□w mierz□cych cz□stoliwo□□ FS - frequency sensor
#define number_of_FS 4
byte FS_pins[number_of_FS]        = { FLOW_3, FLOW_4, REV_1, REV_2 };
boolean FS_states[number_of_FS]     = { LOW, LOW, LOW, LOW };
float FS_values[number_of_FS]     = { 0, 0, 0, 0};
float (*FS_conv_funs[number_of_FS])(float) = {  convert_adafruit_flow,
                        convert_adafruit_flow,
                        convert_freq_sens1,
                        convert_freq_sens2};
String FS_quantities[number_of_FS]    = {"2","2","3","3"};
String FS_places[number_of_FS]      = {"12","20","30","31"};

// Czujniki cyfrowe, protoko□owe
SFE_BMP180 BMP180;
double TEMP0, PRESS0;

// Czujniki temperatury DS18B20
#define number_of_DS18B20  7
byte DS18B20_pin = 14;
OneWire  ds(DS18B20_pin); // inicjalizacja magistrali OneWire obslugujacej czujniki temp ds18b20
byte DS18B20_ROMS[number_of_DS18B20 * 8] = {0x28,   0xFF,   0x47,   0x7,    0x11,   0x14,   0x0,    0x40, // Gaz przed KS
                    0x28, 0xFF, 0xB1, 0x39, 0x10, 0x14, 0x0,  0x8C, // Olej przed WC
                    0x28, 0xFF, 0xC4, 0xC,  0x11, 0x14, 0x0,  0x51, // Olej za WC
                    0x28,   0xFF,   0xB4,   0x11,   0x11,   0x14,   0x0,  0x99, // Powietrze za sprezarka
                    0x28, 0xFF,   0xC0,   0x33,   0x10,   0x14,   0x0,  0x37, // Woda KN
                    0x28, 0xFF,   0x4C,   0x7,    0x11,   0x14,   0x0,    0x30, // Woda zbiornik
                    0x28,   0xFF,   0xC0,   0x36,   0x11,   0x14, 0x0,  0x1D,}; // Spaliny za KN
String DS18B20_places[] = {"03",  "10", "11", "02", "20", "22", "08"}; // 11 nie istnieje, 3 pkt
float DS18B20_temps[number_of_DS18B20];

//Czujniki cyfrowe do termopar MAX6675 (x2) i MAX31855 (do KS)
float KS_temp;
float therm_1;
float therm_2;
float prev_KS_temp = 0;
Adafruit_MAX31855 thermocouple_KS(11,13,12); // W kolejności: CLK, CS, DO
MAX6675 thermocouple_1(8,9,3);              // W kolejności: CLK, CS, DO
MAX6675 thermocouple_2(17,15,16);          // W kolejności: CLK, CS, DO
```

```
82
83 // Transmisja parametr□w do Arduino control - większosc usunieta
84 #define num_of_tx_parameters 5
85 boolean com_started = false;
86 unsigned int tx_packets = 0;
87 unsigned int rx_packets = 0;
88 boolean started = false;
89
90 // Lancuchy znakow do transmisji
91 String vals_str;
92 String serial_communication;
93
94
95
96 void setup()
97 {
98   Serial.begin(115200);
99   Serial1.begin(115200);
100  BMP180.begin();
101 }
102
103 // Timing
104 unsigned int FS_timeout;        // czas na odpytywanie czujnikow czestotliwosci w ms - minimalna mierzona cz□stotliwosc to 1/FS_timeout. wyliczany dynamicznie
105 unsigned int conversion_time = 750;
106 unsigned int cycle_time = 1000; // 1s na cykl
107 unsigned long cycle_start;
108 unsigned long loop_counter = 0;
109
110 void loop()
111 {
112   cycle_start = millis();
113   start_DS18B20_conversion();
114   read_BMP180();
115   read_AS();
116   convert_AS();
117   FS_timeout = conversion_time - (millis() -  cycle_start);
118   read_FS_freqs();
119   read_DS18B20();
120   thermocouples();
121   communicate_with_control();
122   vals_str += ("\n");
123   do
124   {
125   Serial.print(vals_str); vals_str = "";
126   }
127   while ((millis()-cycle_start)<(cycle_time-1));
128   loop_counter++;
129 }
130
131 void start_DS18B20_conversion()
132 {
133   for (byte DS18B20_num = 0; DS18B20_num < number_of_DS18B20; DS18B20_num++)
134   { byte DS18B20_rom[8];
135     select_DS18B20(DS18B20_num, DS18B20_rom);
136     start_conversion(DS18B20_rom); }
137 }
138
139 void read_AS()
140 {
141   for (byte AS_index = 0; AS_index<number_of_AS; AS_index++)
142   {AS_raw_values[AS_index] = analogRead(AS_pins[AS_index]);}
143 }
144
145 void convert_AS()
146 {
147   for (byte AS_index=0; AS_index<number_of_AS; AS_index++)
148   {
149     float voltage = analog_to_voltage(AS_raw_values[AS_index]);
150     AS_values[AS_index] = AS_conv_funs[AS_index](voltage);
151       vals_str+=(AS_places[AS_index]);
152       vals_str+=(AS_quantities[AS_index]);
153       vals_str+=(":");
154       vals_str+=(AS_values[AS_index]);
155       vals_str+='\n';
156   }
157 }
158
159 void read_BMP180()
160 {
161   int conversion_time;
162   conversion_time = BMP180.startTemperature();
```

```
  delay(conversion_time);
  BMP180.getTemperature(TEMP0);
  conversion_time = BMP180.startPressure(3);  // argument to poziom dok□□dno□ci odczytu ci□nienia. 0-3
  delay(conversion_time);
  BMP180.getPressure(PRESS0, TEMP0);
    vals_str+=("000:"); //wyrzucone /t między temp i ooeczeniem, otoczeniem, a \t i \t na koncu
    vals_str+=(TEMP0); vals_str+='\n';
    vals_str+=("001:");
    vals_str+=PRESS0; vals_str+='\n';
}

void read_FS_freqs()
{
  unsigned int pulses[number_of_FS] = {0, 0, 0, 0};
  boolean state = LOW;
  boolean prev_state[number_of_FS] = {HIGH, HIGH, HIGH, HIGH};
  unsigned long start = millis();         // odczytujemy czas pocz□tkowy
  unsigned long loop_counter = 0;
  boolean previous_state = HIGH;
  unsigned int my_pulses = 0;
  while (millis() - start <=FS_timeout) // p□tla kt□ra b□dzie dzia□a□a przez czas freq_sensor_poll_time
  {
    for (byte FS_num = 0; FS_num<number_of_FS; FS_num++)
    {
      state = digitalRead(FS_pins[FS_num]);
      if (state == HIGH && prev_state[FS_num] == LOW)
      {pulses[FS_num]++;}
      prev_state[FS_num] = state;
    }
    delayMicroseconds(200);
    loop_counter++;
  }
  for (byte FS_num = 0; FS_num<number_of_FS; FS_num++)
  {
    FS_values[FS_num] = FS_conv_funs[FS_num](pulses[FS_num] / ((float)FS_timeout/1000));
    vals_str+=(FS_places[FS_num]); vals_str+=(FS_quantities[FS_num]); vals_str+=(":"); vals_str+=(FS_values[FS_num]); vals_str+='\n';
  }
}

void read_DS18B20()
{
  for (byte DS18B20_num = 0; DS18B20_num < number_of_DS18B20; DS18B20_num++)
  {
    byte DS18B20_rom[8];
    select_DS18B20(DS18B20_num, DS18B20_rom);
    float temp = read_temperature_sensor(DS18B20_rom);
    if (temp >0 && temp < 200){ DS18B20_temps[DS18B20_num] = temp; }
    vals_str+=(DS18B20_places[DS18B20_num]); vals_str+=("0"); vals_str+=(":"); vals_str+=(DS18B20_temps[DS18B20_num]); vals_str+='\n';
  }
}


// KOMUNIKACJA Z ARDUINO STERUJACYM         <>          KOMUNIKACJA Z ARDUINO STERUJĄCYM         <>          KOMUNIKACJA Z ARDUINO STERUJĄCYM

void communicate_with_control()
{
  establish_communication();
  unsigned long start = millis();
  communicate();
}

void establish_communication()
{
  do
  {Serial1.print('s'); }
  while (Serial1.read() !='s');
}

void communicate()
{
  serial_communication = ("");
  serial_communication += (AS_values[1]); serial_communication += (";");
  serial_communication += (AS_values[2]); serial_communication += (";");
  serial_communication += (FS_values[2]); serial_communication += (";");
  serial_communication += (AS_values[5]); serial_communication += (";");
  serial_communication += (KS_temp);
  Serial1.println(serial_communication);
  Serial1.flush();
  tx_packets++;
}

```

Dodatek B: Kod mikrokontrolera sterującego

```
#include <Arduino.h>
#include <Wire.h>   // standardowa biblioteka Arduino
#include <LiquidCrystal_I2C.h> // dolaczenie pobranej biblioteki I2C dla LCD
enum{
  MANUAL,
  AUTO,
  TEST,
};
#define ON false
#define OFF true
//Debugowanie ogólne
#define in  0 //Sprawdzanie stanu wejść
#define out 1 //Sprawdzanie stanu wyjść
#define com 2 //Podgląd komunikacji
bool debug[] = {LOW, LOW, LOW};

// Piny analogowe
byte POT_GAS = 15;
byte POT_AIR = 14;

// Sterowanie wyj□ciami
byte PWM_AIR = 2;
byte PWM_GAS = 3;
byte RELAY_PUMP_OIL = 4;
byte RELAY_SPARK = 5;
byte RELAY_VALVE_AIR = 6;
byte RELAY_NC = 7;
byte RELAY_PUMP_WATER = 8;
byte RELAY_VALVE_GAS = 9;

// Panel kontrolny
byte SW_MAN_AUTO = 43;
byte DIODE_START = 44;
byte SW_PUMP = 45;
byte SW_SPARK = 46;
byte SW_START = 47;
byte SW_STOP = 48;
byte SW_VALVE_GAS = 49;
byte SW_VALVE_AIR = 50;
byte DIODE_STOP = 51;
byte SW_TACT2 = 52;
byte SW_TACT1 = 53;

//Zmienne opisujące wejścia cyfrowe tj. przelaczniki
#define num_of_DI 9
String DI_names[num_of_DI]  = { "SW_MAN_AUTO", "SW_PUMP", "SW_SPARK", "SW_START", "SW_STOP", "SW_VALVE_GAS", "SW_VALVE_AIR", "SW_TACT2", "SW_TACT1" };
byte DI_pins[num_of_DI]   = { SW_MAN_AUTO, SW_PUMP, SW_SPARK, SW_START, SW_STOP, SW_VALVE_GAS, SW_VALVE_AIR, SW_TACT2, SW_TACT1 };
boolean DI_vals[num_of_DI]  = { LOW, LOW, LOW, LOW, LOW, LOW, LOW, LOW, LOW };

//Zmienne opisujace wejsca analogowe tj. potencjometry
#define num_of_AI 2
String AI_names[num_of_AI]    = {  "POT_AIR", "POT_GAS", };
byte AI_pins[num_of_AI]   = { POT_AIR, POT_GAS };
unsigned int AI_vals[num_of_AI]= { 0, 0 };

//Zmienne opisujace wyjscia cyfrowe tj przekazniki i diody
#define num_of_DO 8
String DO_names[num_of_DO]    = { "RELAY_PUMP_OIL", "RELAY_SPARK", "RELAY_VALVE_AIR", "RELAY_NC", "RELAY_PUMP_WATER", "RELAY_VALVE_GAS", "DIODE_START", "DIODE_STOP" };
byte DO_pins[num_of_DO]   = { RELAY_PUMP_OIL, RELAY_SPARK, RELAY_VALVE_AIR, RELAY_NC, RELAY_PUMP_WATER, RELAY_VALVE_GAS, DIODE_START, DIODE_STOP };
bool DO_vals[num_of_DO] = { OFF, OFF, OFF, OFF, OFF, OFF, OFF, OFF };
enum {PUMP_OIL, SPARK, VALVE_AIR, NC, PUMP_WATER, VALVE_GAS, START, STOP };

//Zmienne opisujace wyjsca PWM czyli sterownik ZR i SR
#define num_of_PWM 2
String PWM_names[num_of_PWM]  = { "PWM_AIR", "PWM_GAS" };
byte PWM_pins[num_of_PWM]   = { PWM_AIR, PWM_GAS, };
int PWM_vals[num_of_PWM]    = { 0, 0 };
byte PWM_adjust[2][2] = {{40, 140},{51, 206}};      // Tablica korygująca sterowanie. Sterowanie PWM zawiera się od wartości pierwszej do drugiej. Tablica 1 dla powietrza 2 dla gazu.
enum { AIR, GAS, };

//Tablica grupujaca wszystkie wejscia
byte control_panel_DI[] = {SW_PUMP, SW_SPARK, SW_VALVE_GAS, SW_VALVE_AIR, SW_TACT2};
byte control_panel_AI[] = {POT_GAS, POT_AIR};

//Zmienne komunikacji z Measure
bool transmit = false;
bool com_started = false;
unsigned int tx_packets = 0;
unsigned int rx_packets = 0;

```

```
//Zmienne otrzymywane od Measure
float oil_press =2.22;
float air_flow;
float air_press;
float crank_rev = 1000;
float exhoust_temp = 30;
#define history_number 3
float historia_termopar[history_number] = {30,30,30};

//Progi na zmienne
float oil_press_treshold    = -1; // Prog ponizej, ktorego turbina sie wylacza
float air_flow_start_treshold = -1; // Prog, ktory wskazuje, ze turbina ruszyla [kg/h]
float air_press_start_treshold  = -1; // Prog, ktory wskazuje, ze turbina ruszyla [bar]
float crank_rev_start_treshold  = 1000; // Prog, ktory wskazuje, ze turbina ruszyla [Hz]
float exhaust_temp_ignition_treshold = 400;// Prog, ktory zatrzymuje silnik rozruchowy podczas wychladzania turbiny [C]

//Definicja parametrów wyswietlacza
#define max_lcd_col 16
#define max_lcd_row 2
LiquidCrystal_I2C lcd(0x27, 2, 1, 0, 4, 5, 6, 7, 3, POSITIVE);  // Ustawienie adresu ukladu na 0x27
bool lcd_update[] = {LOW, LOW, LOW, LOW};
enum { OIL=2, REV=3 };

//Zmienne wyznaczajace stan turbiny
bool start, started;
bool stop = false;

//Zmienne parametryzujace auto start
int autostart_delay = 2000;       //standardowy delay do operacji w trybie auto
int autostart_abort_time = 15000; //jesli po tym czasie nie dojdzie do zaplonu to procedura zatrzymania
int air_start_level = 300;        //poczatkowe ustawienie poziomu SR - wartosc 0-1023
int gas_start_level = 300;        //poczatkowe ustawienie poziomu ZR - wartosc 0-1023
int air_auto_level = 300;
int gas_auto_level = 300;
byte air_auto_incr = 20;          //krok zwiekszania mocy SR
byte gas_auto_incr = 20;          //krok zwiekszania otwarcia ZR
float air_press_drop = 0.2;
boolean tryb_auto = 0;               // 0 i nie dotykac


void setup()
{
  Serial.begin(115200);
  Serial1.begin(115200);
  lcd.backlight();
  lcd.begin(max_lcd_col, max_lcd_row);
  set_pins();
  zero_outputs();
}

void set_pins()
{
  // Piny analogowe
  pinMode(POT_GAS, INPUT);
  pinMode(POT_AIR, INPUT);
  // Sterowanie wyj□ciami
  pinMode(PWM_AIR, OUTPUT);
  pinMode(PWM_GAS, OUTPUT);
  pinMode(RELAY_PUMP_OIL, OUTPUT);
  pinMode(RELAY SPARK, OUTPUT);
  pinMode(RELAY_VALVE_AIR, OUTPUT);
  pinMode(RELAY_NC, OUTPUT);
  pinMode(RELAY_PUMP_WATER, OUTPUT);
  pinMode(RELAY_VALVE_GAS, OUTPUT);
  // Panel kontrolny
  pinMode(SW_MAN_AUTO, INPUT);
  pinMode(DIODE_START, OUTPUT);
  pinMode(SW_PUMP, INPUT);
  pinMode(SW_SPARK, INPUT);
  pinMode(SW_START, INPUT);
  pinMode(SW_STOP, INPUT);
  pinMode(SW_VALVE_GAS, INPUT);
  pinMode(SW_VALVE_AIR, INPUT);
  pinMode(DIODE_STOP, OUTPUT);
  pinMode(SW_TACT2, INPUT);
  pinMode(SW_TACT1, INPUT);
}

void zero_outputs()
{
  for (byte DO_device_index = 0; DO_device_index < num_of_DO-2; DO_device_index++) {
    byte pin = DO_pins[DO_device_index];
    digitalWrite(pin, OFF);}

  for (byte PWM_device_index = 0; PWM_device_index < num_of_PWM ; PWM_device_index++) {
    byte pin = PWM_pins[PWM_device_index];
    analogWrite(pin, 0); }
}

void loop()
{
  boolean mode = mode_select();
  if (mode == AUTO)   { start_auto(); } // Zrealizuj tryb rozruchu auto
  if (mode == MANUAL) { start_manual(); } // Zrealizuj tryb rozruchu manual
  if (started) { turbine_work(); }    // Jeśli turbina wystartowała poprawnie to wejdz w tryb pracy
  turbine_stop();             // Wyłącz turbinę.
}

void time_delay(unsigned int time, bool check_oil = false)
{
```

```
  unsigned long timer = millis();
  while (millis() - timer < time)
  {
    communicate_with_measure();
    update_lcd();
    if (check_oil){ check_oil_press(); }
  }
}

boolean mode_select()
{
  lcd_print(0, "Wybierz tryb");
  lcd_print(1, "Zatwierdz PP");
  boolean mode;
  while(!digitalRead(SW_TACT2))
  {
    mode = digitalRead(SW_MAN_AUTO);
    communicate_with_measure();
    //if (digitalRead(SW_TACT1)){ return TEST;}
  }
  start = true;
  return mode;
}

void start_auto()
{
  tryb_auto = true;

  // Sprawdz wyzerowanie wejsc panelu kontrolnego
  lcd_print(0, "Tryb Auto");
  lcd_print(1, "Wyzeruj wejscia");
  while (!check_control_panel_I())
  {communicate_with_measure();}
  ////////////////////////////////////////////////////////////////////////

  //Tryb auto, czekaj na inicjalizacje
  lcd_print(0, "Tryb Auto");
  lcd_print(1, "Wcisnij start");
  while(!digitalRead(SW_START))
  {communicate_with_measure();}
  ////////////////////////////////////////////////////////////////////////

  //Wlacz pompe
  lcd_update[OIL] = HIGH;
  lcd_update[REV] = HIGH;
  lcd_print(0, "Wlaczam pompe   ");
  lcd_print(1, "p:      n:");
  time_delay(autostart_delay, false);
  set_DO(PUMP_OIL, ON);
  ////////////////////////////////////////////////////////////////////////

  //Wlacz zaplon
  if (!digitalRead(SW_STOP)) {
  lcd_print(0, "Wlaczam zaplon");
  time_delay(autostart_delay, true);
  set_DO(SPARK, ON);
  } else {stop = true;}
  ////////////////////////////////////////////////////////////////////////

  //Wlacz zawor odc
  if (!stop) {
    if (!digitalRead(SW_STOP)) {
      lcd_print(0, "Wlaczam zawor od");
      time_delay(autostart_delay, true);
      set_DO(VALVE_GAS, ON);}
      else {stop = true;} }
  ////////////////////////////////////////////////////////////////////////

  //Ustawienia wartosci poczatkowej powietrza, oraz stopniowe zwiekszanie gazu do 30%
  if (!stop) {
    lcd_print(0, "air:    gas:");
    lcd_update[GAS] = true;
    lcd_update[AIR] = true;
    if (!digitalRead(SW_STOP)) {
      set_PWM(AIR, air_start_level);
      air_auto_level = air_start_level;
      time_delay(autostart_delay*2, true);
      } else {stop = true;} }
  gas_auto_level = 0;
  while ((gas_auto_level < gas_start_level) && !stop) {
    gas_auto_level = gas_auto_level + 50;
    if (digitalRead(SW_STOP))
      {stop = true;} else {
      time_delay(autostart_delay/2, true);
      if (!digitalRead(SW_STOP))
        {set_PWM(GAS, gas_auto_level);}
        else {stop = true;} } }
  ////////////////////////////////////////////////////////////////////////

```

```
 // Sprawdzenie zaplonu przez max 15 sekund
unsigned long timer = millis();
while (((historia_termopar[0] < ((exhaust_temp_ignition_treshold*3)/2)) || (exhoust_temp < exhaust_temp_ignition_treshold)) && !stop) {
  if (((millis() - timer)>autostart_abort_time) || digitalRead(SW_STOP))
    {stop = true;}
    else {time_delay(1, true);}}
////////////////////////////////////////////////////////////////////////

//Petla zwiekszajaca moc powietrza do 65%
float before_air_press = air_press;
while ((air_auto_level < 660) && !stop) {
  if (digitalRead(SW_STOP))
    {stop = true;}
    else {time_delay(autostart_delay/2, true);}
  air_auto_level += air_auto_incr;
  if ((exhoust_temp < exhaust_temp_ignition_treshold) || digitalRead(SW_STOP))
    {stop = true;}
  if (!stop) {
    if ((before_air_press - air_press)>air_press_drop)    // turbina zaczyna kaszlec. brakuje gazu
      {gas_auto_level += gas_auto_incr*3;}                // to dodaj gazu
    set_PWM(AIR, air_auto_level);
    set_PWM(GAS, gas_auto_level);
    before_air_press = air_press;} }
////////////////////////////////////////////////////////////////////////

// Sprawdzenie zaplonu 2
timer = millis();
while (((historia_termopar[0] < ((exhaust_temp_ignition_treshold*3)/2)) || (exhoust_temp < exhaust_temp_ignition_treshold)) && !stop) {
  if (((millis() - timer)>autostart_abort_time) || digitalRead(SW_STOP))
    {stop = true;}
    else {time_delay(1, true);}}
////////////////////////////////////////////////////////////////////////

//Petla zwiekszajaca moc turbiny poprzez dodawanie gazu az do osiafnieca wymaganych obrotow
while ((crank_rev < crank_rev_start_treshold) && !stop) {
  if (digitalRead(SW_STOP))
    {stop = true;} else {
      time_delay(autostart_delay/2, true);
      if (crank_rev < crank_rev_start_treshold) {
        gas_auto_level += gas_auto_incr;
        if ((gas_auto_level > 650) || (exhoust_temp < exhaust_temp_ignition_treshold) || digitalRead(SW_STOP))
          {stop = true;}
          else {set_PWM(GAS, gas_auto_level);} } } }
////////////////////////////////////////////////////////////////////////

// Po osiagnieciu obrotow
if (!digitalRead(SW_STOP) && !stop) {
  time_delay(autostart_delay/2, true);
  set_DO(VALVE_AIR, ON);
  set_PWM(AIR, 0);}
else {stop = true;}
set_DO(SPARK, OFF);
////////////////////////////////////

// Sprawdź czy turbina wystartowała
if (((crank_rev) > crank_rev_start_treshold) && (exhoust_temp > exhaust_temp_ignition_treshold) && !stop && !digitalRead(SW_STOP))
  {started = true; start = false;}
  else {stop = true;}
////////////////////////////////////
}

void start_manual()
{
  // Sprawdz wyzerowanie wejsc panelu kontrolnego
  lcd_print(0, "Tryb Manual");
  lcd_print(1, "Wyzeruj wejscia");
  while (!check_control_panel_I())
  {
    communicate_with_measure();
  }
  ////////////////////////////////////////////

  // Czekaj na sygnal wlaczenia pompy
  lcd_print(0, "Wlacz pompe     ");
  lcd_print(1, "p:      n:");
  lcd_update[OIL] = HIGH;
  lcd_update[REV] = HIGH;
  while (!digitalRead(SW_PUMP) && !digitalRead(SW_STOP))
  {
    communicate_with_measure();
    update_lcd();
```

```
  }
  set_DO(PUMP_OIL, ON);
  set_DO(PUMP_WATER, ON);
  // Czekaj na sygnal wlaczenia zaplonu
  lcd_print(0, "Wlacz zaplon");
  while (!digitalRead(SW_SPARK) && !digitalRead(SW_STOP))
  {
    communicate_with_measure();
    update_lcd();
    check_oil_press();
  }
  set_DO(SPARK, ON);

  // Czekaj na sygnal wlaczenia zaworu odcinajacego
  lcd_print(0, "Wlacz zawor odc.");
  while (!digitalRead(SW_VALVE_GAS) && !digitalRead(SW_STOP))
  {
    communicate_with_measure();
    update_lcd();
    check_oil_press();
  }
  set_DO(VALVE_GAS, ON);

  // Steruj silnikiem rozruchowym i zaworem gazu
  lcd_print(0, "air:    gas:");
  lcd_update[GAS] = true;
  lcd_update[AIR] = true;
  while (!digitalRead(SW_VALVE_AIR) && !digitalRead(SW_STOP))
  {
    set_PWM(AIR, analogRead(POT_AIR));
    set_PWM(GAS, analogRead(POT_GAS));

    communicate_with_measure();
    update_lcd();
    check_oil_press();
  }
  set_PWM(AIR, 0);
  set_DO(VALVE_AIR, ON);
  set_DO(SPARK, OFF);

  // Sprawdź czy turbina wystartowała
  if (check_start_tresholds()){
    started = true; start = false;
  }
}

void turbine_work()
{
  lcd_update[AIR] = false;
  lcd_update[GAS] = true;
  if (tryb_auto == true)
  {
    lcd_update[GAS] = false;
    lcd_print(0, "Ustaw regulator");
    lcd_print(1, "gazu");

    // Czekaj na ustawienie regulatora gazu
    while ((analogRead(POT_GAS) < gas_auto_level) && !digitalRead(SW_STOP) && !stop
     && (exhoust_temp > exhaust_temp_ignition_treshold) && (crank_rev > (crank_rev_start_treshold/2.))) {
      communicate_with_measure();
      check_oil_press();}
      lcd_print(0, "Praca   gaz:");
      lcd_print(1, "p:      n:");
      lcd_update[GAS] = true;
    ////////////////////////////////////////

    // Praca turbiny, sterowanie moca poziomem otwarcia zaworu gazu
    while (!digitalRead(SW_STOP) && !stop && (exhoust_temp > exhaust_temp_ignition_treshold)
     && (crank_rev > (crank_rev_start_treshold/2.))) {
      set_PWM(GAS, analogRead(POT_GAS));
      communicate_with_measure();
      update_lcd();
      check_oil_press();}
    ////////////////////////////////////////

    stop = true;
    tryb_auto = false;
    started = false;
  }
  else
  {
  lcd_update[AIR] = false;
  lcd_update[GAS] = true;
  lcd_print(0, "Praca   gas:");
  while (!digitalRead(SW_STOP) && !stop)
    {
    set_PWM(GAS, analogRead(POT_GAS));
    communicate_with_measure();
    update_lcd();
    check_oil_press();
    if (!check_start_tresholds()){ stop = true; }
    }
  started = false;
  }
}
```

```
446
447 void turbine_stop()
448 {
449   lcd_print(0, "Stop turbiny");
450   stop = true;
451   set_DO(STOP, HIGH);
452   set_DO(START, LOW);
453   set_DO(VALVE_GAS, OFF);
454   time_delay(1000, true);
455   set_DO(VALVE_AIR, OFF);
456   set_PWM(AIR, 0);
457   set_PWM(GAS, 0);
458   lcd_print(0, "air:        stop");
459   lcd_update[GAS] = false;
460   lcd_update[AIR] = true;
461
462   // Ramp UP
463   int air_ramp_time = 10000;
464   for (int air_reading = 0; air_reading< 1023; air_reading++)
465   {
466     set_PWM(AIR, air_reading);
467
468     unsigned long start = millis();
469     while (millis() - start < air_ramp_time / 1024)
470     {
471       if (air_reading % 100 == 0){
472         communicate_with_measure();
473         update_lcd();
474       }
475     }
476   }
477
478   unsigned long start_cool = millis();
479   while (millis() - start_cool<30000){
480     communicate_with_measure();
481     update_lcd();}
482
483   // Ramp DOWN
484   for (int air_reading = 1023; air_reading > 0; air_reading--)
485   {
486     set_PWM(AIR, air_reading);
487     unsigned long start = millis();
488     while (millis() - start < air_ramp_time / 1024)
489     {
490       if (air_reading % 100 == 0){
491         communicate_with_measure();
492         update_lcd();
493       }
494     }
495   }
496   set_PWM(AIR, 0);
497   set_DO(PUMP_OIL, OFF);
498   set_DO(PUMP_WATER, OFF);
499   lcd_update[AIR] = false;
500   started = false;
501   stop = false;
502   set_DO(STOP, LOW);
503 }
504
505 boolean check_control_panel_I()
506 {
507   byte control_panel_DI_quantity  = 5;
508   byte control_panel_AI_quantity  = 2;
509   byte on_num = 0;
510   int AI_check_treshold = 20;
511   boolean checked = false;
512
513   for (int DI_num = 0; DI_num < control_panel_DI_quantity; DI_num++)
514   {if(digitalRead(control_panel_DI[DI_num])){on_num++;}}
515
516   for (int AI_num = 0; AI_num < control_panel_AI_quantity; AI_num++)
517   {if(analogRead(control_panel_AI[AI_num]) > AI_check_treshold){on_num++;}}
518
519   if (on_num == 0)
520   {checked = true;}
521   return checked;
522 }
523
524 boolean check_start_tresholds()
525 {
526   boolean started = true;
527   if(air_flow < air_flow_start_treshold){started = false;}
528   if(air_press < air_press_start_treshold){started = false;}
529   if(crank_rev < crank_rev_start_treshold){started = false;}
530   return true;
531 }
532
533 void check_oil_press()
534 {
535   if (oil_press < oil_press_treshold) {
536     stop = true;
537     lcd_print(0, "p olej nis stop!;");}
538 }
539
540 void communicate_with_measure()
541 {
542   if (establish_communication()) { communicate(); }
543 }
```

```
544
545 boolean establish_communication()
546 {
547   if (Serial1.read() == 's'){
548     if (debug[com]){Serial.println('s');}
549     Serial1.print('s');
550     return true;
551   } else{ return false; }
552 }
553
554 void communicate()
555 {
556   float termopara;
557   while (Serial1.available()<=15){} // wait for buffer to load
558   oil_press = Serial1.parseFloat();
559   air_press = Serial1.parseFloat();
560   crank_rev = Serial1.parseFloat();
561   air_flow  = Serial1.parseFloat();
562   termopara = Serial1.parseFloat();
563   if (termopara != 0)
564   {
565   historia_termopar[2] = historia_termopar[1];
566   historia_termopar[1] = historia_termopar[0];
567   historia_termopar[0] = termopara;
568   exhoust_temp = ((historia_termopar[1] + historia_termopar[0])/2);
569   }
570   transmit = true;
571   rx_packets++;
572
573   // CZYSZCZENIE RESZTY BUFORA
574   while (Serial1.available() > 0) {
575     char x = Serial1.read();}
576 }
577
578 void set_PWM(byte device, int value)
579 {
580   byte pin = PWM_pins[device];
581   if (value < 6) {
582     analogWrite(pin, 0);
583     PWM_vals[device] = 0;
584   }
585   else
586   {
587     byte PWM_control_level = map(value, 0, 1023, PWM_adjust[device][0], PWM_adjust[device][1]); // z 1 0 sie zrobilo
588     analogWrite(pin, PWM_control_level);
589     PWM_vals[device] = map(value, 0, 1023, 0, 100);
590   }
591 }
592
593 void set_DO(byte device, boolean state)
594 {
595   byte pin = DO_pins[device];
596   digitalWrite(pin, state);
597   DO_vals[device] = state;
598 }
```

SPIS RYSUNKÓW

SPIS TABEL

BIBLIOGRAFIA

[1] J. Pleszyński, *Budowa układu turbiny gazowej z odbiorem mocy elektrycznej wyposażonej w układy automatycznego rozruchu i opomiarowania,* Warszawa, 2015.

[2] K. Setlak, *Visualization of Operational Parameters of a Gas Microturbine*, Warszawa, 2016.

[3] Arduino, [Online]. Available: https://www.arduino.cc/. [Data uzyskania dostępu: styczeń 2016].

[4] E. Tołyż i A. Chełmińska, *Badanie małej turbiny gazowej,* Warszawa, 2015.

[5] S. Monk, Arduino dla początkujących, Helion S.A, 2014.

[6] Plociennik. [Online]. Available: www.plociennik.info/index.php/informatyka/systemy-wbudowane/arduino. [Data uzyskania dostępu: styczeń 2016].