POLITECHNIKA POZNAŃSKA

WYDZIAŁ ELEKTRYCZNY

INSTYTUT AUTOMATYKI, ROBOTYKI I INŻYNIERII

INFORMATYCZNEJ

ZAKŁAD STEROWANIA I ELEKTRONIKI PRZEMYSŁOWEJ

PROJEKT ZALICZENIOWY

MIKROPROCESOROWY SYSTEM STEROWANIA I POMIARU NATĘŻENIA ŚWIATŁA

GRZEGORZ AWTUCH

JĘDRZEJ GRZEBISZ

(AUTOR/AUTORZY)

SYSTEMY MIKROPROCESOROWE

(NAZWA PRZEDMIOTU)

LABORATORIUM

(FORMA ZAJĘĆ {PROJEKT/LABORATORIUM/ĆWICZENIA})

PROWADZĄCY:

MGR INŻ. ADRIAN WÓJCIK

ADRIAN.WOJCIK@PUT.POZNAN.PL

POZNAŃ 27/02/2020

DANE SZCZEGÓŁOWE

Rok studiów: INŻ. III

Rok akademicki: 2019/2020

Termin zajęć: czwartek g. 9:45

Data wykonania: 2020/02/27

Temat projektu: Mikroprocesorowy system sterowania i pomiaru natężenia światła

Prowadzący: Adrian Wójcik

Skład grupy (nazwisko, imię, nr indeksu):

1. Awtuch Grzegorz 135794
2. Grzebisz Jędrzej 135825

OCENA (*WYPEŁNIA PROWADZĄCY*)

Spełnienie wymogów redakcyjnych: ……

……

Wykonanie, udokumentowanie oraz opis wykonanych zadań: ……

……

Zastosowanie prawidłowego warsztatu programistycznego: ……

……

Spis treści

1 SPECYFIKACJA

1.1 ZAŁOŻENIA PROJEKTOWE

Celem projektu było stworzenie mikroprocesorowego systemu sterowania oraz pomiaru natężenia światła. Całość została zrealizowana w oparciu o mikrokontroler STM32F746ZG umieszczony na płytce Nucleo-144. Podczas realizacji projektu dążyliśmy do spełnienia następujących wymogów:

a) Uchyb ustalony na poziomie 1% wartości zakresu regulacji(wybrany przez nas zakres to 1200 lx – 4200 lx)
b) Możliwość zadawania wartości referencyjnej za pomocą komunikacji szeregowej
c) Możliwość podglądu aktualnej wartości mierzonej za pomocą komunikacji szeregowej oraz urządzenia wyjścia
d) Podział kodu na moduły funkcjonalne i dokumentacja kodu zgodna ze standardem generatora dokumentacji Doxygen
e) Protokół komunikacji szeregowej odpornej na błędy

1.2 WYKORZYSTYWANY SPRZĘT ORAZ OPROGRAMOWANIE

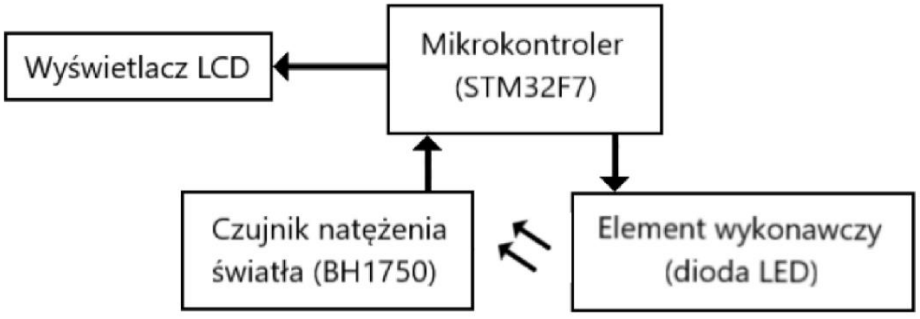
Do realizacji projektu wykorzystywaliśmy następujące elementy oraz urządzenia elektroniczne(rzeczywisty układ widoczny jest na Rysunku 3., natomiast schemat blokowy oraz elektryczny odpowiednio na Rysunku 1. i Rysunku 2.):

a) Zestaw uruchomieniowy Nucleo-144 wraz z mikrokontrolerem STM32F746ZG
b) Czujnik natężenia światła BH1750
c) Wyświetlacz LCD1602 wraz z konwerterem do I2C – PCF8574T
d) Tranzystor MOSFET 2N7000
e) Dioda LED 5mm, niebieska
f) Rezystory THT 10kΩ oraz 1kΩ
g) Płytka stykowa oraz przewody połączeniowe
h) Przewód USB do zasilania oraz komunikacji szeregowej

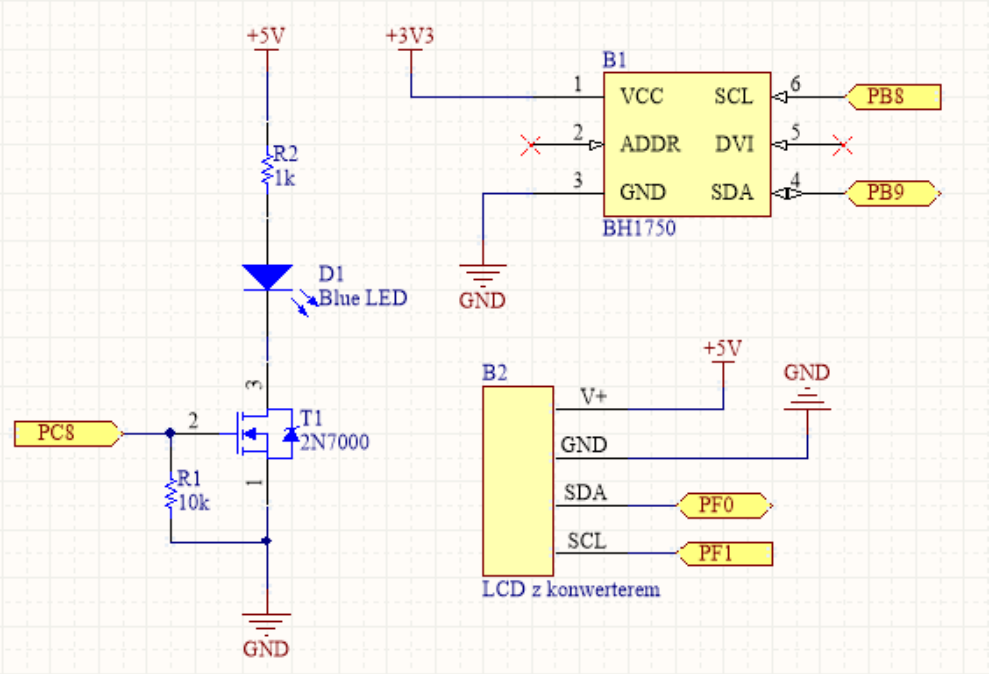
Wykorzystywane oprogramowanie:

a) Zintegrowane środowisko programistyczne STM32CubeIDE
b) Emulator terminala portu szeregowego

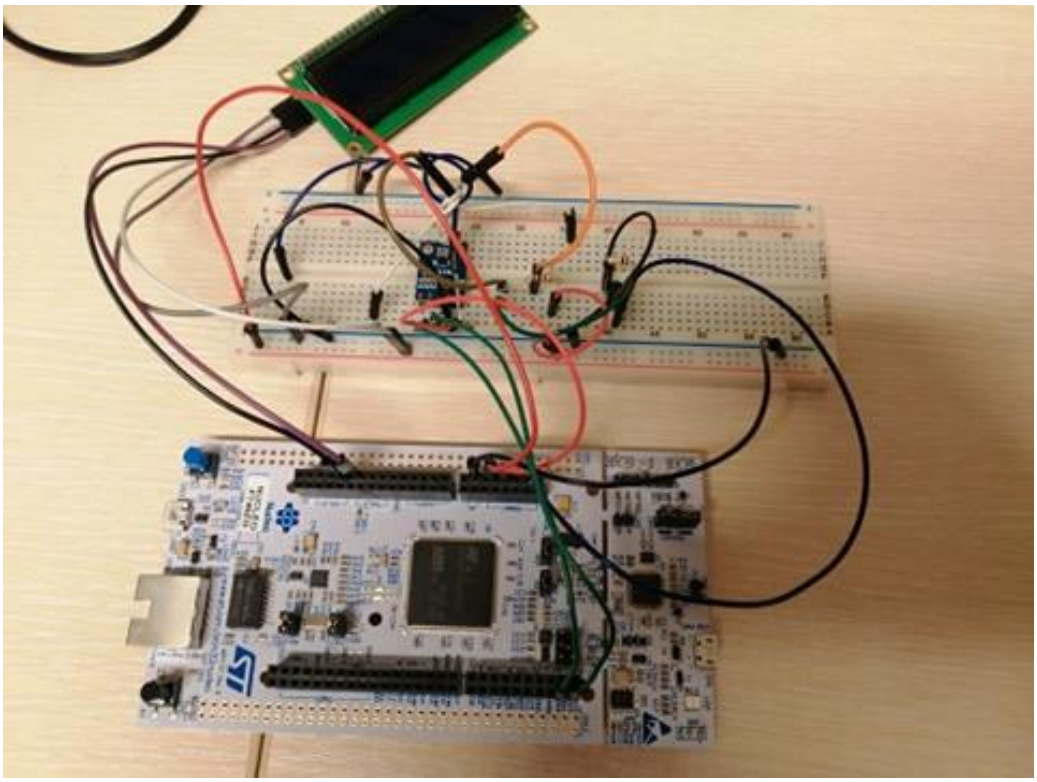
1.3 SCHEMATY POŁĄCZEŃ



Rysunek 1 Schemat blokowy układu



Rysunek 2 Schemat elektryczny układu



Rysunek 3 Rzeczywisty układ

2 IMPLEMENTACJA

W celu większej czytelności raportu punkt opisujący implementację został podzielony na kilka podpunktów, w których zostaną zaprezentowane konkretne funkcje programu wraz z odpowiednimi fragmentami kodów, które obsługują daną funkcjonalność.

2.1 OBSŁUGA CZUJNIKA BH1750

Czujnik BH1750 jest cyfrowym czujnikiem natężenia światła. W celu komunikacji z mikrokontrolerem wykorzystuje on magistralę I2C, którą należy odpowiednio skonfigurować w CubeMX wybierając odpowiednie wyjścia mikrokontrolera jako SCL oraz SDA. W celu łatwiejszego korzystania z czujnika warto stworzyć parę plików *.c oraz *.h, w których zaimplementowana zostanie inicjalizacja czujnika oraz funkcja odpowiedzialna za odbiór danych. Obie te funkcje zostały zaprezentowane na Listingu 1., natomiast stworzone zostały już podczas zajęć laboratoryjnych, na których poznawaliśmy interfejs I2C.

Listing 4 Implementacja funkcji inicjalizacji oraz odczytu danych z czujnika

```
1.  void BH1750_Init()
2.  {
3.          uint8_t power_on=0x01;
4.          uint8_t mode=0x10;
5.
6.          HAL_I2C_Master_Transmit(&hi2c1, 0x23<<1, &power_on, 1, 0xffff);
7.          HAL_I2C_Master_Transmit(&hi2c1, 0x23<<1, &mode, 1, 0xffff);
8.  }
9.
10. float BH1750_Receive(void)
11. {
12.         uint8_t Data[2];
13.         float rslt;
14.
15.         HAL_I2C_Master_Receive(&hi2c1, 0x23<<1, Data, 2, 0xffff);
16.         rslt=((Data[0]<<8)| Data[1])/1.2;
17.
18.         return rslt;
19. }
```

W celu przechowywania w pamięci aktualnej wartości odczytanej przez czujnik stworzona została w kodzie zmienna typu int: *BH_1750_int*, która konwertuje wartość odczytaną z czujnika(typ float) do zmiennej stałoprzecinkowej(int). Zabieg ten został zaimplementowany w pętli while(1), gdyż chcemy na bieżąco odbierać mierzoną przez czujnik wartość. Całą pętlę while można zobaczyć na Listingu 9., natomiast blok utworzonych zmiennych na Listingu 7.

2.2 KOMUNIKACJA PRZEZ UART, TERMINAL SZEREGOWY

W naszym projekcie, w celu komunikacji wykorzystaliśmy zgodnie z założeniem interfejs UART. Pomocny w tej kwestii jest emulator portu szeregowego, który służył nam do podglądu informacji o aktualnym natężeniu światła, wyświetlania informacji o błędnej wartości zadanej oraz samego zadawania wartości.

W celu obsługi wysyłania informacji z mikrokontrolera do terminalu, zaimplementowane zostały w kodzie dwie funkcje, jedna z nich informowała o aktualnie zmierzonym natężeniu światła(wyświetlana w terminalu co 0.5s), natomiast druga wysyłała informację o błędnym formacie wartości zadanej, gdy takowa została wpisana. Funkcje w swojej konstrukcji są praktycznie identyczne, zaprezentowane na Listingu 2. poniżej

Listing 2 Funkcje transmisji komunikatów na terminal

```
1.  void transmisja_danych()
2.  {
3.          transmit_size_lx = sprintf(bufor_lx, "Natezenie swiatla [lux]: %d\n\r", BH1750_int);
4.          HAL_UART_Transmit(&huart3, (uint8_t*)bufor_lx, transmit_size_lx, 100);
5.  }
6.
7.  void transmisja_error()
8.  {
9.          transmit_size_error = sprintf(bufor_error, "Blad wartosci zadanej!!\n\r");
10.         HAL_UART_Transmit(&huart3, (uint8_t*)bufor_error, transmit_size_error, 100);
11. }
```

W linijce 3. oraz 9. zobaczyć można funkcję *sprintf*, która podany tekst(wraz z ewentualnymi zmiennymi) zapisuje do bufora, a sama zwraca wielkość zapisanej tam informacji. Natomiast funkcja *HAL_UART_Transmit* jest odpowiedzialna za wysłanie informacji zawartej w buforze przez interfejs UART. Tak jak wcześniej zostało wspomniane komunikat o natężeniu światła wysyłany jest co pewien stały okres czasu w pętli while(1) – Listing 9. Natomiast komunikat o błędzie w odpowiednim momencie, zaznaczonym w dalszej części raportu. Typy poszczególnych zmiennych utworzonych na potrzeby obsługi tychże funkcji zobaczyć można w bloku na Listingu 7.

Druga część wykorzystywania interfejsu UART to odbiór danych przez mikrokontroler, dane te to zadawana wartość natężenia światła w luxach, którą chcemy uzyskać. Podawanie tej wartości odbywa się przez wcześniej wspominany terminal szeregowy. W przeciwieństwie do transmisji danych, odbiór odbywa się nie poprzez cykliczne odpytywanie, tylko przez zastosowanie przerwania od UART. Fragment kodu odpowiedzialny za działanie odbioru został przedstawiony na Listingu 3. Na wstępie warto wyjaśnić obecność zmiennych *lx_zadana_char* oraz *lx_zadana_int*. Pierwsza z nich przyjmuje dane bezpośrednio z interfejsu szeregowego, jednakże w innych częściach kodu niezbędne było posługiwanie się tą wartością w postaci liczby, a nie tablicy znaków.

Listing 3 Funkcja przerwania od UART, odczyt wartości zadanej

```
1.  void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
2.  {
3.          for (int i=0; i<4; i++)
4.          {
5.                  if(lx_zadana_char[i] > 47 && lx_zadana_char[i] < 58)
6.                  {
7.                          kontrola_znaku = 1;
8.                  }
9.                  else
10.                 {
11.                         kontrola_znaku = 0;
12.                         break;
13.                 }
14.         }
15.         if(kontrola_znaku)
16.         {
17.                 lx_zadana_int = atoi(lx_zadana_char);
18.         }
19.         else
20.         {
21.                 transmisja_error();
22.                 memset(lx_zadana_char, 0, sizeof(lx_zadana_char));
23.         }
24.         port_status = HAL_UART_Receive_IT(&huart3, (uint8_t*)lx_zadana_char, receive_size);
25. }
```

Jak widać w funkcji przerwania została zaimplementowana jednocześnie funkcja obsługi błędu wpisanej wartości zadanej. Analizę kodu warto rozpocząć jednak od ostatniej linijki 24. w której nasłuchujemy informacji o przychodzących danych z UART. Ostatnim argumentem funkcji *HAL_UART_Receive_IT* jest zmienna *receive_size* deklaruje ona ilu znaków zadanej wartości ma spodziewać się UART, w naszym przypadku zmienna ta została zadeklarowana(Listing 7) jako *const int* o wartości 4. W teorii zadane wartości mogą więc wynosić od 0000 do 9999, wartości mniejsze od 1000 należy podawać w formacie z zerami z przodu uzupełniającymi liczbę do 4 znaków(np. 0800 – 800lx). Odczytane w taki sposób wartości zapisywane są do tablicy *lx_zadana_char*. Pozostała część kodu to już sprawdzenie czy w wartości zadanej nie występują żadne niepożądane znaki(np. litery, znaki interpunkcyjne itp.). Sprawdzone zostało to pętlą for(dla każdego znaku tablicy) i jeżeli wykryto choć jeden niepoprawny znak, to nie wykonuje się konwersja tablicy znaków do wartości

liczbowej(funkcja *atoi())* w linijce 17. Natomiast wywołana jest opisana w poprzednim punkcie raportu funkcja *transmisja_error()*, a bufor przechowujący niepoprawną wartość zadaną jest wypełniany zerami. W celu poprawnego działania procedury odbioru należy dodatkowo jednorazowo wywołać funkcję *HAL_UART_Receive_IT* z tymi samymi argumentami w funkcji main po inicjalizacji wszystkich niezbędnych peryferiów. Wywołanie to widoczne jest w bloku na Listingu 8.

2.3 REGULATOR

Kluczową sprawą w projekcie było zaimplementowanie regulatora, który ustala wypełnienie sygnału PWM sterującego jasnością świecenia diody, tak aby natężenie światła(mierzone przez czujnik) wynosiło wartość jak najbliższą wartości zadanej. W przypadku tego projektu zastosowany regulator to regulator czteropołożeniowy z całkowaniem. Funkcja odpowiedzialna za regulację zaprezentowana została na Listingu 4.

Listing 4 Funkcja regulacji wypełnienia PWM,w celu uzyskania wartości zadanej

```
1.  void regulacja()
2.  {
3.          if(uchyb > 70)
4.          {
5.                  if (BH1750_int <= lx_zadana_int && duty <= 1000)
6.                  {
7.                          duty +=10;
8.                          TIM3->CCR3=duty;
9.                  }
10.                 else if (BH1750_int > lx_zadana_int && duty > 0)
11.                 {
12.                         duty -=10;
13.                         TIM3->CCR3=duty;
14.                 }
15.         }
16.
17.         if(uchyb <= 70)
18.         {
19.                 if (BH1750_int <= lx_zadana_int && duty <= 1000)
20.                 {
21.                          duty +=1;
22.                          TIM3->CCR3=duty;
23.                 }
24.                 else if (BH1750_int > lx_zadana_int && duty > 0)
25.                 {
26.                          duty -=1;
27.                          TIM3->CCR3=duty;
28.                 }
29.         }
30. }
```

Odpowiednio wcześniej należy oczywiście skonfigurować timer w CubeMX, jako generator PWM oraz wybrać wyjście do sterowania diodą. Natomiast co się tyczy samego kodu, to początkowo należy wyjaśnić pojawiające się w kodzie regulacji zmienne. Dwie z nich(*lx_zadana_int* oraz *BH1750_int*) pojawiały się już wcześniej. Jednakże nowymi zmiennymi są *uchyb* oraz *duty*. Pierwsza z nich symbolizuje zgodnie z nazwą aktualnych uchyb, czyli z matematycznego punktu widzenia różnicę pomiędzy wartością zadaną, a wartością odczytaną. Uchyb liczony jest w pętli głównej programu(Listing 9) i jest to de facto jego wartość bezwzględna. Zmienna *duty* symbolizuje wypełnienie naszego sygnału PWM(w rzeczywistości jest to wartość countera, timera który obsługuje PWM). Sama regulacja została zrealizowana w taki sposób, że zależnie od różnicy jaka dzieli wartość zadaną i odczytaną, to wypełnienie PWM zwiększa się lub zmniejsza się z odpowiednią szybkością. Zabieg ten ma na celu zmniejszenie oscylacji pojawiających się gdy wartość aktualna jest już bliska wartości zadanej. W celu przypisania odpowiedniej wartości do countera zastosowane zostało polecenie *TIM3→CCR3*. Prawidłowe działanie regulacji wypełnienia wymaga również wystartowania timera w odpowiednim miejscu funkcją *HAL_TIM_PWM_Start()*, można również ustalić w kodzie początkową wartość wypełnienia, obie te funkcje zaprezentowane zostały na Listingu 8.

2.4 OBSŁUGA WYŚWIETLACZA

W ramach projektu zastosowaliśmy wyświetlacz LCD1602 wraz z konwerterem do I2C, w celu uzyskania łatwiejszej komunikacji z mikrokontrolerem. Podobnie jak w przypadku czujnika BH1750 należało w CubeMX skonfigurować piny mikrokontrolera jako SCL oraz SDA. W Internecie można znaleźć wiele przykładowych realizacji obsługi wyświetlacza przez mikrokontroler STM32, w naszym projekcie wykorzystaliśmy jedną z takich gotowych bibliotek, w postaci pary plików i2c-lcd.h oraz i2c-lcd.c(możliwe do podglądu w plikach projektu), zaimplementowane mają one w sobie podstawowe funkcje takie jak wysyłanie na wyświetlacz tekstu, znaku, ustawienie pozycji kursora, czy wyczyszczenie wyświetlacza. W naszym przypadku wyświetlacz ma pokazywać jedynie aktualną wartość mierzoną oraz symbol jednostki „lx”. Jako że miejsce wyświetlania oraz sam napis „lx” jest niezmienny przez cały czas działania układu, to jego wypisanie na wyświetlacz zrealizowane jest tylko raz przed pętlą główną programu. Należy użyć funkcji widocznych na poniższym Listingu 5.

Listing 5 Umieszczenie napisu „lx” na wyświetlaczu w określonej pozycji

```
1. lcd_put_cur(0, 5);
2. lcd_send_string("lx");
```

Wyświetlanie wartości natężenia realizowane jest natomiast w pętli głównej której fragment za to odpowiedzialny został przedstawiony na Listingu 6.

Listing 6 Wysyłanie wartości natężenia na wyświetlaczu

```
1. itoa(BH1750_int, lx_lcd, 10);
2. lcd_put_cur(0, 0);
3. lcd_send_string(lx_lcd);
```

W pierwszej linii kodu widzimy funkcję *itoa*, która konwertuje wartość stałoprzecinkową(zmienną *BH1750_int*) do wartości zapisanej jako tablica znaków(*lx_lcd*) zabieg ten jest konieczny ze względu na to, że na wyświetlacz nie można wysłać bezpośrednio

wartości liczbowej. W kolejnej linii ustawiamy kursor wyświetlacza w pozycji bazowej, a następnie wysyłamy nasz ciąg znaków, przedstawiający aktualną wartość natężenia światła. Dokładne umiejscowienie dwóch powyższych listingów w kodzie programu widoczne jest w blokach kodu na Listingu 8. oraz Listingu 9.

2.5 BLOKI KODU

Poniższy listing kodu zawiera wszystkie zmienne, które zostały utworzone w celu poprawnego działania programu. Do każdej z nich zamieszczony został krótki opis, do czego dana zmienna była potrzebna. Wykorzystywanie ich w praktyce zostało przedstawione we wcześniejszych punktach raportu.

Listing 7 Zmienne programu

```
1.  int BH1750_int;                  //Aktualna wartość natężenia światła
2.  int lx_zadana_int;               //Zadana wartość natężenia światła
3.  int duty;                        //Wypełnienie PWM regulowane w zakresie 0-1000
4.  int uchyb;                       //Uchyb regulacji
5.  int kontrola_znaku;              //Zmienna kontrolująca błąd w zadawanej wartości
6.  int transmit_size_lx;            //Rozmiar wysyłanego komunikatu o natężeniu
7.  int transmit_size_error;         //Rozmiar wysyłanego komunikatu o błędzie
8.  const int receive_size = 4;      //Oczekiwany rozmiar odbieranej wiadomości
9.  char bufor_lx[200];              //Bufor na wysyłany komunikat o natężeniu
10. char bufor_error[200];           //Bufor na wysyłany komunikat o błędzie
11. char lx_zadana_char[4];          //Bufor na odebraną wartość zadaną
12. char lx_lcd[20];                 //Bufor na wartość dla wyświetlacza
13. HAL_StatusTypeDef port_status;   //Status odbioru
```

Kolejny blok kodu, to polecenia wywoływane w funkcji main, ale przed pętlą główną programu. Tak jak w przypadku zmiennych każde polecenie zostało krótko opisane.

Listing 8 Wywołania poleceń przed pętlą główną programu

```
1.  BH1750_Init();                          //inicjalizacja czujnika
2.  lcd_init();                             //inicjalizacja wyświetlacza
3.  HAL_TIM_PWM_Start(&htim3, TIM_CHANNEL_3); //wystartowanie PWM
4.  TIM3->CCR3 = 0;                         //ustalenie startowej wartości wypełnienia
5.  lcd_put_cur(0, 5);        //umieszczenie kursora wyświetlacza w zadanej pozycji
6.  lcd_send_string("lx");                  //wysłanie symbolu jednostki na
    wyświetlacz
7.
8.  //Jednorazowe wywołanie odbioru danych przez UART
9.  port_status = HAL_UART_Receive_IT(&huart3, (uint8_t*)lx_zadana_char, receive_size);
```

Ostatnim przedstawionym wycinkiem kodu są polecenia, które wykonują się cyklicznie w pętli while(1). Występują tutaj wcześniej wspominane funkcje(np. *transmisja_danych(), itoa(), HAL_Delay()).* Tutaj również zostawione zostały krótkie komentarze wyjaśniające cel umieszczenia danego polecenia.

Listing 9 Pętla główna programu

```
1. BH1750_int=(int)BH1750_Receive();      //Odczyt wartości natężenia światła z czujnika
2. transmisja_danych();                   //Transmisja komunikatu w terminalu
3. itoa(BH1750_int, lx_lcd, 10);          //Konwersja odczytanej wartości int do string
4. lcd_put_cur(0, 0);                     //Umieszczenie kursora na wyświetlaczu
5. lcd_send_string(lx_lcd);               //Wysyłanie wartości natężenia na wyświetlacz
6. HAL_Delay(500);                        //Odczekanie 0.5s(częstotliwość wysyłania)
7. uchyb = abs(lx_zadana_int - BH1750_int);     //Obliczenie uchybu
8. regulacja();                                 //Regulacja natężenia światła
```

3 WYNIKI TESTÓW

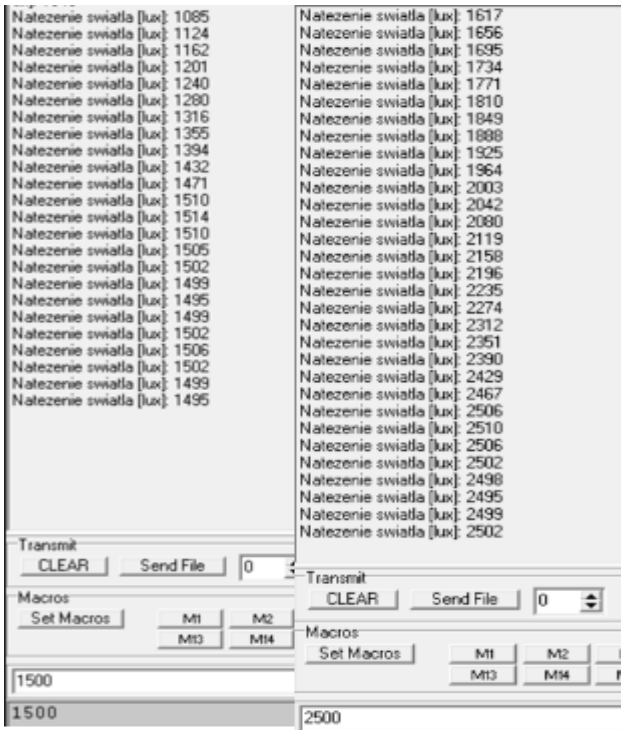
3.1 PREZENTACJA MIERZONYCH WARTOŚCI DLA RÓŻNYCH WARTOŚCI ZADANYCH

Lp.	Wartość zadana	Wartość minimalna	Wartość maksymalna	Błąd bezwzględny	Błąd względny
-	[lx]	[lx]	[lx]	[lx]	[%]
1	1200	1194	1208	8	0,66
2	1500	1493	1507	7	0,46
3	1800	1794	1807	7	0,39
4	2100	2094	2107	7	0,33
5	2400	2393	2408	8	0,33
6	2700	2694	2707	7	0,26
7	3000	2995	3009	9	0,3
8	3300	3292	3310	10	0,21
9	3600	3595	3607	7	0,19
10	3900	3893	3907	7	0,18
11	4200	4193	4208	8	0,19

Tabela 1 Wyniki pomiarów

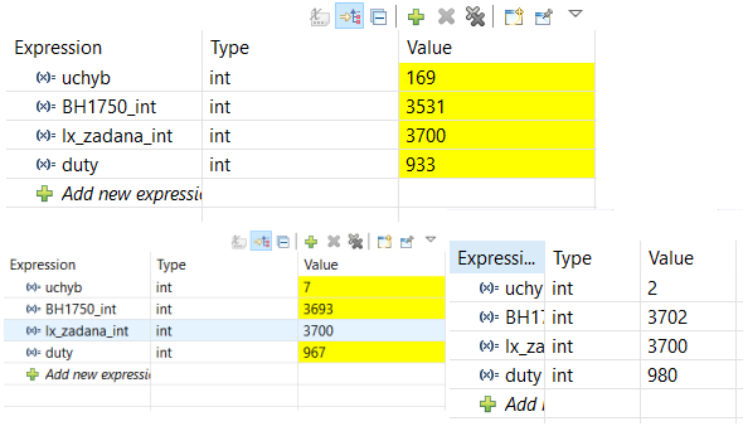
W powyższej tabeli zestawione zostały zmierzone maksymalne oraz minimalne wartości natężenia światła dla 10 różnych wartości zadanych. Ponadto określone zostały błędy bezwzględne oraz względne(w procentach). Jak widać udało się tutaj uzyskać jedno z założeń projektu jakim był uchyb ustalony na poziomie 1%. Wartość zmierzona w żadnym przypadku nie różni się o 30 od wartości zadanej(1% z 3000 => 30).

Ustalanie się wartości natężenia zgodnie z wartością zadaną można również potwierdzić poniższym zrzutem ekranu z terminalu, które prezentują jak zmienia się wartość zmierzona i dąży do zadanej wartości. Jednocześnie mamy potwierdzenie prawidłowego działania transmisji oraz odbioru danych przez interfejs UART. Jak widać natężenie światła w obu przypadkach(wartość zadana 1500 lx oraz 2500 lx) powoli ustala się z niewielkim błędem.



Rysunek 5 Terminal szeregowy, ustalanie wartości

Ostatnim dowodem na prawidłowość działania regulacji w naszym układzie są zaprezentowane na Rysunku 6 aktualne wartości natężenia zmierzonego, zadanego, uchybu oraz wypełnienia sygnału PWM.

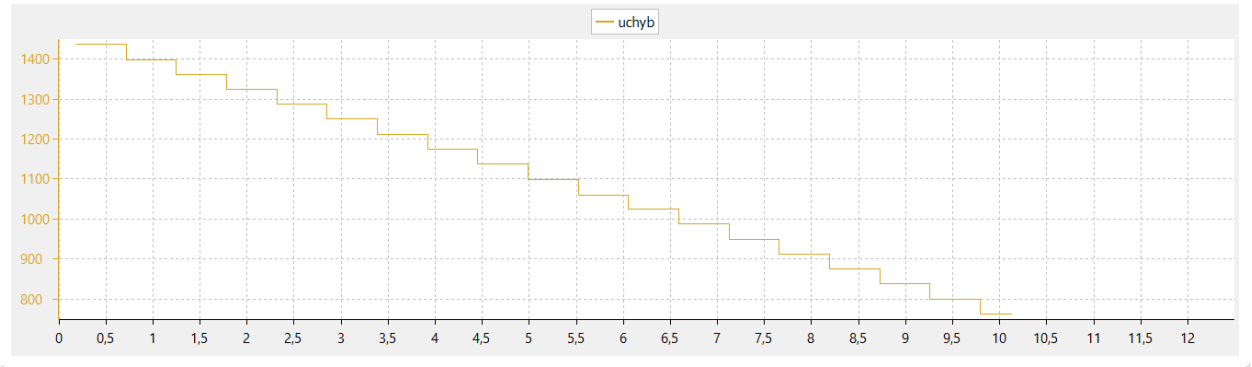


Rysunek 6 Aktualne wartości

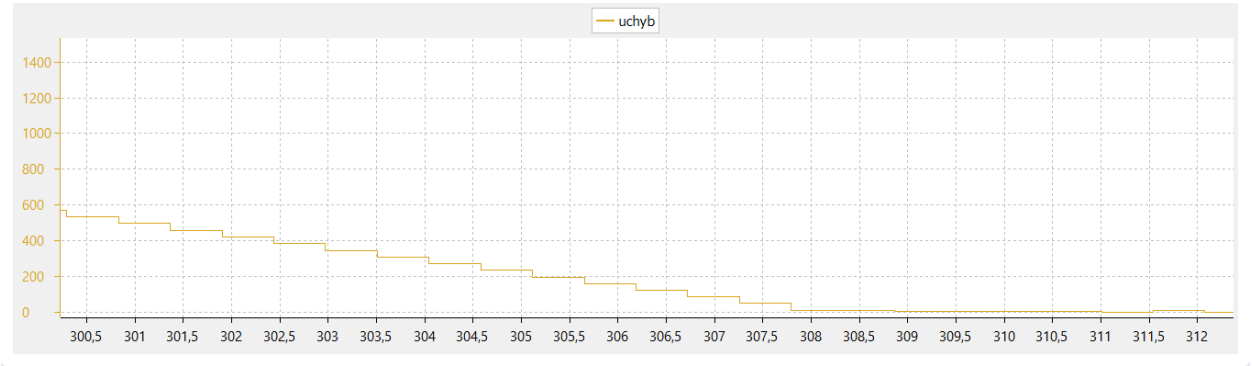
Jak widać zadaną wartością było w tym przypadku 3700 lx, z czasem działania układu uchyb zmierzał to zera, zwiększało się natomiast wypełnienie PWM. W stanie ustalonym przy zmierzonej wartości 3700 lx wypełnienie wynosiło 98%.

3.2 PRZEBIEG CZASOWY – ZMIANA UCHYBU

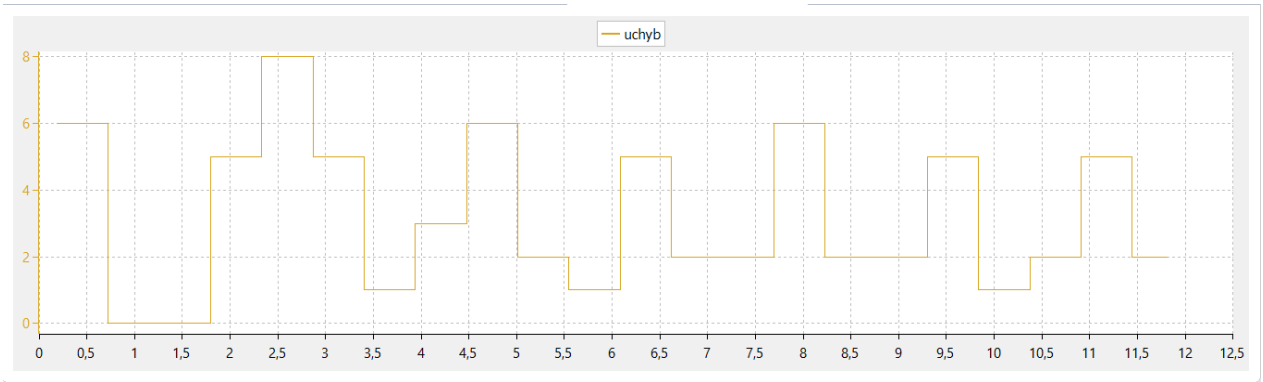
Korzystając z funkcjonalności oprogramowania STMCubeIDE, jakim jest Serial Wire Viewer możliwe było podejrzenie jak w czasie zmienia się dana wartość. Na poniższych rysunkach prezentujemy jak uchyb dochodzi do wartości zbliżonej do zera. Na osi x widać aktualny czas w sekundach, natomiast na osi y uchyb w luxach.



Rysunek 7 Początkowe ustalanie uchybu



Rysunek 8 Uchyb dochodzi od zerowej wartości

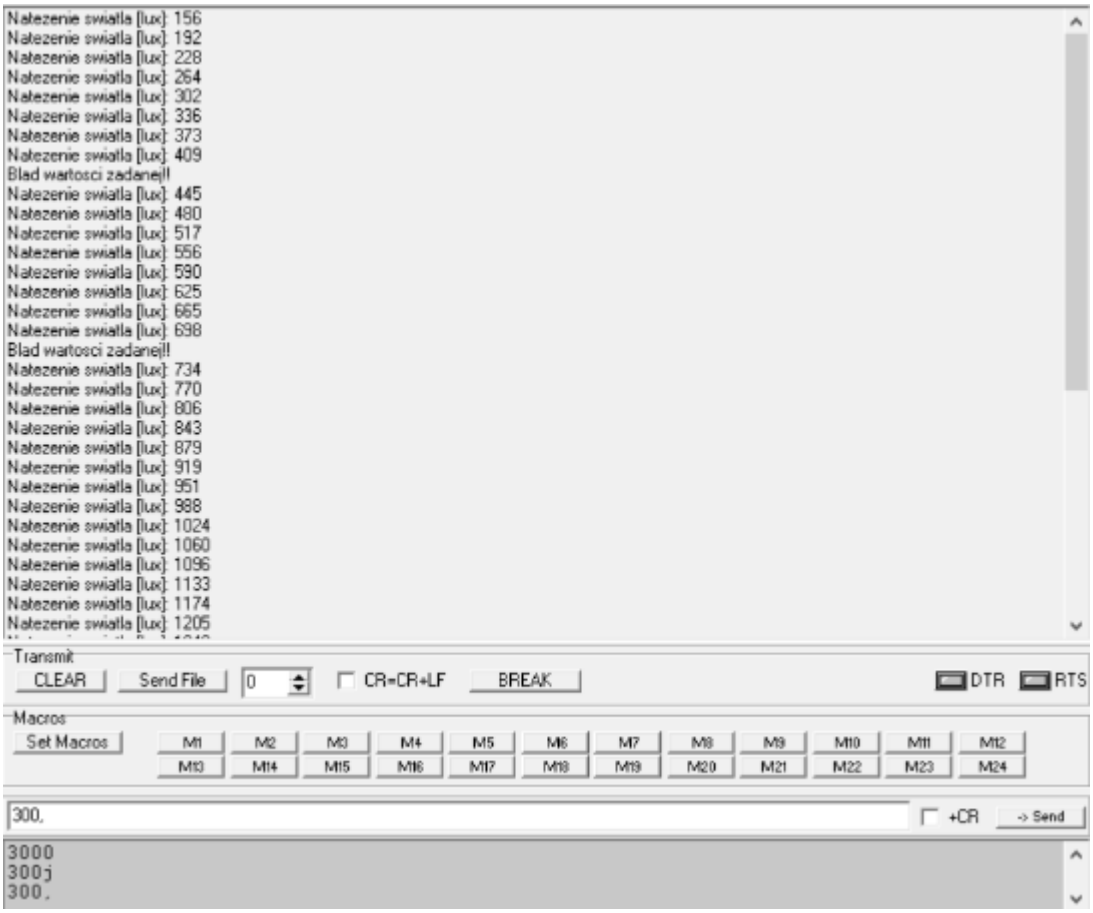


Rysunek 9 Oscylacje w granicach zera

Jak można zauważyć w czasie jednej sekundy uchyb zmniejsza się o wartość nieco mniejszą niż 50 lx. Regulator nie należy do najszybszych, jednak nie było to celem całego projektu. Z czasem jednak uchyb spada do wartości bliskiej 0 i oscyluje wokół niej, co widać na Rysunku 9. Tak duże wartości czasu na osi x w przypadku rysunku 8. spowodowane są faktem, że program działał dłuższy czas, nie mają nic wspólnego z ewentualnym tak długim czasem ustalania.

3.3 OBSŁUGA BŁĘDÓW

W punkcie 2.2 raportu zaprezentowana została implementacja obsługi błędnego wprowadzenia wartości zadanej. W momencie wykrycia takowej sytuacji w terminalu powinien zostać wysłany stosowny komunikat. Działanie tej funkcjonalności prezentuje Rysunek 10.

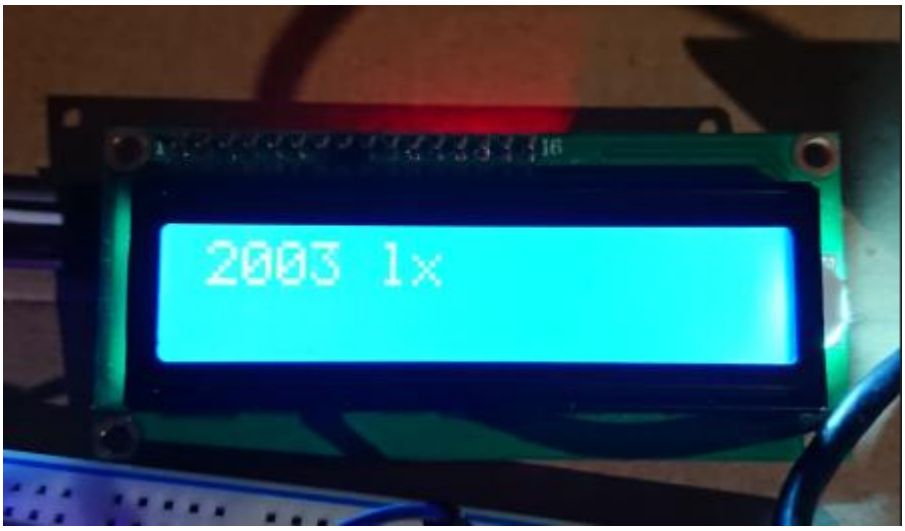


Rysunek 10 Obsługa błędów

W terminalu została zadana wartość 3000 lx i jak widać, zgodnie z oczekiwaniami aktualna wartość natężenia światła zwiększa się. Jednakże w trakcie ustalania tej wartości, dwukrotnie zadana została błędna wartość natężenia(zawierająca w sobie znak spoza przedziału 0-9). Zgodnie z oczekiwaniami w terminalu pojawił się stosowny komunikat, jednak w żadnym przypadku nie został przerwany proces zwiększania natężenia do ostatniej prawidłowej wartości zadanej 3000 lx. W ten sposób układ odporny jest na wprowadzanie danych w nieprawidłowym formacie.

3.4 DZIAŁANIE WYŚWIETLACZA

Ciągłość działania wyświetlacza została zaprezentowana wcześniej w trakcie prezentacji projektu. Poniższe zdjęcia ukazują, jak wyświetlacz prezentuje aktualną wartość mierzoną. Jak widać na zdjęciach wyświetlacz w jednej linii prezentuje wartość liczbową oraz jednostkę „lx”, gdyż w taki sposób został zaprogramowany w kodzie.



Rysunek 11 Działanie wyświetlacza 1

Rysunek 12 Działanie wyświetlacza 2

4 PODSUMOWANIE

Projekt mikroprocesorowego systemu sterowania i pomiaru natężenia światła został przez nas pomyślnie zrealizowany. Udało nam się spełnić w większości zapisane w punkcie 1.1 założenia. Bez większych problemów uzyskaliśmy uchyb na poziomie co najwyżej 1%(wielokrotnie udokumentowane w raporcie). Możliwe było zadawanie wartości referencyjnej przy pomocy komunikacji szeregowej(interfejs UART, terminal szeregowy), Terminal służył również do podglądu aktualnie zmierzonej wartości natężenia oraz wysyłania komunikatu o błędnej formie wartości zadanej. Dodatkowym elementem w układzie jest wyświetlacz LCD, na którym również możliwy był odczyt natężenia. Protokół komunikacji jest odporny na podstawowy rodzaj błędu jakim jest podanie niepoprawnej formy natężenia. Całość kodu została podzielona na różne funkcje, obsługujące konkretne zagadnienia projektu(opisane w punkcie 2. Implementacja). W kodzie staraliśmy się używać komentarzy objaśniających działanie konkretnych funkcji(niewidoczne na listingach, możliwe do sprawdzenia w plikach projektu) oraz opisywać rolę tworzonych zmiennych.