Politechnika Poznańska

Wydział Automatyki, Robotyki i Elektrotechniki

Instytut Robotyki i Inteligencji Maszynowej

Zakład Sterowania i Elektroniki Przemysłowej

Automatyczna regulacja oświetlenia

Systemy mikroprocesorowe

Raport laboratoryjny

Konstanty Odważny, 144514
konstanty.odwazny@student.put.poznan.pl

Jędrzej Szczerbal, 144510
jedrzej.szczerbal@student.put.poznan.pl

Paweł Chumski, 144392
pawel.chumski@student.put.poznan.pl

Prowadzący:
mgr inż. Adrian Wójcik
Adrian.Wojcik@put.poznan.pl

02-02-2022

Spis treści

Wstęp

Tematem projektu jest zaprojektowanie oraz zrealizowanie układu sterowania natężeniem oświetlenia. Cały model pracuje w oparciu płytkę rozwojową NUCLEO F767ZI. Zastosowany został regulator I, natomiast obiekt stanowi LED RGB.

Celem projektowym było umożliwienie sterowania diodą poprzez zadanie konkretnej jasności wyrażonej w luxach. Napisany program umożliwia sterowanie wypełnieniem oraz kolorem LED RGB. Poprawnie działający układ powinien osiągać wartość referencyjną w czasie do paru sekund przy 1% uchybie ustalonym.

Komunikacja z układem odbywa się poprzez port szeregowy mikroprocesora. Wykorzystany został w tym celu terminal oraz aplikacja desktopowa napisana w języku C#. Dodatkowo zastosowano wyświetlacz LCD na którym możliwe jest zaprezentowanie poszczególnych parametrów działającego układu.

Zadanie #1

1.1 Specyfikacja

Komponenty wykorzystane w trakcie realizacji projektu:

- STM32 NUCLEO F767ZI,
- czujnik cyfrowy BH1750,
- wyświetlacz LCD 2x16 z konwerterem I2C LCM1602,
- podstawowe elementy elektroniki, pasywne,
- LED RGB pełniący rolę obiektu regulacji,

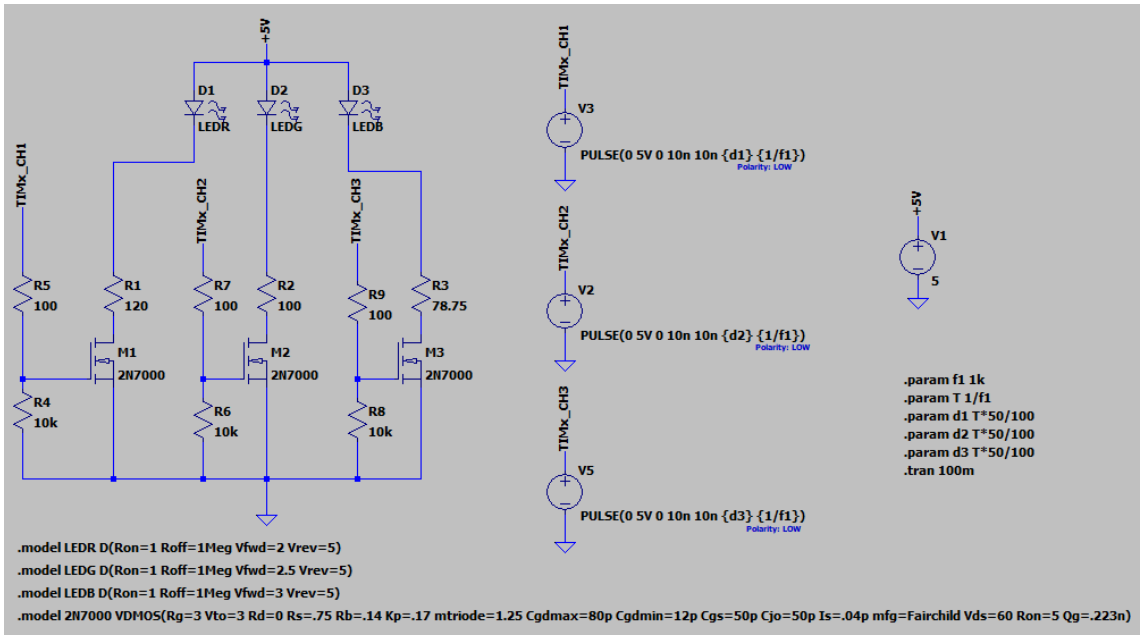
Sterowanie oświetlenia odbywa się za sprawą regulacji PWM sygnału podawanego na tranzystory kluczujące. Każdy z trzech stanowi przełącznik dla kolejnych barw diody Red, Green oraz Blue. Proces regulacji rozpoczyna się po zadaniu przez port szeregowy wartości referencyjnej jasności oraz procentowego wypełnienia każdej z barw. W tym momencie mikroprocesor z pomocą regulatora I dostosowuje sygnały PWM na trzech timerach, by sygnał odczytywany na czujniku cyfrowym dążył do wartości referencyjnej. Sygnały sterujące przeskalowane są natomiast przez procentową wartość podaną przez użytkownika. Tym sposobem możliwe jest ograniczenie maksymalnej mocy podanej na LED.

Dodatkowo sposób komunikacji rozszerzony został o aplikację desktopową napisaną w języku C#. Umożliwia ona automatyczne wysłanie wiadomości stanowiącej o wartościach sygnału oraz pożądanym stanie układu. Reprezentacja odczytywanych wartości została również rozbudowana o wyświetlacz LCD z konwerterem podłączone przez protokół I2C. Do zmiany wyświtlanego parametru oraz wysłania komunikatu służa kolejna USER_btn oraz zewnętrznie podłączony przycisk.

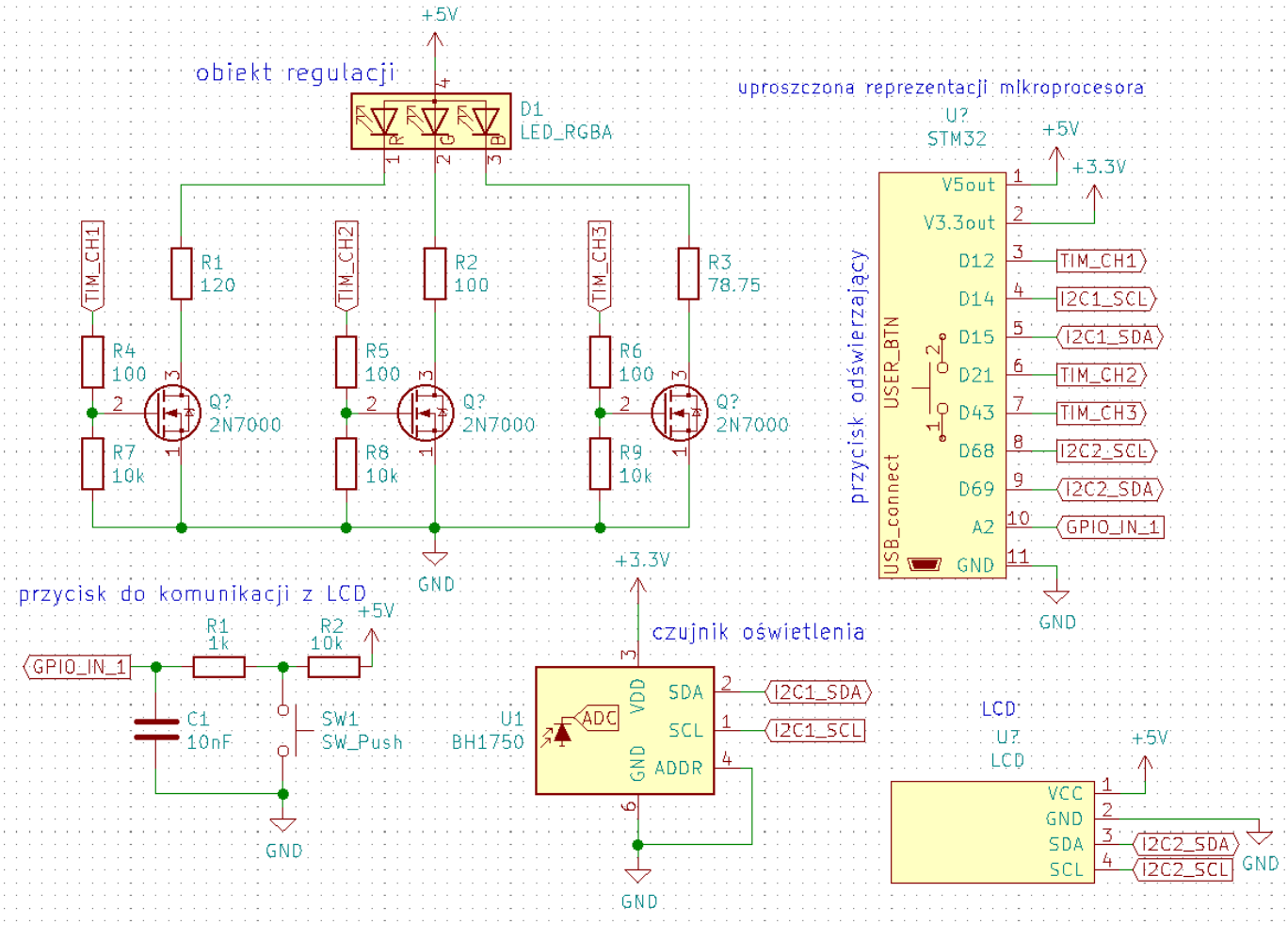
Cały układ zasilany jest przez port mikroUSB stanowiący również połączenie mikrokontrolera z jednostką PC.

1.2 Implementacja

Wstepne symulacje obiektu przeprowadzone zostały w środowisku LTspice [1]. Następnie w programie KiCad [2] zaprojektowany został schemat elektryczny całego układu. W celu uproszczenia zaprezentowania komunikacji miedzy mikrokontrolerem, a elementami wykonywawczymi, poszczególne połączenia pinów zostały zaprezentowane w tabelach.



Rys. 1. Układ symulacyjny obiektu regulacji



Rys. 2. Schemat podłączenia układu elektrycznego

Tab. 1. Połącznie cyfrowego czujnika natężenia światła BH1750 do zestawu NUCLEO-F746ZG za pomocą magistrali I2C [3]

NUCLEO-F746ZG		BH1750	
Pin #	Pin name	Pin #	Pin name
-	3V3	1	VCC
-	GND	2	GND
D15	I2C1 SCL	3	SCL
D14	I2C1 SDA	4	SDA
-	GND	5	ADDR

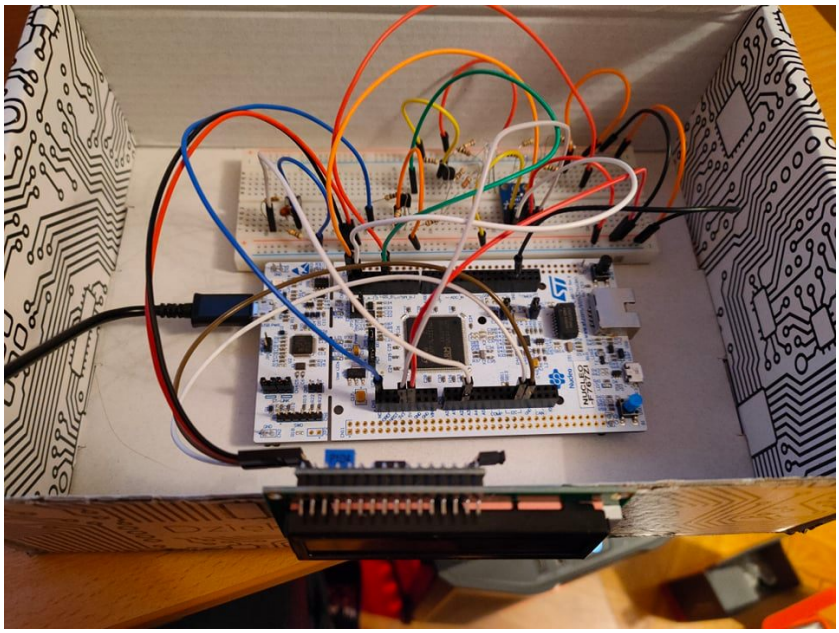
Tab. 2. Połącznie konwertera wyświetlacza LCD 2x16 do zestawu NUCLEO-F746ZG za pomocą magistrali I2C [3]

NUCLEO-F746ZG		LCM1602	
Pin #	Pin name	Pin #	Pin name
-	5V	1	VCC
-	GND	2	GND
D69	I2C2 SCL	3	SCL
D68	I2C2 SDA	4	SDA

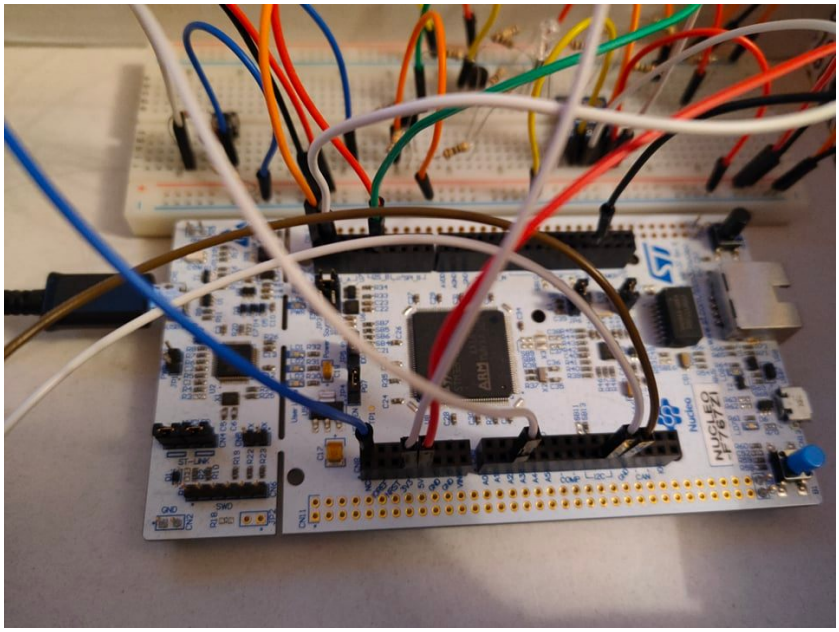
Tab. 3. Połącznie obiektu regulacji do zestawu NUCLEO-F746ZG za pomocą wyjść cyfrowych [3]

NUCLEO-F746ZG		BH1750	
Pin #	Pin name	Pin #	Pin name
D12	PA6	Tim_ch1	R
D21	PC7	Tim_ch2	G
D43	PC8	Tim_ch3	B

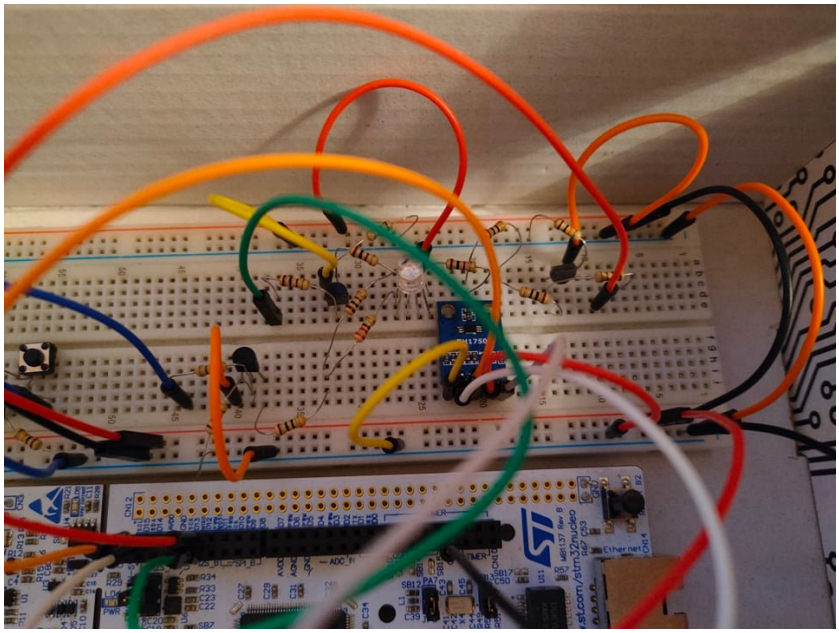
Zdjęcia zbudowanego układu, na którym były przeprowadzane testy:



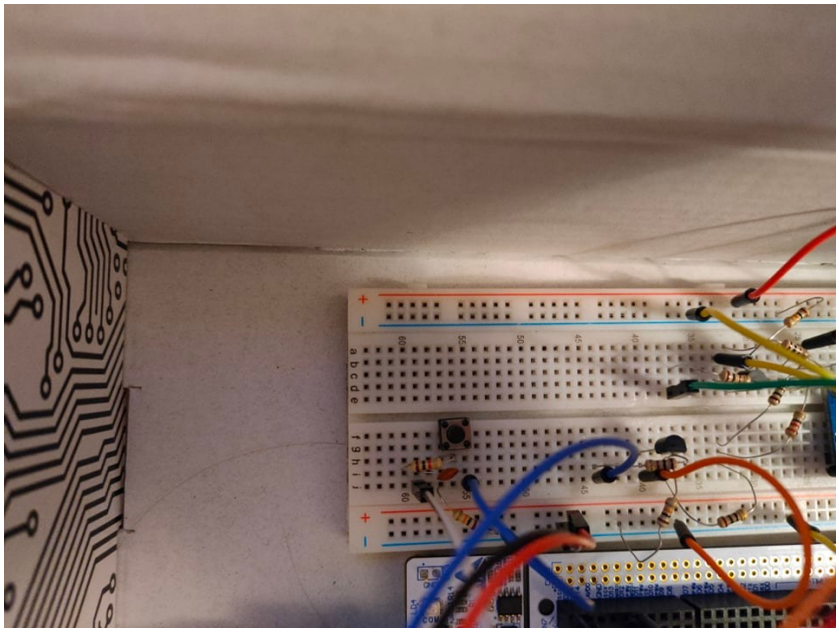
Rys. 3. Zbudowany układ



Rys. 4. Przyłączenia przy mikrokontrolerze STM32



Rys. 5. Podłączony czujnik oraz LED



Rys. 6. Podłączony przycisk z rezystorem podciągającym (pull up)

Kod projektu napisanego w środowisku CubeIDE [4].

Listing 1. Kod regulatora .h

```
/*
 * regulator.h
 *
 *  Created on: Jan 15, 2022
 *      Author: Konstanty
 */

#ifndef INC_REGULATOR_H_
#define INC_REGULATOR_H_


#define reg_Type regulator_Handle_TypeDef*

typedef struct{
         float Ki;
         float Kd;
         float Kp;
         float Ts;
         float e_int;
         float e_der;
         float e_prev;
         float limitdown;
         float limitup;
 }regulator_Handle_TypeDef;


 float Reg_SignalControl(regulator_Handle_TypeDef* Reg,float y_ref, float pomiar)
     ;



#endif /* INC_REGULATOR_H_ */
```

Listing 2. Kod regulatora .c

```c
/*
 * regulator.c
 *
 *  Created on: Jan 15, 2022
 *      Author: Konstanty
 */
#include "regulator.h"

float Reg_SignalControl(regulator_Handle_TypeDef* Reg,float y_ref, float pomiar)
{
        float e;
        float u;
        float u_sat;
        float N=0.01f;
        e = y_ref - pomiar;
                        Reg->e_int += Reg->Ki*Reg->Ts*e;
                        Reg->e_der = (Reg->Kd*N)*(e-Reg->e_prev) + (1.0f - N*Reg
                            ->Ts)*Reg->e_der;
                        Reg->e_prev = e;

                        u = Reg->e_int + Reg->e_der + (Reg->Kp*e);

                        if(u > Reg->limitup)
                        {
                          u_sat = Reg->limitup;
                        }
                        else if(u < Reg->limitdown)
                        {
                          u_sat = Reg->limitdown;
                        }
                        else
                        {
                          u_sat = u;
                        }

                        if(u!=u_sat)
                        {
                          Reg->e_int -=Reg->Ki*Reg->Ts*e;
                        }
        return u_sat;
};
```

Listing 3. Kod cyfrowego czujnika natężenia światła .h

```c
/*
 * bh1750.h
 *
 *  Created on: Nov 13, 2021
 *      Author: konst
 */

#ifndef BH1750_H_
#define BH1750_H_

// Includes
#include "stm32f7xx_hal.h"
#include "i2c.h"

#define BH1750_I2CType I2C_HandleTypeDef*

typedef struct{
```

```
18.         BH1750_I2CType I2C;
19.         uint8_t Address;
20.         uint32_t Timeout;
21. }BH1750_HandleTypeDef;
22.
23. // Definicje
24. #define BH1750_ADDRESS_L (0x23 << 1)
25. #define BH1750_ADDRESS_H (0x5C << 1)
26. #define BH1750_POWER_DOWN 0x00
27. #define BH1750_POWER_ON 0x01
28. #define BH1750_RESET 0x07
29. #define BH1750_CONTINOUS_H_RES_MODE 0X10
30. #define BH1750_CONTINOUS_H_RES_MODE2 0x11
31. #define BH1750_CONTINOUS_L_RES_MODE 0x13
32. #define BH1750_ONE_TIME_H_RES_MODE 0x20
33. #define BH1750_ONE_TIME_H_RES_MODE2 0x21
34. #define BH1750_ONE_TIME_L_RES_MODE 0x23
35.
36. // Funkcje
37. void BH1750_Init(BH1750_HandleTypeDef* hbh1750,uint8_t command);
38. float BH1750_ReadLux(BH1750_HandleTypeDef* hbh1750);
39.
40.
41. #endif /* BH1750_H_ */
```

Listing 4. Kod cyfrowego czujnika natężenia światła .c

```
01. /*
02.  * bh1750.c
03.  *
04.  *  Created on: Nov 13, 2021
05.  *      Author: konst
06.  */
07. #include "bh1750.h"
08.
09. void BH1750_Init(BH1750_HandleTypeDef* hbh1750, uint8_t command){
10. uint8_t start = BH1750_POWER_ON;
11. HAL_I2C_Master_Transmit(hbh1750->I2C, hbh1750->Address, &start, 1, hbh1750->
        Timeout);
12. HAL_I2C_Master_Transmit(hbh1750->I2C, hbh1750->Address, &command, 1,hbh1750->
        Timeout);
13. }
14.
15.
16. float BH1750_ReadLux(BH1750_HandleTypeDef* hbh1750){
17. float light = 0;
18. uint8_t buff[2];
19. HAL_I2C_Master_Receive(hbh1750->I2C, hbh1750->Address, buff, 2, hbh1750->Timeout)
        ;
20. light = ((buff[0] << 8) | buff[1]) / 1.2;
21. return light;
22. }
```

Listing 5. Kod diody LED .h

```
01. /*
02.  * led.h
03.  *
04.  *  Created on: Jan 15, 2022
05.  *      Author: Konstanty
06.  */
```

```
07.
08. #ifndef INC_LED_H_
09. #define INC_LED_H_
10.
11. #define led_Timer LED_HandleTypeDef*
12.
13. typedef struct{
14.         float R;
15.         float G;
16.         float B;
17.         float duty_R;
18.         float duty_G;
19.         float duty_B;
20. }LED_HandleTypeDef;
21.
22.
23. void ColorsGenerator(LED_HandleTypeDef* led, float rgb_duty);
24.
25.
26.
27. #endif /* INC_LED_H_ */
```

Listing 6. Kod diody LED .c

```
01. /*
02.  * led.c
03.  *
04.  *  Created on: Jan 15, 2022
05.  *      Author: Konstanty
06.  */
07. #include "led.h"
08.
09.
10. void ColorsGenerator(LED_HandleTypeDef* led, float rgb_duty){
11.         led->duty_B = rgb_duty*led->B;
12.         led->duty_G = rgb_duty*led->G;
13.         led->duty_R = rgb_duty*led->R;
14. };
```

Listing 7. Kod wyświetlacza LCD .h

```
01. /*
02.  * lcd.h
03.  *
04.  *  Created on: 20 sty 2022
05.  *      Author: Konstanty
06.  */
07.
08. #ifndef INC_LCD_H_
09. #define INC_LCD_H_
10.
11. #include "stm32f7xx_hal.h"
12.
13. void lcd_init(void);   // initialize lcd
14.
15. void lcd_send_cmd(char cmd);  // send command to the lcd
16.
17. void lcd_send_data(char data);  // send data to the lcd
18.
19. void lcd_send_string(char *str);  // send string to the lcd
20.
```

```
21. void lcd_put_cur(int row, int col);  // put cursor at the entered position row (0
        or 1), col (0-15);
22.
23. void lcd_clear(void);
24.
25. #define lcd_type LCD_HandleTypeDef*
26.
27. typedef struct{
28.         char u[20];
29.         char yref[20];
30.         char y[20];
31.         char Red[20];
32.         char Green[20];
33.         char Blue[20];
34.         char message1[20];
35.         char message2[20];
36. }LCD_HandleTypeDef;
37.
38. void send_message_to_lcd(LCD_HandleTypeDef* mlcd,int index);
39.
40.
41.
42.
43. #endif /* INC_LCD_H_ */
```

Listing 8. Kod wyświetlacza LCD .c

```
01. /*
02.  * lcd.c
03.  *
04.  *  Created on: 20 sty 2022
05.  *      Author: Konstanty
06.  */
07.
08. #include "lcd.h"
09. extern I2C_HandleTypeDef hi2c2;  // change your handler here accordingly
10.
11. #define SLAVE_ADDRESS_LCD 0x4E // change this according to ur setup
12.
13. void lcd_send_cmd (char cmd)
14. {
15.   char data_u, data_l;
16.         uint8_t data_t[4];
17.         data_u = (cmd&0xf0);
18.         data_l = ((cmd<<4)&0xf0);
19.         data_t[0] = data_u|0x0C;  //en=1, rs=0
20.         data_t[1] = data_u|0x08;  //en=0, rs=0
21.         data_t[2] = data_l|0x0C;  //en=1, rs=0
22.         data_t[3] = data_l|0x08;  //en=0, rs=0
23.         HAL_I2C_Master_Transmit (&hi2c2, SLAVE_ADDRESS_LCD,(uint8_t *)data_t, 4,
              100);
24.
25. }
26.
27. void lcd_send_data (char data)
28. {
29.         char data_u, data_l;
30.         uint8_t data_t[4];
31.         data_u = (data&0xf0);
32.         data_l = ((data<<4)&0xf0);
33.         data_t[0] = data_u|0x0D;  //en=1, rs=0
34.         data_t[1] = data_u|0x09;  //en=0, rs=0
```

```
35.         data_t[2] = data_l|0x0D;  //en=1, rs=0
36.         data_t[3] = data_l|0x09;  //en=0, rs=0
37.         HAL_I2C_Master_Transmit(&hi2c2, SLAVE_ADDRESS_LCD,(uint8_t *)data_t, 4,
                100);
38.
39. }
40.
41. void lcd_clear (void)
42. {
43.         lcd_send_cmd (0x80);
44.         for (int i=0; i<70; i++)
45.         {
46.                 lcd_send_data (' ');
47.         }
48. }
49.
50. void lcd_put_cur(int row, int col)
51. {
52.     switch (row)
53.     {
54.         case 0:
55.             col |= 0x80;
56.             break;
57.         case 1:
58.             col |= 0xC0;
59.             break;
60.     }
61.
62.     lcd_send_cmd (col);
63. }
64.
65.
66. void lcd_init (void)
67. {
68.         // 4 bit initialisation
69.         HAL_Delay(50);  // wait for >40ms
70.         lcd_send_cmd (0x30);
71.         HAL_Delay(5);  // wait for >4.1ms
72.         lcd_send_cmd (0x30);
73.         HAL_Delay(1);  // wait for >100us
74.         lcd_send_cmd (0x30);
75.         HAL_Delay(10);
76.         lcd_send_cmd (0x20);  // 4bit mode
77.         HAL_Delay(10);
78.
79.   // dislay initialisation
80.         lcd_send_cmd (0x28); // Function set --> DL=0 (4 bit mode), N = 1 (2 line
              display) F = 0 (5x8 characters)
81.         HAL_Delay(1);
82.         lcd_send_cmd (0x08); //Display on/off control --> D=0,C=0, B=0  --->
              display off
83.         HAL_Delay(1);
84.         lcd_send_cmd (0x01);  // clear display
85.         HAL_Delay(1);
86.         HAL_Delay(1);
87.         lcd_send_cmd (0x06); //Entry mode set --> I/D = 1 (increment cursor) & S
              = 0 (no shift)
88.         HAL_Delay(1);
89.         lcd_send_cmd (0x0C); //Display on/off control --> D = 1, C and B = 0. (
              Cursor and blink, last two bits)
90. }
91.
92. void lcd_send_string (char *str)
```

```
93. {
94.         while (*str) lcd_send_data (*str++);
95. }
96.
97.
98. void send_message_to_lcd(LCD_HandleTypeDef* mlcd,int index){
99.
100.
101.         if(index == 0)
102.         {
103.                 lcd_put_cur(0,0);
104.                 lcd_send_string(mlcd->yref);
105.                 lcd_put_cur(1,0);
106.                 lcd_send_string(mlcd->y);
107.         }
108.         else if(index == 1)
109.         {
110.                 lcd_put_cur(0,0);
111.                 lcd_send_string(mlcd->y);
112.                 lcd_put_cur(1,0);
113.                 lcd_send_string(mlcd->u);
114.         }
115.         else if(index == 2)
116.         {
117.                 lcd_put_cur(0,0);
118.                 lcd_send_string(mlcd->u);
119.                 lcd_put_cur(1,0);
120.                 lcd_send_string(mlcd->Red);
121.         }
122.         else if(index == 3)
123.         {
124.                 lcd_put_cur(0,0);
125.                 lcd_send_string(mlcd->Red);
126.                 lcd_put_cur(1,0);
127.                 lcd_send_string(mlcd->Green);
128.         }
129.         else if(index == 4)
130.         {
131.                 lcd_put_cur(0,0);
132.                 lcd_send_string(mlcd->Green);
133.                 lcd_put_cur(1,0);
134.                 lcd_send_string(mlcd->Blue);
135.         }
136.         else if(index == 5)
137.         {
138.                 lcd_put_cur(0,0);
139.                 lcd_send_string(mlcd->Blue);
140.                 lcd_put_cur(1,0);
141.                 lcd_send_string(mlcd->yref);
142.         }
143. }
```

Listing 9. Zmienne globalne w funckji main

```
01. #define START 1
02. #define STOP 0
03.
04. int akcja = 2;
05. int pulse =0;
06.
07. float wartosc_zadana = 130.0f;
08.
```

```
float duty = 0.0f;
BH1750_HandleTypeDef hbh1750_1 = {
.I2C = &hi2c1, .Address = BH1750_ADDRESS_L, .Timeout = 0xffff};
regulator_Handle_TypeDef reg_I = {
   .Ki = 50.0f,.Kd=0.0f,.Kp = 0.0f, .Ts = 0.0007f, .e_int = 0.0f,.e_der = 0.0f,.
      e_prev = 0.0f, .limitdown = 0.0f, .limitup=100.0f};
LED_HandleTypeDef led = {
  .R = 1.0f, .G = 0.0f, .B = 1.0f, .duty_R=0.0f, .duty_G=0.0f, .duty_B=0.0f};

LCD_HandleTypeDef moj_lcd = {
                .yref = "0", .Blue = "0", .Green = "0", .Red = "0", .message1 = "
                    ", .u = "", .y = "",.message2 = ""};
float LightIntensity = -0.1;
int light = -1;
int wartzad = 0;
int u_dutyi = 0;
char komunikat[20];

float led_R;
float led_G;
float led_B;
int pulseR=0,pulseG=0,pulseB=0;

char on[2];
char kolor[12];
char errors[50] = "Zly wzor lub wartosc poza zakresem!";
char yr[4];
char wiadomosc[23];

int lcd_yr = 0;
int lcd_light = -1;
int lcd_index = 0;
_Bool button_state;
```

Listing 10. Komunikacja szeregowa

```
// Odbieranie wiadomosci z aplikacji badz terminala
void HAL_UART_RxCpltCallback(UART_HandleTypeDef* huart)
{
        if(huart->Instance == USART3)
        {

                if(wiadomosc[0] == 'K' && wiadomosc[1] == ':' && wiadomosc[2] ==
                    'R' && wiadomosc[6] == 'G' && wiadomosc[10] == 'B' &&
                    wiadomosc[14] == ',' && wiadomosc[15] == 'Y' && wiadomosc[16]
                    == ':' && wiadomosc[20] == ',' && wiadomosc[21] == '0')
                {

                        if(wiadomosc[22] == 'N')
                        {
                        sscanf(wiadomosc,"K:R%dG%dB%d,Y:%d,ON",&pulseR,&pulseG,&
                            pulseB,&wartzad);
                        akcja = 1;
                        if(pulseR >=0 && pulseR <=100)
                        {
                                led.R = (float)(pulseR/100.0f);

                        }
                        if(pulseG >=0 && pulseG <=100)
                        {
                                led.G = (float)(pulseG/100.0f);

```

```
23.                             }
24.                             if(pulseB >=0 && pulseB <=100)
25.                             {
26.                                     led.B = (float)(pulseB/100.0f);
27.
28.                             }
29.                             wartosc_zadana = (float)(wartzad*1.0f);
30.
31.
32.                             lcd_yr = wartzad;
33.
34.                             }
35.                             else if(wiadomosc[22] == 'F')
36.                             {
37.                                     akcja = 0;
38.
39.                             }
40.                             else
41.                             {
42.                                     HAL_UART_Transmit(huart, (uint8_t*)errors, strlen
                                            (errors), 1000);
43.                             }
44.                     }
45.                     HAL_UART_Receive_IT(&huart3, (uint8_t*)wiadomosc, 23);
46.
47.             }
48. }
```

Listing 11. Program główny

```
01. void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
02. {
03.         if(htim->Instance == TIM2)
04.         {
05.                 // Zczytywanie danych z czujnika
06.                 LightIntensity = BH1750_ReadLux(&hbh1750_1);
07.                 light = LightIntensity*100;
08.                 lcd_light = LightIntensity*100;
09.                 u_dutyi = duty*100;
10.
11.                 if(akcja == START)
12.                 {
13. //                sprintf(komunikat,"%d.%d\r\n",light/100,light%100);
14. //                HAL_UART_Transmit(&huart3, komunikat, strlen(komunikat), 150);
15.
16.                 // Obliczanie sygnalu sterujacego
17.                 duty = Reg_SignalControl(&reg_I, wartosc_zadana, LightIntensity);
18.                 ColorsGenerator(&led, duty);
19.                 __HAL_TIM_SET_COMPARE(&htim3,TIM_CHANNEL_1,(uint32_t)((led.duty_R
                        )*10));
20.                 __HAL_TIM_SET_COMPARE(&htim3,TIM_CHANNEL_2,(uint32_t)((led.duty_G
                        )*10));
21.                 __HAL_TIM_SET_COMPARE(&htim3,TIM_CHANNEL_3,(uint32_t)((led.duty_B
                        )*10));
22.                 }
23.                 else if(akcja == STOP)
24.                 {
25.                         // Zatrzymanie pracy ukladu
26.                         led.duty_R = 0.0f;
27.                         led.duty_G = 0.0f;
28.                         led.duty_B = 0.0f;
29.                         duty = 0.0f;
```

```
30.                         __HAL_TIM_SET_COMPARE(&htim3,TIM_CHANNEL_1,(uint32_t)((
                                led.duty_R)*10));
31.                         __HAL_TIM_SET_COMPARE(&htim3,TIM_CHANNEL_2,(uint32_t)((
                                led.duty_G)*10));
32.                         __HAL_TIM_SET_COMPARE(&htim3,TIM_CHANNEL_3,(uint32_t)((
                                led.duty_B)*10));
33.
34.                 }
35.         }
36.
37. }
```

Listing 12. Przyciski przerywania

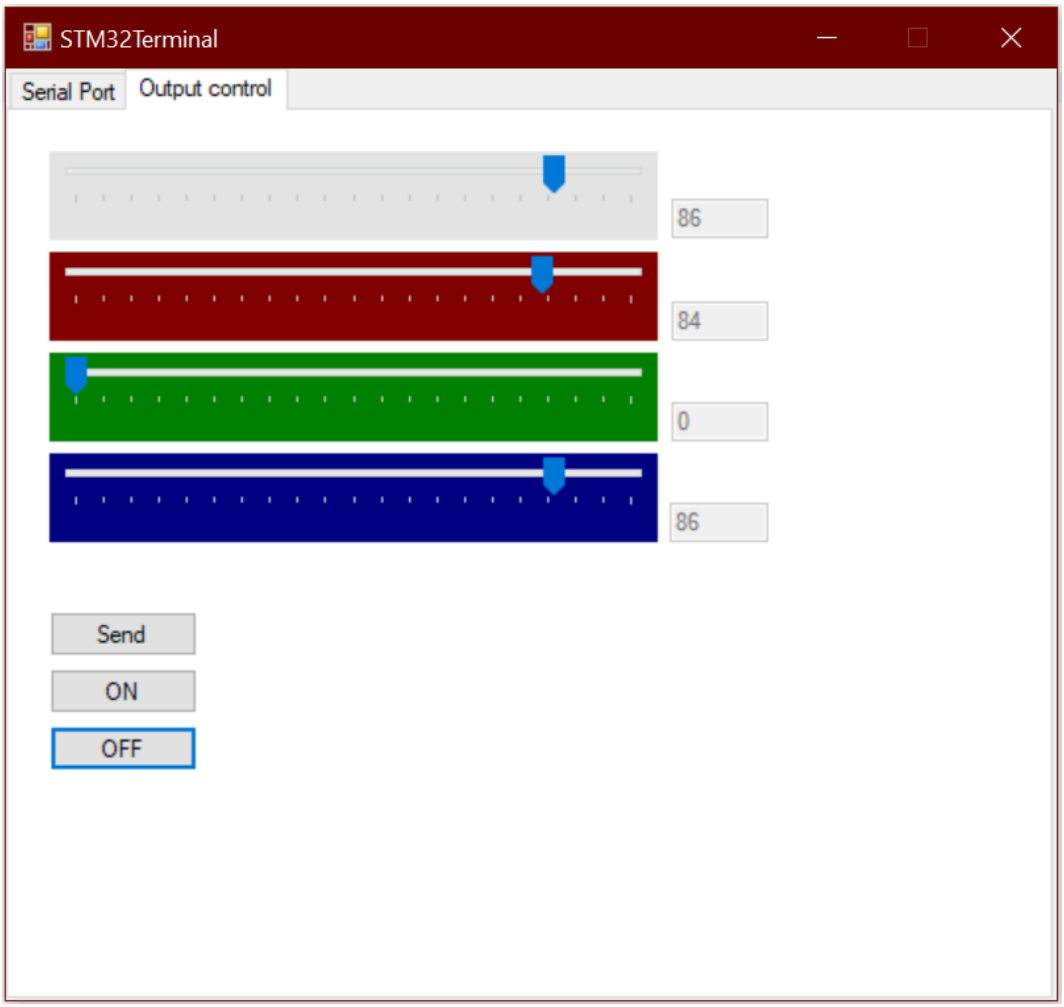
```
01. void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
02. {
03.         // Wysylanie wiadomosci na lcd (aktualnych z listy)
04.   if(GPIO_Pin == USER_Btn_Pin)
05.   {
06.           sprintf(moj_lcd.y,"Y=%d.%d[lux]",lcd_light/100,lcd_light%100);
07.           sprintf(moj_lcd.yref,"Yref=%d[lux]",lcd_yr);
08.           sprintf(moj_lcd.u,"u=%d.%d[lux]",u_dutyi/100,u_dutyi%100);
09.           sprintf(moj_lcd.Red,"RED=%d[%]",pulseR);
10.           sprintf(moj_lcd.Green,"GREEN=%d[%]",pulseG);
11.           sprintf(moj_lcd.Blue,"BLUE=%d[%]",pulseB);
12.           lcd_clear();
13.           send_message_to_lcd(&moj_lcd, lcd_index);
14.           HAL_UART_Receive_IT(&huart3, (uint8_t*)wiadomosc, 23);
15.   }
16.   else if(GPIO_Pin == button_Pin) // Interiwanie po liscie wartosci wysiwetlanych
          na lcd
17.   {
18.           if(lcd_index >=0 && lcd_index < 5)
19.           {
20.                   lcd_index++;
21.           }
22.           else
23.           {
24.                   lcd_index = 0;
25.           }
26.           sprintf(moj_lcd.y,"Y=%d.%d[lux]",lcd_light/100,lcd_light%100);
27.           sprintf(moj_lcd.yref,"Yref=%d[lux]",lcd_yr);
28.           sprintf(moj_lcd.u,"u=%d.%d[lux]",u_dutyi/100,u_dutyi%100);
29.           sprintf(moj_lcd.Red,"RED=%d[%]",pulseR);
30.           sprintf(moj_lcd.Green,"GREEN=%d[%]",pulseG);
31.           sprintf(moj_lcd.Blue,"BLUE=%d[%]",pulseB);
32.           lcd_clear();
33.           send_message_to_lcd(&moj_lcd, lcd_index);
34.           HAL_UART_Receive_IT(&huart3, (uint8_t*)wiadomosc, 23);
35.   }
36. }
```

Listing 13. Inicjalizacja czujnika, timera, lcd i uart

```
01. /* USER CODE BEGIN 2 */
02.
03.
04.    // Inicjalizacja czujnika cyfrowego
05.    uint8_t TrybPracy = BH1750_CONTINOUS_H_RES_MODE;
06.    BH1750_Init(&hbh1750_1, TrybPracy);
07.
```

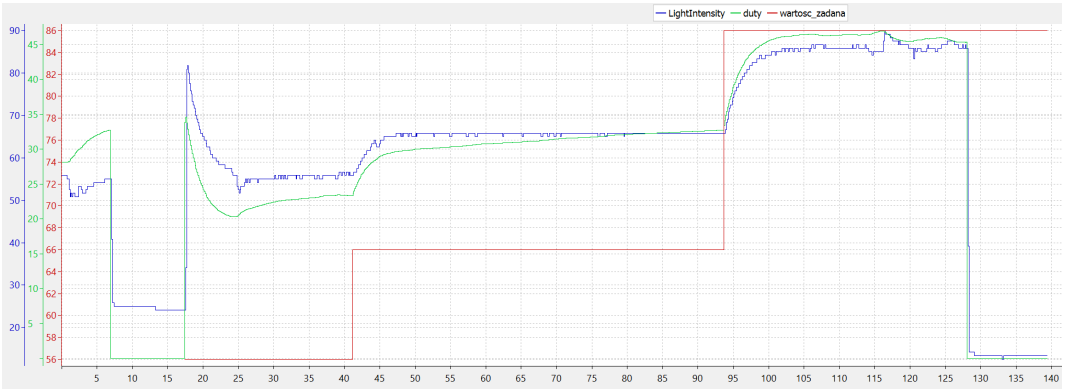
```
08.     // Wystartowanie zegarow
09.     HAL_TIM_PWM_Start(&htim3,TIM_CHANNEL_1);
10.     HAL_TIM_PWM_Start(&htim3,TIM_CHANNEL_2);
11.     HAL_TIM_PWM_Start(&htim3,TIM_CHANNEL_3);
12.     HAL_TIM_Base_Start_IT(&htim2);
13.
14.     // Oczekiwanie na komende UART
15.     HAL_UART_Receive_IT(&huart3, (uint8_t*)wiadomosc, 23);
16.
17.     //Inicjalizacja LCD
18.     lcd_init();
19.     lcd_send_string("Witamy");
20.     lcd_put_cur(1, 0);
21.     lcd_send_string("LED␣RGB");
22.
23.
24.   /* USER CODE END 2 */
```

Aplikacja napisana w C# wykorzystywana do zastąpienia terminala, służąca do połączenia się z STM w celu z automatyzowania przesyłania danych przez użytkownika na płytkę.

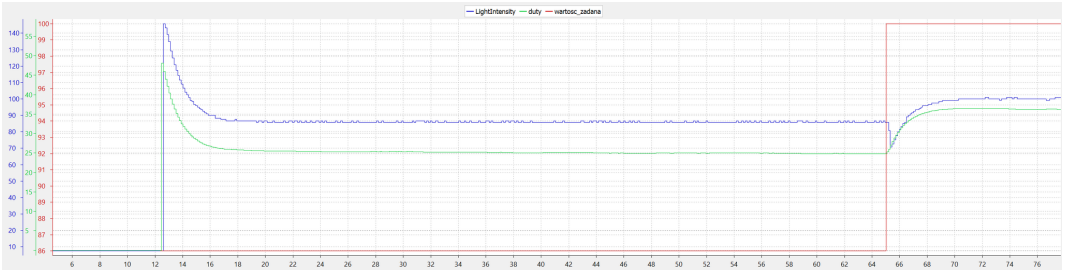


Rys. 7

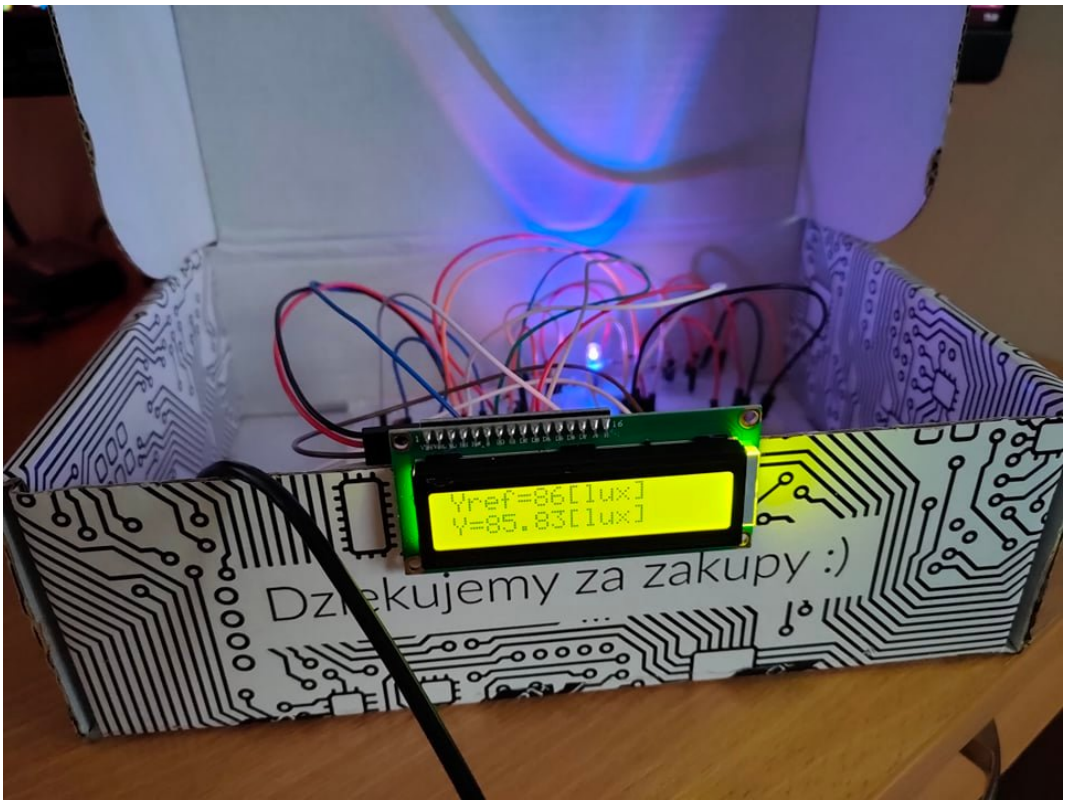
1.3 Wyniki testów



Rys. 8. Przebiegi na odpowiednie wartości zadane

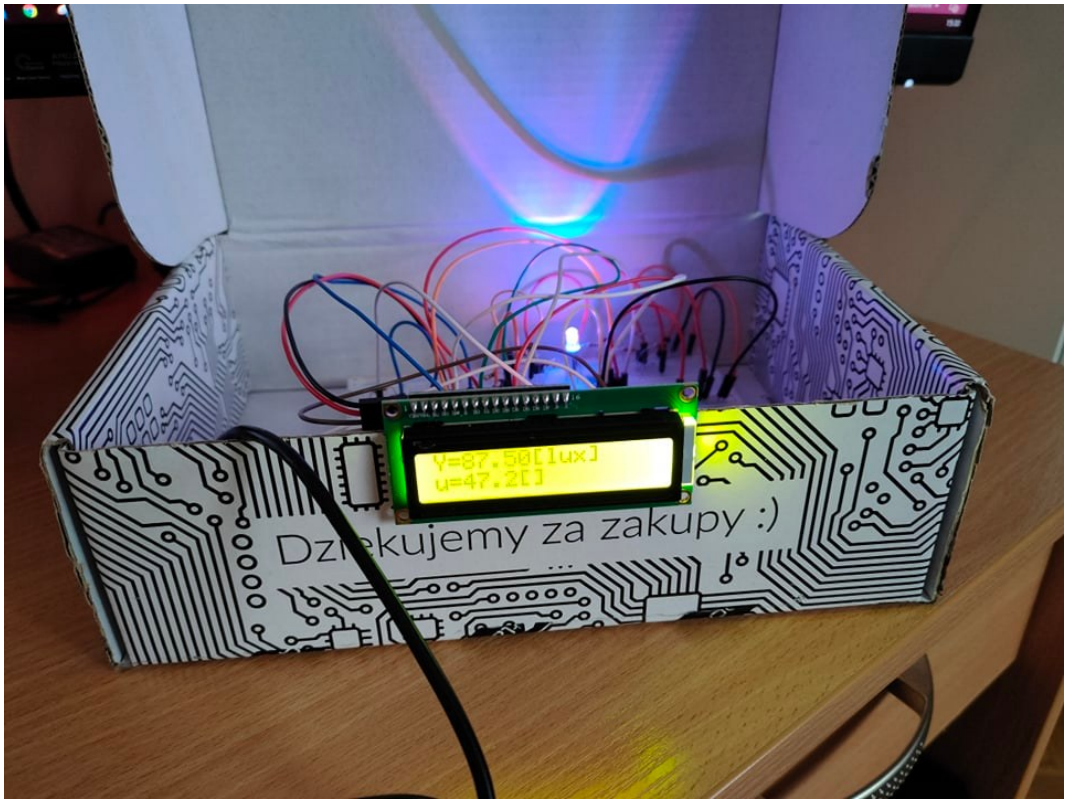


Rys. 9. Przebiegi na odpowiednie wartości zadane

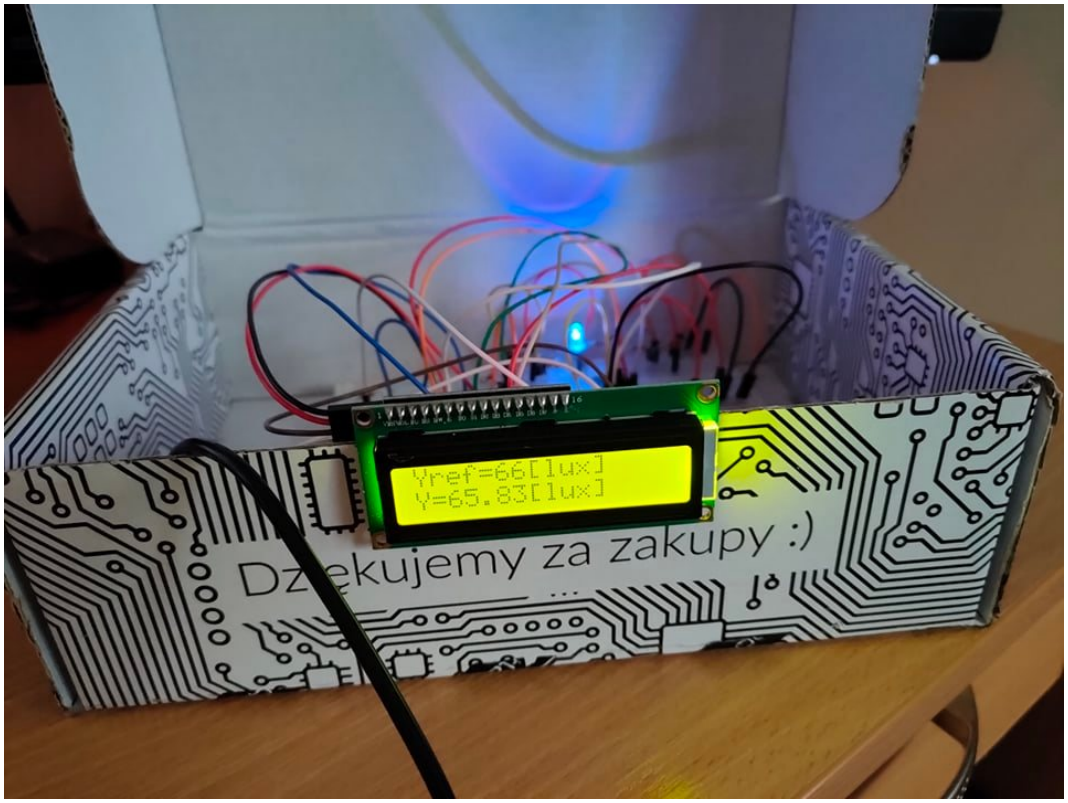


Rys. 10. Zdjęcie działającego układu - kolor fioletowy

Rys. 11. Zdjęcie działającego układu - kolor zielony



Rys. 12. Zdjęcie działającego układu - kolor biały



Rys. 13. Zdjęcie działającego układu - kolor błękitny

1.4 Wnioski

Wszystkie założone cele zostały osiągnięte. Na podstawie przebiegów odpowiedzi układu na konkretną wartość zadaną, można stwierdzić, że regulacjia przebiega zgodnie z założeniami. LED RGB przyjmuje zadane przez użytkownika barwy, natomiast sterowanie osiaga wartość zadaną z 1% uchybu ustalonego. Problem wystąpił przy jednoczesnym wysyłaniu informacji przez port szeregowy, wraz z komunikacją z wyświetlaczem LCD. Dodatkowa komunikacja przez aplikacje C# pozwoliła na sprawną zmiane wartości zadanych.

Podsumowanie

Zadanie zostało w większości zrealizowane zgodnie z założeniami. W całym projekcie zaimplementowano wszystkie podstawowe zagadnienia, a z rozszerzonych następujące:
- Wykorzystanie systemu kontroli wersji [5],
- System zapewnia uchyb ustalony poziomie 1% zakresu regulacji,
- Dedykowana aplikacja desktopowa jako graficzny interfejs użytkownika,
- Dodatkowe urządzenie wyjścia użytkownika(LCD),
- Dodatkowe urządzenie sterujące, ponieważ mieliśmy 3 źródła światła,
- Dodatkowe urządzenie wejścia użytkownika(przyciski),

Bibliografia

1. *LTspice Simulator | Analog Devices* [online] [udostępniono 2021-10-12]. Dostępne z: https://www.analog.com/en/design-center/design-tools-and-calculators/ltspice-simulator.html.

2. *KiCad Simulator | Analog Devices*. Dostępne także z: https://www.kicad.org/.
3. *BH1750 Light Sensor Pinout, Features & Datasheet* [Components101] [online] [udostępniono 2021-03-09]. Dostępne z: https://components101.com/sensors/bh1750-ambient-light-sensor.
4. *STM32CubeIDE | ST.com* [online]. [B.d.] [udostępniono 2019-09-30]. Dostępne z: https://www.st.com/en/development-tools/stm32cubeide.html.
5. *Github Link do projektu na githubie*. Dostępne także z: https://github.com/KonstantyOdwazny/Projekt_Systemy_Mikroprocesorowe.