POLITECHNIKA
BYDGOSKA
im. Jana i Jędrzeja Śniadeckich
Wydział Telekomunikacji,
Informatyki i Elektrotechniki

# DOKUMENTACJA PROJEKTU

## Mikroprocesory

Michalina Matuszak

# Temat Projektu:

Detekcja i pomiar składowych RGB światła widzialnego z wykorzystaniem czujnika kolorów.

# Opis projektu:

Celem projektu jest wykorzystanie modułu z czujnikiem koloru umożliwiającego pomiar składowych RGB (Red, Green, Blue) światła z wykorzystaniem mikrokontrolera STM32 Nucleo. Wyniki pomiarów oraz prezentacja danych zostaną przedstawione na wyświetlaczu alfanumerycznym LCD.

# Specyfikacja projektu:

**1.** Komunikacja z wykorzystaniem interfejsu USART z buforem kołowym (dane w buforze min. 1000 wpisów) i obsługą przerwań, implementacja odpowiedniego protokołu komunikacyjnego.

**2.** Podłączenie i konfiguracja wyświetlacza LCD w trybie 4-bitowym z odczytem flagi zajętości.

**3.** Podłączenie i konfiguracja czujnika kolorów. Czujnik ma wykorzystywać timer PWN IN przy wsparciu DMA.

**4.** Możliwość ustawienia interwału pomiarowego zadawanego w milisekundach.

**4.** Obsługa wyświetlenia pomiarów z czujnika na wyświetlacz oraz możliwość przeglądania danych archiwalnych i bieżących.

**Mikrokontroler:**

- Model: STM32 NUCLEO-F103RB
- Rdzeń: ARM Cortex M3 32-bit
- Częstotliwość taktowania: 72 MHz
- Pamięć programu Flash: 128 kB
- Pamięć SRAM: 20 kB
- 2x przetwornik analogowo-cyfrowy: 12-bitowy, 16-kanałowy
- Ilość Timerów: 7
- Interfejsy: 3x USART, 2x SPI 18Mbit/s, 2x I2C, USB Full Speed, CAN 2,0B

**Czujnik światła:**

- Model: TCS3200D
- Napięcie zasilania: 2,7 V do 5,5 V
- Programowalny wybór mierzonego koloru
- Programowalna częstotliwość wyjściowa
- Błąd nieliniowości na poziome 0,2 % przy 50 kHz
- Wyprowadzenia: goldpin

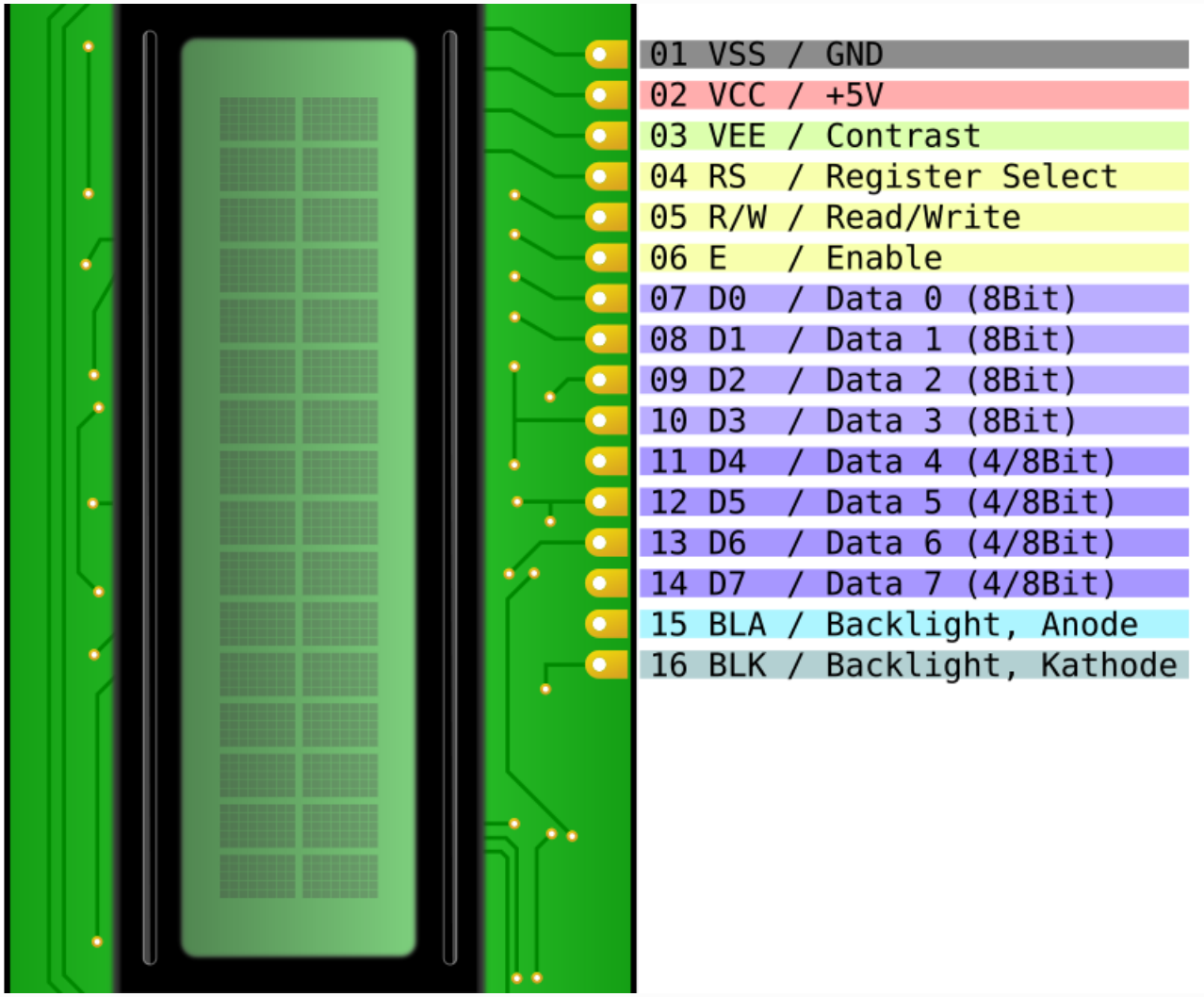
**Wyświetlacz:**

- Model: JHD162A-YG.
- Rodzaj: LCD 2x16 znaków,
- Sterownik zgodny z HD44780
- Podświetlanie: żółto-zielone, czarne znaki

Sterowanie w trybie 4-bitowym i 8-bitowym

# Wyświetlacz LCD

Wyświetlacz LCD ma dołączony moduł I2C, którego model to `PCF874` i jest on zgodny ze standardowym sterownikiem `HD44780`, którego rozkład pinów wygląda następująco:



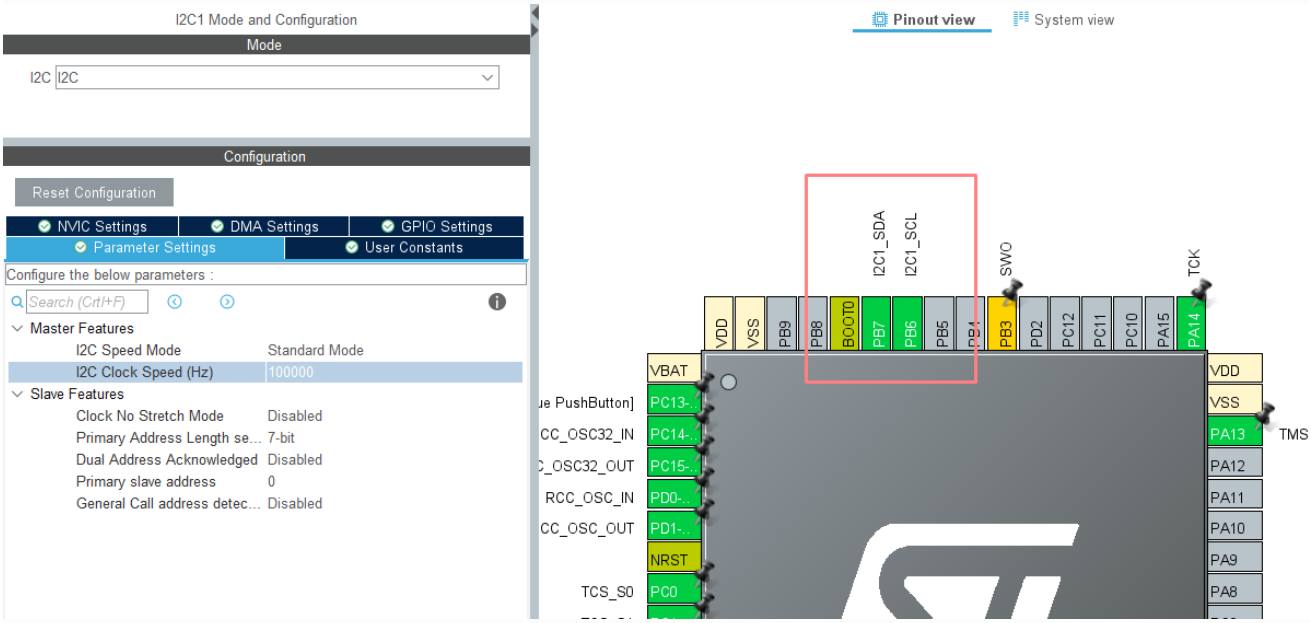

**RS** - bit przełączający między komendą funkcyjną, a zapisem do rejestru

**RW** - tryb odczytu/ zapisu. Odpowiednio bit 1 oznacza zapis, a bit 0 -odczyt

**EN** - pin zatwierdzający przesłanie

**BL** - załączenie podświetlenia wyświetlacza

Konfiguracja interfejsu I2C po którym następuje komunikacja z wyswietlaczem:




Zgodnie z dokumentacją modułu wyświetlacza w projekcie zaimportowane zostały odpowiednie komendy w pliku nagłówkowym `lcd_i2c.h` , prezentują się one w następujący sposób:

```
#define HI2C_DEF hi2c1

//komendy z dokumentacji wyswietlacza
#define RS_PIN 0x01
#define RW_PIN 0x02
#define EN_PIN 0x04
#define BL_PIN 0x08

#define INIT_8_BIT_MODE 0x30
#define INIT_4_BIT_MODE 0x02

#define CLEAR_LCD   0x01

#define UNDERLINE_OFF_BLINK_OFF     0x0C
#define UNDERLINE_OFF_BLINK_ON      0x0D
#define UNDERLINE_ON_BLINK_OFF      0x0E
#define UNDERLINE_ON_BLINK_ON       0x0F

#define FIRST_CHAR_LINE_1   0x80
#define FIRST_CHAR_LINE_2   0xC0
```

Od góry: Ustawienie odpowiednich bitów. Zdefiniowanie trybu w jakim chcemy, aby działał wyświetlacz. Czyszczenie wyświetlacza. `UNDERLINE_ON/OFF_BLINK_ON/OFF` określa załączenie migania kursora oraz podkreślenia. Określenie na jakiej lini ma pracować LCD.

Poniżej zdefiniowana jest struktura z informacjami o adresie urządzenia I2C, zmienne przechowujące znaki potrzebne do wyświetlenia, bit oznaczający podświetlenie.

```
struct lcd_disp {
    uint8_t addr;
    char f_line[17];
    char s_line[17];
    bool bl;
};
```

Do inicjalizacji wyświetlacza została zdefiniowana funkcja `lcd_init`, która jest zawarta w pliku `lcd_i2c.c`

Ponieważ wyświetlacz nie zapamięta stanu bitu BL, funkcja w projekcie odczytuje wartość bitu odpowiadającego za załączenie podświetlenia i zapisuje ten stan, aby wykorzystać go do innych komend. Wyświetlacz działał będzie w trybie 4-bitowym. Inicjalizacja takiego trybu przeprowadzona jest zgodnie z dokumentacją i wygląda następująco:

```
void lcd_init(struct lcd_disp * lcd)
{
    uint8_t xpin = 0;
    /* set backlight */
    if(lcd->bl)
    {
        xpin = BL_PIN;
    }

    /* init sequence */
    HAL_Delay(40);
    lcd_write(lcd->addr, INIT_8_BIT_MODE, xpin);
    HAL_Delay(5);
    lcd_write(lcd->addr, INIT_8_BIT_MODE, xpin);
    HAL_Delay(1);
    lcd_write(lcd->addr, INIT_8_BIT_MODE, xpin);

    /* set 4-bit mode */
    lcd_write(lcd->addr, INIT_4_BIT_MODE, xpin);

    /* set cursor mode */
    lcd_write(lcd->addr, UNDERLINE_OFF_BLINK_OFF, xpin);

    /* clear */
    lcd_clear(lcd);

}
```

W funkcji `lcd_write` tworzymy pomocniczą tabelę, która określi stany bitów na wyświetlaczu. Najpierw musimy przesłać bardziej znaczące bity, następnie mniej znaczące. Do tego posłużą zdefiniowane maski. Do przesłania informacji na wyświetlacz służy komunikacja `HAL_I2C_Master_Transmit`. Wszystko zaimplementowane jest zgodnie z dokumentacją wyświetlacza:

```
void lcd_write(uint8_t addr, uint8_t data, uint8_t xpin)
{
    uint8_t tx_data[4];

    /* split data */
    tx_data[0] = (data & 0xF0) | EN_PIN | xpin;
    tx_data[1] = (data & 0xF0) | xpin;
    tx_data[2] = (data << 4) | EN_PIN | xpin;
    tx_data[3] = (data << 4) | xpin;

    /* send data via i2c */
    HAL_I2C_Master_Transmit(&HI2C_DEF, addr, tx_data, 4, 100);

    HAL_Delay(5);
}
```

W funkcji `lcd_display` odczytujemy stan bitu BL. Czyścimy poprzednio wyświetlone znaki, aby nie dopuścić do nadpisywania się kolejnych znaków na poprzedni tekst. Wybieramy adresację linii. W pętli while, dopóki nie napotkamy zerowego znaku oznaczającego koniec linii, wysyłamy po kolei znaki w kodzie ASCII.

```
void lcd_display(struct lcd_disp * lcd)
{
    uint8_t xpin = 0, i = 0;

    /* set backlight */
    if(lcd->bl)
    {
        xpin = BL_PIN;
    }

    lcd_clear(lcd);

    /* send first line data */
    lcd_write(lcd->addr, FIRST_CHAR_LINE_1, xpin);
    while(lcd->f_line[i])
    {
        lcd_write(lcd->addr, lcd->f_line[i], (xpin | RS_PIN));
        i++;
    }

    /* send second line data */
    i = 0;
    lcd_write(lcd->addr, FIRST_CHAR_LINE_2, xpin);
    while(lcd->s_line[i])
    {
        lcd_write(lcd->addr, lcd->s_line[i], (xpin | RS_PIN));
        i++;
    }
}
```

Funkcja `lcd_clear` nadpisuje wartość wszystkich kolumn komendą CLEAR_LCD i zostawia załączone podświetlenie.

```
void lcd_clear(struct lcd_disp * lcd)
{
    uint8_t xpin = 0;

    /* set backlight */
    if(lcd->bl)
    {
        xpin = BL_PIN;
    }

    /* clear display */
    lcd_write(lcd->addr, CLEAR_LCD, xpin);
}
```

```
struct lcd_disp disp;
```

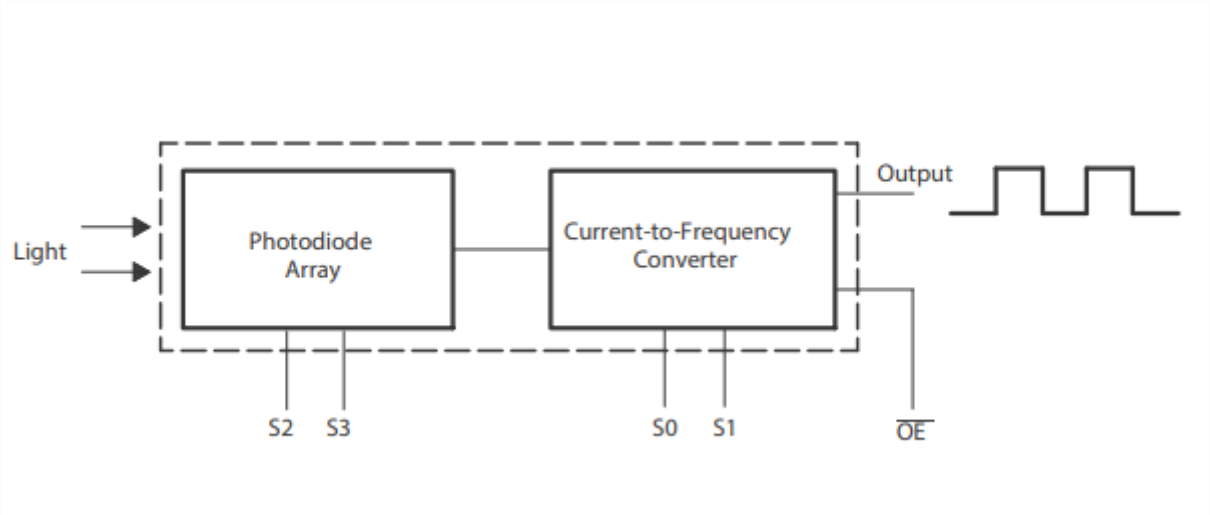
Aby uruchomić wyświetlacz na mikroprocesorze w głównej funkcji programu inicjalizujemy I2C poprzez `MX_I2C1_Init()` . Następnie określamy do uprzednio stworzonej struktury wszystkie potrzebne informacje. W tym przypadku adres urządzenia to 0x27. Adres musi być przesunięty bitowo o 1, ponieważ interfes ustawiony jest na obsługę 7-bitowy adresu.

```
//Wyświetlacz
   disp.addr = (0x27<<1);
   disp.bl = true;
   lcd_init(&disp);
   sprintf((char *)disp.f_line, "WYSWIETLACZ");
   sprintf((char *)disp.s_line, "URUCHOMIONY");
   lcd_display(&disp);
```

# Czujnik TCS3200D

Czujnik TCS3200D to programowany konwerter światło-częstotliwość, który wykorzystuje konfigurowalne fotodiody krzemowe i przetworniki prądu na częstotliwość. Na wyjściu czujnika otrzymujemy falę kwadratową o okresie wypełnienia 50% oraz z częstotliwością wprost proporcjonalną do natężenia światła.

Czujnik odczytuje tablicę 8 × 8 fotodiod. Szesnaście fotodiod ma niebieskie filtry, 16 fotodiody ma filtry zielone, 16 fotodiod ma filtry czerwone, a 16 fotodiod jest bez filtrów.



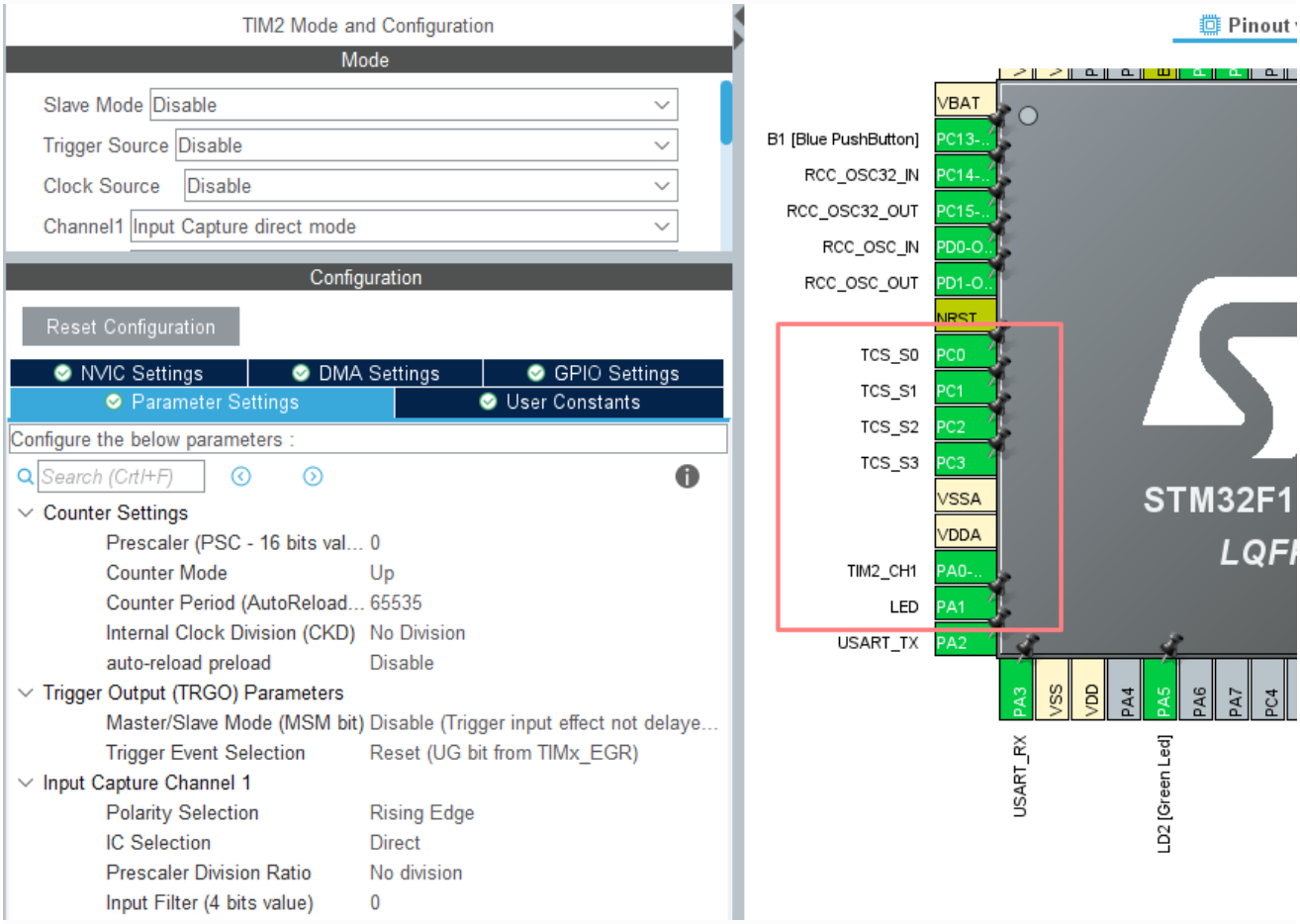

Funkcje wyprowadzeń czujnika:

- **VCC** - zasilanie, 2,7-5,5 V
- **GND** - masa zasilania
- **LED** - sterowanie diodami podświetlającymi
- **OUT** - linia wyjściowa, sygnał kwadratowy o częstotliwości zależnej od natężenia światła koloru wybranego na liniach S2 i S3
- **S0, S1** - skalowanie częstotliwości na linii OUT na 2%, 20% i 100%.
- **S2, S3** - linie służące do wyboru badanego koloru składowego

Konfigurowalne wejścia czujnika S0, S1, S2, S3:

| S0 | S1 | Output Frequency Scaling ($f_o$) | S2 | S3 | Photodiode Type |
|---|---|---|---|---|---|
| L | L | Power down | L | L | Red |
| L | H | 2% | L | H | Blue |
| H | L | 20% | H | L | Clear (no filter) |
| H | H | 100% | H | H | Green |

Output czujnika podłączony jest do STM jako timer PWM w trybie **Input Capture**. Timer jest wspierany przez kanał DMA.  Konfiguracja reszty pinów dla czujnika podłączona jest w następujący sposób:



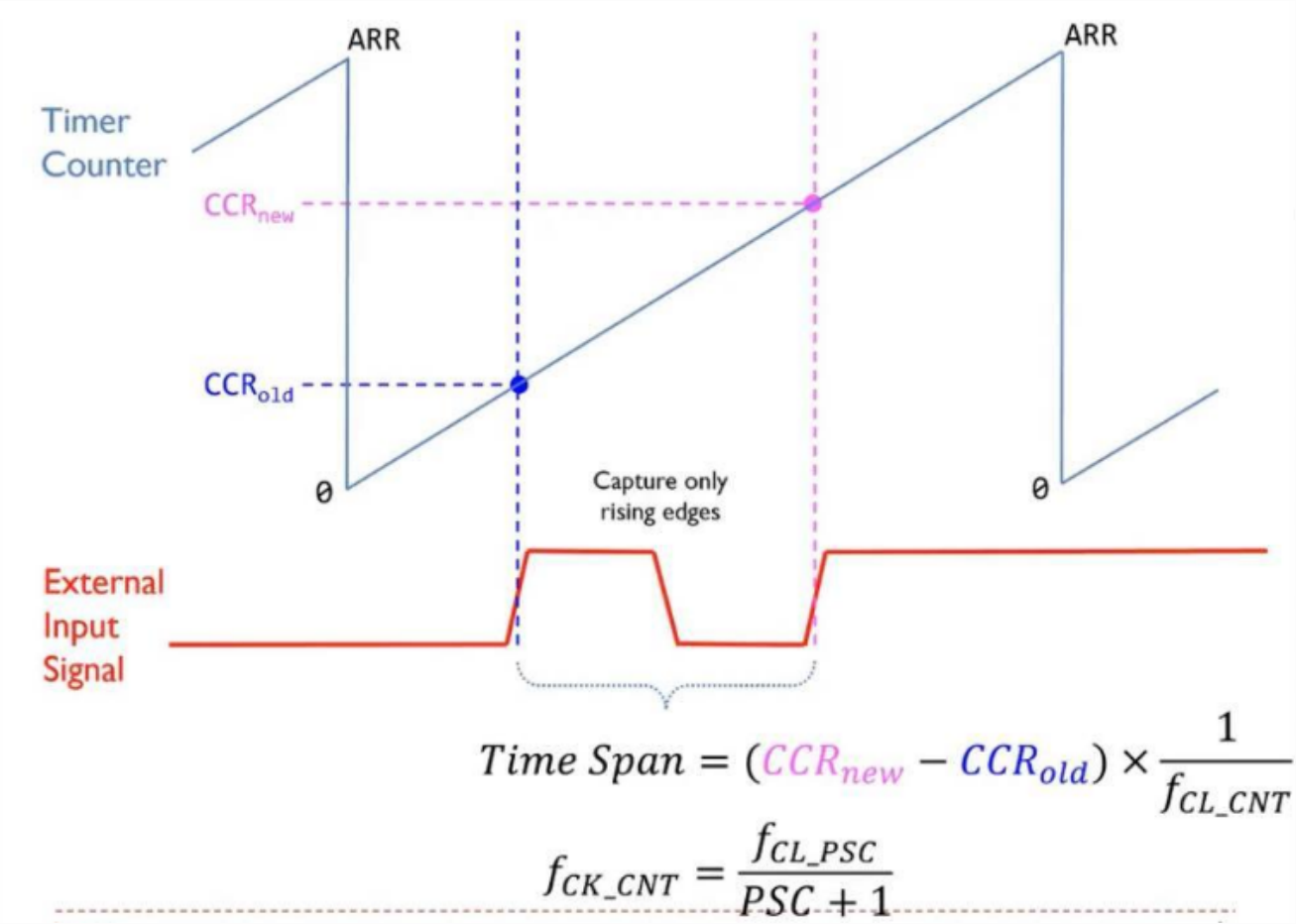

Ustawienie skalowania częstotliwości na 100% - zgodnie z dokumentacją czujnika ustawiamy odpowiednie wejścia S0 i S1 na stan wysoki. Załączamy podświetlenie LED w czujniku.

```
void TCS_Init(){
    // Output frequency scaling dla czujnika = 100%
    HAL_GPIO_WritePin(TCS_S0_GPIO_Port, TCS_S0_Pin, HIGH);
    HAL_GPIO_WritePin(TCS_S1_GPIO_Port, TCS_S1_Pin, HIGH);
    HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, HIGH);
}
```

Do konfiguracji wyboru filtra kolorów posłuży wcześniej zdefiniowane enum, które w funkcjach automatycznie ustawi stan pinów S2 i S3 na odpowiadający filtr.:

```
typedef enum FilterType FilterType;
enum FilterType {
    Red=0, Blue, Clear, Green
};
#define LOW RESET
#define HIGH SET
const FlagStatus TCS_S2_Logic_Input[4]={LOW, LOW, HIGH, HIGH};
const FlagStatus TCS_S3_Logic_Input[4]={LOW, HIGH, LOW, HIGH};
```

Do zmierzenia częstotliwości z jaką nadaje na wyjściu czujnik służy funkcja `TCS_GetLightIntensity` , która przyjmuje parametry pinów S2 i S3, aby określić jakiego koloru intensywność badamy. Do wyliczenia częstotliwości potrzebujemy wartość pełnego okresu, czyli zmierzenia odstępu między 2 zboczami narastającymi sygnału czujnika. Działanie takiego procesu przedstawione jest na instrukcji poniżej:




$$Time\ Span = (CCR_{new} - CCR_{old}) \times \frac{1}{f_{CL\_CNT}}$$

$$f_{CK\_CNT} = \frac{f_{CL\_PSC}}{PSC + 1}$$

Gdzie:

- ***CCR*** *- wartość rejestru CCR. Im większa wartość rejestru, tym większa wartość licznika timera.*
- ***f_CL_CNT*** *- częstotliwość zegara wewnętrznego*
- ***PSC*** *- wartość preskalera*
- ***f_CL_PSC*** *- częstotliwość po  zastosowaniu preskalera.*

Przy wsparciu DMA timer w trybie Input Capture wychwytuje 2 następujące po sobie zbocza narastające i wychwytuje wartość timera do tablicy `captures`. Dzięki temu możemy obliczyć wartość okresu obliczając różnice czasową między 2 zboczami narastającymi.

```
static float TCS_GetLightIntensity(FilterType filter){
    captureDone=0;
    HAL_GPIO_WritePin(TCS_S2_GPIO_Port, TCS_S2_Pin, TCS_S2_Logic_Input[filter]);
    HAL_GPIO_WritePin(TCS_S3_GPIO_Port, TCS_S3_Pin, TCS_S3_Logic_Input[filter]);

    HAL_TIM_IC_Start_DMA(&htim2, TIM_CHANNEL_1, (uint32_t*) captures, 2);
    while(!captureDone) {}
    HAL_TIM_IC_Stop_DMA(&htim2, TIM_CHANNEL_1);

    uint32_t diffCapture = 0;
    if (captures[1] >= captures[0])
        diffCapture = captures[1] - captures[0];
    else
        diffCapture = (htim2.Instance->ARR - captures[0]) + captures[1] + 1;

// TIM2 Clock = PCLK1
    float frequency = HAL_RCC_GetPCLK1Freq()/ (htim2.Instance->PSC + 1.0);
    frequency = (float) frequency / (float) diffCapture;
    return frequency;
}
```

Następnie odczytujemy zgodnie ze wzorem częstotliwość taktowania PCLK1, który jest zegarem timera2 i dzielimy tę wartość przez okres, co daje nam w wyniku częstotliwość nadawania czujnika. Finalnie znamy intensywność badanego koloru.

Kolejna fukcja, która potrzebna jest do kalibracji wyników to `TCS_GetLightIntensities` . W fukcji `main` następuje poproszenie użytkownika o przyłożenie czarnego koloru, następnie wywołana jest poniższa funkcja w celu określenia poziomu czerni. Analogiczny proces przeprowadzany jest na kolorze białym. Dzięki temu posiadamy odwołanie do wartości minimalnej i maksymalnej RGB (0 - 255).

```
void TCS_GetLightIntensities(Frequency *freq){
    freq->clear=TCS_GetLightIntensity(Clear);
    freq->red =TCS_GetLightIntensity(Red);
    freq->green=TCS_GetLightIntensity(Green);
    freq->blue =TCS_GetLightIntensity(Blue);
}
```

Ostatnim krokiem jest przeliczenie uzyskanej częstotliwości na wartość RGB. Ponieważ czujnik określa intensywność światła wprost proporcjonalne do częstotliwości, możliwe jest mapowanie między częstotliwością, a wartością koloru RGB (0-255 dla każdego z R, G i B) przy użyciu interpolacji liniowej.

Dzięki poprzedniej kalibracji dwa punkty na linii RGB są już dobrze określone – czysta czerń (RGB 0, 0, 0) i czysta biel (255, 255, 255). Wartości zwracane przez czujnik można odczytać za pomocą łatwo dostępnych próbek kolorów:

- Karta koloru czarnego daje nam stałą warunku ciemności $f_D$ . Jest to początek (wartość zerowa) konwersji linii prostej RGB.
- Karta koloru białego daje nam ekstremalny punkt RGB $f_W$ , znany również jako balans bieli. Znając $f_D$ , wartość ta może być użyta do skalowania wszystkich częstotliwości pośrednich do odpowiedniej wartości

RGB.

Zależność proporcjonalna jest wyrażona przez standardowe równanie linii prostej `y = mx + b` , gdzie

- ***y*** - to otrzymany odczyt (w naszym przypadku ***f*** *O* )
- **x** - to znormalizowana wartość RGB
- **b** - jest wartością *y* , gdy *x* wynosi 0 (w naszym przypadku ***f*** *D* )
- ***m*** - jest nachyleniem lub stałą proporcjonalności linii (w naszym przypadku [ ***f*** *W* – ***f*** *D* ]/255).

Otrzymane równanie to

$$f_O = f_D + \frac{x \cdot (f_W - f_D)}{255}$$

lub zmieniając kolejność, aby uzyskać żądaną wartość RGB

$$x = \frac{255 \cdot (f_O - f_D)}{(f_W - f_D)}$$

```
void TCS_EstmateRGB(RGB *rgb){
    Frequency freq;
    TCS_GetLightIntensities(&freq);
    float ared = (freq.red -freq_DC.red) /(freq_WB.red -freq_DC.red);
    float agreen = (freq.green-freq_DC.green)/(freq_WB.green-freq_DC.green);
    float ablue = (freq.blue -freq_DC.blue) /(freq_WB.blue -freq_DC.blue);
    float amax=ared;
    if (agreen>amax) amax=agreen;
    if (ablue>amax) amax=ablue;
    if (amax<1.0) amax=1.0;
    rgb->red = (uint8_t) 255*ared/amax;
    rgb->green = (uint8_t) 255*agreen/amax;
    rgb->blue = (uint8_t) 255*ablue/amax;
}
```

Wspomniane wcześniej wywołanie kalibracji balansu bieli w funkcji `main` :

```
//Czujnik
   TCS_Init();

   WysylanieDanych(" *Kalibracja* Przybliz BIALY kolor i kliknij niebieski przycisk. ");
   while(!calibrationRequest){}
   calibrationRequest=0;
   TCS_GetLightIntensities(&freq_WB);

   WysylanieDanych(" *Kalibracja* Przybliz CZARNY kolor i kliknij niebieski przycisk. ");
   while(!calibrationRequest){}
   calibrationRequest=0;
   TCS_GetLightIntensities(&freq_DC);
```

Wykorzystane wyżej `calibrationRequest` następuje jako przerwanie po naciśnięciu niebieskiego przycisku na płytce:

```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
    if(GPIO_Pin == B1_Pin) {
        calibrationRequest=1;
    }
}
```

# Protokół komunikacyjny

## Opis komunikacji:

Komunikacja między urządzeniem, a PC odbywać się będzie poprzez interfejs USART, przy pomocy Terminala użytkownik będzie mógł wysyłać odpowiednie komendy oraz otrzymywać komunikaty zwrotne zgodnie z zaprojektowanym protokołem komunikacyjnym.

Odpowiednie znaki komendy oraz dane wartości będą odbierane przez STM w postaci ramki protokołu i przechowywane w formacie ASCII kodowane szesnastkowo.

W protokole zostało dodane obliczanie **sumy kontrolnej**, aby wykryć ewentualne błędy w ramce. W celu obliczania sumy kontrolnej zostanie wykorzystany algorytm CRC8. Suma kontrolna jest podzielona na 2 bajty więc liczba nie przychodzi w całości tylko znak po znaku (co uniemożliwi pojawienie się znaku początku lub końca ramki w środku ramki). Jest reprezentowana w liczbach szesnastkowych np. 4D.

Obliczana jest ona na zasadzie poniższego algorytmu:

```
char CRC8(const char *data,int length)
{
   char crc = 0xF4;
   char extract;
   char sum;
   for(int i=0;i<length;i++)
   {
      extract = *data;
      for (char tempI = 8; tempI; tempI--)
      {
         sum = (crc ^ extract) & 0x01;
         crc >>= 1;
         if (sum)
            crc ^= 0x8C;
         extract >>= 1;
      }
      data++;
   }
   return crc;
}
```

## Ramka Protokołu:

| Początek ramki | Długość dane + komenda | Polecenie | Dane | Suma kontrolna | Koniec ramki |
|---|---|---|---|---|---|
| 1 B | 1 B | 3 B | 0 B – 6 B | 2 B | 1 B |
| znak ASCII<br>{<br>0x7B | znaki ASCII<br>0- 9<br>(0x30 – 0x39) | znaki ASCII<br>A-Z<br>(0x41 – 0x5A) | znaki ASCII<br>0-9 , A-Z<br>(0x41 – 0x5A)<br>(0x30 – 0x39) | znaki ASCII<br>wartość sumy<br>(0x00 – 0xFF) | znak ASCII<br>}<br>0x7D |
| 0x7B | z wyjątkiem znaków początku i końca ramki ( 0x7B i 0x7D ) | | | | 0x7D |
| 1 znak | 1 znak | 3 znaki | 0 – 6 znaków | 2 znaki | 1 znak |

## Analizowana ramka zostanie odrzucona, gdy:

- Polecenie będzie składało się ze znaków innych niż 0-9, A-Z
- Suma kontrolna ramki będzie nieprawidłowa
- Gdy do urządzenia zostanie wysłana prawidłowa ramka, w środku której znajdą się znaki rozpoczęcia lub zakończenia ramki
- Długość komendy będzie nieprawidłowa
- Maksymalna długość polecenia to 3 znaki
- Maksymalna długość wartości danych to 6 znaków, co w sumie daje 9 razem z poleceniem
- Minimalna długość ramki wynosi 8 znaków (wliczając znak początku i końca ramki)
- Maksymalna długość ramki wynosi 41 znaków (wliczając znak początku i końca ramki)
- Długość ramki musi zawierać znaki liczbowe 0-9 (0x30 do 0x39)
- Polecenie składa się z 3 znaków nagłówka komendy. A-Z (0x41 – 0x5A)
- Dodatkowo polecenie może zawierać dane liczbowe (np. informujące o żądanym numerze wybranego pomiaru z archiwum lub o wartości ustawianego interwału pomiarowego)
- W polu dane mogą zostać przesłane komunikaty zwrotne dotyczące wprowadzanych komend

## Obsługa błędów:

- Aby zapewnić poprawny odbiór ramki, analizowane są wystąpienia znaków początku i końca ramki.
- Gdy dojdzie do inicjalizacji odbioru danych, wszystkie wartości niepoprzedzone znakiem początku ramki są odrzucane.
- Jeżeli po znaku początku ramki, pojawi się jeszcze raz znak początku ramki to brany jest pod uwagę tylko ostatni wysłany znak początku.

· Jeśli ramka zostanie wprowadzona prawidłowo, a polecenie zostanie nierozpoznane, zostanie zwrócona informacja o błędnej komendzie lub błędnej wartości

## Nagłówek komendy:

Polecenia składają się z 3 znaków i możliwe jest przypisanie do nich wartości w polu Dane - jeśli chcemy ustawić interwał pomiarowy czujnika lub zażądać odczytania danych archiwalnych z konkretnej pozycji bufora

| Polecenie | Dane | Funkcja |
|---|---|---|
| SET | 1 - 9999 | Ustawienie interwału pomiarowego dla czujnika (zadawany w milisekundach). |
| ARH | 1 - 1000 | Wyświetlanie danych archiwalnych (odczyt z bufora). |
| NOW | - | Wyświetlanie danych bieżących. |
| CLR | - | Wyczyszczenie wyświetlacza. |

Przykładowe polecenie z wartością:

**ARH450** – odczytanie pomiaru archiwalnego nr. 450

Suma takiego komunikatu wynosi 6 bajtów.

## Komunikaty zwrotne:

W protokole możliwe jest też przesyłanie komunikatów zwrotnych otrzymywanych od mikroprocesora. Każdy z komunikatów zawiera nagłówek- czyli nazwę polecenia, którego dotyczy oraz treść komunikatu.

# Wysyłanie znaków

W celu wysłania łańcucha znaków, muszę umieścić dane do wysłania w buforze nadawczym. Do tego celu służy funkcja zapisDoBufora():

```
void zapisdobuforaTx(char* znak, uint8_t dlugosc){ //przekazywanie słowa do zapisu na Tx
    for(uint8_t x=0; x < dlugosc; x++){
            TxBUF[TxEmpty] = znak[x]; //zapis do TxEmpty kazdego kolejnego znaku
            TxEmpty = TxEmpty + 1;
            if(TxEmpty==128) TxEmpty=0;
        }
}
```

W dalszym procesie przesyłania znaków uczestniczy funkcja „WysylanieDanych()”, która odbiera znaki i przekazuje je za pomocą przerwań do powyższej funkcji. Wygląda ona następująco:

```
void WysylanieDanych(char* DOWYSLANIA) {
    uint8_t dlugosc = strlen(DOWYSLANIA);
    if(dlugosc > 128) return; //ograniczenie  długości danych
    zapisdobuforaTx(DOWYSLANIA,dlugosc); //zapis znaków do bufora

    __disable_irq(); //zablokowanie przerwan
    if(flaga == 0) {
            flaga = 1; //ustawienie flagi na "transmisja rozpoczęta"
            HAL_UART_Transmit_IT(&huart2, TxBUF + TxBusy, 1);
            TxBusy = TxBusy + 1;

            if(TxBusy==128) TxBusy=0;
        }
    __enable_irq();//odblokowanie przerwan
}
```

# Bufor kołowy

Zaimplementowano obsługę bufora kołowego, mogącego przechowywać 1000 znaków. Mechanizm pozwala na przechowywanie danych, których nie można od razu obsłużyć. Odbieranie i wysyłanie znaków odbywa się poprzez interfejs komunikacyjny USART, którego funkcję przedstawia poniższy kod:

```
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart) {
    if(huart->Instance == USART2)
    {
        RxBUF[RxEmpty] = Received;
        RxEmpty = RxEmpty + 1;
        if(RxEmpty==128) RxEmpty=0;
    }
    HAL_UART_Receive_IT(&huart2, &Received, 1);
}

void HAL_UART_TxCpltCallback(UART_HandleTypeDef *huart) {
    if(huart->Instance == USART2){
        if(TxBusy != TxEmpty) {
            HAL_UART_Transmit_IT(&huart2, TxBUF + TxBusy, 1);
            TxBusy = TxBusy+1;
            if(TxBusy==128) TxBusy=0;
        }
        else {
            flaga = 0; //transmisja zak.
        }
    }
}
```

Aby uruchomić nasłuchiwanie na kanale USART (zawsze po odebraniu danych lub przy inicjalizacji pracy bufora), wywołuję polecenie:

```
HAL_UART_Receive_IT(&huart2, &Received, 1);
```

W pętli głównej uruchamiam funkcję służącą do odczytu bufora odbiorczego oraz kolekcjonowania ramki:

```
void get_frame(){ //ODBIERANIE DANYCH KTÓRE MOGĄ BYĆ POTENCJALNĄ RAMKĄ
    static int dlugosc_pol; //dlugosc polecenia komendy
    static int dlugosc_danych; //dlugosc danych w ramce (nie licząc sumy kontrolnej)
    if(RxEmpty != RxBusy){
        if(RxBUF[RxBusy] == 0x7B ){ //ZNAK POCZĄTKU
                reset_checking = yes;
                frame_status = true;
                clear_data(); //czyszczenie poprzedniej ramki
                ramka[idx_rx] = 0x7B;
                idx_rx++;
        }
        else if(RxBUF[RxBusy] == 0x7D ){    //ZNAK KOŃCA
                if(frame_status == true){ //uwzględniamy tylko jeśli wczesniej był znak począt
                    ramka[idx_rx] = 0x7D;
                    WysylanieDanych(ramka);
                    check_frame(ramka); // OK - wysyłamy ramke do sprawdzenia
                    frame_status = false;
                    dlugosc_danych = 0;
                    //clear_data();
                }
        }
        else { //JEŚLI ZNAK JEST INNY NIŻ ZNAK POCZĄTKU I KOŃCA
            if((frame_status == true) && (reset_checking == no)){   //wpisywanie znaków do ram
                uint8_t znak = RxBUF[RxBusy];
                if (etap==0){
                    WysylanieDanych("ETAP 0 ");
                    if(znak >= 0x33 && znak<= 0x39){ //deklaracja dlugosci danych od 3 do 9;
                        ramka[1] = znak;
                        dlugosc_danych = ramka[1] - '0';
                        idx_rx++;
                        etap++;
                    }else { WysylanieDanych(ramka);
                            frame_status = false;
                            frame_error(etap);//nieprawidlowa dlugosc danych
                            clear_data();
                    }
                }
                else if(etap==1){
                        if(znak >= 0x41 && znak<= 0x5A){ //znaki polecenia A-Z
                            ramka[idx_rx] = znak;
                            dlugosc_pol++;
                            idx_rx++;
                            if(dlugosc_pol == 3) { //max. długość komendy = 3 litery
                                etap++;
                                dlugosc_pol = 0;}
                        }else { WysylanieDanych(ramka);
                                frame_status = false;
                                frame_error(etap); //blad znaków polecenia
                                clear_data();
                                dlugosc_pol = 0;}
                }
```

```
            else if(etap==2){          //jeśli długość ramki przekroczy zadeklarowaną "długość danych"
                    if (idx_rx > dlugosc_danych + 1){ //to znaczy, że to już są znaki sumy kontrolnej
                        ramka[idx_rx] = znak;
                        idx_rx++;
                        if (idx_rx > dlugosc_danych + 4){ //po przekroczeniu miejsc na sume kontrloną - błąd
                            dlugosc_danych = 0;
                            frame_status = false;
                            frame_error(3); //przekroczono zakres ramki
                            clear_data();}
                    } else{
                            if(znak >= 0x30 && znak<= 0x39){ //wpisujemy dane 0-9
                                ramka[idx_rx] = znak;
                                idx_rx++;
                                WysylanieDanych(ramka);
                            }else { frame_error(etap); //oczekiwano 0 - 9;
                                    clear_data();
                                    frame_status = false;}
                    }
            }
        }
        reset_checking = no;
        RxBusy++;
        if(RxBusy == 128) RxBusy = 0;
    }
}
```

W powyższej funkcji odczytujemy znaki zapisane w buforze Rx. W pierwszej kolejności analizujemy, czy odebrany znak jest znakiem początku lub końca ramki. Jeżeli odebraliśmy znak początku to ustawiamy „frame_status” na ‘true’

Oraz resetujemy wcześniejsze znaki zapisane do tablicy ‘ramka’. Jeżeli odebraliśmy znak końca (oraz wcześniejsze dane do ramki) wtedy kompletowanie ustawiamy na zakończone i wysyłamy ramkę do innej fukcji w celu analizy sumy kontrolnej.

W ciele funkcji analizujemy odebrane znaki. Jeżeli odebraliśmy znak początku dalsze dane muszą mieścić się w zakresie określonym przez protokół komunikacyjny. W tym celu w funkcji get_frame zaimplementowane zostały liczne warunki sprawdzające poprawność odbieranych danych.

Z powyższej funkcji wysyłania znaków do użytkownika korzystają inne funkcje projektu. Między innymi funkcja zwracająca komunikaty błędów:

```
void frame_error(int etap){

    if (etap == 1) {
        WysylanieDanych("Podano znaki polecenia inne niz A-Z. etap 1");
    }
    else if (etap == 2){
        WysylanieDanych("Przekroczono zakres dlugosci danych. etap 2 ");
    }
    else if (etap == 0){
        WysylanieDanych("Nieprawidlowa dlugosc danych ");
    }
    else if (etap == 3){
        WysylanieDanych("Przekroczono zakres ramki ");
    }
}
```

# Obliczanie sumy kontrolnej:

Suma kontrolna obliczana jest za pomocą algorytmu CRC8. Dokładniej proces obliczania sumy przedstawia poniższa funkcja, w której dokładnie widać jak trakturemy obliczenia:

```
void check_frame(char* ramka){
    size_t len = strlen(ramka);
    uint8_t crc = 0xF4;
    size_t i, j;
    for (i = 0; i < len - 3 ; i++) {
        crc ^= ramka[i];
        for (j = 0; j < 8; j++) {
            if ((crc & 0x80) != 0)
                crc = (uint8_t)((crc << 1) ^ 0x31);
            else
                crc <<= 1;
        }
    }
    char suma[2];
    sprintf(suma,"%x",crc);
    WysylanieDanych(", Obliczona suma: ");
    WysylanieDanych(suma);
    char temp_suma[2];
    temp_suma[0]= ramka[len-3];
    temp_suma[1]= ramka[len-2];
    WysylanieDanych(", Podana suma: ");
    WysylanieDanych(temp_suma);

    if (!strncmp(suma,temp_suma,2)){
        WysylanieDanych(", SUMA KONTROLNA: OK ");
    }
    else {
        WysylanieDanych(", SUMA KONTROLNA: ZLE ");
        return;
    }
```

Jeżeli podana przez użytkownika suma kontrolna w ramce zgadza się z tą obliczoną przez algorytm następuje przejście do procesowania wykrytych komend, jeżeli suma jest błędna występuje komunikat zwrotny.

### Komenda NOW:

Zadaniem tej komendy jest wyłowanie funkcji `LCD_now`, która odpowiada za wyświetlanie na LCD wartości koloru przyłożonego do czujnika w czasie rzeczywistym w odstępie zadanym przez odpowiedni Timer3. Wykrywanie tej komendy następuje poprzez funkcje `strncmp`:

```
uint8_t y = ramka[1] - '0'; //ramka[1] - długosc danych (pol+dane)
char command[y];
for(uint8_t x=0; x < y; x++){
    command[x]=ramka[x+2];
        }

WysylanieDanych(", komenda: ");
WysylanieDanych(command);

    /* ============================
     *          KOMENDA NOW
     *============================ */
if (!strncmp("NOW",command,3)){
    WysylanieDanych(" Odebrano komende: NOW ");
    HAL_NVIC_EnableIRQ(TIM3_IRQn);
 }
```

Wywołanie funkcji `LCD_now` wywoływane jest w takt Timera, który odlicza co 1 sek i ustawia wartość `LCD_now_active = 1`, dzięki temu funkcja wywoływana jest z poziomu maina co 1 sek.

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim){
    if(htim->Instance == TIM3){
        LCD_now_active = 1;
        }
}
```

Ciało funkcji wygląda w następujący sposób:

```
void LCD_now(){
    RGB rgb;
    TCS_EstmateRGB(&rgb);
    sprintf((char *)disp.f_line, "RED GREEN BLUE");
    sprintf((char *)disp.s_line, "%3u  %3u  %3u", rgb.red, rgb.green, rgb.blue);
    lcd_display(&disp);
    zapisTCS_BUF(rgb.red, rgb.green, rgb.blue);
}
```

## Komenda CLR:

Aby zastopować wyświetlanie RGB, musimy wyłączyć przerwanie Timera3. Następnie czyścimy wszystkie znaki:

```
    /* ============================
     *          KOMENDA CLR
     *============================ */
else if( !strncmp("CLR",command,3)){
    HAL_NVIC_DisableIRQ(TIM3_IRQn);
    WysylanieDanych(" Odebrano komende: CLR ");
    lcd_clear(&disp);

}
```

### Komenda ARH:

Zadaniem tej komendy jest wyświetlenie na LCD danych archiwalnych, które zostały zapisane do oddzielnych buforów zapisujących wartości RGB odczytywane z funkcji `LCD_now`. Zapisywanie do bufora wygląda następująco:

```
void zapisTCS_BUF(uint8_t red, uint8_t green, uint8_t blue){
/*
uint16_t TCS_IDX;
uint16_t TCS_BUF_RED[1000];
uint16_t TCS_BUF_GREEN[1000];
uint16_t TCS_BUF_BLUE[1000];

 */
    TCS_BUF_RED[TCS_IDX] = red;
    TCS_BUF_GREEN[TCS_IDX] = green;
    TCS_BUF_BLUE[TCS_IDX] = blue;
    TCS_IDX++;
    if (TCS_IDX == 1000) {
        TCS_IDX = 0;
        cycle = new;
    }

}
```

Następnie po wykryciu komendy w ramce odczytujemy wartość z pola dane, jaką przekazał użytkownik - będzie ona numerem odczytu z bufora. Ponieważ bufor ten działa na zasadzie bufora kołowego, po przekroczeniu 1000 wpisów każdy kolejny jest nadpisywany. Po określeniu wartości odczytu przekazujemy dane z bufora na wyświetlacz:

```
        /* ============================
         *         KOMENDA ARH
         *============================ */
   else if(!strncmp("ARH",command,3)){
       HAL_NVIC_DisableIRQ(TIM3_IRQn);
       char command_data[3];
       for(uint8_t x=0; x < y - 3; x++){
            command_data[x]=command[x+3];
                }
       uint16_t arh_num = atoi(command_data);
       WysylanieDanych(", Odebrano komende: ARH ");
       sprintf((char *)disp.f_line, "Odczyt nr %d", arh_num);

       if(cycle == new){
           arh_num = arh_num + TCS_IDX;
               if (arh_num > 999){
                   arh_num = arh_num - 1000;
               }
       }
       sprintf((char *)disp.s_line, "%3u  %3u  %3u", TCS_BUF_RED[arh_num], TCS_BUF_GREEN[arh_num], TCS_BUF_BLUE[arh_num]);
       lcd_display(&disp);
   };
}
```