Czujnik natężenia światła

07.05.2020

Kacper Kupiszewski

Wstęp

Zdecydowałem się na taki projekt, ponieważ chciałem zaprezentować działanie różnych interfejsów wykorzystywanych w programowaniu mikrokontrolerów. Skonstruowane przeze mnie urządzenie poprzez fotorezystor rejestruje natężenie światła i przetwarza te dane dalej do programu.

Napisany przeze mnie program uwzględnia trzy tryby pracy.

- Pierwszy najbardziej prymitywny na podstawie wartości natężenia światła zapala kolejne LED'y na linijce.
- Drugi tryb dodaje funkcjonalność diody RGB, która zmienia swój kolor w czasie.
- Trzeci a zarazem ostatni tryb wykorzystuje diodę RGB, której kolor sterowany jest wartością natężenia światła.

To tyle jeżeli chodzi o funkcjonalność tego urządzenia, aspekty techniczne rozwiązania danych problemów są poruszone w dalszej części dokumentacji.

Filmik przedstawiający funkcjonalność urządzenia - > https://youtu.be/EjyMuQAQO4s

Niestety urządzenie, którym nagrywałem filmik nie posiada funkcji High Dynamic Range, więc miało problemy z rzeczywistym zarejestrowaniem koloru diody RGB.

Dla drugiego trybu urządzenia dosyć dobrze widać zmianę koloru na diodzie.

Gorzej jest dla trzeciego, gdzie ,,na żywo" dokładnie widać, że dla słabego oświetlenia dioda świeci się na krwistoczerwony kolor, a w przypadku wystawienia na mocne światło mieni się lekko różowym kolorem.

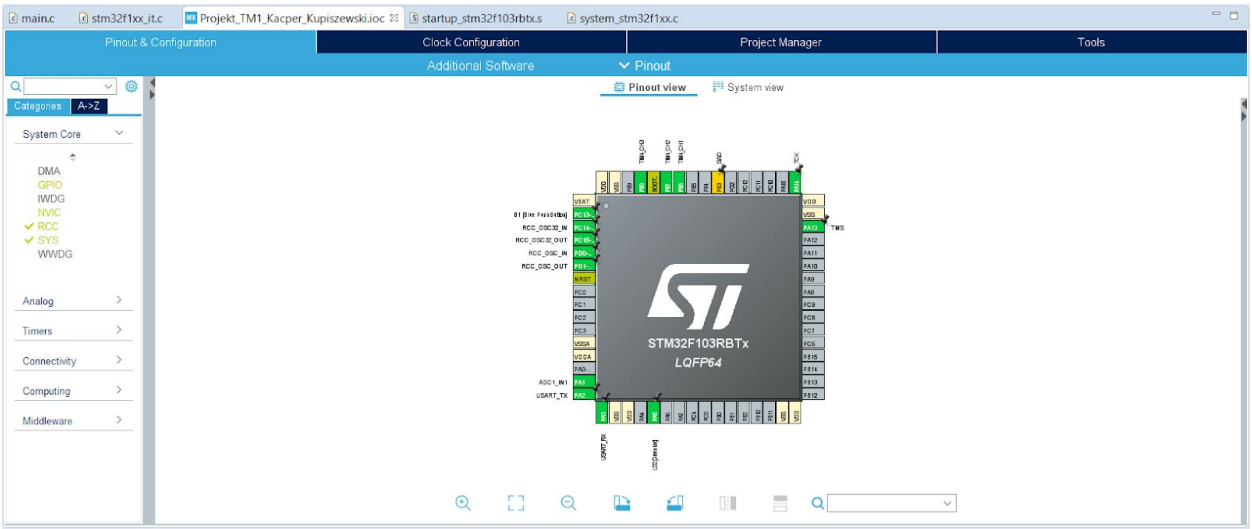
Specyfikacje

Projekt oparty jest o mikrokontroler Nucleo-F103RB firmy STMicroelectronics. Jest to platforma z mikroprocesorem ARM Cortex M4. Niestety nie posiadam płytki z mikroprocesorem z rodziny AVR, więc po uzgodnieniu z prowadzącym zajęcia użyłem platformy którą posiadam.

Jeżeli chodzi o dodatkowe peryferia wykorzystane w moim projekcie to użyłem:

- 10 LED'ów o kolorze czerwonym
- 1 dioda RGB ze wspólną anodą
- 1 fotorezystor
- 13 rezystorów 330Ω
- 1 rezystor 10kΩ
- przewody
- płytka stykowa

Program napisano w programie STM32CubeIDE, w języku C z pomocą biblioteki ,,hardware abstraction layer" (HAL). Kolejnym pomocnym dodatkiem w tworzeniu aplikacji był STMCubeMX, z pomocą którego łatwo wygenerowałem kod inicjalizujący wybrane przeze mnie funkcjonalności takie jak timer, przetwornik ADC, GPIO oraz zegary systemowe.



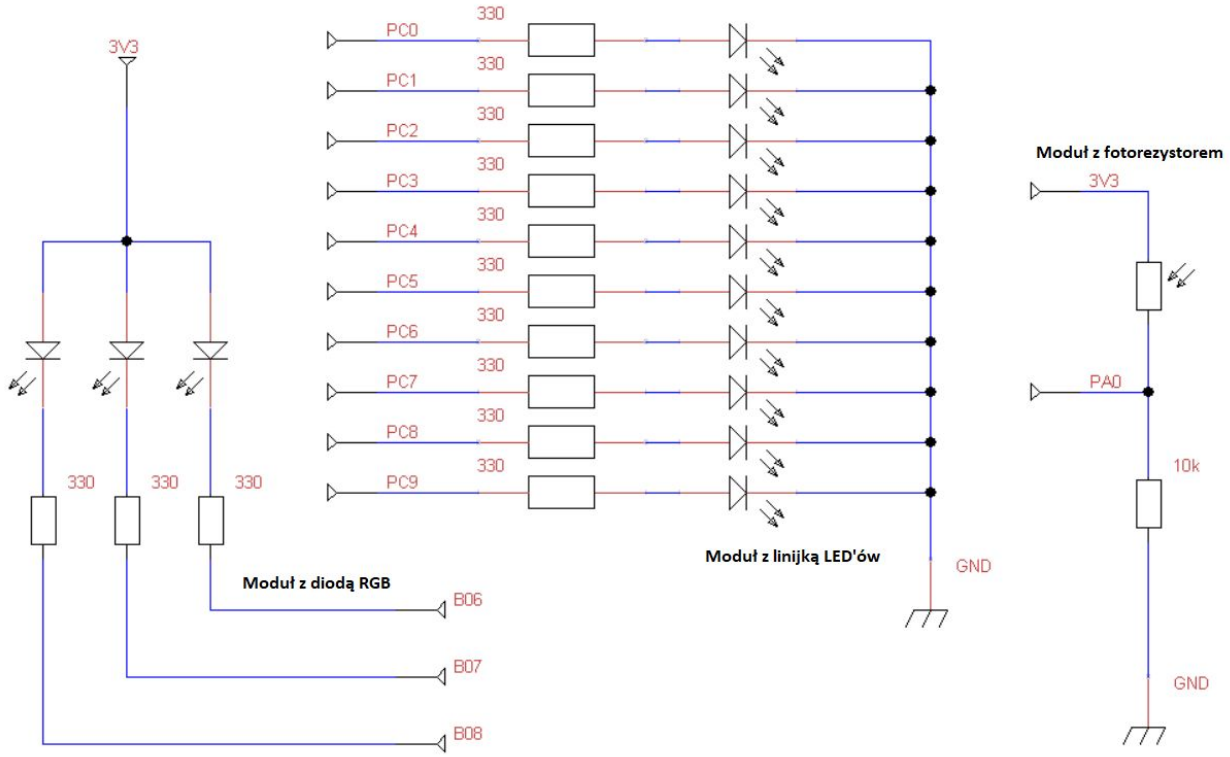
Rys.1. Interface dodatku STMCubeMX

Można tutaj także w prosty sposób dobrać częstotliwości pracy interesujących nas peryferiów, zobaczyć pobór prądu, zużycie pamięci lub na przykład przewidywany czas pracy przy zasilaniu baterią.

Napisany przeze mnie kod znajduje się w dwóch plikach:

- main.c , w którym zapisane są funkcje inicjalizujące oraz główna funkcjonalność programu
- stm32f1xx_it.c , w którym zawarte są funkcjonalności związane z obsługą przerwań

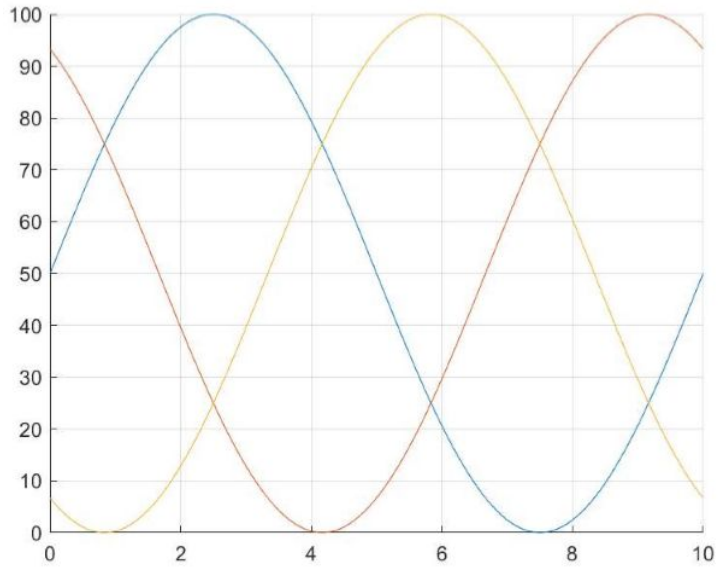
Dodatkowo poniżej załączam podłączenie urządzeń peryferyjnych.



To tyle jeżeli chodzi o ogólną specyfikację projektu. Dalej zajmę się omówieniem poszczególnych problemów wraz z rozwiązaniami oraz elementów kodu programu.

Problemy

Głównym z problemów było znalezienie sposobu na pokazanie działania diody RGB. Wpadłem na rozwiązanie, w którym generujemy trzy przebiegi sinusoidalne z różnymi fazami, każdy odpowiada za dany kolor diody RGB.



Rys.2. Trzy sinusoidy reprezentujące dane kolory w diodzie RGB

Wartość 100 w danym momencie reprezentuje maksymalne natężenie danej barwy, możemy w ten sposób niezależnie zmieniać każdy z trzech kolorów poprzez manipulację wartością fazy. Zjawisko to jest dokładniej przedstawione w dalszej części dokumentacji.

Drugim problemem była nieliniowość diody RGB. Udało się odnaleźć wzór na linearyzację

$$f(x) = \frac{L}{1+e^{-k(x-x_0)}}$$

L - maksymalna wartość

k - współczynnik narastania

x0 - współrzędna punktu zerowego

Omówienie elementów kodu programu

Funkcja inicjalizująca zegary systemowe

```
87
88
89  void SystemClock_Config(void)
90  {
91    RCC_OscInitTypeDef RCC_OscInitStruct = {0};
92    RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
93    RCC_PeriphCLKInitTypeDef PeriphClkInit = {0};
94
95    RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
96    RCC_OscInitStruct.HSIState = RCC_HSI_ON;
97    RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
98    RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
99    RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI_DIV2;
100   RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL2;
101   HAL_RCC_OscConfig(&RCC_OscInitStruct);
102
103   RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
104                                 |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
105   RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
106   RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
107   RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
108   RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
109   HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0);
110
111   PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_ADC;
112   PeriphClkInit.AdcClockSelection = RCC_ADCPCLK2_DIV2;
113   HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit);
114 }
115
```

Powyższy kod został wygenerowany przez dodatek STMCubeMX, po wcześniejszym ustawieniu zegara szyny AHB na częstotliwość 8MHz oraz częstotliwości przetwornika ADC na 4MHz.

Funkcja inicjalizująca GPIO

```
138
139
140 static void MX_GPIO_Init(void)
141 {
142   GPIO_InitTypeDef GPIO_InitStruct = {0};
143
144   __HAL_RCC_GPIOC_CLK_ENABLE();
145   __HAL_RCC_GPIOA_CLK_ENABLE();
146   __HAL_RCC_GPIOB_CLK_ENABLE();
147
148   GPIO_InitStruct.Pin = GPIO_PIN_0|GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3|GPIO_PIN_4|
149             GPIO_PIN_5|GPIO_PIN_6|GPIO_PIN_7|GPIO_PIN_8|GPIO_PIN_9;
150   GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
151   GPIO_InitStruct.Pull = GPIO_NOPULL;
152   GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
153   HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);
154
155   GPIO_InitStruct.Pin = GPIO_PIN_6|GPIO_PIN_7|GPIO_PIN_8;
156   GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
157   GPIO_InitStruct.Pull = GPIO_NOPULL;
158   GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_HIGH;
159   HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
160
161   GPIO_InitStruct.Pin = GPIO_PIN_0;
162   GPIO_InitStruct.Mode = GPIO_MODE_ANALOG;
163   HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
164
165   GPIO_InitStruct.Pin = GPIO_PIN_13;
166   GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
167   GPIO_InitStruct.Pull = GPIO_PULLUP;
168   HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);
169
170   HAL_NVIC_SetPriority(EXTI15_10_IRQn, 0, 0);
171   HAL_NVIC_EnableIRQ(EXTI15_10_IRQn);
172
173 }
174
175
```

Powyższa funkcja inicjalizuje porty generalnego użytku podłączając GPIOA, GPIOB oraz GPIOC do zegara szyny AHB.

PINy GPIOC od 0 do 9 odpowiedzialne są za obsługę linijki LED'ów w trybie wyjściowym PushPull bez rezystora podciągającego, nie zależy nam na szybkości więc zostawiamy niską częstotliwość pracy na tych PIN'ach.

Dalej PINy od 6 do 8 GPIOB odpowiedzialne są za timery, potrzebujemy tutaj dużej częstotliwości oraz pracy w trybie alternatywnym aby korzystać z PWM.

PIN 0 GPIOA jest używany jako wejście analogowe do naszego przetwornika analogowo cyfrowego.

Jako ostatni mamy PIN 13 GPIOC, który de facto jest wbudowanym w płytkę przyciskiem. Narastające zbocze sygnału po jego naciśnięciu będzie aktywowało przerwanie. Oczywiście do poprawnego działania przycisku potrzebujemy jeszcze aktywować PULLUP.

Funkcja inicjalizująca przetwornik ADC

```
109
110
111 static void MX_ADC1_Init(void)
112 {
113
114   ADC_ChannelConfTypeDef sConfig = {0};
115
116   hadc1.Instance = ADC1;
117   hadc1.Init.ScanConvMode = ADC_SCAN_DISABLE;
118   hadc1.Init.ContinuousConvMode = ENABLE;
119   hadc1.Init.DiscontinuousConvMode = DISABLE;
120   hadc1.Init.ExternalTrigConv = ADC_SOFTWARE_START;
121   hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
122   hadc1.Init.NbrOfConversion = 1;
123   hadc1.Init.NbrOfDiscConversion = 1;
124   HAL_ADC_Init(&hadc1);
125
126   sConfig.Channel = ADC_CHANNEL_0;
127   sConfig.Rank = ADC_REGULAR_RANK_1;
128   sConfig.SamplingTime = ADC_SAMPLETIME_1CYCLE_5;
129   HAL_ADC_ConfigChannel(&hadc1, &sConfig);
130
131   HAL_ADC_Start(&hadc1);
132   HAL_ADC_PollForConversion(&hadc1, 20);
133
134 }
135
136
```

Powyższa funkcja inicjalizuje ustawienia przetwornika ADC. Ustawiamy ciągły pomiar z PINu 0 GPIOA, tak jak było to wcześniej omawiane. Reszta to domyślne ustawienia przetwornika.

Warto dodać, że zamontowany na płytce przetwornik jest 12-bitowy a jego wartości znajdują się w granicy 0-4096, jest to ważne przy przeskalowaniu wyniku z LSB.

Funkcja inicjalizująca TIMER

```
175
176 static void MX_TIM4_Init(void)
177 {
178   TIM_OC_InitTypeDef sConfigOC = {0};
179
180   __HAL_RCC_TIM4_CLK_ENABLE();
181   htim4.Instance = TIM4;
182   htim4.Init.Prescaler = 8 - 1;
183   htim4.Init.CounterMode = TIM_COUNTERMODE_UP;
184   htim4.Init.Period = 1000 - 1;
185   htim4.Init.ClockDivision = 0;
186   htim4.Init.RepetitionCounter = 0;
187   htim4.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_ENABLE;
188   HAL_TIM_PWM_Init(&htim4);
189
190   sConfigOC.OCMode = TIM_OCMODE_PWM2;
191   sConfigOC.Pulse = 100;
192   sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
193   sConfigOC.OCNPolarity = TIM_OCNPOLARITY_LOW;
194   sConfigOC.OCFastMode = TIM_OCFAST_ENABLE;
195   sConfigOC.OCIdleState = TIM_OCIDLESTATE_SET;
196   sConfigOC.OCNIdleState = TIM_OCNIDLESTATE_RESET;
197   HAL_TIM_PWM_ConfigChannel(&htim4, &sConfigOC, TIM_CHANNEL_1);
198   HAL_TIM_PWM_ConfigChannel(&htim4, &sConfigOC, TIM_CHANNEL_2);
199   HAL_TIM_PWM_ConfigChannel(&htim4, &sConfigOC, TIM_CHANNEL_3);
200
201   HAL_TIM_PWM_Start(&htim4, TIM_CHANNEL_1);
202   HAL_TIM_PWM_Start(&htim4, TIM_CHANNEL_2);
203   HAL_TIM_PWM_Start(&htim4, TIM_CHANNEL_3);
204
205 }
206
207
```

Wybieramy TIMER4, musimy przeskalować dołączony zegar z 8MHz na 1MHz, w tym celu używamy prescalera, który liczy od zera więc odejmujemy jedynkę. Tak samo należy ustawić period na 1000, aby timer liczył do 1s i następnie zaczynał od nowa. Oczywiście zlicza w górę.

Następnie ustawiamy PWM domyślnymi ustawieniami, gdzie wartość 0 będzie odpowiadała 0% wypełnienia oraz 100 maksymalnemu wypełnieniu.

Po inicjalizacji należy włączyć TIMERY aby zaczęły zliczanie.

Funkcja linearyzująca nieliniowość diody RGB

```
206
207
208 float PWM_Function_Linear(float val)
209 {
210  const float k = (float)0.1;
211  const float x0 = (float)60.0;
212  return (float)(300.0 / (1.0 + exp(-k * (val - x0))));
213 }
214
215
```

Jak wiadomo dioda RGB posiada nieliniową charakterystykę, więc aby efektywnie ją wykorzystać w programie należy odpowiednio ją zlinearyzować. Wzory potrzebne do tego można z łatwością wyszukać w internecie, lub spróbować wyprowadzić.

Funkcja obsługi przerwania

```
76
77
78 void EXTI15_10_IRQHandler(void)
79 {
80
81   if(mode == 0) mode = 1;
82   else if(mode == 1) mode = 2;
83   else mode = 0;
84
85   HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_13);
86
87 }
88
89
```

Powyższy kod zmienia tryb pracy wraz z każdorazowym naciśnięciem przycisku na mikrokontrolerze.

Funkcja main

```
15
16
17  int main(void)
18  {
19    HAL_Init();
20    SystemClock_Config();
21
22    MX_GPIO_Init();
23    MX_ADC1_Init();
24    MX_TIM4_Init();
25
26    uint16_t counter = 0;
27    mode = 0;
28    uint16_t data_LSB;
29
30    while (1)
31    {
32        data_LSB = HAL_ADC_GetValue(&hadc1);
33
34        HAL_GPIO_WritePin(GPIOC, GPIO_PIN_0|GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3|GPIO_PIN_4|
35                  GPIO_PIN_5|GPIO_PIN_6|GPIO_PIN_7|GPIO_PIN_8|GPIO_PIN_9, GPIO_PIN_RESET);
36
37        if(mode == 1){
38            float r = (float)(50 * sin(counter / 100.0) + 50);
39            float g = (float)(50 * sin(counter / 100.0 - 1 ) + 50);
40            float b = (float)(50 * sin(counter / 100.0 - 2 ) + 50);
41
42            __HAL_TIM_SET_COMPARE(&htim4, TIM_CHANNEL_1, PWM_Function_Linear(r));
43            __HAL_TIM_SET_COMPARE(&htim4, TIM_CHANNEL_2, PWM_Function_Linear(g));
44            __HAL_TIM_SET_COMPARE(&htim4, TIM_CHANNEL_3, PWM_Function_Linear(b));
45
46            counter++;
47        }
48        else if(mode == 2){
49            float data_scaled = (float)((data_LSB * 200) / 4096);
50
51            float r = (float)(50 * sin(data_scaled / 100.0) + 50);
52            float g = (float)(50 * sin(data_scaled / 100.0 - 1 ) + 50);
53            float b = (float)(50 * sin(data_scaled / 100.0 - 2 ) + 50);
54
55            __HAL_TIM_SET_COMPARE(&htim4, TIM_CHANNEL_1, PWM_Function_Linear(r));
56            __HAL_TIM_SET_COMPARE(&htim4, TIM_CHANNEL_2, PWM_Function_Linear(g));
57            __HAL_TIM_SET_COMPARE(&htim4, TIM_CHANNEL_3, PWM_Function_Linear(b));
58        }
59        else{
60            __HAL_TIM_SET_COMPARE(&htim4, TIM_CHANNEL_1, 0);
61            __HAL_TIM_SET_COMPARE(&htim4, TIM_CHANNEL_2, 0);
62            __HAL_TIM_SET_COMPARE(&htim4, TIM_CHANNEL_3, 0);
63        }
64
65        if(data_LSB >= 2000) HAL_GPIO_WritePin(GPIOC, GPIO_PIN_0, GPIO_PIN_SET);
66        if(data_LSB >= 2200) HAL_GPIO_WritePin(GPIOC, GPIO_PIN_1, GPIO_PIN_SET);
67        if(data_LSB >= 2400) HAL_GPIO_WritePin(GPIOC, GPIO_PIN_2, GPIO_PIN_SET);
68        if(data_LSB >= 2600) HAL_GPIO_WritePin(GPIOC, GPIO_PIN_3, GPIO_PIN_SET);
69        if(data_LSB >= 2800) HAL_GPIO_WritePin(GPIOC, GPIO_PIN_4, GPIO_PIN_SET);
70        if(data_LSB >= 3000) HAL_GPIO_WritePin(GPIOC, GPIO_PIN_5, GPIO_PIN_SET);
71        if(data_LSB >= 3200) HAL_GPIO_WritePin(GPIOC, GPIO_PIN_6, GPIO_PIN_SET);
72        if(data_LSB >= 3400) HAL_GPIO_WritePin(GPIOC, GPIO_PIN_7, GPIO_PIN_SET);
73        if(data_LSB >= 3600) HAL_GPIO_WritePin(GPIOC, GPIO_PIN_8, GPIO_PIN_SET);
74        if(data_LSB >= 3800) HAL_GPIO_WritePin(GPIOC, GPIO_PIN_9, GPIO_PIN_SET);
75
76        HAL_Delay(20);
77
78    }
79
80  }
81
82
```

Funkcja main to serce naszego programu. Jeszcze przed rozpoczęciem nieskończonej pętli programu inicjalizujemy peryferia, o których była mowa wcześniej oraz zmienne.

Wewnątrz pętli while zaczynamy pobierać dane z przetwornika ADC oraz gasimy wszystkie LED'y. Następnie w zależności od wybranego trybu pracy poprzez naciśnięcie przycisku aktywujemy dane funkcjonalności programu.

- pierwszy i domyślny tryb pracy to świecąca linijka LED'ów, ustawiamy wypełnienie PWM na 0% - dioda RGB się nie świeci.
- drugi tryb pracy zaświeca diodę RGB i powoli zmienia wyświetlany kolor mieszając trzy kolory w różnym stopniu.
- trzeci tryb pracy także zaświeca diodę RGB, ale steruje jej kolorem poprzez wartości natężenia światła zmierzone poprzez fotorezystor.

Na koniec widać kod do sterowania linijką LED'ów oraz wprowadzający opóźnienie, ta część kodu wykonuje się bez względu na aktywny w danym momencie tryb pracy.

Podsumowanie

Projekt został zrealizowany oraz szczegółowo omówiony w dokumentacji, dowód na poprawność jego działania jest załączony jako krótki filmik na platformie YouTube.

Podczas pisania kodu korzystano głównie z książki Donalda Norrisa pt. ,, Programming with STM32 - Getting Started with Nucleo Board and C/C++. W szczególnych przypadkach posiłkowano się także dokumentacją biblioteki HAL, płytki Nucleo F103RB oraz mikroprocesora ARM Cortex M4.