Systemy Wbudowane

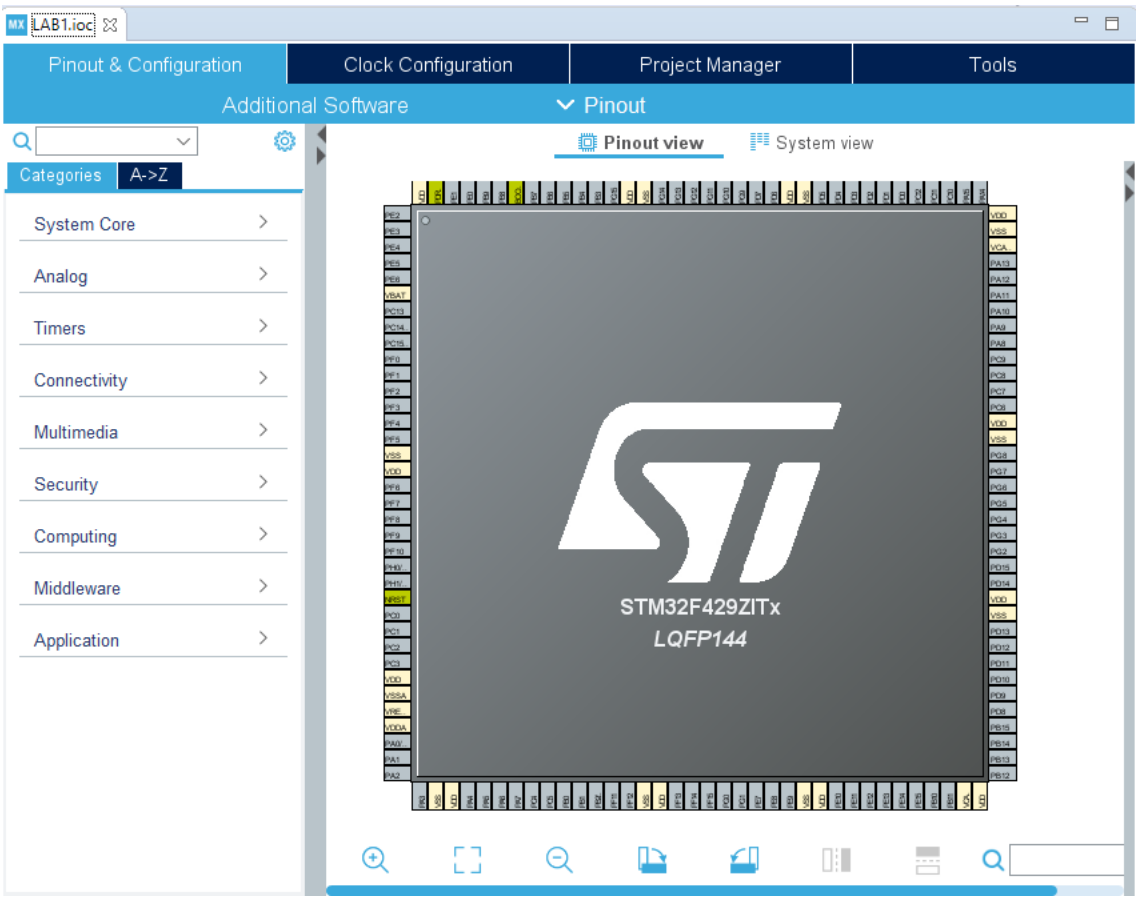
Laboratorium 2:

Konfiguracja peryferiów na przykładzie LCD oraz wykorzystanie przetwornika ADC

2.1 Konfiguracja LCD

2.1.1 Peryferia

Oprócz konfiguracji poszczególnych pinów STM32CubeMX pozwala na konfigurację poszczególnych elementów płytki rozwojowej. Konfiguracja ta automatycznie ustawia odpowiednie rejestry dotyczące peryferiów, a także zaznacza które piny muszą być do tego zarezerwowane (jeżeli takowe istnieją). Umożliwia to panel po lewej stronie, podzielony na odpowiednie kategorie dotyczące peryferiów:

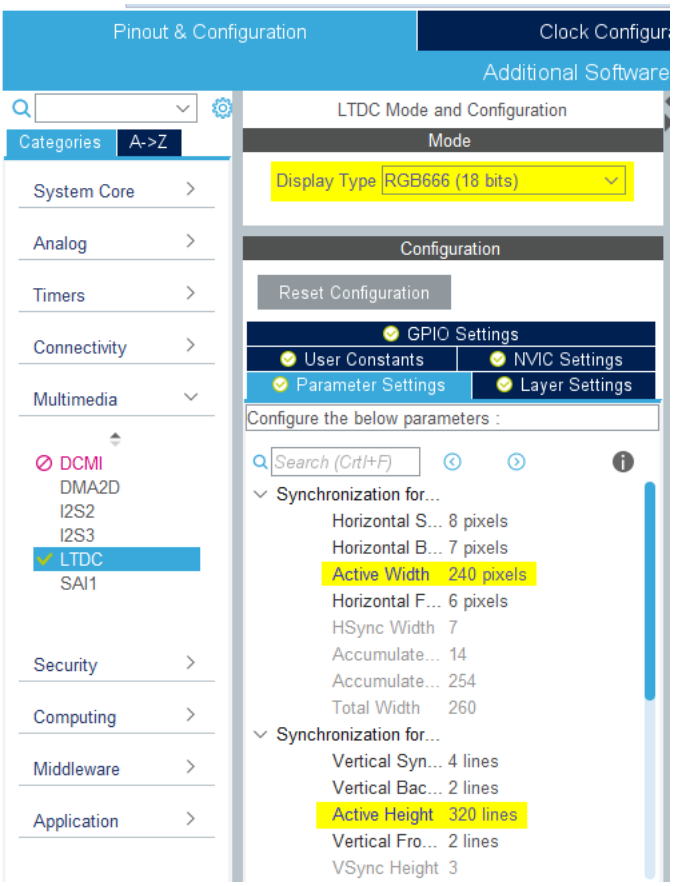


Rysunek 2.1: Widoczny panel konfiguracji peryferiów

2.1.2 LTDC

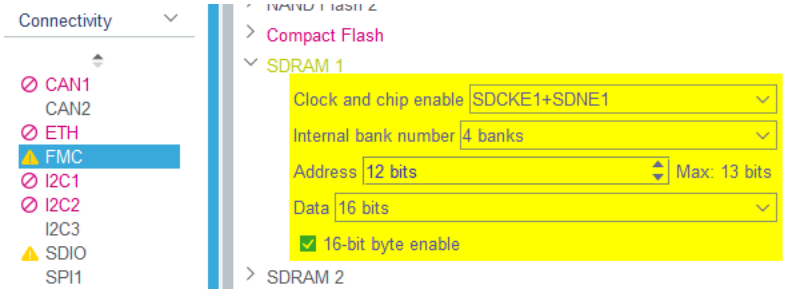
Sposobów obsługi wyświetlania na ekranie wbudowanym w płytkę rozwojową STM32F429ZI jest wiele. Jednym z nich jest wykorzystanie zalecanego przez producenta kontrolera LTDC. LTDC (kontroler ekranu LCD-TFT) to kontroler obrazu zapewniający 24 bitowe równoległe połączenie RGB do rozdzielczości maksymalnej 1024x768 i maksymalnej częstotliwości 83 MHz. Interfejs ten można też wykorzystać do sterowania obrazem bez wbudowanego kontrolera (ekran znajdujący się na płytce rozwojowej STM32F429ZI posiada kontroler ILI9341).

Wyświetlacz z STM32F429ZI zgodnie ze specyfikacją posiada rozdzielczość QVGA (240 x 320) obsługujący 262144 kolorów (RGB666 - 6 bitów na każdą barwę). Należy zatem odpowiednio skonfigurować kontroler LTDC dostępny w zakładce multimedia:



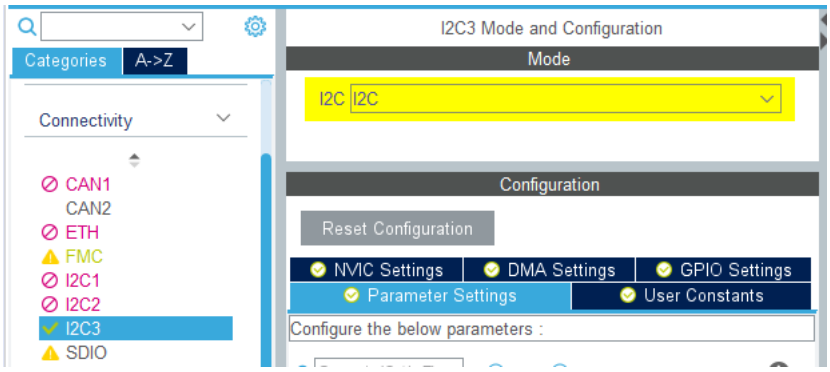
Rysunek 2.2: Konfiguracja LTDC

LTDC potrzebuje pamięci na przechowywane ramki obrazów, w tym celu można wykorzystać elastyczny kontroler pamięci FMC (*ang. Flexible Memory Controller*) i aktywować w nim dostęp do pamięci SDRAM:



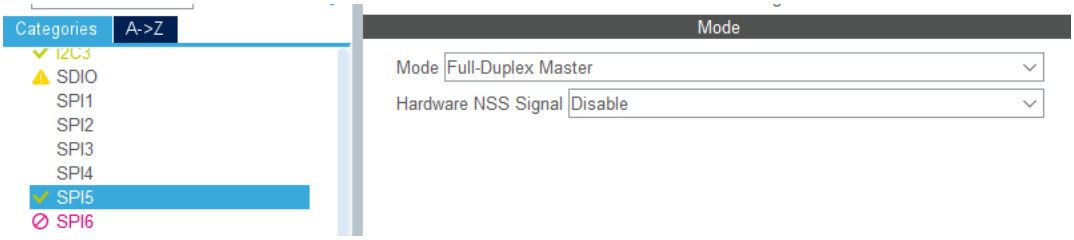
Rysunek 2.3: Konfiguracja FMC

LTDC wymaga także wykorzystania interfejsu I2C (do komunikacji):



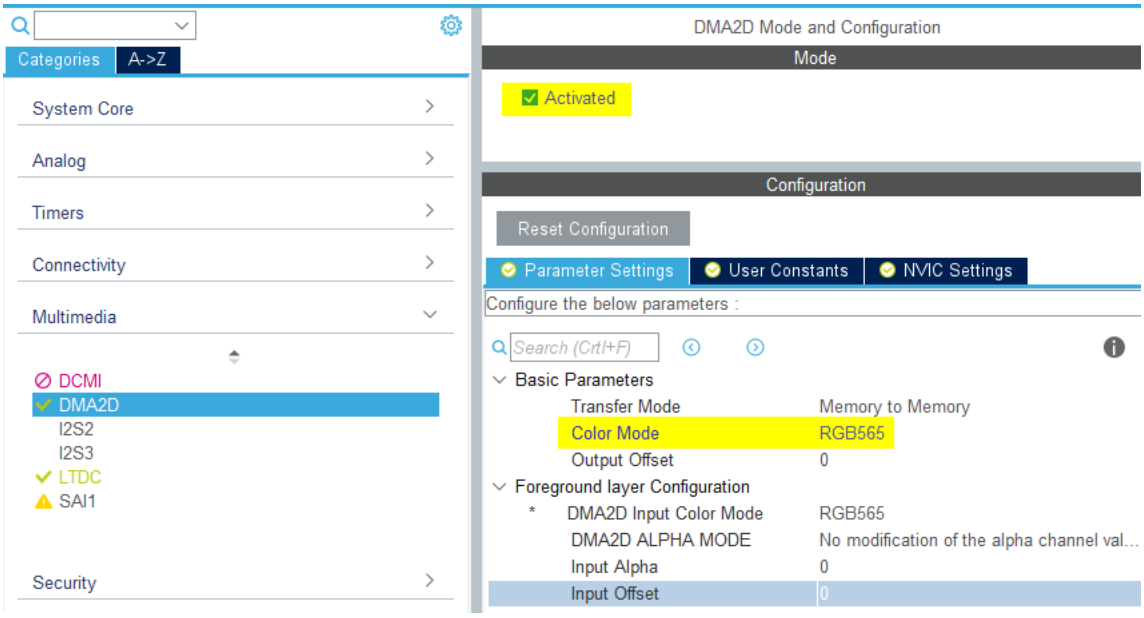
Rysunek 2.4: Konfiguracja I2C

Interfejsu SPI do konfiguracji:



Rysunek 2.5: Konfiguracja SPI

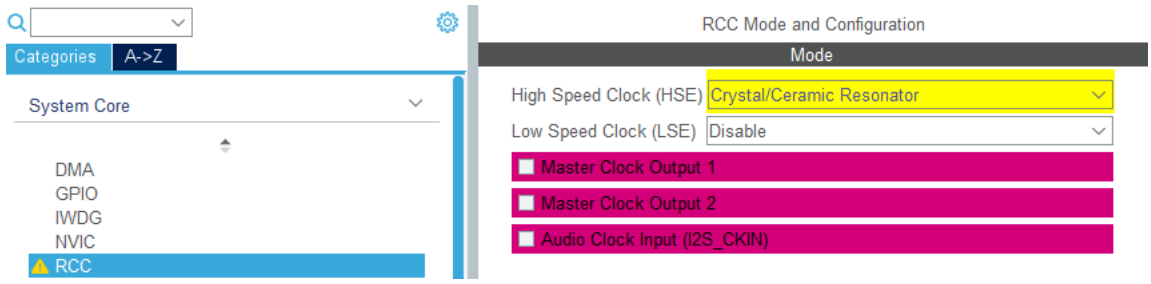
Wykorzystania DMA2D (koprocesor graficzny korzystający z DMA):



Rysunek 2.6: Konfiguracja DMA2D

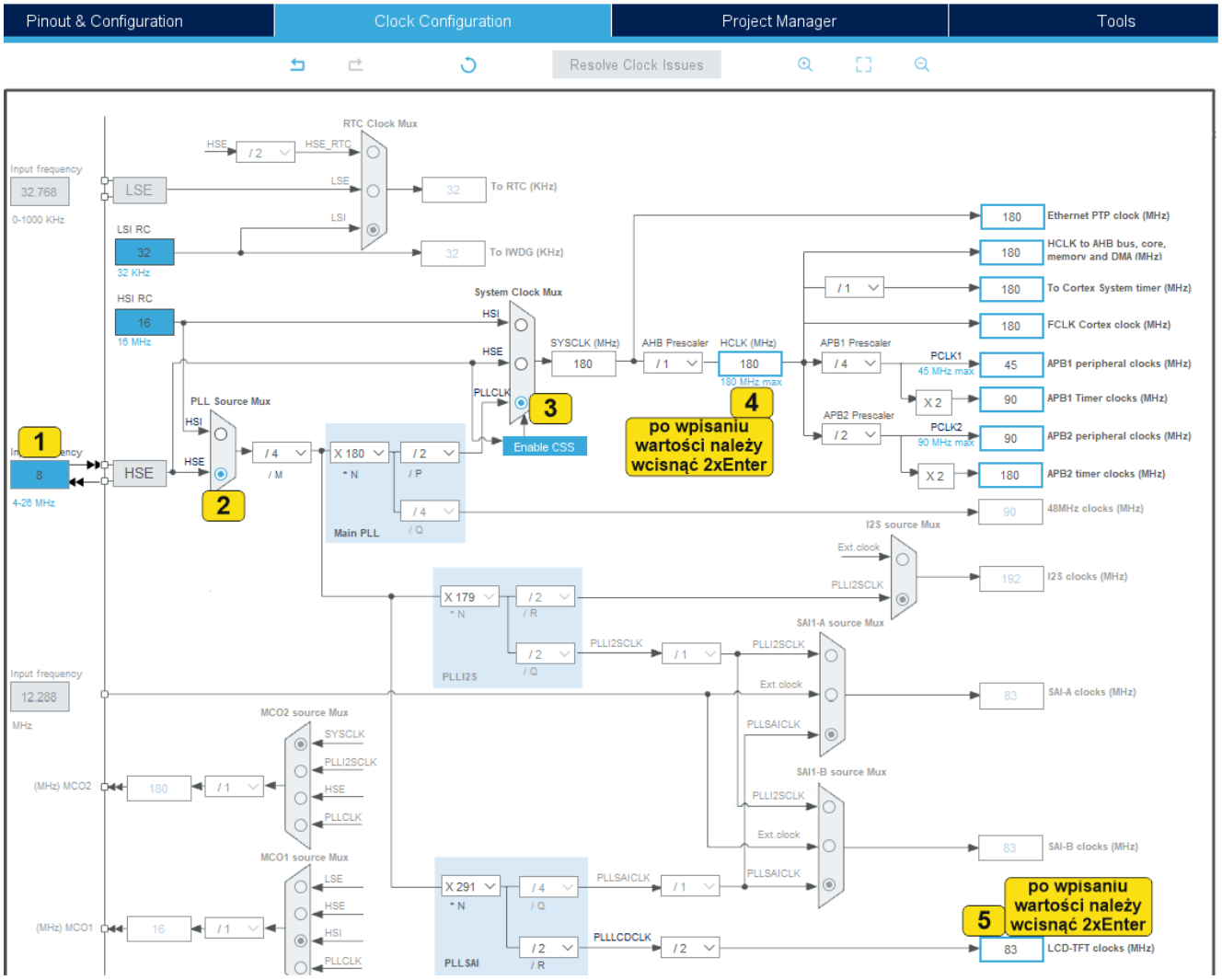
2.1.3 Konfiguracja zegarów

Wykorzystanie LTDC wymaga także zwiększenia taktowania zegarów. Najpierw należy włączyć korzystanie z zegara HSE (*ang. High Speed Clock*):



Rysunek 2.7: Konfiguracja HSE

Następnie należy odpowiednio zwiększyć taktowanie zegarów, służy do tego zakładka *Clock Configuration* (należy wyłączyć anulować automoatyczne rozwiązanie problemów z zegarami). Wrtości częstotliwości i mnożników należy skonfigurować zgodnie z kolejnością przedstawioną poniżej:



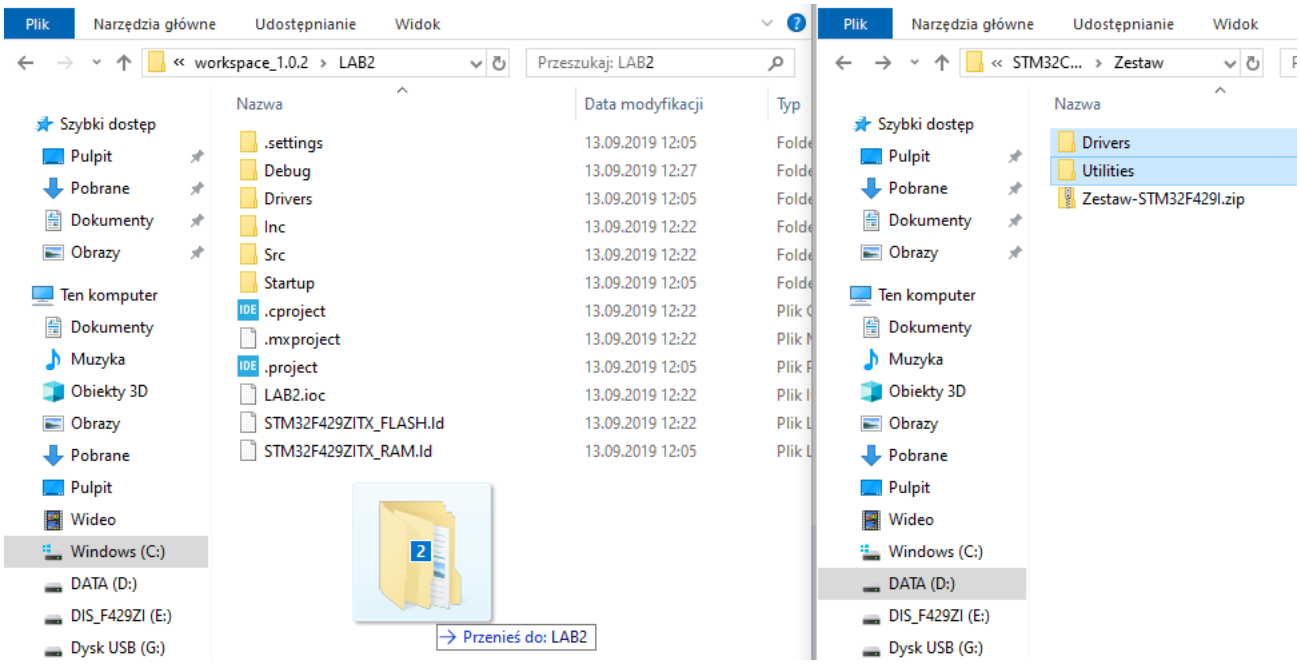
Rysunek 2.8: Konfiguracja zegarów

2.1.4 Biblioteki BSP

Po konfiguracji zegarów można przejść do wygenerowania kodu projektu. Warto zauważyć, że dla każdego konfigurowanego peryferia program wygenerował odpowiednie funkcje:

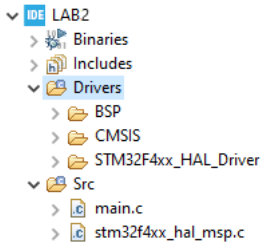
- static void MX_GPIO_Init(void)
- static void MX_DMA2D_Init(void)
- static void MX_FMC_Init(void)
- static void MX_I2C3_Init(void)
- static void MX_LTDC_Init(void)
- static void MX_SPI5_Init(void)

W tej chwili można już rozpocząć pracę z kontrolerem LTDC (opis). Przykład takiego wykorzystania można znaleźć np. tutaj. W ramach zajęć będą jednak wykorzystane dodatkowe biblioteki BSP (*Board Support Package*). Są to biblioteki oficjalnie załączane w pakiecie STM32Cube MCU Package for STM32F4 series. Po pobraniu pakietu można je odnaleźć w folderze *STM32Cube_FW_F4_V1.24.0/Drivers/BSP/STM32F429I-Discovery*. Biblioteki te wymagają również zbioru narzędzi i dodatków (m.in. czcionek) znajdujących się w katalogu *STM32Cube_FW_F4_V1.24.0/Utilities*. Zestaw potrzebnych plików został przygotowany i umieszczony na stronie instytutu (Zestaw-STM32F429I.zip) - wystarczy go pobrać, rozpakować i przenieść odpowiednie pliki do folderu projektu:



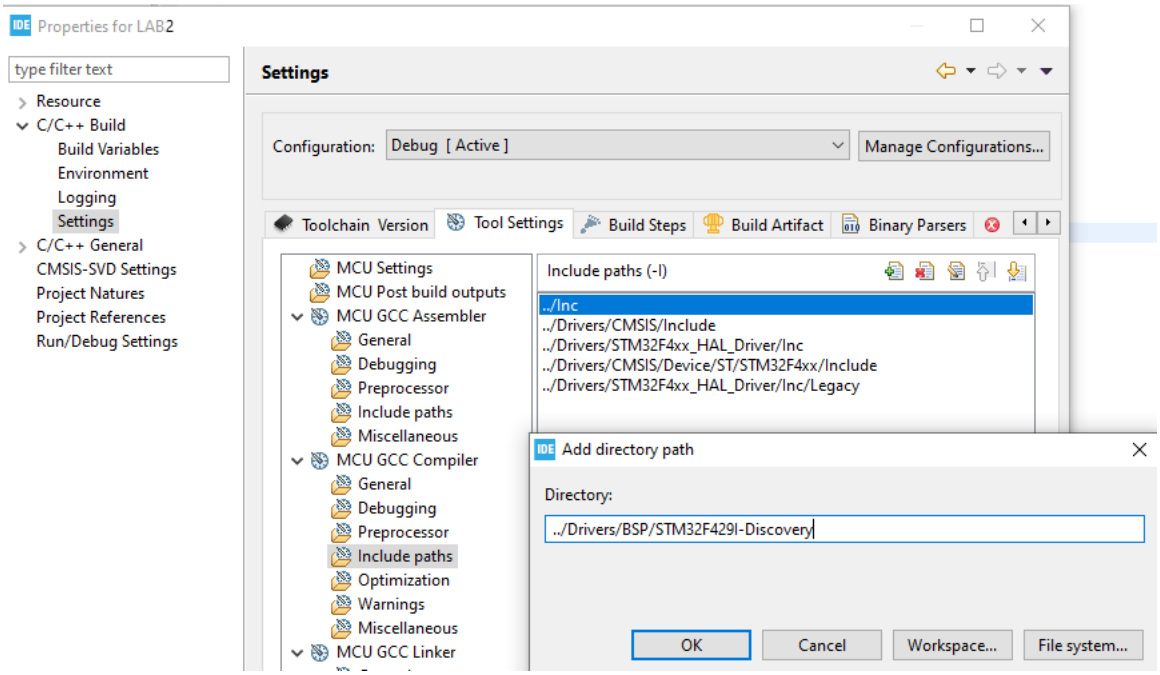
Rysunek 2.9: Kopiowanie bibliotek

Po przekopiowaniu bibliotek i odświeżeniu projektu (F5) powinien pojawić się katalog BSP:



Rysunek 2.10: Katalog BSP

Należy jeszcze odpowiednio skonfigurować kompilator, tak aby widział katalog z bibliotekami (należy pamiętać także o opcjach generujących pliki .hex lub .bin - zakładka *MCU Post build outputs*):



Rysunek 2.11: Dodanie katalogu bibliotek

Po takiej konfiguracji można już korzystać z gotowych bibliotek, wystarczy w sekcji kodu pliku main.c oznaczonej *USER CODE BEGIN Includes* dodać następujący wpis:
#include "stm32f429i_discovery_lcd.h"

Jeżeli wszystko zostało skonfigurowane pomyślnie można przetestować działanie przykładowego programu:

Rysunek 2.12: "hello world!" wyświetlone na LCD

Najważniejsze funkcje biblioteki BSP_LCD:

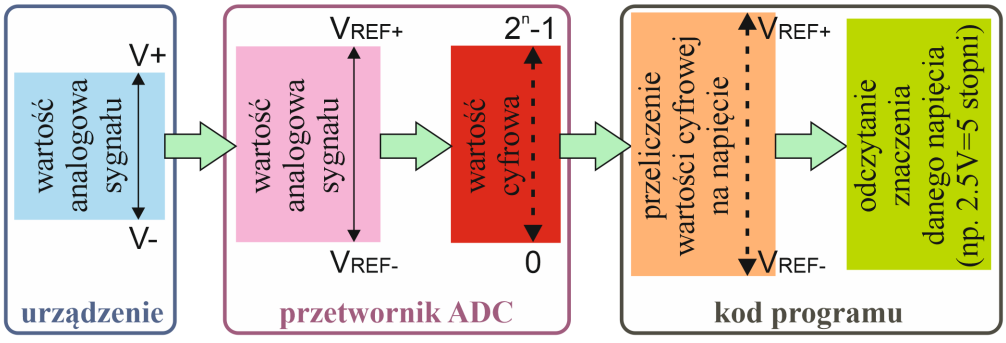
- *BSP_LCD_Init();* - Inicjalizacja ekranu LCD
- *BSP_LCD_LayerDefaultInit(LAYER, BUFFER);*
  LAYER przyjmuje odpowiednio wartości LCD_FOREGROUND_LAYER lub LCD_BACKGROUND_LAYER, buffer domyślnie powinien mieć wartość LCD_FRAME_BUFFER
- *BSP_LCD_SelectLayer(LAYER);* - Wybór aktywnej warstwy
- *BSP_LCD_SetBackColor(COLOR);*
  Ustawienie koloru obramowania i tła tekstu, COLOR przyjmuje zdefiniowane wartości LCD_COLOR_X (gdzie X jest nazwą koloru, np. RED, BLUE, BLACK, itd.) lub dowolne wartości postaci 0xAARRGGBB (AA oznacza szesnastkową wartość przeźroczystości, RR wartość odcienia czerwonego, GG wartość odcenia zielonego, BB wartość odcienia niebieskiego - przykład: 0xFFFF00AA)
- *BSP_LCD_SetTextColor(COLOR);* - Wybór koloru tekstu i wypełnienia
- *BSP_LCD_DisplayStringAtLine(LINIA, TEKST);*
  Wypisanie tekstu (typu byte[] / uint8_t[]) w danej linii ekranu (ekran dzielony jest automatycznie według ustawień czcionki)
- *BSP_LCD_SetFont(&FONT);*
  Wybór czcionki FONT, domyślne czcionki z katalogu Utilities to: &Font24 &Font20 &Font16 &Font12 &Font8. Warto podejrzeć jak wyglądają pliki z czionkami - np. plik Utilities/Fonts/font8.c
- *BSP_LCD_DrawCircle(X, Y, PROMIEN);* - Narysowanie okręgu*
- *BSP_LCD_FillCircle(X, Y, PROMIEN);* - Wypełnienie okręgu*
- *BSP_LCD_DrawRect(X, Y, W, H);* - Narysowanie prostokąta*
- *BSP_LCD_FillRect(X, Y, W, H);* - Wypełnienie prostokąta*
- *BSP_LCD_DrawPixel(X, Y, COLOR);* - Narysowanie piksela

* - kształty nie mogą wychodzić rozmiarem poza obszar ekranu (bufora)

2.2 Przetwornik ADC

2.2.1 Działanie przetworników

Przetworniki analogowo-cyfrowe (ADC - *ang. Analog to Digital*) to jak sama nazwa wskazuje układy zamieniające sygnał analogowy na sygnał cyrfowy. Przez sygnał analogowy rozumie się sygnał o zmiennym napięciu w zakresie od $V-$ do $V+$. W uproszczeniu, jeżeli na danym pinie mikrokontrolera podpięty będzie sygnał analogowy (pochodzący z dowolnego urządzenia) to przetwornik zamieni je na wartości cyfrowe całkowite z zakresu od 0 do $2^n - 1$ (gdzie n jest rozdzielczością przetwornika). A zatem przetwornik o rozdzielczości $n = 8$ pozwoli uzyskać wartości od 0 do 255. Odczytane wartości całkowite należy odpowiednio przekształcić, najczęściej opierając się na specyfikacji dostarczonej przez producenta danego elementu generującego sygnał analogowy. Przykład działania przetwornika ADC (w uproszczeniu):



Rysunek 2.13: Uproszczony schemat działania przetwornika ADC

2.2.2 Alternatywa bibliotek HAL

Wykorzystane w poprzednich ćwiczeniach metody komunikacji z mikrokontrolerem bazowały najczęściej na wykorzystaniu biblioteki HAL (*ang. Hardware Abstraction Layer*). Celem powstania bibliotek HAL było ułatwienie tworzenia aplikacji oraz konfiguracji bloków peryferyjnych mikrokontrolerów. Korzystanie z bibliotek HAL ma jednak następujące wady:

- Zmniejsza kontrolę nad kodem oraz elementami układu
- Zmniejsza wydajność mikrokontrolera
- W pewnym stopniu ogranicza możliwości oferowane przez mikrokontroler

Powyższe wady powodują że biblioteki HAL powinny być wykorzystywane jedynie do szybkiego prototypowania lub w sytuacjach w których nie przeszkadza gorsza wydajność działania układu.

Jak zatem w inny sposób oprogramować mikrokontroler? Można to robić ustawiając i odczytując odpowiednie wartości rejestrów. Takie zmiany automatycznie konfigurują dany element mikrokontrolera, pozwalając odczytać zapisaną wcześniej konfigurację lub odczytać wartość liczbową wynikającą z pracy układu (np. wartość przekonwertowaną przez przetwornik ADC).

Rejestry

Definicje adresów poszczególnych rejestrów można znaleźć w pliku *stm32f[NR]xx.h* (np. *stm32f429xx.h* dla płytki STM32F429ZI). Adresy rejestrów są zdefiniowane albo w sposób bezpośredni, albo przez odpowiednie struktury:

```
// ...
#define APB1PERIPH_BASE       PERIPH_BASE
#define APB2PERIPH_BASE       (PERIPH_BASE + 0x00010000UL)
#define AHB1PERIPH_BASE       (PERIPH_BASE + 0x00020000UL)
#define AHB2PERIPH_BASE       (PERIPH_BASE + 0x10000000UL)
// ...
typedef struct
{
  __IO uint32_t MODER;    /*!< GPIO port mode register,               Address offset: 0x00      */
  __IO uint32_t OTYPER;   /*!< GPIO port output type register,        Address offset: 0x04      */
  __IO uint32_t OSPEEDR;  /*!< GPIO port output speed register,       Address offset: 0x08      */
  __IO uint32_t PUPDR;    /*!< GPIO port pull-up/pull-down register,  Address offset: 0x0C      */
  __IO uint32_t IDR;      /*!< GPIO port input data register,         Address offset: 0x10      */
  __IO uint32_t ODR;      /*!< GPIO port output data register,        Address offset: 0x14      */
  __IO uint32_t BSRR;     /*!< GPIO port bit set/reset register,      Address offset: 0x18      */
  __IO uint32_t LCKR;     /*!< GPIO port configuration lock register, Address offset: 0x1C      */
  __IO uint32_t AFR[2];   /*!< GPIO alternate function registers,     Address offset: 0x20-0x24 */
} GPIO_TypeDef;
// ...
#define GPIOA               ((GPIO_TypeDef *) GPIOA_BASE)
#define GPIOB               ((GPIO_TypeDef *) GPIOB_BASE)
#define GPIOC               ((GPIO_TypeDef *) GPIOC_BASE)
// ...
#define GPIOA_BASE            (AHB1PERIPH_BASE + 0x0000UL)
#define GPIOB_BASE            (AHB1PERIPH_BASE + 0x0400UL)
#define GPIOC_BASE            (AHB1PERIPH_BASE + 0x0800UL)
// ...
```

Rysunek 2.14: Wybrane fragmenty kodu pliku stm32f429xx.h

Szczegółowe dane o rejestrach znajdują się w instrukcjach *"Reference Manual"* dostępnych dla każdej z płytki STM32 (np. dla STM32F429ZI). Ważne są także pliki *"Datasheet"* zawierające szczegółowe specyfikacje poszczególnych płytek (np. dla STM32F429ZI).

Chcąc np. ustawić odpowiedni pin z portu A w tryb analogowy należy ustawić odpowiednie bity rejestru *GPIOA->MODER*. Aby port działał należy jeszcze wykonać szereg innych czynności - m.in. uaktywnić zegar dla danego portu. Warto zauważyć że metoda *MX_GPIO_Init();* włącza automatycznie zegary dla wybranych portów wykorzystując bibliotekę HAL - np. *__HAL_RCC_GPIOF_CLK_ENABLE()*.

Operacje na rejestrach

Zapis bitu 1 do rejestru:	REJESTR |= (0b1 << POZYCJA)
Zapis bitu 0 do rejestru:	REJESTR &= ~(0b1 << POZYCJA)
Odwrócenie bitu w rejestrze:	REJESTR ^= (0b1 << POZYCJA)
Odczyt bitu z rejestru:	REJESTR & (0b1 << POZYCJA)

POZYCJA oznacza przesunięcie bitowej jedynki (zapisywanej jako 0b1 lub 0x1 lub 1U) w lewo o konkretną liczbę miejsc. Chcąc ustawić szósty bit należy zatem zapisać 0b1 << 6 lub bezpośrednio podać wartość bitu 0b01000000.

Często zachodzi też potrzeba zapisania kilku bitów jednocześnie. Można to zrobić najpierw wygaszając odpowiednie bity, a następnie ustawiając w ich miejscu konkretne wartości:

Wygaszenie wielu bitów:	REJESTR &= ~MASKA
Ustawienie wielu bitów:	REJESTR |= (BITY << POZYCJA)

MASKA oznacza maską ze zdefiniowanym miejscem do wyzerowania bitów, np. wykorzystanie maski 0b00011000 oznacza wyzerowanie 3 i 4 bitu (bity maski są odwracane), BITY oznaczają bity do ustawienia (np. 0b101). Wartości bitów można również podawać w formie szesnastkowej, np. 0x1F (co odpowiada 0b00011111).
Wygaszenie oraz ustawienie bitów można również połączyć w jedną linię kodu:

REJESTR = (REJESTR & ~MASKA) | (BITY << POZYCJA)

Dla przykładu przyjrzymy się rejestrom portu GPIO (strona 281 z Reference Manual dla STM32F429ZI):

8.4.1 GPIO port mode register (GPIOx_MODER) (x = A..I/J/K)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MODER15[1:0]		MODER14[1:0]		MODER13[1:0]		MODER12[1:0]		MODER11[1:0]		MODER10[1:0]		MODER9[1:0]		MODER8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MODER7[1:0]		MODER6[1:0]		MODER5[1:0]		MODER4[1:0]		MODER3[1:0]		MODER2[1:0]		MODER1[1:0]		MODER0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 2y:2y+1 **MODERy[1:0]:** Port x configuration bits (y = 0..15)
These bits are written by software to configure the I/O direction mode.
00: Input (reset state)
01: General purpose output mode
10: Alternate function mode
11: Analog mode

Rysunek 2.15: Rejestr pozwalający ustawić kierunek działania pinów - źródło: Reference Manual dla STM32F429ZI

Spróbujmy zatem skonfigurować pin 6 (MODER6 - bit 12 i 13) jako wejście analogowe (11: Analog mode):

```
GPIOA->MODER |= 0b1 << 12; // ustawienie 12-tego bitu na wartość 1
GPIOA->MODER |= 0b1 << 13; // ustawienie 13-tego bitu na wartość 1
```

Biblioteki STM przynoszą jednak pewne usprawnienia i wartości posiadają zdefiniowane maski i pozycje:

GPIO_MODE_ANALOG - wartość trybu analogowego 0x3U (0b11)
GPIO_MODER_MODER6_Pos - pozycja pinu 6-tego w rejestrze MODER
GPIO_MODER_MODER6_Msk - maska pinu 6-tego w rejestrze MODER

Najeżdżając kursorem w programie STM32CubeIDE można również podejrzeć zdefiniowane wartości. Zapis grupy bitów zawierających zera i jedynki z wykorzystaniem definicji będzie wyglądał następująco:

```
GPIOA->MODER &= ~GPIO_MODER_MODER6_Msk;
GPIOA->MODER |= (GPIO_MODE_ANALOG << GPIO_MODER_MODER6_Pos);
```

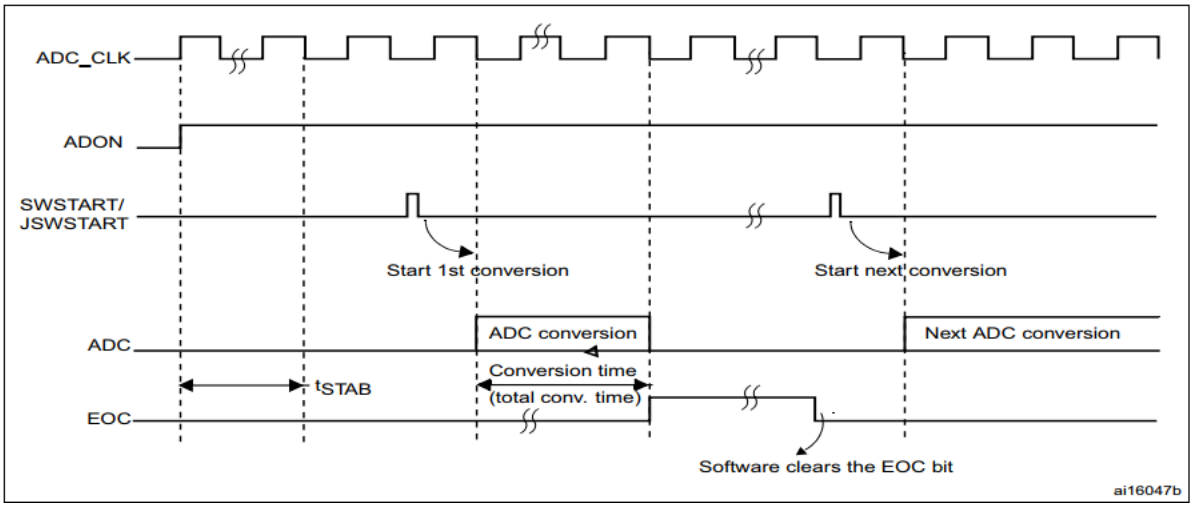
2.2.3 Konfiguracja przetwornika ADC za pomocą rejestrów

Informacje na temat rejestrów związanych z przetwornikami ADC dla płytki STM32F429ZI zaczynają się od strony 388 z Reference Manual dla STM32F429ZI. Najważniejsze informacje o przetworniku to możliwość konfiguracji rozdzielczości trzech przetworników (12, 10, 8 albo 6 bitów), 19 kanałów umożliwiających pomiar z 16 zewnętrzych źródeł (dla każdego przetwornika), możliwość wykorzystania DMA.

Najważniejsze informacje o konfiguracji (na potrzeby najprostszej konfiguracji):

- Włączenie zegara dla odczytu/zapisu do rejestrów przetworników umożliwiają bity RCC_APB2ENR_ADCXEN z rejestru RCC->APB2ENR
- Bit ADON rejestru ADCX->CR2 steruje zasilaniem przetworników
- Bit ADC_CCR_TSVREFE rejestru ADC->CCR pozwala włączyć wbudowany czujnik temperatury podpięty do 18 kanału pierwszego przetwornika
- Bity rejestru ADCX->SQR3 pozwalają na wpisywanie kanałów na których ma być dokonywana konwersja
- Ropoczęcie konwersji ADC rozpoczyna ustawienie bitu SWSTART w rejestrze ADCX->CR2
- Informacja o zakończeniu konwersji znajduje się w bicie ADC_SR_EOC rejestru ADC1->SR (po jej zakończeniu należy ten bit wyzerować)
- Przekonwertowaną wartość można odczytać z rejestru ADC1->DR;

Warto zaznajomić się z całą dokumentacją i możliwymi konfiguracjami przetworników (strony 388-433), a także przyjrzeć się następującemu diagramowi:



Rysunek 2.16: Działanie przetwornika ADC - źródło: Reference Manual dla STM32F429ZI

Najprostsza konfiguracja oraz odczyt wartości z przetwornika może zatem wyglądać następująco:

```
RCC->APB2ENR |= RCC_APB2ENR_ADC1EN; // włączenie zegara ADC1
ADC1->CR2 |= ADC_CR2_ADON; // włączenie zasilania ADC1
ADC->CCR |= ADC_CCR_TSVREFE; // włączenie czujnika temp
ADC1->SQR3 |= 18U << ADC_SQR3_SQ1_Pos; // konfiguracja odczytu z 18 kanału (temperatura)
while(1) // główna pętla programu
{
    ADC1->CR2 |= ADC_CR2_SWSTART; // rozpoczęcie konwersji na ADC1
    while((ADC1->SR & ADC_SR_EOC) == 0) { } // oczekiwanie na koniec konwersji
    uint32_t value = ADC1->DR; // odczytanie wartości z ADC1
    ADC1->SR &= ~ADC_SR_EOC; // wykasowanie bitu EOC
}
```

2.3 Ćwiczenie 1

Celem ćwiczenia jest napisanie programu, który wyświetli na ekranie tekst oraz podstawowe kształty geometryczne. W celu realizacji ćwiczenia należy:

- Stworzyć nowy projekt o nazwie LAB2A dla otrzymanej płytki rozwojowej
- Skonfigurować peryferia płytki zgodnie z tym co opisano w instrukcji w punkcie 2.1
- Wykorzystać bibliotekę BSP i przykładowe komendy przedstawione w instrukcji w punkcie 2.1.4

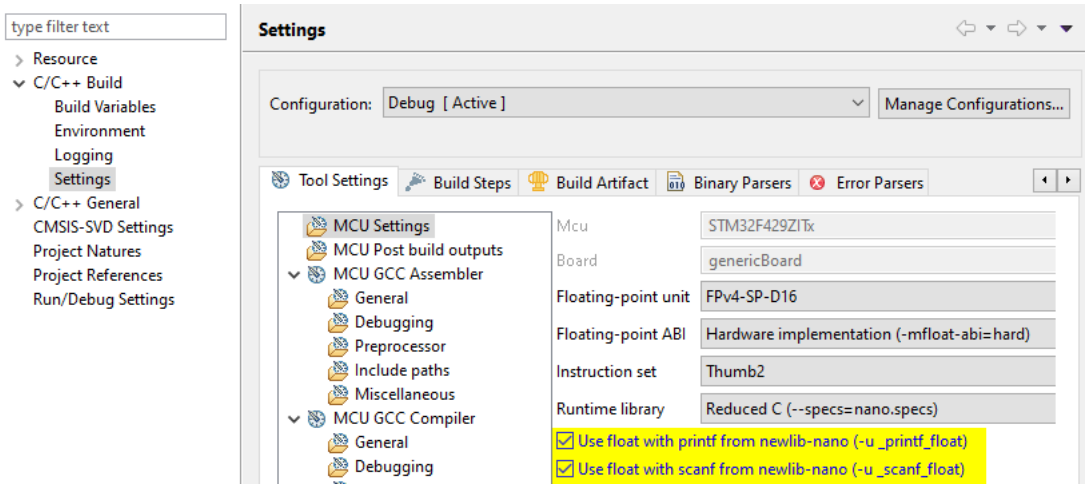
2.4 Ćwiczenie 2

Celem ćwiczenia jest napisanie programu, który wyświetli na ekranie wartość temperatury odczytaną z wbudowanego analogowego czujnika temperatury. W celu realizacji ćwiczenia należy:

- Wykorzystać program utworzony dla Ćwiczenia nr. 1
- Skonfigurować przetwornik ADC zgodnie z przykładem umieszczonym w instrukcji w punkcie 2.2.3 (należy również dołączyć w nagłówku bibliotekę *stm32f429xx.h*)
- Odczytaną wartość temperatury *value* zamienić na wartość napięcia:
  float vval = (3.0f * value) / 4095.0f;
- Wartość napięcia zmienić na wartość temperatury zgodnie ze specyfikacją czujnika:
  $float\ temperature = (vval - V_{25})/Avg\_slope + 25.0f$
  Wartości zmiennych V_{25} oraz Avg_slope można odczytać ze specyfikacji płytki rozwojowej
- Następnie należy przekonwertować wartość liczbową na tekst, aby móc go wyświetlić na ekranie za pomocą metody *BSP_LCD_DisplayStringAtLine(...);*. Służy do tego metoda *sprintf(bufor, tekst, zmienna1, zmienna2, ...)* z biblioteki "stdio.h". Oto przykład jej wykorzystania:

  ```
  char str[100];
  sprintf(str, "T=%0.2fC", temperature);
  BSP_LCD_DisplayStringAtLine(1, (uint8_t*)&str);
  ```

  Do prawidłowego działania funkcji należy też odpowiednio skonfigurować IDE:



Rysunek 2.17: Konfiguracja umożliwiająca użycie sprintf i scanf

- Dodatkowo należy również zaprezentować graficznie odczytaną wartość temperatury - np. za pomocą kwadratu o zmiennym kolorze (od zielonego dla 30 stopni do czerwonego przy 50 stopniach) lub prostokąta którego wysokość zmienia się zależnie od temperatury.

2.5 Ćwiczenie 3

Celem ćwiczenia jest napisanie programu, który wyświetli na ekranie informację o wartości czujnika analogowego. W celu realizacji ćwiczenia należy:

- Wykorzystać program utworzony dla Ćwiczenia nr. 2
- Zgłosić się do prowadzącego po czujnik analogowy
- Odnaleźć odpowiednie piny na płytce i podpiąć czujnik (polecany pin dla wejścia analogowego: PA7)
- Skonfigurować pin z linią analogową jako wyjście analogowe (za pomocą modyfikacji rejestrów lub konfiguratora STM32CubeMX)
- Zmodyfikować odpowiednio kod modyfikacji rejestrów, tak aby zamiast linii 18, odczytywany był sygnał z odpowiedniej linii analogowej (według specyfikacji)
- Wyświetlić w dowolnej postaci (tekstowej lub graficznej) informacje odczytane z czujnika

Dla tego laboratorium należy przygotować sprawozdanie zawierające odpowiednie kody źródłowe oraz zdjęcia przedstawiające działanie programów (dla wszystkich ćwiczeń)