STM32CubeMX v 5.0

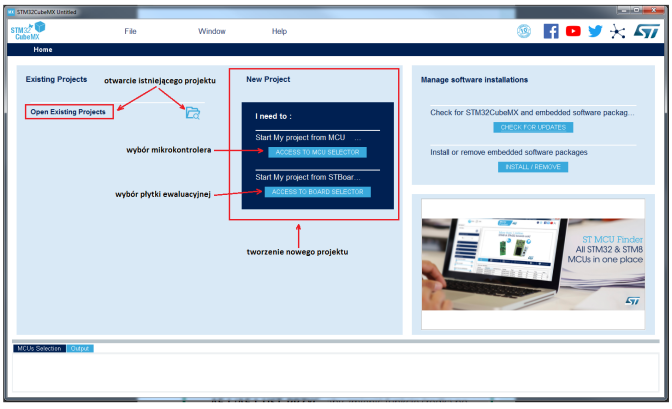
Nowoczesne, skomplikowane i wydajne mikrokontrolery mają wiele wbudowanych bloków funkcjonalnych, od nieskomplikowanych układów GPIO poprzez moduły komunikacyjne UART, SPI, I²C, liczniki/timery, do najbardziej skomplikowanych interfejsów typu USB OTG. Konfigurowanie rozbudowanych peryferii jest zajęciem żmudnym i podatnym na możliwość popełnienia błędów. Dlatego producenci oprogramowania narzędziowego wbudowują w pakiety IDE konfiguratory, które znacznie redukują te niedogodności. Ich graficzne interfejsy pozwalają mocno przyspieszyć proces konfigurowania i znacząco redukują możliwość pomyłki przy zapisywaniu rejestrów konfiguracyjnych.

Program STM32CubeMX to środowisko graficzne dobrze znane programistom wykorzystującym mikrokontrolery STM32, wspierające konfigurację bloków funkcjonalnych i generujące szkielet projektu dla dostępnych na rynku środowisk projektowych IDE. Ostatnio firma ST udostępniła odświeżoną wersję tego pakietu – STM32CubeMX v 5.0.0.

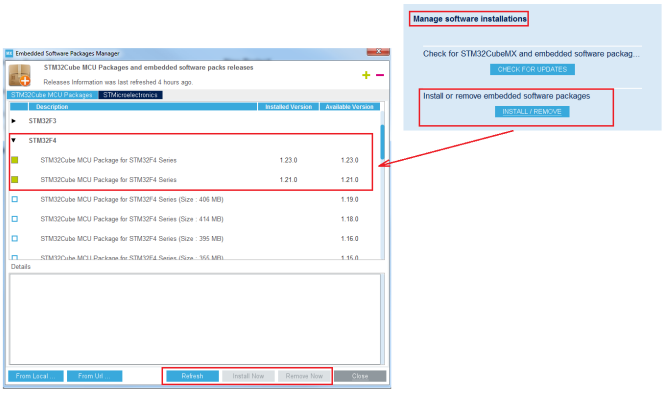
Pierwszą rzeczą, która zwraca uwagę w nowej wersji, jest zmodyfikowane okno startowe. To okno w poprzednich wersjach, delikatnie rzecz ujmując, nie było zbyt atrakcyjne. Teraz to się zmieniło według mojej opinii zdecydowanie na plus – **rysunek 1**.

Po uruchomieniu STM32CubeMX możemy:

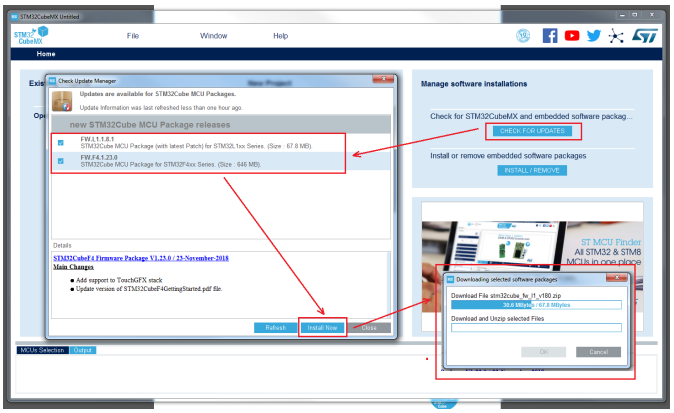
- Otworzyć istniejący projekt.
- Utworzyć nowy projekt z kryterium wyboru mikrokontrolera lub płytki ewaluacyjnej ST.



Rysunek 1. Okno startowe STM32CubeMX V5.0.0



Rysunek 2. Instalowanie paczki z warstwą HAL i middleware



Rysunek 3. Aktualizowanie bibliotek

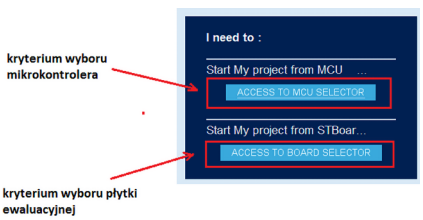
- Pobrać najnowszą wersję platformy programowej dla każdej z rodziny mikrokontrolerów (warstwa HAL plus middleware).
- Zainstalować lub usunąć składniki programowe platformy programowej.

Pierwszą rzeczą, którą można zrobić po zainstalowaniu STM32CubeMX, jest zainstalowanie składników bibliotek właściwych dla każdej z rodzin mikrokontrolerów STM32. Jest to wykonywane po kliknięciu na przycisk „Install/Remove" w oknie *Manage software installations* → *Installor Remove embedded software packages* (**rysunek 2**). Aby mieć pewność, że są wgrywane biblioteki w najnowszej wersji, można przed ich zainstalowaniem wykonać uaktualnienie, jak to zostało pokazane na **rysunku 3**. Konfigurator łączy się z serwerem firmy ST, automatycznie sprawdza najnowszą, dostępną wersję i po kliknięciu na „Install Now" proces uaktualniania wykonuje się automatycznie.

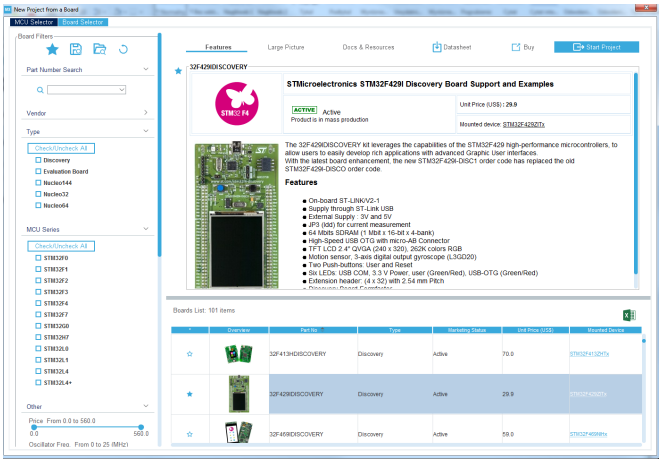
Wybór mikrokontrolera w czasie tworzenia nowego projektu ułatwia selektor działający w oparciu na kryterium doboru konkretnego modelu MCU (opcja Access to MCU Selector) lub płytki ewaluacyjnej (opcja Access To Board Selector), jak na **rysunku 4**.

Po kliknięciu na przykład na opcję „Access To Board Selector" oprogramowanie STM32CubeMX łączy się z serwerem ST i pobiera wykaz wszystkich dostępnych modułów. Dzięki temu mamy do dyspozycji zawsze aktualną listę modułów, łącznie z najnowszymi. Na **rysunku 5** pokazano okno selektora modułów z wybranym modułem 32F429I DISCOVERY. Moduł można wybrać z listy umieszczonej na dole okna. Poszukiwania można zawęzić, używając okna „Board Filters" umieszczonego z lewej strony okna dzięki wybraniu dodatkowych opcji:

- Dostawca (Vendor) – na razie jest możliwy tylko wybór STMicroelectronics.
- Typ – Discovery, Nucleo itp.
- Seria MCU – STM32F0, STM32F1 itp.
- Cena.



Rysunek 4. Kryterium wyboru mikrokontrolera



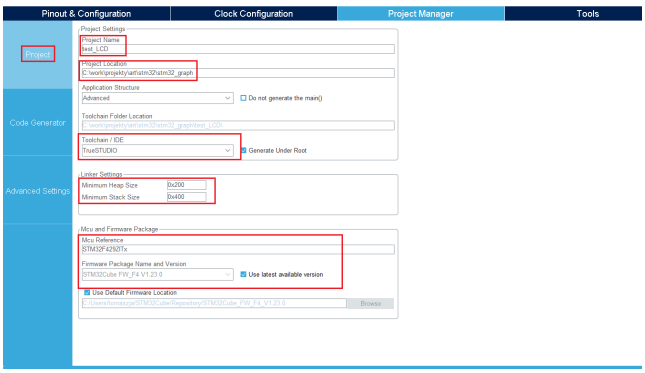
Rysunek 5. Okno selektora modułów z wybranym 32F429IDISCOVERY

- Częstotliwość taktowania mikrokontrolera.
- Niezbędne układy peryferyjne.

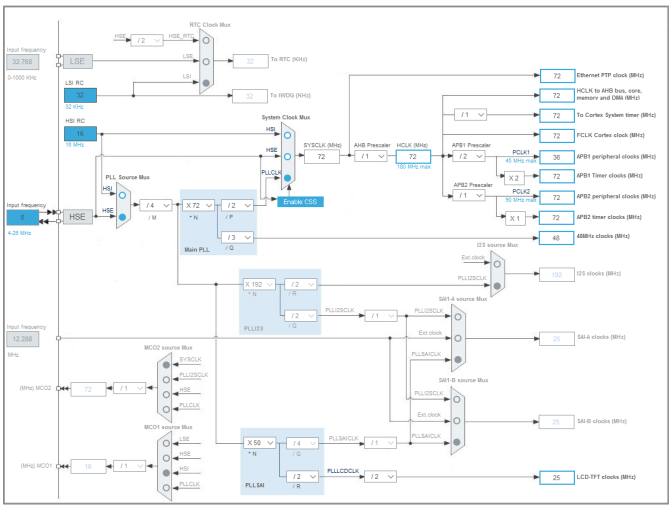
Fotografia wybranego modułu z krótkim opisem i wypunktowanymi właściwościami jest wyświetlana w górnym oknie. Oprócz tego, w tym oknie można znaleźć informację o statusie dostępności, na przykład *Acitve* (produkt w produkcji masowej) i sugerowanej cenie w dolarach amerykańskich.

W górnym oknie można wyświetlić powiększone zdjęcie modułu, uzyskać dostęp do dokumentów, takich jak instrukcja użytkownika lub noty aplikacyjne i zamówić moduł przez Internet. Jeżeli wybrany moduł spełnia wszystkie kryteria, można przejść do kreatora projektu przez kliknięcie na przycisk „Start Project". Do wygenerowania przykładowego projektu wybrałem moduł 32F429IDISCOVERY.

Kreator projektu jest zbudowany z kilku okien wybieranych poprzez zakładki: *Pinout&Configuration*, *Clock Configuration*, *Project Manager*, *Tools*.



Rysunek 6. Okno ustawień projektu



Rysunek 7. Konfigurowanie modułu taktowania mikrokontrolera

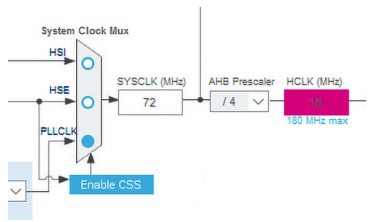
Okno menedżera projektu jest podzielone na trzy kolejne okna: *Project*, *Code Generator*, *Advanced Settings*. W oknie *Project* (**rysunek 6**) użytkownik wybiera nazwę projektu, lokalizację na dysku komputera, IDE, dla którego STM32CubeMX wygeneruje szkielet projektu, ustawienia linkera (rozmiar sterty i stosu) oraz typ wybranego mikrokontrolera i właściwe dla niego biblioteki programowe. Ja wybrałem projekt o nazwie **test_LCD** przeznaczony dla IDE TrueStudio.

Okna *Code Generator* i *Advanced Settings* przy pierwszych testach można pozostawić z ustawieniami domyślnymi.

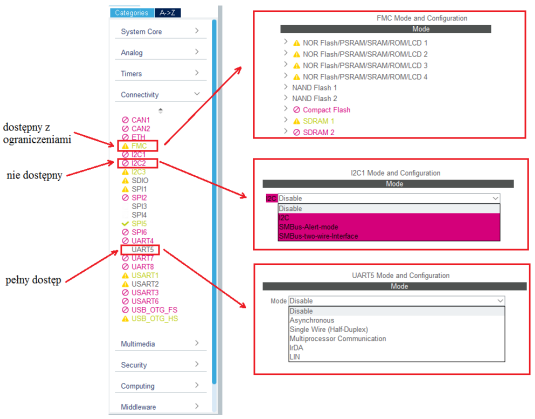
Okno *Clock Configurator* pozwala na szybkie ustawienie układu taktowania rdzenia i układów peryferyjnych mikrokontrolera. Układy z rdzeniem ARM mają dość rozbudowany system taktowania. Użytkownik musi wybrać źródło głównego sygnału zegarowego. Może to być generator HSE stabilizowany zewnętrznym oscylatorem kwarcowym lub wbudowany oscylator RC (HSI RC) o częstotliwości 16 MHz. Sygnał zegarowy z HSE lub HSI RC jest podawany na układ PLL, gdzie jest modyfikowana jego częstotliwość. Dla zastosowanego mikrokontrolera ustawiono taktowanie rdzenia z częstotliwością 72 MHz. Programowany blok taktowania pozwala też na uzyskanie z układów PLL przebiegu o częstotliwości 48 MHz do taktowania bloku interfejsu USB, bloku komunikacyjnego I^2S (na przykład 192 MHz) oraz bloku LTDC sterującego wyświetlaczem LCD TFT. Na **rysunku 7** pokazano okno konfiguracji zegara dla mikrokontrolera modułu 32F429IDISCOVERY. Konfigurator, oprócz graficznej, intuicyjnej możliwości zaprogramowania taktowania wykrywa błędy konfiguracji i podświetla na czerwono nieprawidłowo ustawione wartości (**rysunek 8**).

Okno *Pinout&Configuration* jest zaawansowanym narzędziem konfiguracyjnym pozwalającym na włączanie i konfigurowanie bloków funkcjonalnych, dodawanie funkcji middleware oraz przypisywanie do wyprowadzeń mikrokontrolera funkcji alternatywnych. W lewej części okna jest pokazywana lista z kategoriami bloków funkcjonalnych, middleware i aplikacji. Każdą z kategorii można rozwinąć – wówczas jest wyświetlana lista wykorzystywanych bloków oraz komponentów middleware.

Nazwy bloków funkcjonalnych są oznaczane kolorami: czerwonym, zielonym i czarnym. Kolor czerwony oznacza, że nie można przypisać ich linii do wyprowadzeń mikrokontrolera, bo są już zajęte przez inne peryferie. Nazwa w kolorze zielonym z symbolem żółtego wykrzyknika informuje, że blok może być użyty, ale z ograniczeniami. Jeżeli nazwa bloku funkcyjnego jest wyświetlana bez symbolu i w kolorze czarnym, to można go użyć bez ograniczeń. Na **rysunku 9**

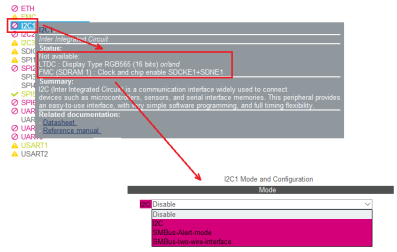


Rysunek 8. Sygnalizacja błędnie ustawionej wartości HCLK



Rysunek 9. Lista bloków funkcjonalnych z zaznaczonym wykrywaniem kolizji przypisania wyprowadzeń alternatywnych

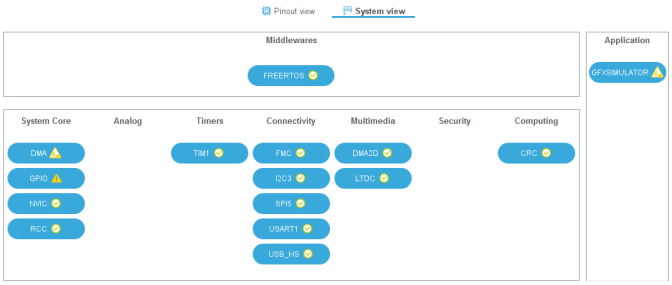
pokazano listę bloków komunikacyjnych (Connectivity) z przykładowymi konfiguracjami dla bloków niedostępnych, dostępnych z ograniczeniami i z pełnym dostępem. Z prawej strony rysunku pokazano opcje konfiguracji dla każdego z tych przypadków. Natychmiastowe wykrywanie kolizji konfiguracji bloków funkcjonalnych jest jedną z moich ulubionych funkcji STM32CubeMX. Przy rozbudowanych mikrokontrolerach wystarczy rzut oka, aby zorientować się, że mogą być problemy z użyciem potrzebnych bloków lub ich użycie w konkretnej konfiguracji w ogóle nie będzie możliwe.



Rysunek 10. Wyświetlenie przyczyny zablokowania możliwości konfiguracji modułu I2C1

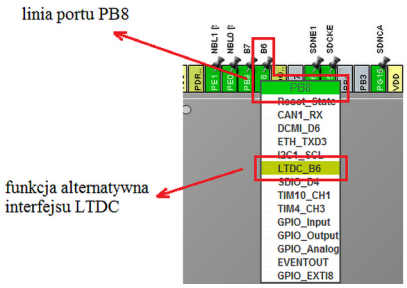
Oprogramowanie STM32CubeMX potrafi podpowiedzieć przyczynę, dla której użycie bloku funkcjonalnego w danej konfiguracji nie jest możliwe lub może być ograniczone. Na **rysunku 10** pokazano, że blok I2C1 nie może być konfigurowany, bo jego wyprowadzenia są zajęte przez już skonfigurowany blok LTDC służący do sterowania wyświetlaczami LCD-TFT oraz alternatywnie przez blok FMC. Jedyna dostępna opcja dla I2C1 to wyłączenie (*Disable*).

W selektorze projektu wybraliśmy moduł ewaluacyjny 32F429IDISCOVERY. Dla wybranej płytki ewaluacyjnej typu 32F429IDISCOVERY oprogramowanie STM32CubeMX domyślnie skonfigurowało kilka bloków funkcjonalnych oraz middleware RTOS (**rysunek 11**). Każdy z bloków funkcjonalnych wymagających połączenia z wyprowadzeniami mikrokontrolera ma takie połączenia definiowane automatycznie. Na **rysunku 12** pokazano fragment okna z rysunkiem obudowy mikrokontrolera z rozwijaną lista dostępnych połączeń

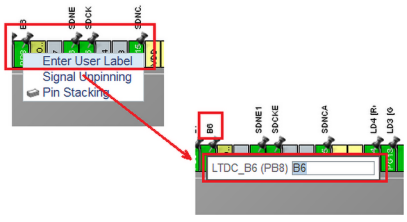


Rysunek 11. Domyślnie skonfigurowane układy peryferyjne dla modułu 32F429IDISCOVERY

alternatywnych dla wyprowadzenia linii portu PB8 (dostępna po kliknięciu prawym przyciskiem myszy na wyprowadzenie), a na **rysunku 13** sposób zmiany przypisanej etykiety.



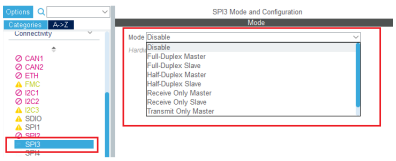
Rysunek 12. Lista z funkcjami alternatywnymi dla wyprowadzenia PB8



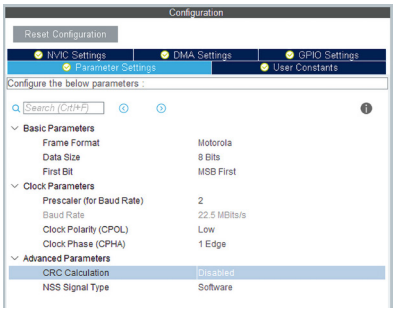
Rysunek 13. Zmiana nazwy etykiety

Sposób konfiguracji nowego bloku funkcjonalnego pokażemy na przykładzie szeregowego interfejsu SPI3. W naszym wypadku jest to układ z dostępem bez ograniczeń (na liście bloków jego nazwa jest wyświetlana w kolorze czarnym). Po kliknięciu jego nazwy na liście jest wyświetlane okno *SPI3 Mode and Configuration*. W rozwijanym oknie *Mode* możemy wybrać tryb pracy interfejsu (**rysunek 14**). Dla przykładu, wybieramy *Full-Duplex Master*. Pod oknem SPI3 zostaje pokazane okno *Configuration*, w którym wykonujemy właściwą konfigurację interfejsu. W oknie konfiguracji jest kilka zakładek. Pierwsza to *Parameter settings*. Ustawiamy tu podstawowe parametry, a więc format ramki (Motorola/TI), długość słowa danych (8/16 bitów) i dosunięcie bitów w ramce danych (LSB/MSB). W następnej kolejności można zaprogramować prędkość transmisji przez ustawienie preskalera (od 2 do 256) oraz polaryzację CPOL i fazę CPHA zegara. Okno ustawień parametrów pokazano na **rysunku 15**. W oknie *GPIO Settings* są zdefiniowane przypisania linii portów mikrokontrolera do linii interfejsu SPI3. STM32CubeMX automatycznie przypisuje te linie do konfiguracji domyślnej (**rysunek 16**).

Jeżeli w zakładce *NVIC Settings* zaznaczymy pole *Enabled*, to zostanie odblokowane globalne przerwanie zgłaszane przez SPI3. W zakładce *DMA Settings* można zdefiniować pracę kanału DMA. Po kliknięciu na przycisk „Add” jest dodawany kanał DMA z dostępem *SPI3_RX* (dane odebrane przez SPI3) lub *SPI3_TX* (dane wysyłane przez SPI3). W okienku *DMA Request Settings* wybiera się tryb *Normal* lub *Circural* oraz inkrementację adresu. Można też zaznaczyć użycie bufora FIFO. Przykładową konfigurację kanału DMA dla interfejsu szeregowego SPI3 pokazano na **rysunku 17**. Jak widać, skonfigurowanie szeregowego interfejsu komunikacyjnego nie jest specjalnie trudne ani pracochłonne.

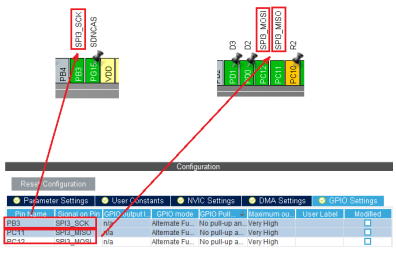


Rysunek 14. Wybranie trybu pracy interfejsu SPI3

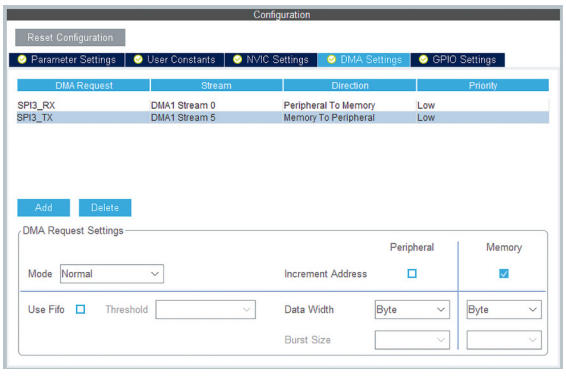


Rysunek 15. Okno ustawień parametrów interfejsu SPI3

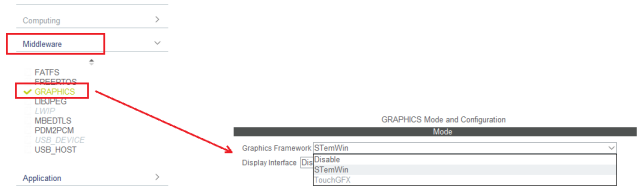
Dla bardziej zaawansowanych użytkowników istotna będzie możliwość dodawania funkcji warstwy middleware. W projekcie z wybranym modułem 32F429IDISCOVERY można dodawać i konfigurować: system plików FATFS, system czasu rzeczywistego FreeRTOS, zaawansowaną bibliotekę graficzną Graphics, bibliotekę



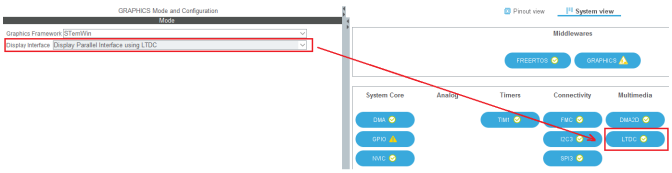
Rysunek 16. Przypisanie sygnałów interfejsu SPI3 do wyprowadzeń mikrokontrolera



Rysunek 17. Konfigurowanie kanału DMA dla interfejsu SPI3



Rysunek 18. Wybranie dostawcy biblioteki graficznej Middleware Graphics



Rysunek 19. Wybór interfejsu LTDC

kryptograficzną MBEDTLS, bibliotekę PDM2PCM do konwersji strumienia danych PDM z mikrofonu MEMS na strumień danych PCM, obsługę interfejsu USB_HOST.

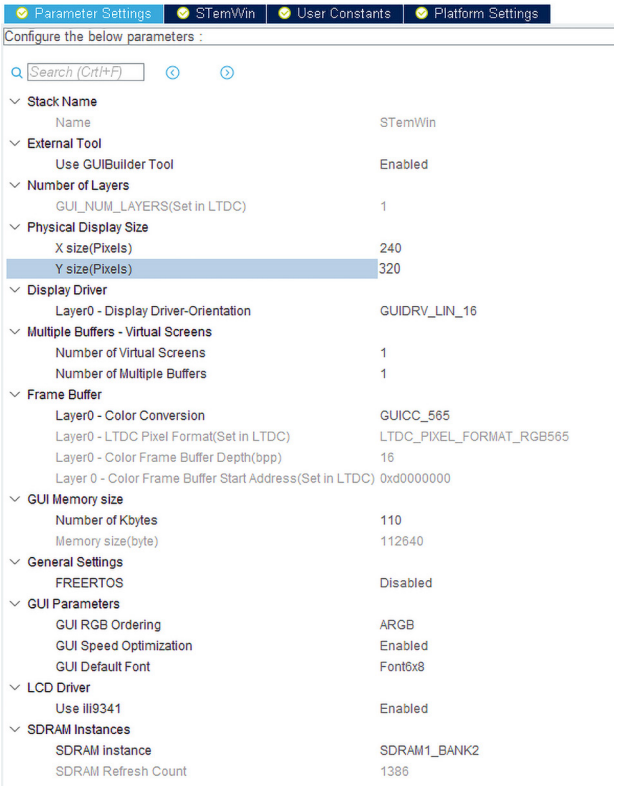
Dodawanie i konfigurowanie middleware pokażemy na przykładzie biblioteki graficznej Graphics. Wybieramy ją z listy middleware i w oknie *Graphics Mode and Configurations* wskazujemy jedne z dostępnych bibliotek i środowisk narzędziowych: STemWin bezpłatną przygotowaną dla mikrokontrolerów STM32 przez firmę Segger lub również bezpłatną TouchGFX (**rysunek 18**). Każda z tych opcji oprócz bibliotek dostarcza wygodne graficzne środowisko projektowe służące do projektowania ekranów z umieszczonymi na nich widżetami, tekstem itp.

W kolejnym kroku trzeba wybrać interfejs komunikacyjny ze sterownikiem panelu LCD-TFT. Możliwy do wybrania interfejs LTDC został dodany i domyślnie skonfigurowany przez STM32CubeMX po wybraniu modułu 32F429IDISCOVERY, który ma zamontowany wyświetlacz LCD TFT ze sterownikiem ILI9341. Oprogramowanie STM32CubeMX domyślnie konfiguruje również interfejs LTDC (**rysunek 20**).

Po wybraniu STemWin możemy przystąpić do konfigurowania biblioteki w oknie *Configuration*. Mamy tu do dyspozycji cztery zakładki: *Parameter Settings*, *STemWin*, *User Constans* i *Platform Settings*.

Pierwsza zakładka *Parameter Settings* (**rysunek 21**) umożliwia:

- Włączenie lub wyłączenie użycia graficznego narzędzia GUIBuilder, przeznaczonego do projektowania ekranów interfejsu użytkownika z użyciem widżetów.
- Zmianę rozdzielczości użytego wyświetlacza (domyślnie x=240, y=320),

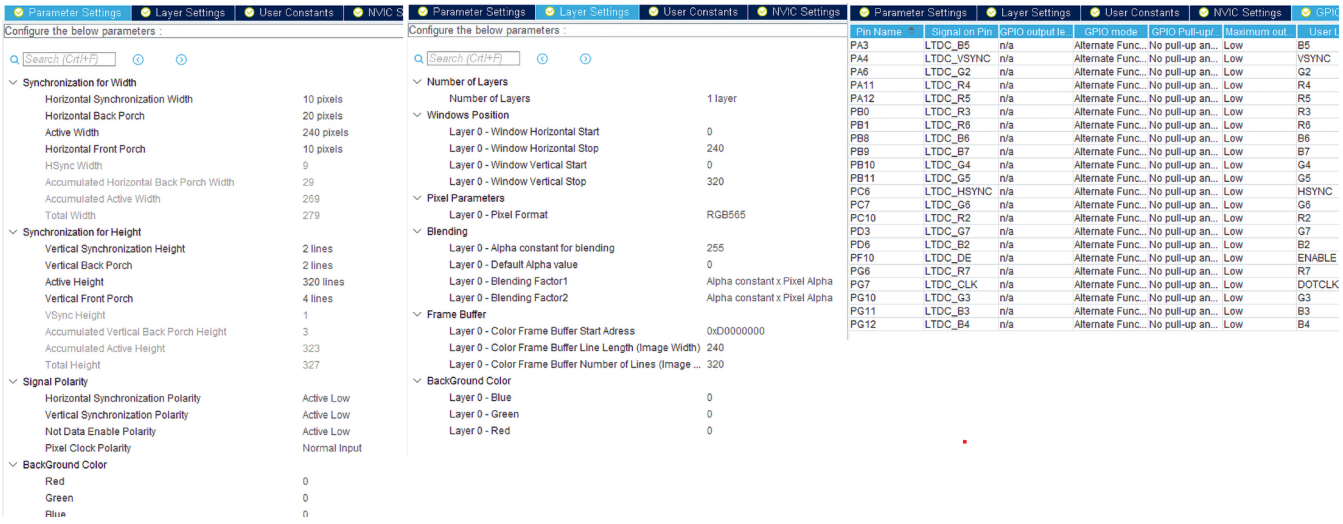


Rysunek 21. Zakładka *Parameter Settings* dla *Middleware Graphics*

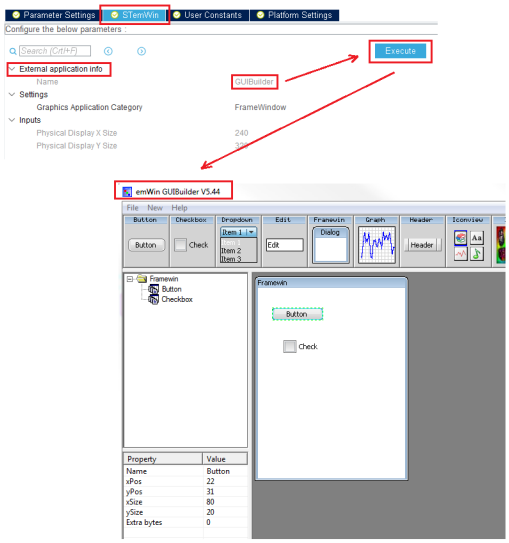
- Zmianę liczby ekranów wirtualnych.
- Wybranie rodzaju konwersji kolorów (domyślnie GUICC_565).
- Rezerwowanie wielkości pamięci przeznaczonej dla GUI.
- Włączanie i wyłączanie użycia systemu czasu rzeczywistego FreeRTOS.
- Włączanie wsparcia dla sterownika wyświetlacza LCD-TFT typu ILI9341.

Jeżeli wybierzemy opcję *FreeRTOS enable*, to oprogramowanie STM32CubeMX przygotuje odpowiednią konfigurację domyślną systemu czasu rzeczywistego i utworzy odpowiedni wątek. Po wybraniu opcji wyłączenia RTOS wygenerowany projekt będzie działał na zasadzie poolingu.

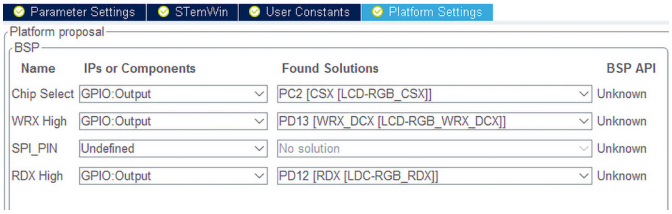
Ustawienie możliwości użycia GUIBuilder w zakładce *Parameter Settings* umożliwia (po kliknięciu na przycisk *Execute*) otworzenia narzędzia emWin GUIBuilder z zakładki STemWIN (**rysunek 22**). Okno *Platform Settings* definiuje przypisanie linii dodatkowego interfejsu szeregowego używanego do komunikacji ze sterownikiem panelu LCD-TFT (**rysunek 23**). W podobny sposób możemy konfigurować pozostałe moduły z listy middleware.



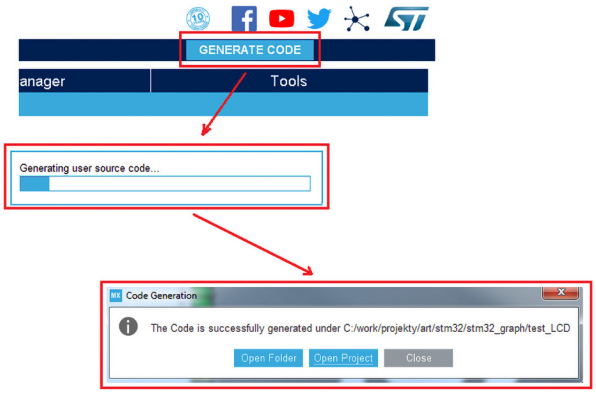
Rysunek 20. Domyślna konfiguracja interfejsu LTDC



Rysunek 22. Uruchomienie emWinGUIBuilder

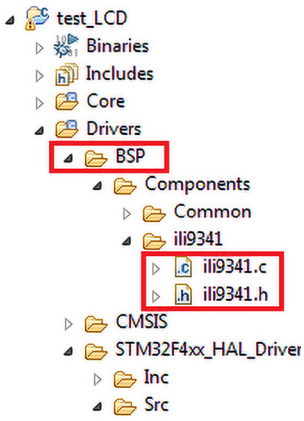


Rysunek 23. Przypisanie sygnałów interfejsu używanego przez sterownik panelu wyświetlacza



Rysunek 24. Generowanie kodu źródłowego

Generowanie szkieletu projektu z wykonanymi przez nas konfiguracjami układów peryferyjnych i ewentualnie middleware jest wykonywane po kliknięciu na przycisk „Generate Code”. Na końcu procesu generowania kodu źródłowego jest wyświetlane okno, w którym możemy otworzyć projekt w środowisku projektowym IDE (tu używamy TrueStudio) wybranym w zakładce *Project Manager* programu STM32CubeMX (**rysunek 24**).

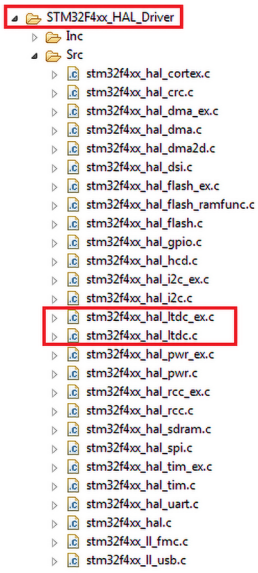


Rysunek 25. Pliki źródłowe z procedurami obsługi sterownika wyświetlacza LCD-TFT

W konfiguracji wybraliśmy bibliotekę Middleware Graphics, więc popatrzmy, jak wygląda obsługa wyświetlacza. Jak wiemy, wyświetlacz LCD-TFT jest sterowany przez kontroler ILI9341. To rozbudowany układ konfigurowany i sterowany przez zapisywanie

Listing 1. Inicjalizacja sterownika wyświetlacza LCD-TFT

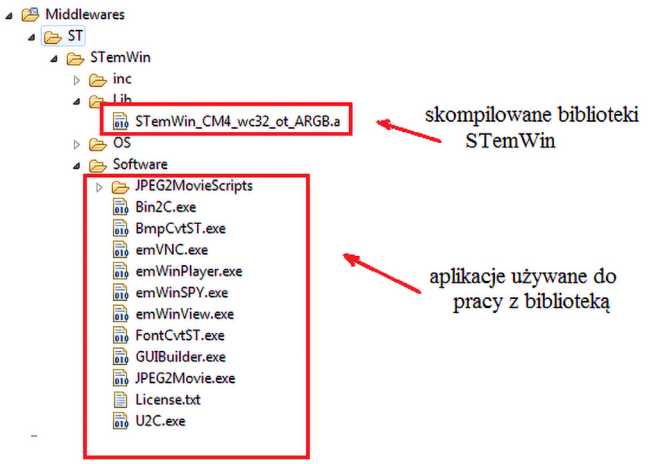
```
void ili9341_Init(void)
{
    /* Initialize ILI9341 low level bus layer */
    LCD_IO_Init();
    /* Configure LCD */
    ili9341_WriteReg(0xCA);
    ili9341_WriteData(0xC3);
    ili9341_WriteData(0x08);
    ili9341_WriteData(0x50);
    ili9341_WriteReg(LCD_POWERB);
    ili9341_WriteData(0x00);
    ili9341_WriteData(0xC1);
    ili9341_WriteData(0x30);
    ili9341_WriteReg(LCD_POWER_SEQ);
    ili9341_WriteData(0x64);
    ili9341_WriteData(0x03);
    ili9341_WriteData(0x12);
    ili9341_WriteData(0x81);
    ili9341_WriteReg(LCD_DTCA);
    ili9341_WriteData(0x85);
    ili9341_WriteData(0x00);
    ili9341_WriteData(0x78);
    ili9341_WriteReg(LCD_POWERA);
    ili9341_WriteData(0x39);
    ili9341_WriteData(0x2C);
    ili9341_WriteData(0x00);
    ili9341_WriteData(0x34);
    ili9341_WriteData(0x02);
    ili9341_WriteReg(LCD_PRC);
    ili9341_WriteData(0x20);
    ili9341_WriteReg(LCD_DTCB);
    ili9341_WriteData(0x00);
    ili9341_WriteData(0x00);
    ili9341_WriteReg(LCD_FRMCTR1);
    ili9341_WriteData(0x00);
    ili9341_WriteData(0x1B);
    ili9341_WriteReg(LCD_DFC);
    ili9341_WriteData(0x0A);
    ili9341_WriteData(0xA2);
    ili9341_WriteReg(LCD_POWER1);
    ili9341_WriteData(0x10);
    ili9341_WriteReg(LCD_POWER2);
    ili9341_WriteData(0x10);
    ili9341_WriteReg(LCD_VCOM1);
    ili9341_WriteData(0x45);
    ili9341_WriteData(0x15);
    ili9341_WriteReg(LCD_VCOM2);
    ili9341_WriteData(0x90);
    ili9341_WriteReg(LCD_MAC);
    ili9341_WriteData(0xC8);
    ili9341_WriteReg(LCD_3GAMMA_EN);
    ili9341_WriteData(0x00);
    ili9341_WriteReg(LCD_RGB_INTERFACE);
    ili9341_WriteData(0xC2);
    ili9341_WriteReg(LCD_DFC);
    ili9341_WriteData(0x0A);
    ili9341_WriteData(0xA7);
    ili9341_WriteData(0x27);
    ili9341_WriteData(0x04);
    /* Colomn address set */
    ili9341_WriteReg(LCD_COLUMN_ADDR);
    ili9341_WriteData(0x00);
    ili9341_WriteData(0x00);
    ili9341_WriteData(0x00);
    ili9341_WriteData(0xEF);
    /* Page address set */
    ili9341_WriteReg(LCD_PAGE_ADDR);
    ili9341_WriteData(0x00);
    ili9341_WriteData(0x00);
    ili9341_WriteData(0x01);
    ili9341_WriteData(0x3F);
    ili9341_WriteReg(LCD_INTERFACE);
    ili9341_WriteData(0x01);
    ili9341_WriteData(0x00);
    ili9341_WriteData(0x06);
    ili9341_WriteReg(LCD_GRAM);
    LCD_Delay(200);
    ili9341_WriteReg(LCD_GAMMA);
    ili9341_WriteData(0x01);
    ili9341_WriteReg(LCD_PGAMMA);
    ili9341_WriteData(0x0F);
    ili9341_WriteData(0x29);
    ili9341_WriteData(0x24);
    ili9341_WriteData(0x0C);
    ili9341_WriteData(0x0E);
    ili9341_WriteData(0x09);
    ili9341_WriteData(0x4E);
    ili9341_WriteData(0x78);
    ili9341_WriteData(0x3C);
    ili9341_WriteData(0x09);
    ili9341_WriteData(0x13);
    ili9341_WriteData(0x05);
    ili9341_WriteData(0x17);
    ili9341_WriteData(0x11);
    ili9341_WriteData(0x00);
    ili9341_WriteReg(LCD_NGAMMA);
    ili9341_WriteData(0x00);
    ili9341_WriteData(0x16);
    ili9341_WriteData(0x1B);
    ili9341_WriteData(0x04);
    ili9341_WriteData(0x11);
    ili9341_WriteData(0x07);
    ili9341_WriteData(0x31);
    ili9341_WriteData(0x33);
    ili9341_WriteData(0x42);
    ili9341_WriteData(0x05);
    ili9341_WriteData(0x0C);
    ili9341_WriteData(0x0A);
    ili9341_WriteData(0x28);
    ili9341_WriteData(0x2F);
    ili9341_WriteData(0x0F);
    ili9341_WriteReg(LCD_SLEEP_OUT);
    LCD_Delay(200);
    ili9341_WriteReg(LCD_DISPLAY_ON);
    /* GRAM start writing */
    ili9341_WriteReg(LCD_GRAM);
}
```



Rysunek 26. Procedury obsługi warstwy HAL interfejsu LTDC

Listing 2. Funkcja mian()

```
int main(void)
{
    /* USER CODE BEGIN 1 */
    /* USER CODE END 1 */
    /* MCU Configuration */
    /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
    HAL_Init();
    /* USER CODE BEGIN Init */
    /* USER CODE END Init */
    /* Configure the system clock */
    SystemClock_Config();
    /* USER CODE BEGIN SysInit */
    /* USER CODE END SysInit */
    /* Initialize all configured peripherals */
    MX_GPIO_Init();
    MX_DMA_Init();
    MX_CRC_Init();
    MX_I2C3_Init();
    MX_SPI5_Init();
    MX_TIM1_Init();
    MX_USART1_UART_Init();
    MX_USB_OTG_HS_HCD_Init();
    MX_SPI3_Init();
    /* USER CODE BEGIN 2 */
    /* USER CODE END 2 */
    /* Initialise the graphical hardware */
    GRAPHICS_HW_Init();
    /* Initialise the graphical stack engine */
    GRAPHICS_Init();
    /* Graphic application */
    GRAPHICS_MainTask();
    /* Infinite loop */
    for(;;);
}
```



Rysunek 27. Zawartość katalogu Middlewares

wewnętrznych rejestrów. Zazwyczaj taki sterownik wymaga bardziej lub mniej zaawansowanej inicjalizacji polegającej na zapisywaniu rejestrów konfiguracyjnych. Należy się spodziewać, że w ramach biblioteki funkcji Middleware Graphics wykonano odpowiednią procedurę. W katalogu BSP umieszczono katalog **ili9341**, w którym są zapisane dwa pliki: ili9341.c i ili9341.h (**rysunek 25**). W pliku ili9341.c umieszczono między innymi funkcję **ili9341_Init()** pokazaną na **listingu 1**. Jak widać, inicjalizacja nie jest banalna i żeby samodzielnie wykonać odpowiednią procedurę, trzeba by było włożyć w to wiele pracy. Procedury obsługi warstwy HAL interfejsu LTDC wykorzystywanego przez sterownik wyświetlacza są zapamiętane w katalogu STM32F4xx_HAL_Driver (**rysunek 26**).

Skompilowaną bibliotekę graficzną emWin w wersji STemWin umieszczono w katalogu Middlewares\STemWin\Lib, jak to pokazano na **rysunku 27**. Co ciekawe, katalog Middlewares\ST-STemWin\Software zawiera programy wykorzystywane w trakcie pracy z biblioteką graficzną.

Na koniec

Oprogramowanie STM32CubeMX bardzo ułatwia konfigurowanie bloków funkcjonalnych i middleware. Po wygenerowaniu kodu otrzymujemy szkielet projektu z plikami konfigurującymi użyte bloki. To oczywiste i spore udogodnienie, ale aplikację musimy napisać sami. Aby jeszcze bardziej ułatwić pracę programiście, w plikach źródłowych umieszczane są sugestie, gdzie użytkownik może dopisać swój kod, jak pokazano na **listingu 2** w funkcji *main()*.

STM32CubeMX jest stale rozwijany przez ST, a jego kolejne wersje są coraz doskonalsze. Wersja piąta jest dojrzałym narzędziem pomagającym programiście w niewdzięcznej i uciążliwej fazie konfigurowania procesora. Dużą zaletą jest bezpłatny, pełny dostęp do wszystkich możliwości programu. Mimo tych niezaprzeczalnych zalet trochę niezrozumiały jest brak konfiguratora STM32CubeMX V5 w wariancie wtyczki *plug-in* instalowanej, na przykład, w IDE opartym na Eclipse. Wcześniejsze wersje wtyczki miały oczywiste błędy. Dla przykładu, nie można było poprawnie skonfigurować układu taktowania, podczas gdy wersja uruchamiana niezależnie pracowała bez problemu. Praca z IDE z zainstalowaną wtyczką konfiguratora jest jak dla mnie wygodniejsza i szybsza. Mimo wspomnianego mankamentu STM32CubeMX wykonuje bardzo dobrze powierzone mu zadanie i jego używanie może być standardem wśród programistów używających mikrokontrolerów STM32.

Tomasz Jabłoński, EP