

Warsztaty rekrutacyjne lato 2023 – spotkanie 1

STM32CubeIDE, debugger, GPIO, UART

Eryk Moźdżen

10 maja 2023

Plan prezentacji

- 1 Wstęp
- 2 Mikrokontroler
 - Sprzęt
 - Program
- 3 Środowisko STM32CubeIDE
 - Projekt
- 4 Piny GPIO
 - Przedstawienie
 - Tryby pracy
 - Konfiguracja CubeMX
 - Wygenerowany kod
 - Użycie
- 5 Debugger
 - Nawigacja
 - Wybrane możliwości
- 6 Port UART
 - Przedstawienie

O prowadzącym



- aktywny członek KNR KoNaR od 2020 roku
- twórca małych robotów mobilnych
- programista embedded w projekcie Ariadna
- półroczne doświadczenie z STM32 w przemyśle

O zajęciach

Cele warsztatów:

- letnia rekrutacja do KNR KoNaR
- zaznajomienie przyszłych inżynierów z czymś innym niż Arduino
- szerzenie wiedzy wśród studentów
- zwiększenie zainteresowania programowaniem embedded

Co to jest to Embedded?

I komu to potrzebne?

System wbudowany (ang. embedded system)

System komputerowy złożony z procesora, pamięci i peryferiów z funkcjonalnościami dedykowanymi do danego urządzenia. Często sercem takiego układu jest mikrokontroler.

Komu to potrzebne?

Wszystkim, którzy chcą wchodzić w interakcję z fizycznym sprzętem.
W szczególności Nam – robotykom.

Mikroporcesor vs. mikrokontroler

	microprocesor	mikrokontroler
budowa	CPU	CPU Flash RAM pin I/O periferia
cel	obliczenia	interakcja

Skrótowa nazwa dla mikrokontrolera (ang. microcontroller):
 $\text{micro} + \text{controller} = \mu + C = \mu C \approx \mu C$

Tworzenie oprogramowania

- target inny niż maszyna z kompilatorem
- Memory Mapped I/O
- kod źródłowy często w więcej niż jednym języku
- podstawa: język C

Nasz ulubiony język programowania – ważne pojęcia

- funkcje
- struktury
- wskaźniki (referencja, dereferencja)
- alokacja pamięci (statyczna vs. dynamiczna)
- keywords-y (sizeof, typedef, volatile, static, extern)

Fixed width integer types #include<stdint.h>

uint8_t	int8_t
uint16_t	int16_t
uint32_t	int32_t
uint64_t	int64_t

HAL – Hardware Abstraction Layer

Charakterystyka uC:

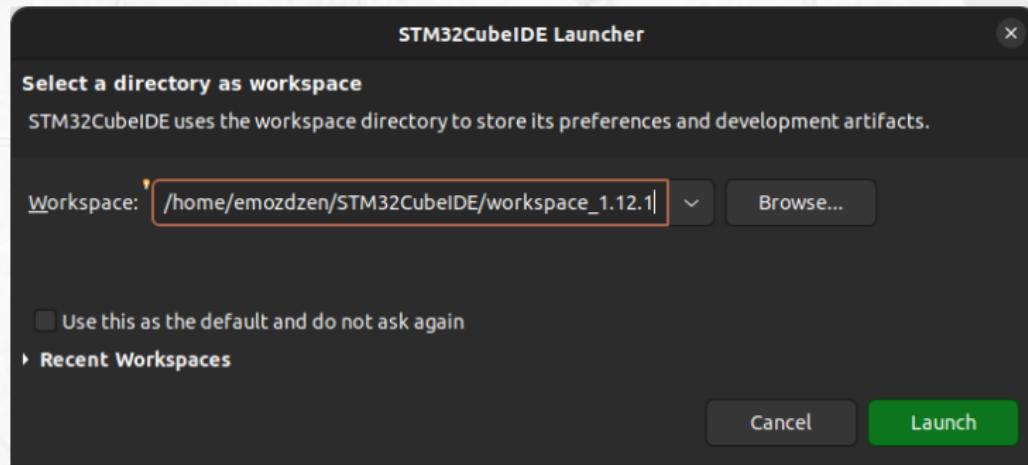
- praca w większości opiera się na wykorzystywaniu sprzętowych peryferiów
- obsługa peryferiów jest często nietrywialna i wymagająca doświadczenia
- każdy jest inny, posiada inny zestaw peryferiów
- dokumentacja jest obszerna (kilka tysięcy stron)

HAL umożliwia przenośność kodu źródłowego pomiędzy różnymi uC.
Programista nie musi być w pełni świadomy tego co się dzieje pod spodem.

Tworzenie nowego projektu

Wybór workspace-a

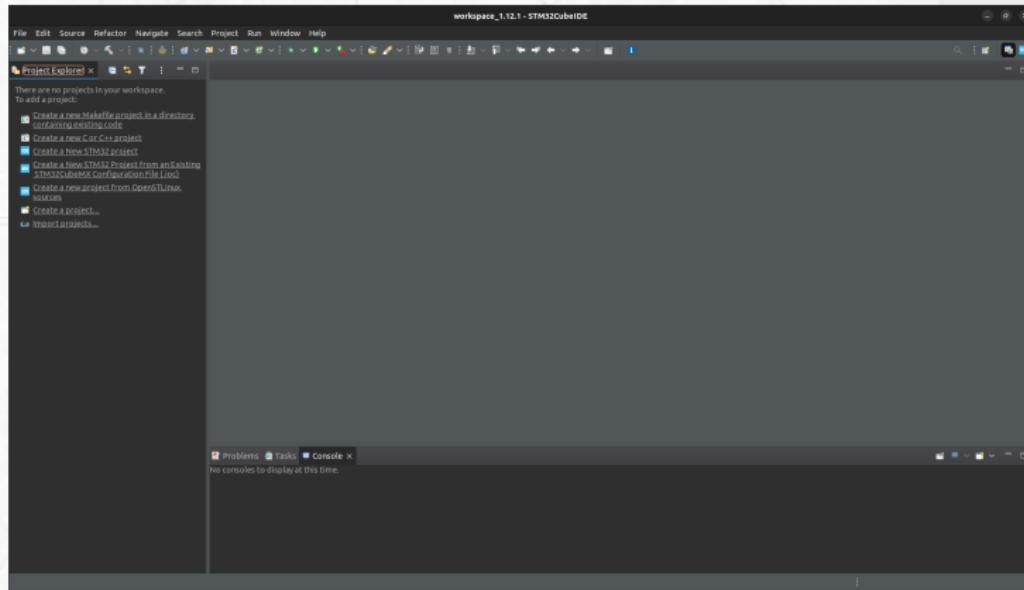
Po uruchomieniu STM32CubeIDE pojawia się okno wyboru przestrzeni roboczej. Jest to miejsce gdzie będą zapisywać się foldery z projektami. Zostawiamy domyślne i klikamy „Launch”.



Tworzenie nowego projektu

Nawigacja

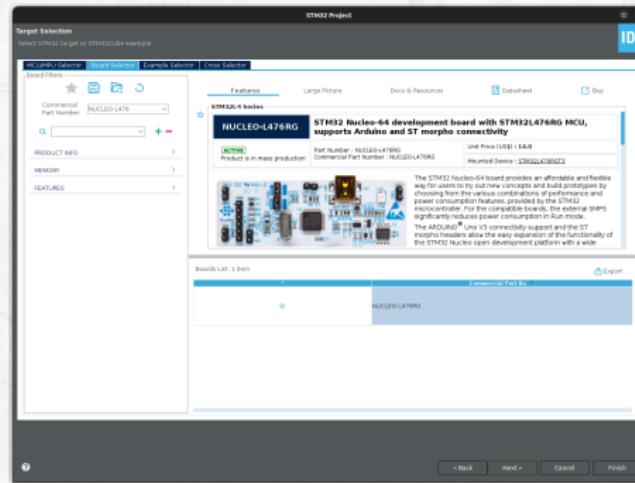
Naszym oczom ukazuje się pusty workspace. Aby stworzyć nowy projekt wchodzimy na górze w : File → New → STM32 Project



Tworzenie nowego projektu

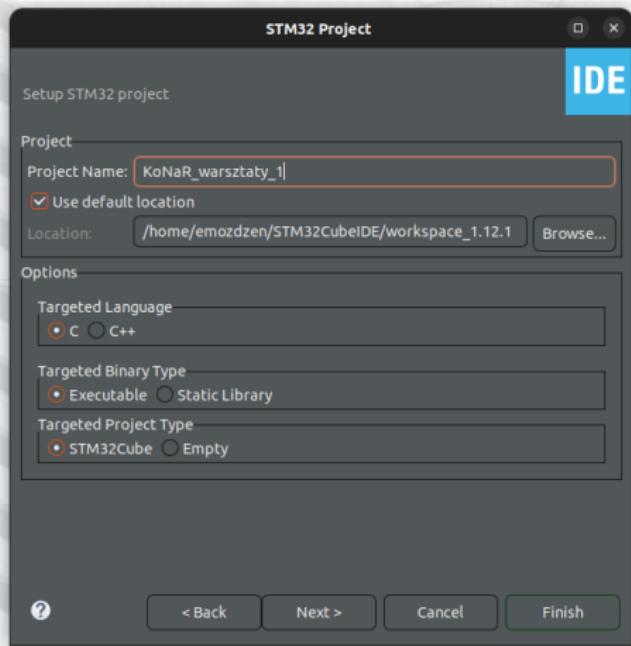
Wybór dostępnego hardwarcu

Teraz otwiera się okno selektora. Na górze wchodzimy w zakładkę „Board Selector”. Następnie w okno „Comercial Number Part” wpisujemy nazwę płytki ewaluacyjnej którą dysponujemy. Jest ich wiele trzeba wpisać tą którą się dysponuje. Dla STM32 są to płytki Nucleo lub Discovery. Wybieramy swoją płytke i klikamy „Next”.



Tworzenie nowego projektu

Nazwa

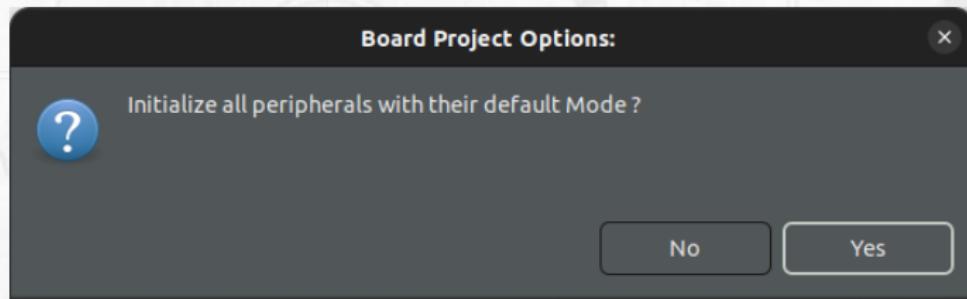


W polu „Project Name” wpisujemy nazwę projektu. Ciężko będzie ją potem zmienić. Resztę rzeczy zostawiamy domyślne. Klikamy „Finish”. Możliwe, że w międzyczasie rozpoczęcie się pobieranie ogromnej paczki z zasobami.

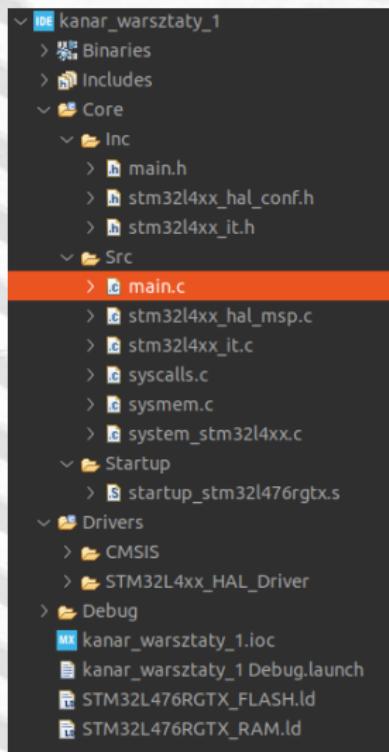
Tworzenie nowego projektu

Domyślna inicjalizacja peryferiów

Wybór odpowiedniej płytki powinien skonfigurować projekt pod istniejący na niej hardware. Dzięki temu będziemy mieli skonfigurowane piny diody oraz przycisku. Naciskamy „Yes”.



Drzewo projektu



Jeśli wszystko poszło dobrze powinno wygenerować się podstawowe drzewo projektu.

- Core – wygenerowane pliki – main.c
- Drivers – biblioteka HAL i rejesty CMSIS
- plik .ioc – graficzna konfiguracja CubeMX

Jeśli macie więcej niż jeden projekt zamknijcie nieużywane klikając prawym przyciskiem, a następnie „Close Project”.

GPIO – okno na świat uC

Piny GPIO (ang. General Purpose Input Output)

Piny GPIO to wyprowadzenia krzemu na zewnątrz obudowy. Pozwalają na wyjście/odczyt binarnych sygnałów logicznych jak i połączenie z wewnętrznym peryferium. Piny są grupowane w porty (w STM32 1 port to 16 pinów).

Możliwe zastosowania dla GPIO:



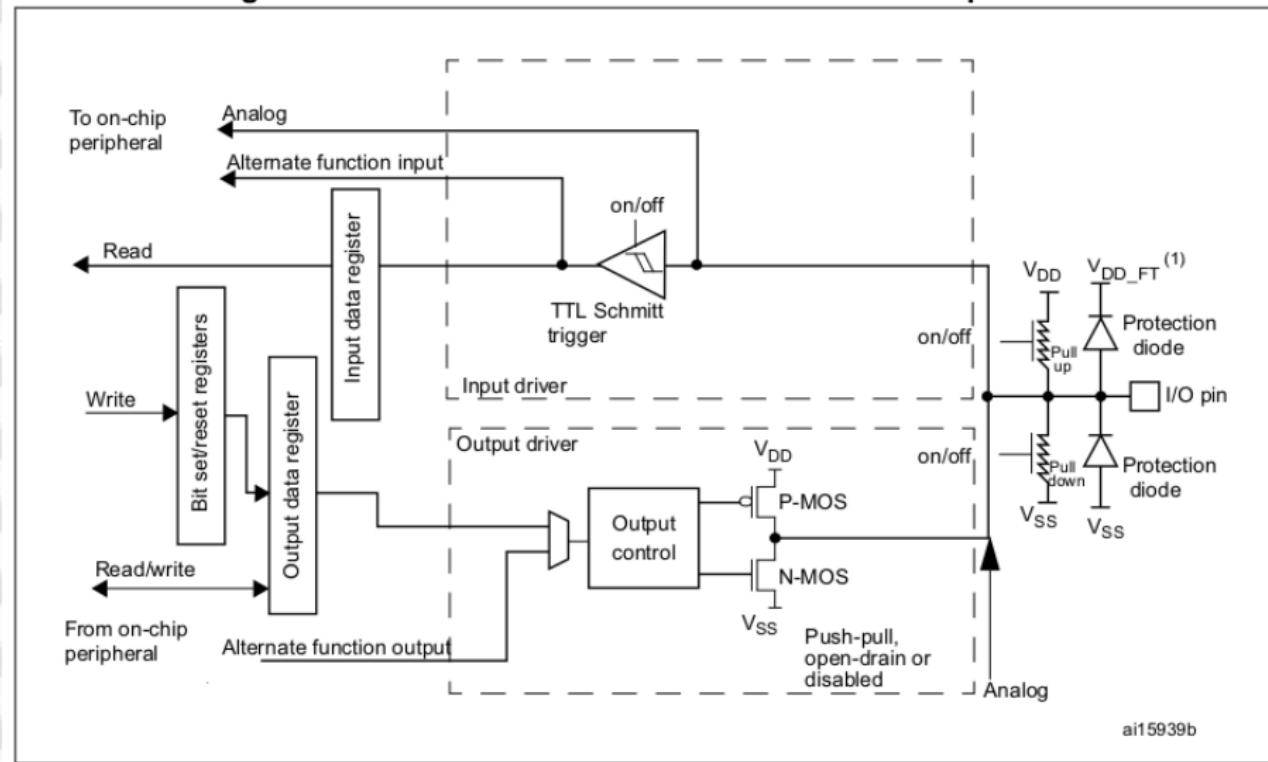
- zewnętrzne oscylatory
- cyfrowe wejścia/wyjścia
- zewnętrzne przerwania
- kanały przetworników ADC
- kanały timerów
- komunikacja UART, I²C, SPI, USB, CAN ...

Tryby pracy

- wejście cyfrowe
- wejście analogowe
- wyjście cyfrowe typu Push–Pull / Open–Drain
- wejście Alternate Function
- wyjście Alternate Function typu Push–Pull / Open–Drain

Wewnętrzna budowa

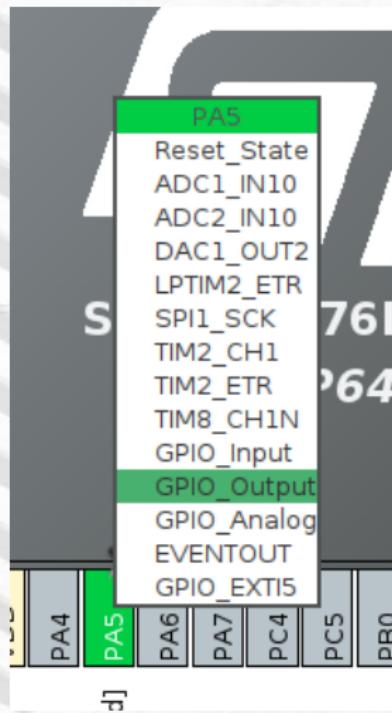
Figure 25. Basic structure of a five-volt tolerant I/O port bit



ai15939b

Konfiguracja w CubeMX

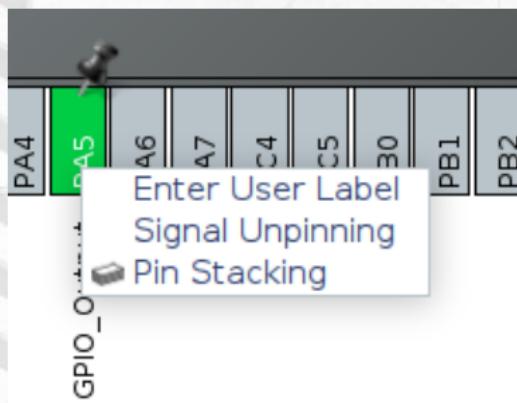
Wybór trybu pracy



Po wejściu w plik o rozszerzeniu .ioc pokazuje się graficzne okno CubeMX. Szare prostokąty to piny bez konfiguracji, zielone już ją mają. Aby dodać swój nowy pin, należy lewym przyciskiem go kliknąć, a następnie wybrać z listy dostępnych opcji odpowiednią. Najwięcej jest trybów AF, czyli tych od peryferiów. W Naszym wypadku pin PA5 powinien być skonfigurowany jako zwykłe wyjście cyfrowe. Lista na każdym pinie jest inna, i zależy od wewnętrznej budowy uC.

Konfiguracja w CubeMX

Nadawanie customowej nazwy

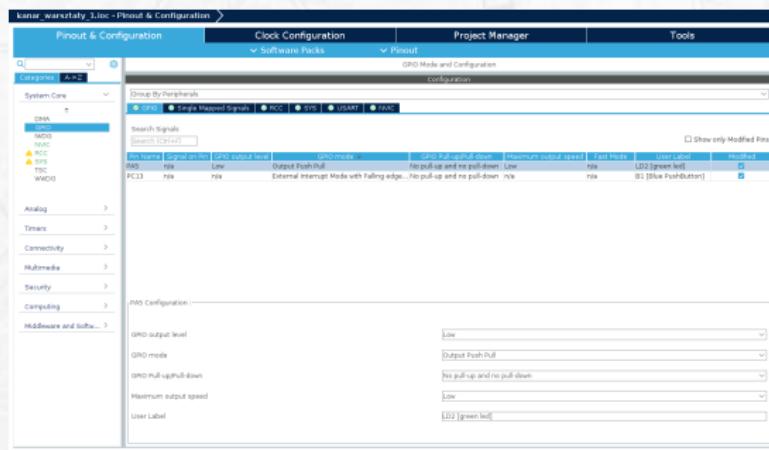


Możliwe jest nadanie nazwy dla łatwiejszego rozróżniania nóżek od siebie. W tym celu klikamy prawym przyciskiem na dany pin i wybieramy „Enter User Label”. Zawartość w kwadratowych nawiasach to tylko komentarz. Domyślna inicjalizacja peryferiów zrobiła to za Nas.

Konfiguracja w CubeMX

Panel boczny

Po lewej stronie można wysunąć panel boczny. W Categories → System Core → GPIO widać listę wszystkich pinów GPIO zgrupowanych po trybach jakie im nadano. Widać dwie nóżki: led oraz guzik domyślnie skonfigurowane. Po kliknięciu w dowolnego z nich widac pełną konfigurację dla danego trybu. Dla wyjścia definiowana jest domyślny stan, prędkość, rodzaj, pull-up/down i nazwa.



Wygenerowane derektywy w main.h

```
58 /* Private defines -  
59 #define B1_Pin GPIO_PIN_13  
60 #define B1_GPIO_Port GPIOC  
61 #define USART_TX_Pin GPIO_PIN_2  
62 #define USART_TX_GPIO_Port GPIOA  
63 #define USART_RX_Pin GPIO_PIN_3  
64 #define USART_RX_GPIO_Port GPIOA  
65 #define LD2_Pin GPIO_PIN_5  
66 #define LD2_GPIO_Port GPIOA  
67 #define TMS_Pin GPIO_PIN_13  
68 #define TMS_GPIO_Port GPIOA  
69 #define TCK_Pin GPIO_PIN_14  
70 #define TCK_GPIO_Port GPIOA  
71 #define SWO_Pin GPIO_PIN_3  
72 #define SWO_GPIO_Port GPIOB  
73  
74
```

Jeśli piny dostały specjalne niedomyślne nazwy porty do których należą oraz ich numery zostały wygenerowane w postaci specjalnych definów w kodzie. Należy ich używać przy wywoływaniu funkcji HAL.

Wygenerowana inicjalizacja w main.c

```
208 static void MX_GPIO_Init(void)
209 {
210     GPIO_InitTypeDef GPIO_InitStruct = {0};
211     /* USER CODE BEGIN MX_GPIO_Init_1 */
212     /* USER CODE END MX_GPIO_Init_1 */
213
214     /* GPIO Ports Clock Enable */
215     __HAL_RCC_GPIOC_CLK_ENABLE();
216     __HAL_RCC_GPIOH_CLK_ENABLE();
217     __HAL_RCC_GPIOA_CLK_ENABLE();
218     __HAL_RCC_GPIOB_CLK_ENABLE();
219
220     /*Configure GPIO pin Output Level */
221     HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_RESET);
222
223     /*Configure GPIO pin : B1 Pin */
224     GPIO_InitStruct.Pin = B1_Pin;
225     GPIO_InitStruct.Mode = GPIO_MODE_IT_FALLING;
226     GPIO_InitStruct.Pull = GPIO_NOPULL;
227     HAL_GPIO_Init(B1_GPIO_Port, &GPIO_InitStruct);
228
229     /*Configure GPIO pin : LD2 Pin */
230     GPIO_InitStruct.Pin = LD2_Pin;
231     GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
232     GPIO_InitStruct.Pull = GPIO_NOPULL;
233     GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
234     HAL_GPIO_Init(LD2_GPIO_Port, &GPIO_InitStruct);
235
236     /* USER CODE BEGIN MX_GPIO_Init_2 */
237     /* USER CODE END MX_GPIO_Init_2 */
238 }
```

Generator kodu tworzy i wywołuje specjalną funkcję z przedrostkiem „MX” w celu prawidłowej inicjalizacji wyklikanych peryferiów, w tym przypadku GPIO. Można zobaczyć, że wymagane jest włączenie sygnału zegarowego do portów, następnie konfiguracja poszczególnych pinów w portach. Argumenty przekazywane są jako wskaźnik na specjalnie przygotowaną i wypełnioną strukturę.

Funkcji MX w żadnym wypadku nie należy modyfikować!

Blink – Najprostszy program

```
/* Infinite loop */
/* USER CODE BEGIN WHILE */

while(1) {
    HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_SET);
    HAL_Delay(1000);
    HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_RESET);
    HAL_Delay(1000);

    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
}

/* USER CODE END 3 */
```

```
/* Infinite loop */
/* USER CODE BEGIN WHILE */

while(1) {
    HAL_GPIO_TogglePin(LD2_GPIO_Port, LD2_Pin);
    HAL_Delay(1000);

    /* USER CODE END WHILE */

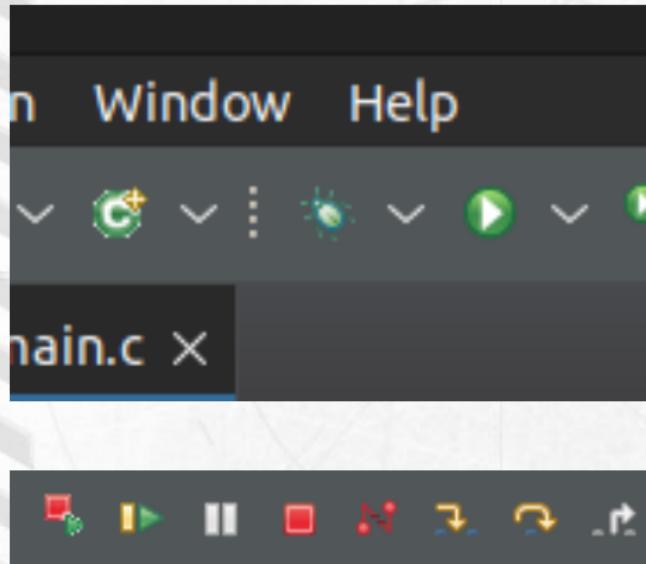
    /* USER CODE BEGIN 3 */
}

/* USER CODE END 3 */
```

Celem jest napisanie programu, który włącza i wyłącza diodę cyklicznie. Służą od tego odpowiednie funkcje HAL dostępne z przedrostkami HAL_GPIO. Podpowiedzi są pod Ctrl + Space.

HAL_Delay to funkcja czekająca podaną ilość milisekund. Wykonywanie pętli zatrzymuje się w jej miejscu.

Debugger



Aby wejść w widok debuggera klikamy ikonę pluskwy koło przycisku „Run”. Jeśli wyskoczy okno z konfiguracją debuggera zostawiamy domyślne i klikamy „Applay” i „Ok”.

Pojawia się nowe przyciski na panelu. Z ich pomocą można od lewej: uruchamiać ponownie, kontynuować, zatrzymywać, zakończyć, rozłączyć się, wejść do funkcji, przeskoczyć wywołanie, wyjść z wywołania kodu.

Widok debugera

The screenshot shows the STM32CubeIDE interface with the following details:

- File**, **Edit**, **Source**, **Refactor**, **Navigate**, **Search**, **Project**, **Run**, **Window**, **Help** menu items.
- Debug X Project Explorer** tab is active.
- Project Explorer** pane shows a project named "F411_Test" with a single file "main.c" selected.
- main.c** code editor pane displays the main function and various initialization sections.
- Variables X Breakpoints Expressions Registers Live Expressions SFRs** tab is active.
- Variables** pane is empty.
- Console X Problems Executables Debugger Console Memory** tab is active.
- Console** pane shows the message "Verifying ...".
- Bottom status bar** shows "Download verified successfully".

Brakpoints

Breakpointy to podstawowe narzędzie debugowania kodu. Pozwala na zatrzymanie wykonywania programu po dojściu w konkretne miejsce. Wykorzystuje się je do sprawdzania:

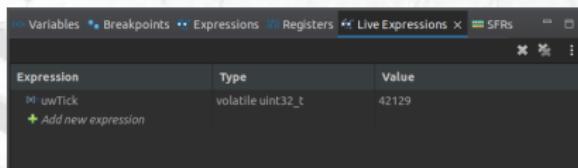
```
54 /* Infinite loop */
55 /* USER CODE BEGIN WHILE */
56 while (1)
57 {
58     HAL_GPIO_TogglePin(LD2_GPIO_Port, LD2_Pin);
59     HAL_Delay(100);
60
61     /* USER CODE END WHILE */
62
63     /* USER CODE BEGIN 3 */
64 }
65 /* USER CODE END 3 */
```

- czy program w ogóle wchodzi w dany warunek
- wartości lokalnych i globalnych zmiennych
- kolejności wywoływania

Stawia się go oraz usuwa poprzez dwukrotne kliknięcie lewym przyciskiem w niebieski pasek po lewej stronie edytora.

Live Expressions

Do podglądu wartości globalnych zmiennych podczas normalnej pracy uC przydatny jest widok „Live Expressions”. Jest to prosty zamiennik ciągłego używania funkcji printf znanego z programowania PC. Wystarczy podać nazwę zmiennych które chcemy obserwować. Mogą być to typy prymitywne jak i struktury oraz tablice.



- wartości globalnych zmiennych
- zmienności globalnych zmiennych

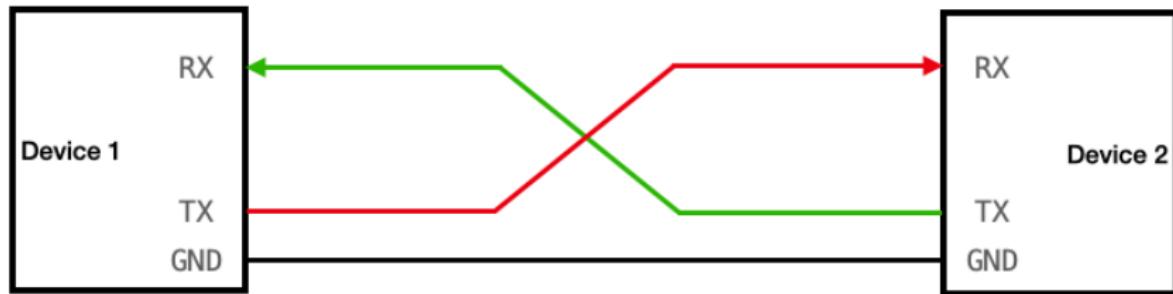
Normalnie pojawia się w oknie po prawej stronie. Jeśli go nie ma można otwprzyć go poprzez Window → Show View → Live Expressions.

UART

Co to jest

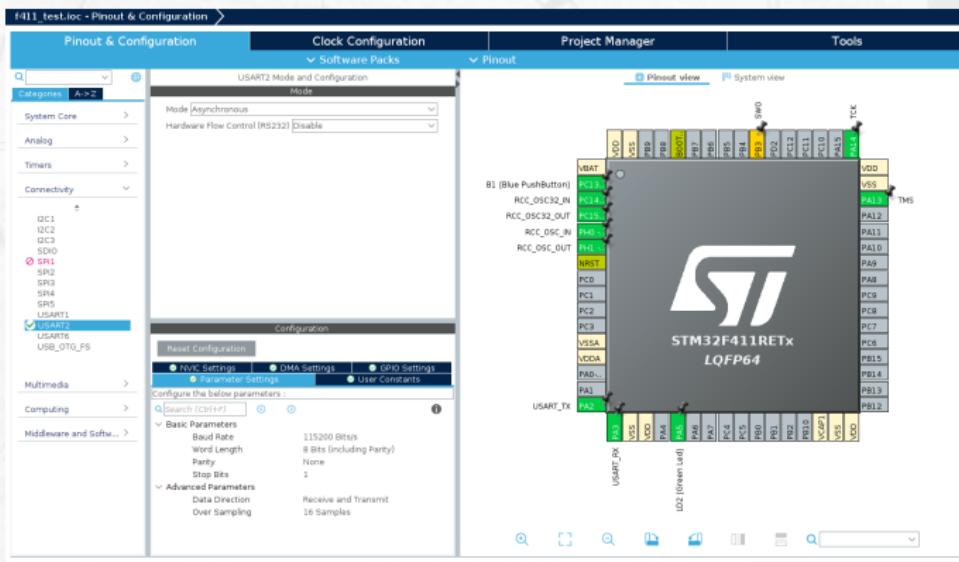
UART (ang. universal asynchronous receiver-transmitter)

Najprostszy interfejs komunikacyjny stworzony do pracy w obrębie urządzenia. Składa się z dwóch linii: RX i TX połączonych na krzyż. TX (transmitter) nadaje, RX (receiver) odbiera. Najważniejszym parametrem jest prędkość, czyli BAUD RATE (po dwóch stronach musi być taki sam).



Konfiguracja CubeMX

W CubeMX otwieramy zakładkę „Connectivity”. Jest skonfigurowany UART2 fizycznie połączony z programatorem ST-Link V2, który ma funkcję wirtualnego portu COM widocznego z poziomu Naszych PC. Aktywnie UARTa na liście powoduje automatyczną konfigurację pinów. Zostawiamy domyślne ustawienia.



Hello world – wyświetlanie sformatowanego tekstu

Funkcje z przedrostkiem HAL_UART pozwalają na korzystanie z UART. Poniżej przedstawiono prosty program, wysyłający napis zformatowany tekst. Użyto biblioteki stdio.h oraz string.h. Konieczne jest zdefiniowanie własnego bufora na dane, a następnie ręczne wysłanie go (wersja blokująca).

```
/* Infinite loop */
/* USER CODE BEGIN WHILE */

char buffer[64];

uint16_t counter = 0;

while(1) {
    counter++;

    snprintf(buffer, sizeof(buffer), "Warsztaty KoNaR %d\r\n", counter);

    HAL_UART_Transmit(&huart2, (uint8_t *)buffer, strlen(buffer), HAL_MAX_DELAY);

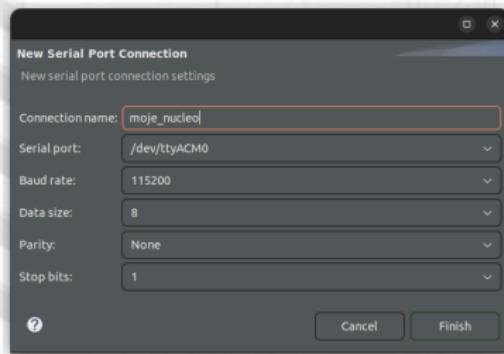
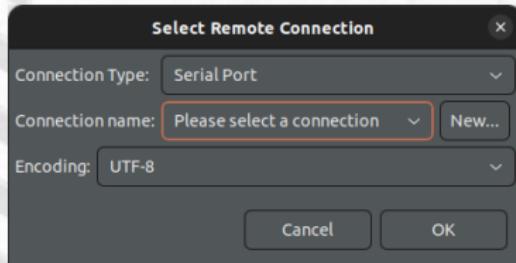
    HAL_Delay(100);

    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
}

/* USER CODE END 3 */
```

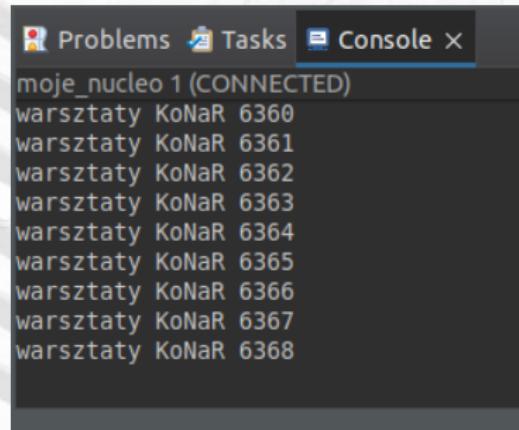
Konsola portu szeregowego



Aby wyświetlić wysyłane dane na PC wystarczy otworzyć port z Nucleo dowolnym programem. W STM32CubeIDE można zrobić to wchodząc na dole w Console → Open Console → 3 Command Shell Console.

Tworzymy nowy „Connection Name”, nadajemy mu nazwę, port COM do którego podłączony jest Nasze Nucleo oraz baudrate.

Konsola portu szeregowego – prawidłowy wynik



A screenshot of a terminal window titled "Console". The window shows a series of identical error messages repeated 10 times. The message is: "moje_nucleo 1 (CONNECTED) warsztaty KoNaR 6360". The window has tabs for "Problems", "Tasks", and "Console", with "Console" being the active tab.

```
moje_nucleo 1 (CONNECTED)
warsztaty KoNaR 6360
warsztaty KoNaR 6361
warsztaty KoNaR 6362
warsztaty KoNaR 6363
warsztaty KoNaR 6364
warsztaty KoNaR 6365
warsztaty KoNaR 6366
warsztaty KoNaR 6367
warsztaty KoNaR 6368
```

W przypadku nieprawidłowego wyświetlania się tekstu: ucięte wyrazy, krzaczki należy zamknąć wszystkie inne programy/terminale korzystające z danego portu COM (same się nie zamkują).