

# Architektura systemów komputerowych

Podczas realizacji poniższych zadań, oprócz instrukcji/dokumentacji podanych na końcu ćwiczeń (sekcja „Materiały”), można korzystać z dowolnych źródeł internetowych. Szczególnie polecany przez prowadzącego jest kurs dostępny na stronie: <https://forbot.pl/blog/kurs-stm32-l4-wstep-spis-tresci-dla-kogo-jest-ten-kurs-id48575>.

Uwaga: Po wykonaniu każdego zadania należy zapisać, kody i inne informacje do sprawozdania (zrzuty ekranu, odpowiedzi na pytania), o ile są potrzebne w danym zadaniu. Po wykonaniu wszystkich zadań niniejszej instrukcji należy powiadomić prowadzącego.

## Instrukcja I

Wprowadzenie, konfiguracja, DEBUG, IT, GPIO

### Wstęp

Architektura ARM wykorzystywana jest w szerokim przekroju mikroprocesorów i mikrokontrolerów. W naszym przypadku zaimplementowana została na mikrokontrolerach serii STM32 firmy STMicroelectronics. Rodzina 32-bitowych układów z rdzeniem ARM od tego producenta składa się z serii ekonomicznej, standardowej oraz wysokowydajnej. Nie zabrakło w niej układów typu *LowPower*, jak również układów z wbudowaną komunikacją Bluetooth i WiFi oraz dwurdzeniowych.

Do pracy posłuży nam kompleksowe środowisko o nazwie STM32CubeIDE, które integruje IDE z konfiguratorem STM32CubeMX.

Mikrokontroler to zrealizowana w postaci pojedynczego układu scalonego architektura zawierająca jednostkę centralną (CPU), pamięć RAM oraz rozbudowane układy wejścia-wyjścia oraz na ogół pamięć programu jako FRAM, MRAM, ROM lub Flash. Do wykorzystania całego potencjału układu należy zapoznać się z konkretnymi peryferiami, ich możliwościami oraz sposobem pracy z nimi. Mimo pewnych standardów, do których są dostosowywane, występują znaczące różnice w obsłudze między producentami, ale również w ramach różnych rodzin jednej marki. Firma STMicroelectronics dla uproszczenia opracowania oraz zwiększenia przenaszalności kodu przygotowała gotowe biblioteki do konfiguracji i obsługi elementów mikrokontrolera:

- HAL (*Hardware Abstraction Layer*),
- LL (*Low Layer*),
- STD (*Standard peripheral library*).

Funkcje biblioteki HAL potrzebne do wykonania poniższych zadań znajdują się w dolnej części instrukcji.

### Przerwania procesora

Przerwania procesora to mechanizm, który pozwala procesorowi na chwilowe przerwanie aktualnie wykonywanego zadania w celu obsługi innego, pilniejszego zdarzenia. Kod obsługi takiego zdarzenia może napisać programista w formie procedury (funkcji) obsługi przerwania. Po zakończeniu obsługi przerwania procesor wraca do pierwotnie wykonywanego zadania. Mechanizm przerwań występuje zarówno w komputerach, jak i mikrokontrolerach.

Istnieją dwa główne rodzaje przerw:

- sprzętowe (ang. hardware interrupts) – zgłaszane np. przez klawiaturę, pamięć masową, przycisk mikrokontrolera, zegar czasu rzeczywistego itp.;
- programowe (ang. software interrupts) – zgłaszane przez programy w celu wykonania operacji systemowych lub w celu obsługi sytuacji wyjątkowych (tzw. wyjątki).

## Przebieg ćwiczenia

1. W środowisku STM32CubeIDE utwórz nowy projekt w języku C dla **NUCLEO-G491RE** (zakładka *Board Selector*), która znajduje się na stanowisku. Jako nazwę projektu wpisz swój numer indeksu. Katalog projektu również powinien mieć nazwę taką, jak numer indeksu. Projekt utwórz zgodnie z poradnikiem „STM32CubeIDE - poradnik”.
2. W konfiguratorze STM32CubeMX sprawdź, czy pin **PA5** (dioda między przyciskami) ma nazwę **LD2**, a pin **PC13** (niebieski przycisk) **B1**. Jeśli nie, to zmień nazwy pinów „Enter user label”.
3. Zaprogramuj miganie diody z częstotliwością 4Hz w pętli głównej `while(1)` wspomagając się funkcją `HAL_DeDelay`. Napisz dwie wersje programu:
  - a) wykorzystując odwracanie stanu wyjścia **LD2** (funkcja `HAL_GPIO_TogglePin`),
  - b) wykorzystując ustawianie stanu wyjścia **LD2** (funkcja `HAL_GPIO_WritePin`).
4. Włącz tryb debugowania i przetestuj dostępne opcje blokowania programu, poruszania się po kodzie (tworzenie breakpointów, Resume, Terminate, Step Into, Step Over).
5. W głównej pętli zapisuj stan fizycznego przycisku (niebieskiego) w zmiennej typu `GPIO_PinState`. Wykorzystując debugger sprawdź wartość zmiennej w czasie działania programu podczas gdy przycisk jest wciśnięty i podczas gdy jest puszczony.
6. Podczas debugowania programu z poprzedniego zadania odnajdź rejestry (zakładka Registers po prawej stronie okna -> General Registers), których wartość zmienia się w zależności od stanu przycisku (stanu na pinie **PC13**). Jakie wartości przyjmują te rejestry w momencie, gdy przycisk jest wciśnięty, a jakie kiedy jest puszczony? Czy w rejestrach zmienia się tylko jeden bit, czy więcej?
7. Tryb blokujący (polling):
  - a) Powiąż stan przycisku z wbudowaną diodą. Naciśnięty przycisk generuje zapalenie się diody, puszczony zgaszenie diody.
  - b) Zmodyfikuj zadanie z poprzedniego punktu. Napisz kod zmieniający stan diody wyłącznie przy zboczu opadającym (puszczenie przycisku).
  - c) Ponownie zaprogramuj miganie diody **LD2**, tak jak w punkcie 3 (korzystając z `HAL_DeDelay`). Następnie napisz funkcję `changeLEDFrequency` (wywołanie w głównej pętli), której zadaniem jest zmiana częstotliwości migania po naciśnięciu przycisku **B1**. Funkcja ma za zadanie wybrać następną częstotliwość ze zbioru {25Hz, 10Hz, 5Hz, 1Hz, 0.5Hz} po każdym naciśnięciu przycisku. Reakcja na przycisk ma być natychmiastowa (bez oczekiwania na puszczenie).
  - d) Zmiana częstotliwości migania diody działa poprawnie jedynie podczas szybszego migania. Podczas wolniejszego migania częstotliwość nie chce się przełączać (lub przełącza się dopiero po kilku próbach). Z czego wynika ten problem? Aby go rozwiązać napisz drugą wersję programu wykorzystując funkcję `HAL_GetTick` zamiast `HAL_DeDelay`.
  - e) Przetestuj przycisk RESET (czarny). Służy on do natychmiastowego zatrzymania i ponownego uruchomienia (ang. restart) programu, po którym powinna wrócić początkowa częstotliwość migania diody: 25Hz. Czy zatrzymanie i ponowne uruchomienie programu następuje po wciśnięciu, czy po zwolnieniu przycisku?

## 8. Tryb przerwaniowy:

Należy w konfiguratorze CubeMX włączyć przerwania dla pinów 15:10. Należy również zmienić ustawienie GPIO mode dla pinu **PC13** tak, aby możliwe było zgłaszanie przerwań zboczem narastającym i opadającym (kurs Forbot - lekcja #7). Domyślnie jest ustawione samo zbocze narastające.

- a) Zrealizuj przełączenie świecenia diody przyciskiem, podobnie jak w zadaniu 7a, jednak tym razem zamiast trybu blokującego wykorzystaj mechanizm przerwań. Kod sterujący diodą powinien być zamieszczony w definicji funkcji obsługi przerwania `HAL_GPIO_EXTI_Callback`.

Podpowiedź: Przerwanie może pochodzić od dowolnego pinu z przedziału PC10-PC15, a więc na początku definicji funkcji należy sprawdzić, czy pochodzi z pinu powiązanego z przyciskiem.

- b) W trybie przerwaniowym zrealizuj przełączenie świecenia diody po puszczeniu przycisku, podobnie jak w zadaniu 7b.

## Funkcje

- **GPIO**
  - `void HAL_GPIO_TogglePin(GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin)`
  - `void HAL_GPIO_WritePin(GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin, GPIO_PinState PinState)`
  - `GPIO_PinState HAL_GPIO_ReadPin(GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin)`
- **TICK**
  - `void HAL_Delay(uint32_t Delay)`
  - `uint32_t HAL_GetTick()`
- **Przerwania - callback**
  - `void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)`

Typy zmiennych: `uint8_t`, `uint16_t` i `uint32_t` reprezentują liczby całkowite bez znaku (podobnie jak `unsigned int`) o ustalonych (niezależnych od środowiska) wielkościach, odpowiednio: 1, 2, 4 bajty.

## Materiały

- „STM32CubeIDE - poradnik” – dokument dostępny na stronie prowadzącego w folderze przedmiotu Architektura systemów komputerowych.
- [STM32G4 Series advanced Arm®-based 32-bit MCUs - Reference manual](#)
- [Datasheet - STM32G491xC STM32G491xE - Arm® Cortex®-M4 32-bit MCU+FPU, 170 MHz / 213 DMIPS, up to 512 KB Flash, 112 KB SRAM, rich analog, math accelerator](#)
- [Description of STM32G4 HAL and low-layer drivers - User manual](#)