

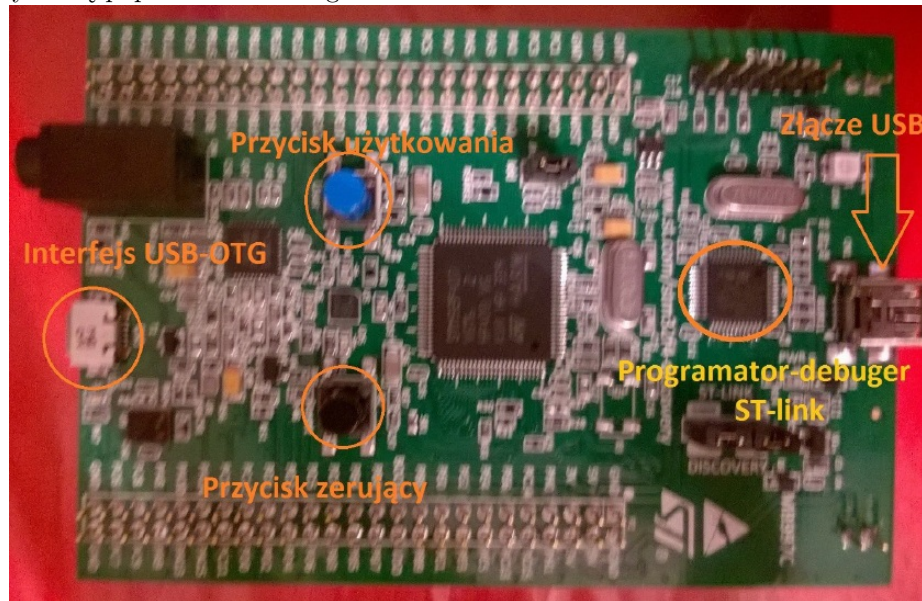
# .NET Micro Framework STM32F4 Discovery

Wojciech Duda

2016.4.21

## 1 Teoria

Rdzeń CortexM4F wykorzystuje architekturę ARMv7M. Pod względem organizacji pamięci jest to architektura harwardzka, tzn. pamięć zawierająca kod programu (Flash) i pamięć danych (SRAM) są rozdzielone i dostęp do nich odbywa się poprzez osobne magistrale.

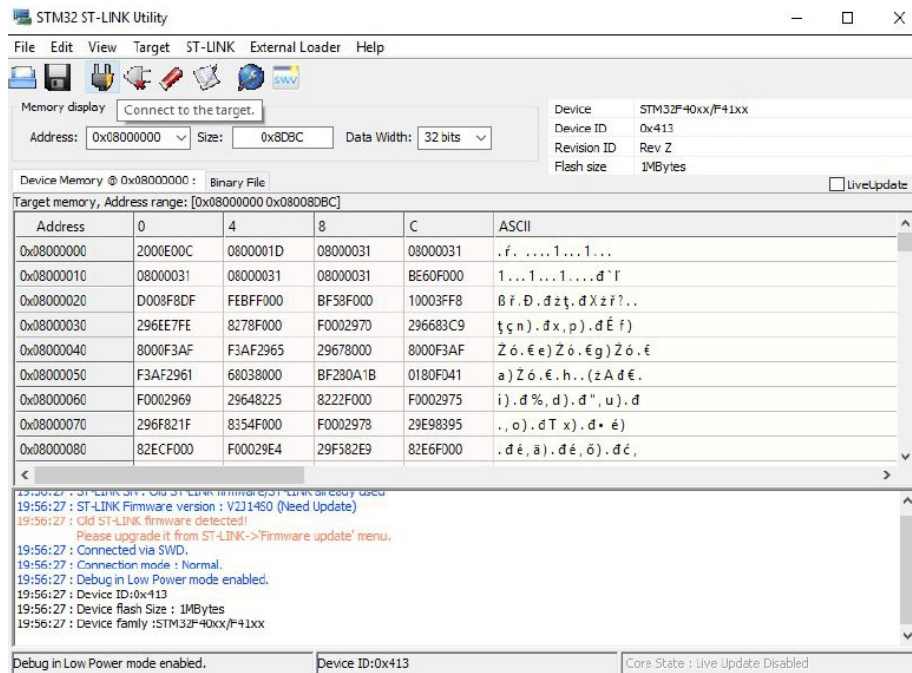


## 2 Instalacja

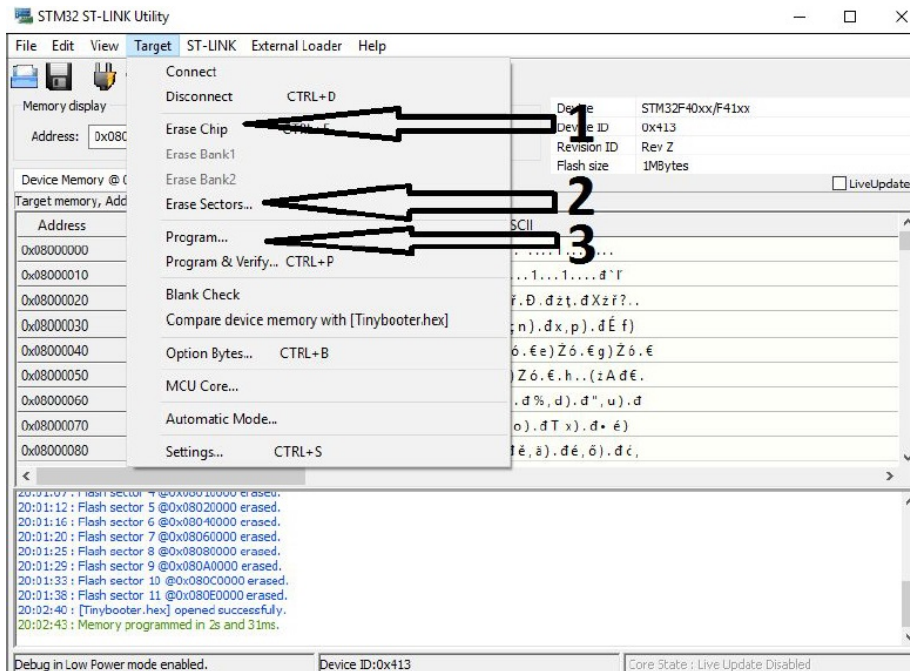
### 2.1 Narzędzia:

- mikrokontroler STM32F4 Discovery
- kable USB Micro oraz USB Mini
- Visual studio
- STM32 ST-LINK Utility
- sterownik USB
- bootloader oraz pliki hex
- .NET MicroFramework SDK

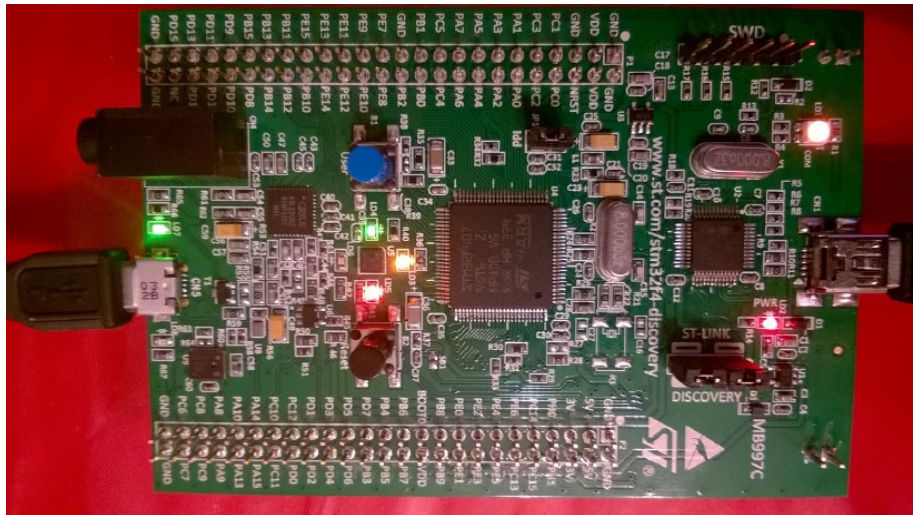
- 2.2 Zainstaluj STLINK, oraz SDK, resztę plików rozpakuj.
- 2.3 Podłącz kabel USB Mini (do wejścia oznaczonego jako “Złącze USB” na zdjęciu powyżej.)
- 2.4 Włącz STLINK Utility , a następnie połącz się z stm32f4 poprzez przycisk: “Connect to the = target”



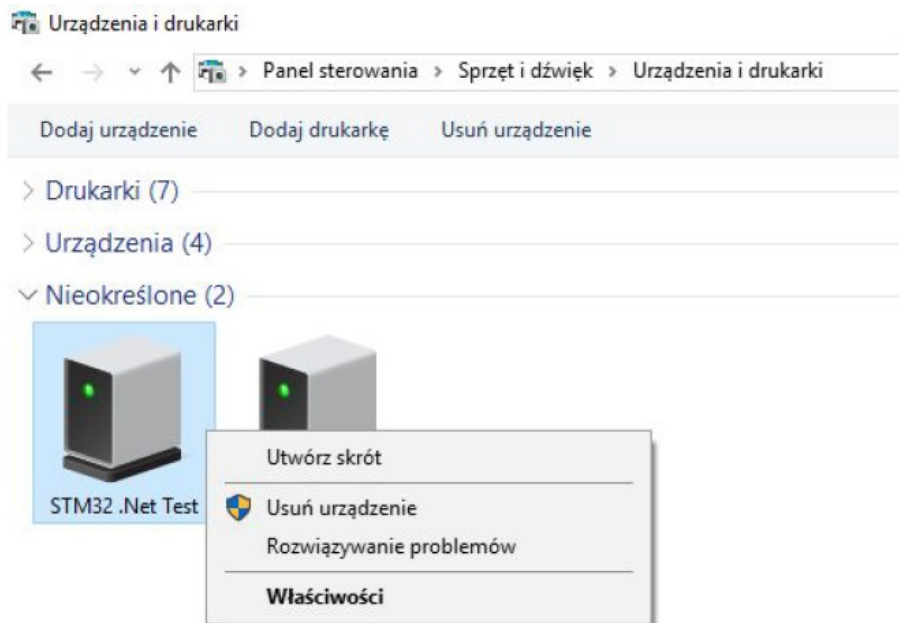
2.5 Następnie wybierz Target >Erase Chip oraz Target>Erase Sectors, wybierz wszystkie i potwierdź. Wybierz Target >Program..., wybierz ścieżkę Tinybooter.hex a następnie wybierz start. Zresetuj mikrokontroler poprzez przycisk zerujący.



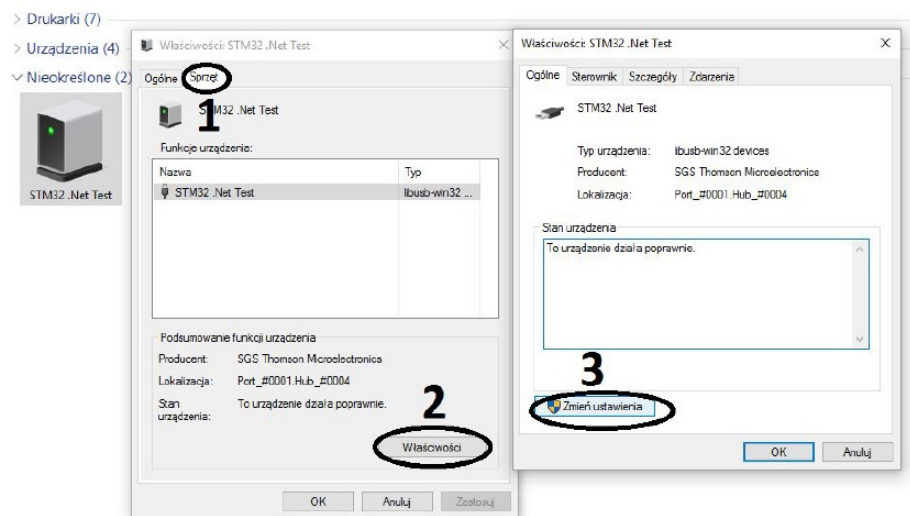
2.6 Jeżeli wszystko przebiegło prawidłowo powinny zapalić się 3 diody użytkowe. Podłącz kabel micro USB (muszą być podłączone oba).



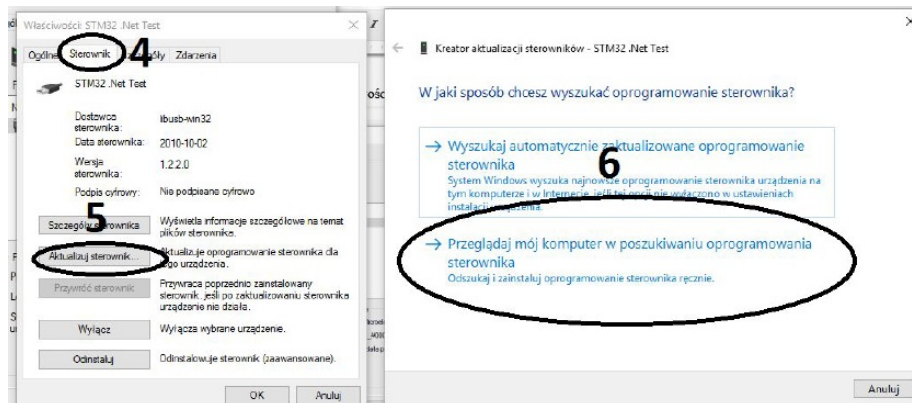
2.7 Przejdź do “urządzenia i drukarki”. Tam w obszarze “nieokreślone” kliknij prawym przyciskiem myszy w “STM .Net Test” i wybierz właściwości.



2.8 Wejdź w sprzęt >właściwości >zmień ustawienia >sterownik >Aktualizuj sterownik...

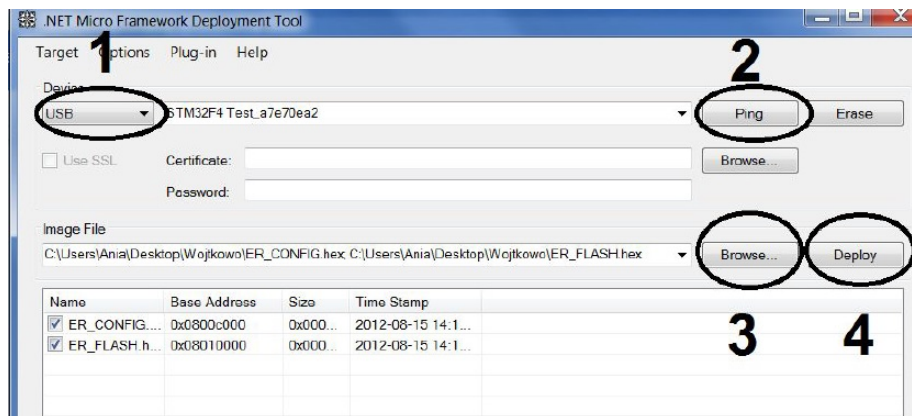




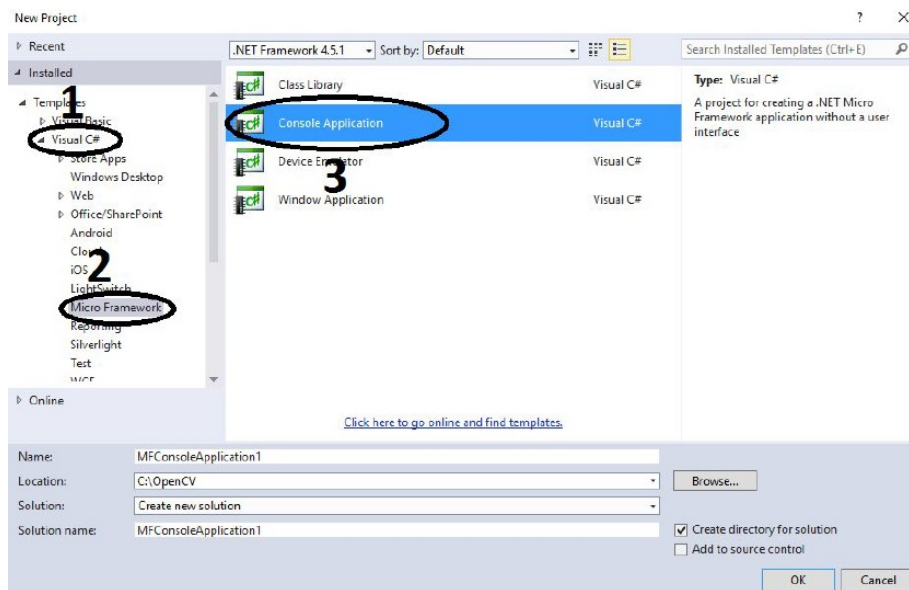


2.9 Wybierz “Przeglądaj mój komputer w poszukiwaniu oprogramowania sterownika” i wybierz ścieżkę gdzie rozpakowałeś na początku sterownik. Podczas instalacji ignoruj ostrzeżenia.

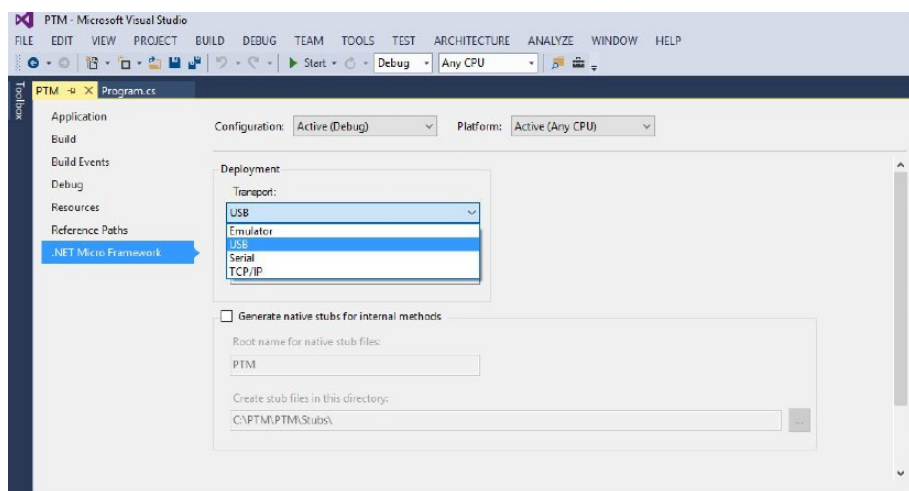
2.10 Teraz uruchom MFDeploy. Wybierz Device: USB. Naduś przycisk Ping. Następnie drugie od góry Browse... , wybierz ścieżkę pozostałych dwóch plików hex: ER\_CONFIG.hex, ER\_FLASH.hex oraz wybierz Deploy.



## 2.11 Włącz Visual studio utwórz nowy projekt i wybierz C# >Micro Framework >Console Application.



## 2.12 W utworzonym projekcie, w Solution Explorer kliknij prawym przyciskiem myszy na projekt i wybierz “Properties”. Tam wybierz .NET Micro Framework i Transport ustaw na USB.





### 3 Instrukcja użycia

Wszystkie przykłady wymienione poniżej są dostępne na: <https://github.com/PUT-PTM/STM-Csharp-tutorial>. Każdy posiada dwie referencje: Microsoft.SPOT.Native oraz mscorlib, są one niezbędne do działania. Mscorlib zawiera zbiór przestrzeni nazw. Aby uruchomić konkretny przykład trzeba uruchomić solucję "STM32F4", w "Solution Explorer" kliknąć prawym przyciskiem myszy konkretny przykład i wybrać opcję "Set as StartUp Project". Niezbędne jest również podłączenie mikrokontrolera kablami USB Micro oraz USB Mini. Następnie wystarczy tylko rozpocząć debugowanie przyciskiem F5.

#### 3.1 Przycisk

Obsługa przycisku oraz LED jest zaimplementowana w przykładzie "Button". Posiada on dodatkową referencję Microsoft.SPOT.Hardware. Klasa OutputPort obsługuje diody, występują one na pinach 60-63. Klasa InterruptPort obsługuje przycisk, jest on na pinie 0. W pętli sterującej while(true) sprawdzany jest stan przycisku za pomocą funkcji Read() klasy InterruptPort. Jeżeli zwróci true, czyli przycisk wciśnięty, włączy się czerwona i niebieska dioda, pozostałe zostaną wyłączone. W przypadku kiedy funkcja zwraca false- zielona i pomarańczowa dioda zostanie włączona, a pozostałe zostaną wyłączone.

#### 3.2 LED

Obsługa diod LED zaimplementowana jest w przykładzie "LED\_GPIO". Posiada on dodatkową referencję Microsoft.SPOT.Hardware. Klasa OutputPort obsługuje diody, występują one na pinach 60-63. W pętli sterującej while(true) wszystkie diody są włączane, a potem wyłączane. Stan diod zmienia się za pomocą metody Write(bool) klasy OutputPort(true-włącz, false-wyłącz). Aby stan diod nie zmieniał się za szybko, dodane są proste pętle for(int i = 0; i < 100000; i++){}

#### 3.3 PWM

Obsługa PWM oraz usypiania wątku zaimplementowana jest w przykładzie "LED\_PWM". Przykład posiada dwie dodatkowe referencje: Microsoft.SPOT.Hardware i Microsoft.SPOT.Hardware.PWM. Klasa PWM obsługuje diody, występują one na kanałach PWM od 0 do 3. W pętli sterującej while(true) diody są po kolei włączane i zmniejsza się ich moc świecenia za pomocą zmiennej DutyCycle klasy PWM. Diody są włączane z mocą 100% i zmniejsza się ich moc o 10% co sekundę aż do 0. Czekanie zaimplementowane jest przez uśpienie wątku: Thread.Sleep(1000), liczba podana jako atrybut to czas podany w milisekundach.

### 3.4 Zegar czasu rzeczywistego

Obsługa zegaru czasu rzeczywistego oraz debuggera zaimplementowana jest w przykładzie "RTC". W pętli sterującej `while(true)` sprawdzane są sekundy z aktualnego czasu (zmienna `DateTime.Now.Second`), co sekundę w debuggerze wyświetlana jest wartość za pomocą metody `Debug.Print(string)`, za argument przyjmuje to co ma wyświetlić w debuggerze. W sekundach parzystych wyświetlana jest aktualna ilość przeskoków zegara, w nieparzystych sekundach wyświetlana jest ilość przeskoków zegara wykonanych w ciągu sekundy.

### 3.5 SPI-Akcelerometr

Obsługa akcelerometru za pomocą SPI, LED oraz obsługa wątków zaimplementowane są w przykładzie "SPIAccelerometer". Posiada on dodatkową referencję `Microsoft.SPOT.Hardware`. Klasa SPI obsługuje akcelerometr, jest on na pinie 67. W przykładzie zaimplementowana jest metoda służąca do zapisywania do akcelerometru `void WriteRegister(byte register, byte data)` i funkcja do odczytywania z akcelerometru `byte ReadRegister(byte register)`. Argument `register` jest adresem rejestru, argument `data` to dane dla akcelerometru. W pętli sterującej `while(true)` sprawdzane są wartości akcelerometru i na ich podstawie włącza się diody. Program włącza diody po tej stronie, w którą pochylony do dołu jest mikrokontroler. W przypadku odwrócenia górą do dołu mikrokontrolera, włącza się wszystkie diody. Na końcu pętli wątek usypiany jest na 100 milisekund aby zmniejszyć ilość niepoprawnych wyników.

### 3.6 Timer

Obsługa timera oraz LED zaimplementowana jest w przykładzie "Timer\_Function". Posiada on dodatkową referencję `Microsoft.SPOT.Hardware`. Klasa `Timer` obsługuje timer, w konstruktorze przyjmuje argumenty jak: funkcja, którą ma wywoływać co tyle milisekund ile jest podanych w ostatnim argumencie. Przedostatni argument konstruktora to opóźnienie z jakim ma być wykonana funkcja. Pętla sterująca `while(true)` istnieje aby program się nie zakończył. Program co sekundę w funkcji wywoływanej przez timer, zmienia stan diod(włączone, wyłączone).