.NET Micro Framework STM32F4 Discovery

Wojciech Duda

2016.4.21

Spis treści

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.

Rysunek 1: Opis urządzenia

2 Instalacja

2.1 Narzędzia:

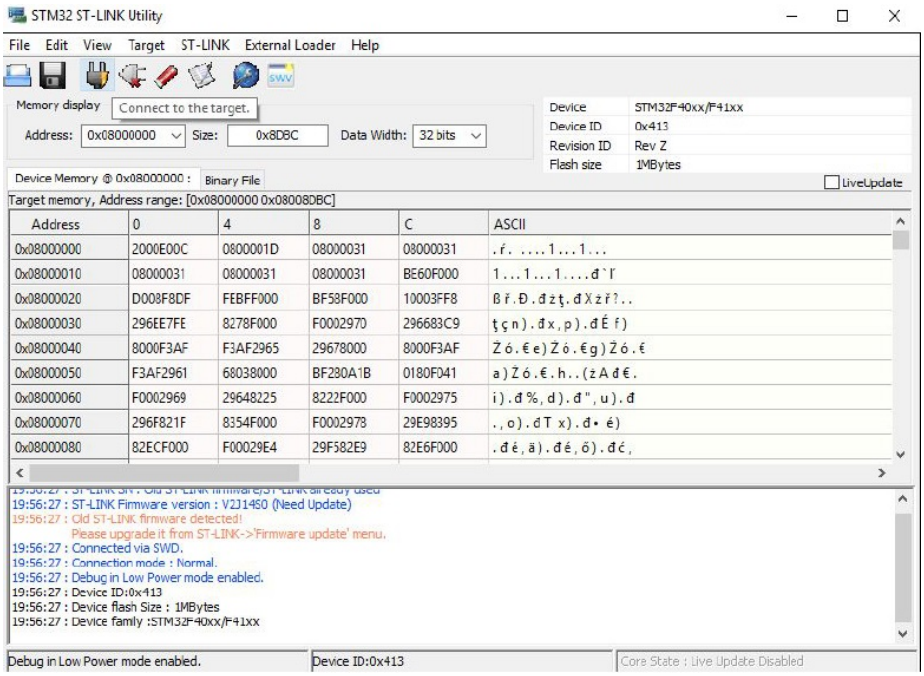
- mikrokontroler STM32F4 Discovery
- kable USB Micro oraz USB Mini
- Visual studio
- STM32 ST-LINK Utility
- sterwonik USB
- bootloader oraz pliki hex
- .NET MicroFramework SDK

2.2 Konfiguracja

1. Zainstaluj STLINK, oraz SDK, resztę plików rozpakuj.

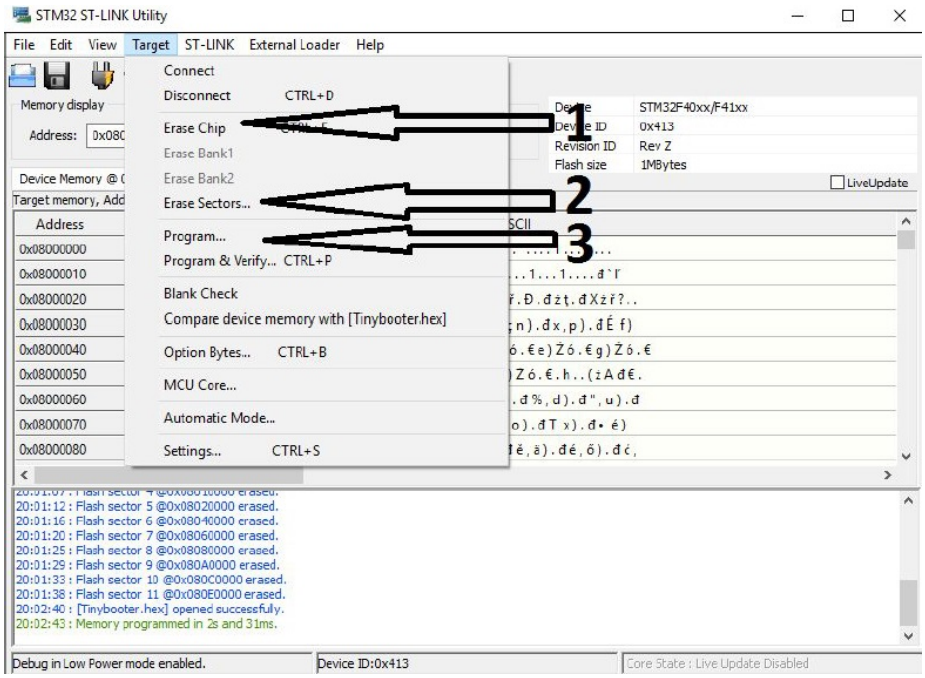
2. Podłącz kabel USB Mini (do wejścia oznaczonego jako “Złącze USB” na zdjęciu powyżej.)

3. Włącz STLINK Utility , a następnie połącz się z stm32f4 poprzez przycisk: “Connect to the = target”



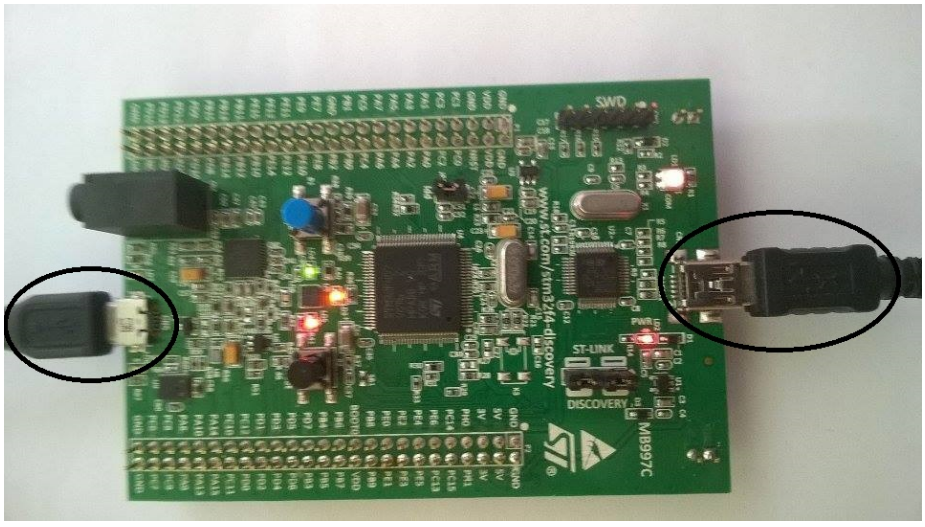
Rysunek 2: STLINK Utility

4. 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.



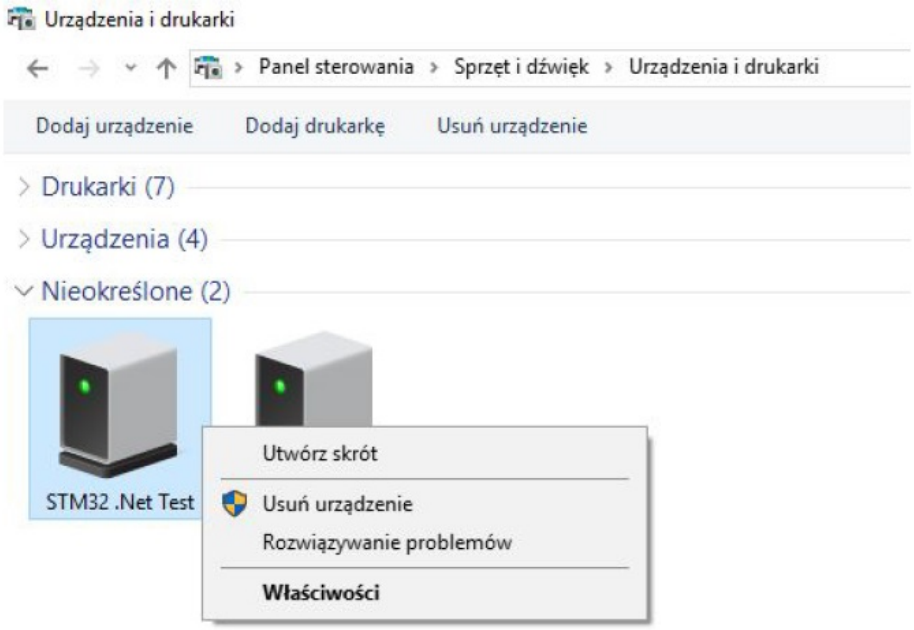
Rysunek 3: Programowanie debuggera

5. Jeżeli wszystko przebiegło prawidłowo powinny zapalić się 3 diody użytkowe. Podłącz kabel micro USB (jak na rysunku 4).



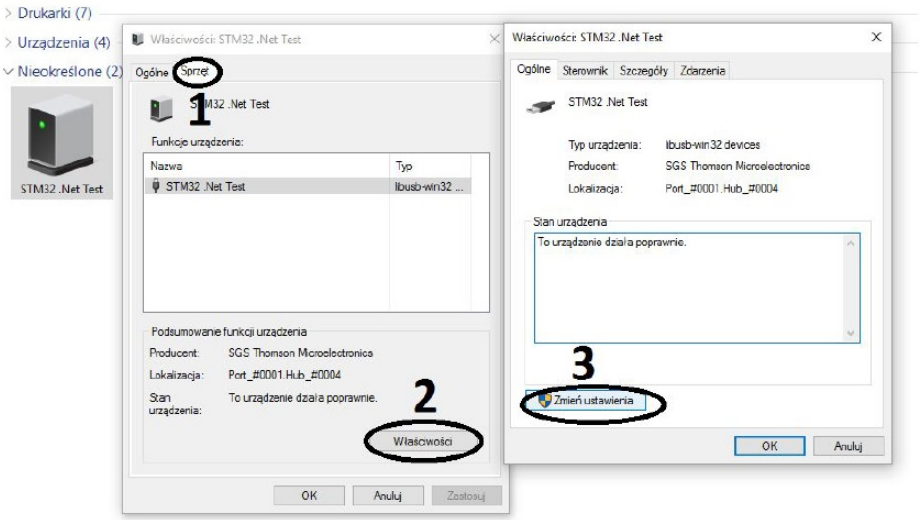
Rysunek 4: Podłączony STM32F4 kablami mikro i mini USB

6. Przejdz do “urządzenia i drukarki”. Tam w obszarze“ nieokreślone” kliknij prawym przyciskiem myszy w “STM .Net Test” i wybierz właściowości.

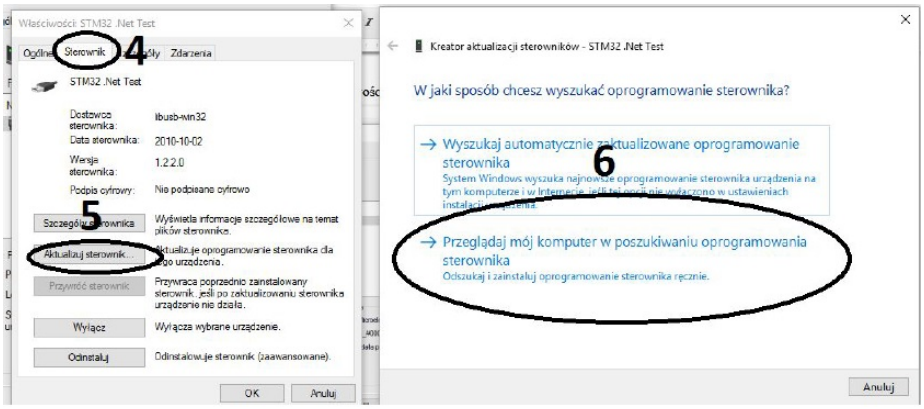


Rysunek 5: Urządzenia i drukarki

7. Wejdź w sprzęt >właściowości >zmień ustawienia >sterownik >Aktualizuj sterownik...



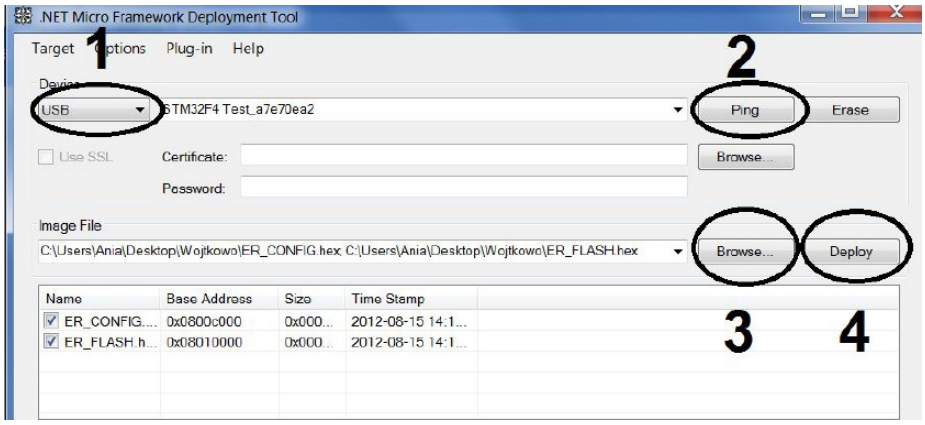
Rysunek 6: Instalacja sterownika krok 1



Rysunek 7: Instalacja sterownika krok 2

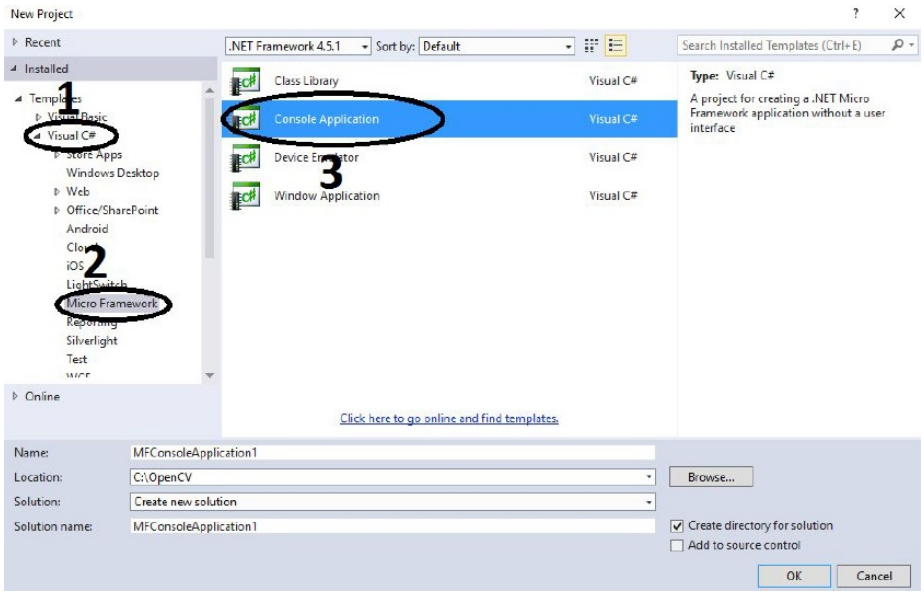
8. Wybierz “Przeglądaj mój komputer w poszukiwaniu oprogramowania sterownika” i wybierz ścieżkę gdzie rozpakowałeś na początku sterownik. Podczas instalacji ignoruj ostrzeżenia.

9. Uruchom MFDeploy. Wybierz Device: USB. Naduś przycisk Ping. Następnie drugie od góry Browse... , wybierz ścieżkę pozostałych dwoch plików hex: ER_CONFIG.hex, ER_FLASH.hex oraz wybierz Deploy.



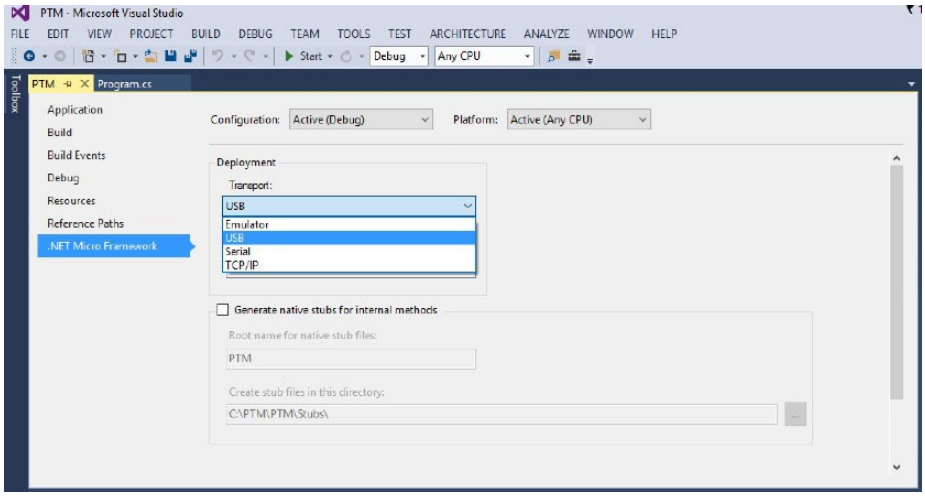
Rysunek 8: MF Deploy

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



Rysunek 9: Tworzenie projektu

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



Rysunek 10: Konfigurowanie Visual Studio

3 Opis przykładów

Przykłady są na stronie: 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 pozsotałe 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 usypaiania wątku zaimplementowana jest w przykładzie "LED_PWM". 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 debugerze wyświetlana jest wartość za pomocą metody Debug.Print(string), za argument przyjmuje to co ma wyświetlić w debugerze. 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 watków zaimplementowane są w przykładzie "SPI_Accelerometer". Posiada on dodatkową referencję Microsoft.SPOT.Hardware. Klasa SPI obsługuje akcelerometr, jest on na pinie 67. W przykładzie zaimplementowana jest moteda 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, włączą się wszystkie diody. Na końcu pętli wątek usypiany jest na 100 milisekund aby zmniejszyć ilość niepoprawnych wyników.

3.6 Timer

Obsługa timera 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 whilte(true) istnieje aby program się nie zakończył. Program co sekundę w funkcji wywoływanej przez timer, zmienia stan diod(włączone, wyłączone).

4 Opis implementacji

4.1 Klasa OutputPort

Klasa zdefiniowana w przestrzeni nazw Microsoft.SPOT.Hardware.

4.1.1 Referencje

- Microsoft.SPOT.Hardware

4.1.2 Konstruktor

OutputPort (Pin portId,bool initialState)

- portId - identyfikator portu.
- initialState - stan początkowy na porcie po aktywacji.

4.1.3 Funkcje

void Write(bool state) - wpisuje wartość do portu.

- state - wartość wpisywana do portu.

4.2 Klasa InterruptPort

Klasa zdefiniowana w przestrzeni nazw Microsoft.SPOT.Hardware.

4.2.1 Referencje

- Microsoft.SPOT.Hardware

4.2.2 Konstruktor

InterruptPort (Pin portId, bool glitchFilter, ResistorMode resistor, InterruptMode interrupt)

- portId - identyfikator portu.
- glitchFilter, - obsługa filtra błędów: true -włączony, false-wyłączony
- resistor - tryb rezystora, który określa stan domyślny dla portu.
- interrupt - tryb przerwania, który określa warunki wymagane do generowania przerwania.

4.2.3 Funkcje

bool Read () - zwraca aktualną wartość portu.

4.3 Klasa Cpu

Klasa zdefiniowana w przestrzeni nazw Microsoft.SPOT.Hardware.

4.3.1 Referencje

- Microsoft.SPOT.Hardware

4.3.2 Atrybuty

- enum Cpu.Pin - identyfikuje porty werjścia/wyjścia.
- enum Cpu.PWMChannel.PWM_X - identyfikuje kanał PWM o numerze X.

4.4 Klasa PWM

Klasa zdefiniowana w przestrzeni nazw Microsoft.SPOT.Hardware.

4.4.1 Referencje

- Microsoft.SPOT.Hardware.PWM
- Microsoft.SPOT.Hardware

4.4.2 Konstruktor

PWM (PWMChannel channel, UInt32 period_us, UInt32 duration_us, bool invert)

- channel - kanał PWM
- period_us - okres pulsowania w mikrosekundach.
- duration_us - czas pulsu w microsekundach.
- invert - Wartość, która wskazuje, czy wyjście jest odwrócone.

4.4.3 Atrybuty

double DutyCycle - Pobiera lub ustawia cykl pracy impulsu jako wartość od 0.0 do 1.0.

4.4.4 Funkcje

void Start () - Uruchamia port PWM na nieokreślony czas.

4.5 Klasa SPI

Klasa zdefiniowana w przestrzeni nazw Microsoft.SPOT.Hardware.

4.5.1 Referencje

- Microsoft.SPOT.Hardware

4.5.2 Konstruktor

SPI (Config)

- Config - Konfiguracja interfejsu SPI

4.5.3 Funkcje

-void Write (byte[] writeBuffer) - wpisuje block danych do interfejsu.

- writeBuffer - buffor który zostanie zapisany do interfejsu.

-void WriteRead (byte[] writeBuffer,ref byte[] readBuffer)

- writeBuffer - buffor który zostanie zapisany do interfejsu.
- readBuffer - buffor do którego zostaną zapisane dane odczytane z interfejsu.

4.6 Klasa SPI.Configuration

Klasa zdefiniowana w przestrzeni nazw Microsoft.SPOT.Hardware.

4.6.1 Referencje

- Microsoft.SPOT.Hardware

4.6.2 Konstruktor

SPI.Configuration (Pin ChipSelect_Port,bool ChipSelect_ActiveState,
UInt16 ChipSelect_SetupTime, UInt16 ChipSelect_HoldTime, bool Clock_IdleState,
bool Clock_Edge, UInt16 Clock_Rate, SPI_module SPI_mod)

- ChipSelect_Port - Port wybranego czipu.
- ChipSelect_ActiveState - Stan aktywny dla portu wybranego czipu. Jeżeli prawda port będzie ustawiany na wysoki w momencie dostępu do czipu, jezeli fałsz port będzie ustawiany na niski w momencie dostępu do czipu.
- ChipSelect_SetupTime - Czas pomiędzy wybraniem urządzenia a momentem kiedy zegar rozpocznie transakcje.
- ChipSelect_HoldTime - Określa jak długo port czipu musi zostać w stanie aktywnym po zakończeniu transakcji czytania lub pisania.
- Clock_IdleState - Stan bezczynności zegara. Jeśli prawda, sygnał zegara SPI zostanie ustawiony na wysoki, gdy urządzenie jest w stanie spoczynku. Jeśli fałsz, sygnał zegara SPI zostanie ustawiony na niski, gdy urządzenie jest w stanie bezczynności.
- Clock_Edge - Jeśli prawda, dane są próbkowane na zboczu wznoszącym zegara SPI. Jeśli fałsz, dane są próbkowane na zboczu opadającym zegara SPI.
- Clock_Rate - Częstotliwość zegara SPI w Hz.
- SPI_mod - Magistrala SPI używana do transakcji.

4.7 Klasa Timer

Klasa zdefiniowana w przestrzeni nazw System.Threading.

4.7.1 Konstruktor

Timer(TimerCallback callback, object state, uint dueTime, uint period)

- callback - nazwa metody która ma być wykonywana.
- state - obiekt z informacjami wykorzysytwanych w metodzie callback lub null.
- dueTime - opóźnienie z jakim będzie wywoływać się metoda callback, podane w millisekundach.
- period - czas między wywołananimy metody callback, podany w millisekundach.

4.8 Klasa DateTime

Klasa zdefiniowana w przestrzeni nazw Microsoft.SPOT.

4.8.1 Atrybuty

- DateTime.Now.Second - zwraca sekundy z aktualnego czasu. Przyjmuje wartości od 0 do 59.
- DateTime.Now.Ticks - zwraca aktualną ilość przeskoków zegara.

4.9 Klasa Thread

Klasa zdefiniowana w przestrzeni nazw System.Threading.

4.9.1 Funkcje

void Sleep(int millisecondsTimeout) - uśpienie wątku.

- millisecondsTimeout - czas na jaki ma być uśpiony wątek podany w milisekundach.