Nowe peryferia Microchipa (1)

Jak ponownie polubić 8-bitowce

Dominacja mikrokontrolerów 32-bitowych stała się faktem. Umysłami konstruktorów zawładnęły 32-bitowe jednostki centralne, rozbudowane peryferia, olbrzymie (jak na układy embeded) pamięci programu i RAM. A mimo wszystko są firmy, który w tym 32-bitowym świecie proponują inne rozwiązania.

Współcześnie, nawet jeśli ktoś chce zbudować urządzenie o skromniejszych możliwościach, to i do niego wkłada okrojoną, 32-bitową jednostkę z mniejszą pamięcią, uboższymi peryferiami i za relatywnie niższą cenę. Z jednej strony to dobrze, bo raz poznane architektura i narzędzia można stosować w wielu aplikacjach. Jednak z drugiej strony, wykreowany brak alternatyw na zmonopolizowanym rynku może wywołać stagnację. Microchip – firma z olbrzymim potencjałem, która od zawsze szła swoja drogą, czasami pod prąd ogólnym tendencjom, wymyśliła sposób na drugą młodość 8-bitowych mikrokontrolerów rodziny PIC16F. Pomysł był tak prosty, jak to tylko możliwe: bierzemy sprawdzony rdzeń i dodajemy specjalne, zaawansowane układy peryferyjne. Pierwsze pytanie jakie się nasuwa: to jakie to są te zaawansowane układy peryferyjne, że konstruktor rozważy wykorzystanie staruszka PIC16F zamiast na przykład nowoczesnego Cortexa M? Przyjrzyjmy się im.

PIC16LF1507 i MPLAB X Code Configurator.

Nic tak dobrze nie ilustruje działania mikrokontrolera i jego układów peryferyjnych jak konkretne przykłady. Dlatego postanowiłem początkowo wykorzystać będący w moim posiadaniu mikrokontroler PIC16LF1507.

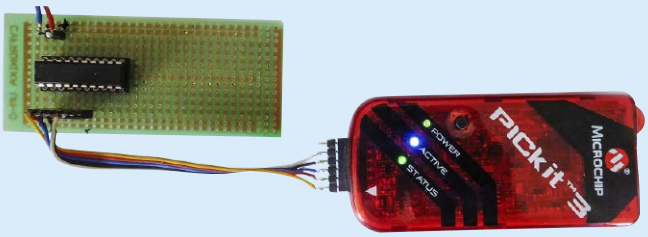
Układ procesorowy PIC16LF1507 został zbudowany zgodnie z wyżej przedstawiona ideą: do rdzenia PIC16F dołączono kilka bardzo interesujących układów peryferyjnych, które to trudno znaleźć w układach innych producentów. Mikrokontroler został umieszczony w podstawce na małej, uniwersalnej płytce drukowanej. Oprócz podstawki na płytce zamontowałem złącze dla programatora-debuggera PICKit3 i całość zasiliłem napięciem +3,3 V. Większość mikrokontrolerów PIC16LF może być zasilana napięciem +5 V. Niestety – PIC16LF jest wyjątkiem, ponieważ jego maksymalne napięcie zasilające wynosi +4 V. Ten układ najlepiej zasilić napięciem +3,3 V, bo przy +5 V może ule uszkodzeniu.

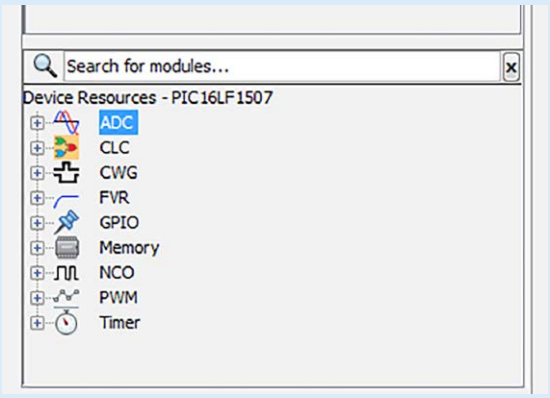
Do testów wykorzystamy bezpłatne środowisko projektowe MPLAB X IDE (v3.05) oraz kompilator C MPLAB XC8 w wersji bezpłatnej (bez optymalizacji kodu). Do programowania pamięci użyjemy firmowego PICkit3.

Pakiet IDE MPLAB X ma możliwość uruchamiania wtyczek (plug-in). Jedną z takich wtyczek jest firmowy *MPALB Code Configurator*. To program narzędziowy, który bardzo ułatwia życie programistom PICów. Jak wiadomo, układy peryferyjne są konfigurowane i sterowane przez zapisywanie i odczytywanie rejestrów konfiguracyjnych umieszczonych w przestrzenia pamięci RAM. Im bardziej rozbudowany układ, tym więcej rejestrów. A więcej rejestrów oznacza więcej wysiłku potrzebnego do ich prawidłowego zaprogramowania. Programowanie rejestrów konfiguracyjnych to żmudna i niezbyt lubiana praca, ale od jej poprawnego wykonania zależy działanie aplikacji. *Code Configurator* znakomicie pomaga w kompletnym skonfigurowaniu układów peryferyjnych mikrokontrolera. Oczywiście skonfigurowany układ trzeba potem programowo obsługiwać, ale to już zadanie dla programisty.

W pierwszym przykładzie użyjmy *Code Configuratora* do skonfigurowania układu taktowania i zaprogramowania bezpieczników (fuse). Narzędzie uruchamiamy z okna *Tools* ➞ *Embeded* (oczywiście, wcześniej musi być ono pobrane i zainstalowane). Po uruchomieniu, z lewej strony ekranu MPLAB X IDE zostanie wyświetlone okno *MPLAB Code Configurator Resources* podzielone na 2 części (**rysunek 1**):

1. *Device Resources* – PIC16LF1507.





Rysunek 1. Okno Device Resources wtyczki Code Configurator

2. *Project Resources*.

W oknie *Device Resources* jest umieszczona lista symboli graficznych wszystkich układów peryferyjnych mikrokontrolera, a w oknie *Project Resources* układy peryferyjne wybrane z listy do konfigurowania.

Taktowanie mikrokontrolera

Taktowanie mikrokontrolera jest konfigurowane z zasobu System. Po kliknięci na ikonę system w okienku *Project Resources* zostaną wyświetlone okienka *Clock* do ustawiania źródła i częstotliwości taktowania oraz okno do konfigurowania bitów konfiguracyjnych mikrokontrolera *Generate Configuration Bits* (fuse). W pierwszym kroku ustawiamy taktowanie mikrokontrolera. Standardowo CPU może być taktowane przez wewnętrzny oscylator RC lub przez zewnętrzny przebieg taktujący z osobnego oscylatora (**rysunek 2**). Co ciekawe, producent nie przewidział taktowania z wewnętrznego oscylatora kwarcowego. Za to taktowanie z wewnętrznego HFINOSC oscylatora daje możliwość wyboru wielu częstotliwości z zakresu od 31 kHz do 16 MHz. Wybierzmy HFINTOSC i częstotliwość 16 MHz, jak pokazano na **rysunku 3**. Po kliknięciu na przycisk Generate Code Configurator wygeneruje funkcję inicjalizującą *OSCILLATOR_Initialize()* pokazaną na **listingu 1**.

Przy okazji ustawiania oscylatora skonfigurujemy bity w oknie *Generate Configuration Bits*. Z ważniejszych ustawiamy: *PWRTE ON*, *MCLRE ON*, *WDTE OFF*. Bity wyboru oscylatora *FOSC INTOSC* są automatycznie ustawiane lub zerowane przy konfiguracji okna *Clock*. Na podstawie ustawionych bitów *MPLAB Code Configurator* wygeneruje fragment programu pokazany na **listingu 2**.

Listing 1. Konfigurowanie oscylatora

```
void OSCILLATOR_Initialize(void) {
    // SCS INTOSC; IRCF 16MHz_HF;
    OSCCON = 0x7A;
    // HFIOFR disabled; HFIOFS not0.5percent_acc; LFIOFR disabled;
    OSCSTAT = 0x00;
    // Set the secondary oscillator
}
```

Listing 2. Ustawienie bitów konfiguracyjnych

```
// Configuration bits: selected in the GUI
// CONFIG1
#pragma config BOREN = ON    // Brown-out Reset Enable->Brown-out Reset enabled
#pragma config PWRTE = ON    // Power-up Timer Enable->PWRT enabled
#pragma config FOSC = INTOSC// Oscillator Selection Bits->Internal Oscillator, I/O function on OSC1
#pragma config MCLRE = ON    // MCLR Pin Function Select->MCLR/VPP pin function is MCLR
#pragma config CP = OFF      // Flash Program Memory Code Protection->Program memory code protection is
disabled
#pragma config WDTE = OFF    // Watchdog Timer Enable->WDT disabled
#pragma config CLKOUTEN = OFF // Clock Out Enable->CLKOUT function is disabled. I/O or oscillator function
on the CLKOUT pin
// CONFIG2
#pragma config WRT = OFF     // Flash Memory Self-Write Protection->Write protection off
#pragma config LPBOR = OFF   // Low-Power Brown Out Reset->Low-Power BOR is disabled
#pragma config LVP = OFF     // Low-Voltage Programming Enable->High-voltage on MCLR/VPP must be used for
programming
#pragma config STVREN = ON   // Stack Overflow/Underflow Reset Enable->Stack Overflow or Underflow will cause
a Reset
#pragma config BORV = LO     // Brown-out Reset Voltage Selection->Brown-out Reset Voltage (Vbor), low trip
point selected.
```

Listing 3. Funkcja inicjująca moduł NCO

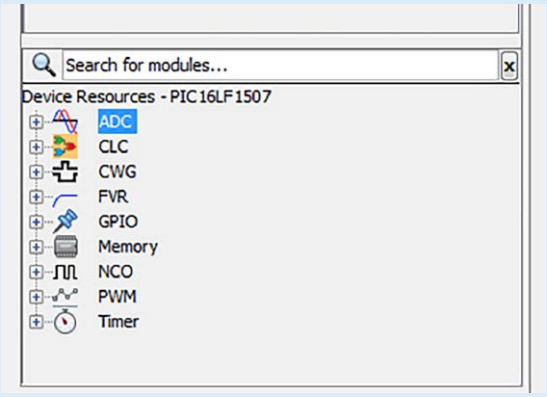
```
void NCO1_Initialize(void) {
    // Set the NCO to the options selected in the GUI
    // N1OUT out_lo; N1PFM FDC_mode; N1POL active_hi; N1EN disabled; N1OE enabled;
    NCO1CON = 0x40;
    // N1PWS 1_clk; N1CKS HFINTOSC_16MHz;
    NCO1CLK = 0x00;
    // NCOACCU 0;
    NCO1ACCU = 0x00;
    // NCOACCH 0;
    NCO1ACCH = 0x00;
    // NCOACCL 0;
    NCO1ACCL = 0x00;
    // NCO1INCH 0
    NCO1INCH = 0;
    // NCO1INCL 131
    NCO1INCL = 131;
    // Enable the NCO module
    NCO1CONbits.N1EN = 1;
}
```

Generator NCO

Przechodzimy teraz do pierwszego z zaawansowanych modułów peryferyjnych – generatora NCO. Moduł generatora NCO jest przeznaczony do użycia w aplikacjach, które wymagają generowania określonej, dokładnej częstotliwości. NCO pozwala również na precyzyjne regulowanie tej częstotliwości. Częstotliwość przebiegu cyfrowego można łatwo dzielić używając do tego celu liczników. W typowym rozwiązaniu do zmiany częstotliwości wykorzystuję się liczniki. Moduł licznika jest podstawowym układem peryferyjnym i zawiera go każdy mikrokontroler, nawet najprostszy rodziny PIC10F. Standardowo, są to liczniki 8- lub 16-bitowe, wyposażone w preskalery i niekiedy w postskalery.

To czy uda nam się wygenerować założoną częstotliwość zależy od częstotliwości sygnału na wejściu i współczynnika podziału. Jeżeli wyliczony współczynnik podziału jest liczbą całkowitą, to można uzyskać dokładnie taką częstotliwość przebiegu, jakiej oczekujemy. Problem pojawia się, kiedy potrzebujemy częstotliwości 130 kHz korzystając z kwarcu (generatora) 20 MHz. Wtedy 2000000/130000 w przybliżeniu 15,38461538. Po podzieleniu przez 15 otrzymujemy 1333333,3333 Hz. Takie proste dzielenie jest albo obarczone sporym błędem, albo wymaga dobierania częstotliwości wejściowej, tak by współczynnik podziału był całkowity.

Microchip w module NCO zastosował pomysłowe połączenie licznika z układem modyfikującym jego wartość w czasie każdego okresu zliczanego przebiegu. Żeby zapewnić odpowiednią rozdzielczość pomiaru licznik ma długość 20 bitów, a modyfikowana wartość jest 16-bitowa. Na **rysunku 6** pokazano schemat blokowy modułu NCO. Sygnał wejściowy jest wybierany selektorem programowanym bitami NxCKS. Można wybrać przebieg dostępny na wejściu NCOCLK, wyjścia wbudowanego układu logicznego (CLC) LC1OUT, zewnętrzny przebieg taktujący Fosc lub wewnętrzny oscylator HFINTOSC o maksymalnej częstotliwości 16 MHz. Wybrany przebieg jest zliczany przez 20-bitowy licznik-akumulator. W czasie każdego narastającego zbocza



Rysunek 2. Układ taktowania PIC16LF1507

sygnału wejściowego jest do niego dodawana 16-bitowa wartość zapisana przez użytkownika w rejestrze NCOxINCL.

$$F_{OVERFLOW} = \frac{NCO\ Clock\ Frequency\ \times Increment\ Value}{2^n},$$

Częstotliwość przepełniania się licznika- akumulatora wylicza się z równania

gdzie :

NCO Clock Frequency: częstotliwość wejściowa.

Increment Value: wartość dodawana do akumulatora.

n: liczba bitów akumulatora, tu $n=20$.

Na **rysunku 7** pokazano przykład zliczania z dodawaniem wartości 0x2000 przy każdym narastającym zboczu przebiegu wejściowego. 20-bitowy licznik-akumulator przepełni się po 8 cyklach (okresach) przebiegu wejściowego. Wybierając kombinację częstotliwości wejściowej i zawartości rejestru NCOxINCL można precyzyjnie ustalić generowaną częstotliwość.

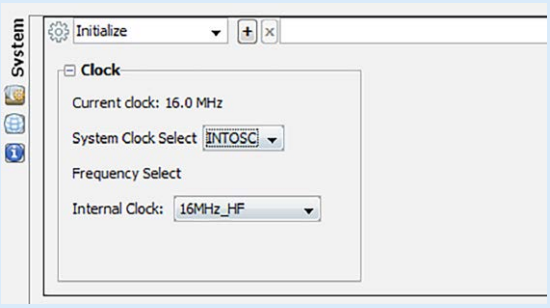
Przepełnienie licznika powoduje zmianę stanu przerzutnika D. Na jego wyjściu Q jest sygnał prostokątny o wypełnieniu 50% i częstotliwości równej połowie częstotliwości przepełnienia licznika. Zapisując bit NxPOL można zmieniać polaryzację sygnału wyjściowego. Przez selektor wyjściowy jest on podawany na wyprowadzenie NCOx, ale jest też dostępny wewnętrznie dla bloków peryferyjnych CLC i CWG. Ten tryb pracy – wybierany wyzerowaniu bitu NxPFM – nazywa się Fixed Duty Cycle (FDC).

Każde przepełnienie licznika – akumulatora powoduje ustawienie znacznika przerwania NCOxIF. Jeżeli przerwanie nie jest zablokowane, to zostanie zgłoszone i musi być obsłużone.

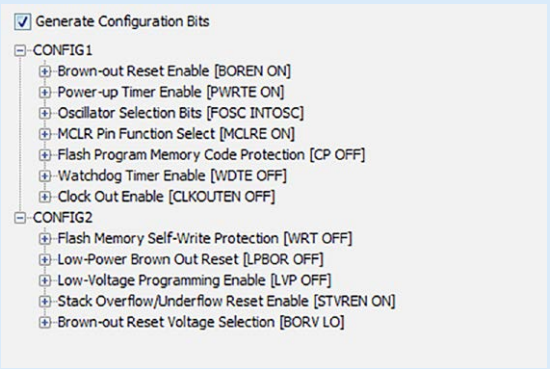
Ustawienie bitu NxPFM wprowadza tryb Pulse Frequency Mode (PF). W tym trybie po każdym przepełnieniu się licznika-akumulatora wyjście NCOx jest ustawiane na czas równy zaprogramowanej wielokrotności okresu zliczanego sygnału wejściowego (zliczany przez *ripple counter*). Czas trwania poziomu aktywnego (wysoki) i nieaktywnego (niski) zależy od bitu określającego polaryzację NxPOL i wartości wpisanej do bitów NxPWS rejestru konfiguracyjnego NCOxCON. Na **rysunku 8** pokazano zależność czasu trwania stanu aktywnego dla NxPWS=0 i dla NxPWS=2. Podobnie jak w trybie FDC, sygnał wyjściowy może być podawany na wyjście NCOx przez selektor wyjściowy lub wewnętrznie do modułów CLC i CFG.

Programowanie NCO jest skomplikowane i wymaga sporo pracy. Lwią jej część może za nas wykonać MPLAB Code Configurator. Po kliknięciu w okienku *Device Resources* na ikonkę *NCO* jest ona automatycznie przenoszona do okienka *Project Resources*. Teraz w tym oknie klikamy na *NCO* i otwiera się okno inicjalizacji modułu pokazane na **rysunku 9**. Przewidziano tu następujące ustawienia:

- **Enable NCO**. Zaznaczenia powoduje włączenie modułu.
- **Enable Pin Out**. Zaznaczenia umożliwia dołączenie sygnału z wyjścia NCO do wyprowadzenia mikrokontrolera.
- **Clock Source**. Wybór zegara taktującego moduł NCO. Można tu wybrać przebieg taktujący mikrokontroler (z wyjścia generatora HFINTOSC lub z wejścia zegara taktującego FOSC), z wyprowadzenia, na które zostanie podany zewnętrzny przebieg taktujący lub z wyjścia modułu Configurable Logic Cell CLC1 (CLC1 OUT). Tutaj wybieramy przebieg z wewnętrznego generatora RC o częstotliwości 16 MHz taktującego mikrokontroler.
- **Output Polarity**. Zanegowanie sygnału wyjściowego.
- **NCO Mode**. Wybór trybu *Pulse Frequency* (PF) lub *Fixed Duty Cycle* (FDC) – sygnał wyjściowy o wypełnieniu 50%. Po wyborze trybu PF w oknie *Pulse Width Modulation Clocks* ustala się czas trwania impulsu w cyklach zegarowych (1…128).



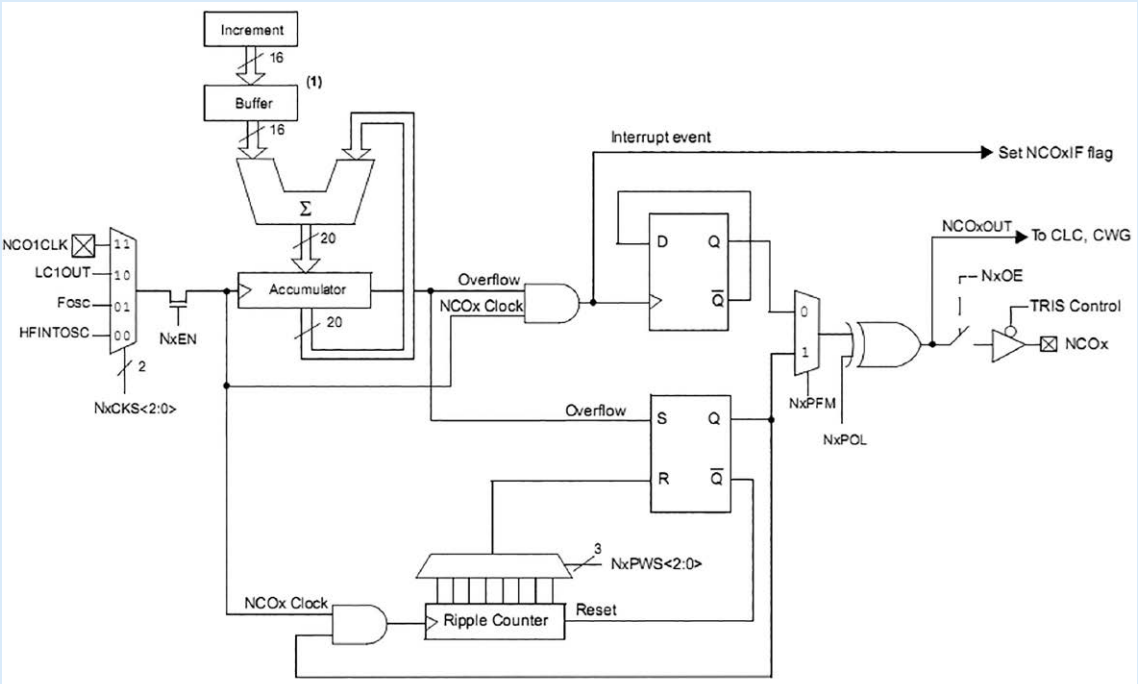
Rysunek 3. Konfiguracja układu taktowania



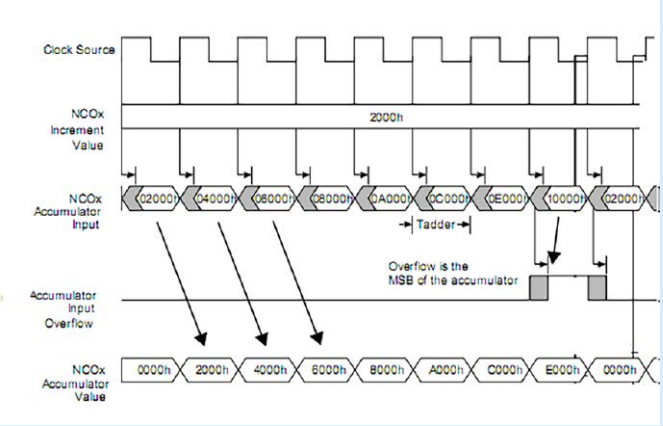
Rysunek 4. Bity konfiguracyjne

Listing 4. Funkcja inicjująca wyprowadzenia

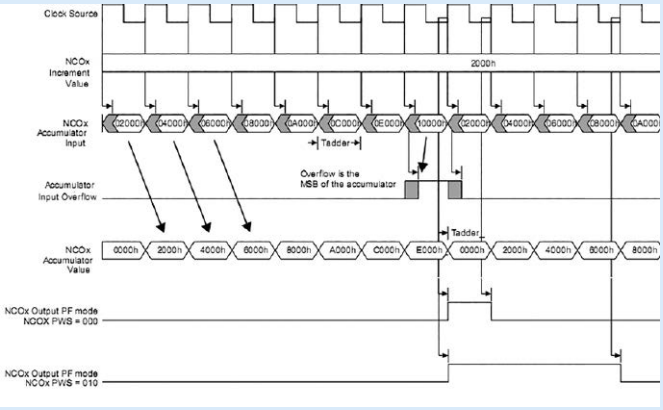
```
void PIN_MANAGER_Initialize(void) {
    LATA = 0x00;
    TRISA = 0x3F;
    ANSELA = 0x17;
    WPUA = 0x00;
    LATB = 0x00;
    TRISB = 0xF0;
    ANSELB = 0x30;
    WPUB = 0x00;
    LATC = 0x00;
    TRISC = 0xFD;
    ANSELC = 0xCD;
    OPTION_REGbits.nWPUEN = 0x01;
    APFCON = 0x00;
}
```



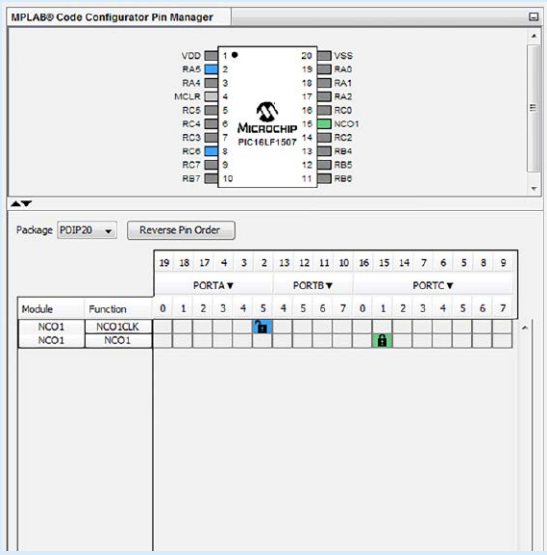
Rysunek 5. Schemat blokowy NCO



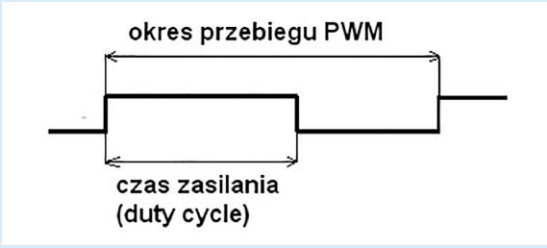
Rysunek 6. Przykład zliczania z dodawaniem 0x2000



Rysunek 7. Generowanie sygnału wyjściowego w trybie PF



Rysunek 9. Przypisane wyjścia do NCO za pomocą Configurator Pin Manager'a



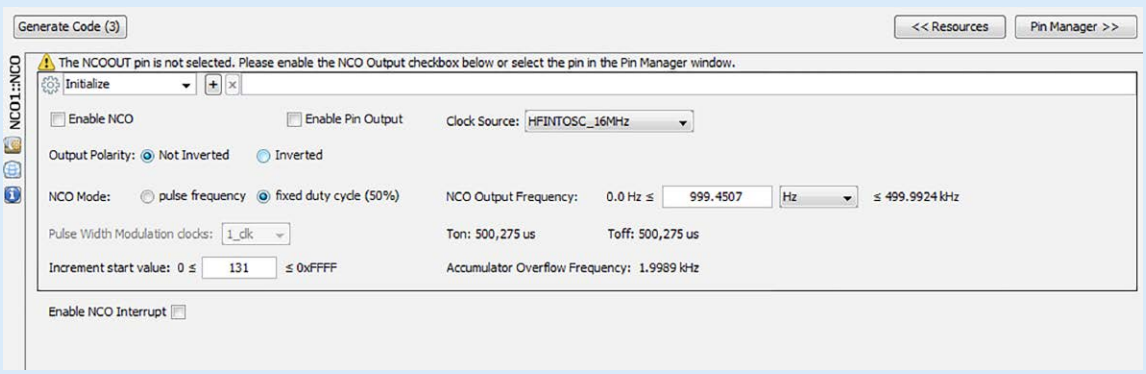
Rysunek 10. Przebieg PWM

- **NCO Output Frequency**. W tym polu wpisuje się generowaną częstotliwość w Hz, kHz, lub MHz.
- **Increment start value**. Wartość wpisywana do rejestru NCO1INCL. Po zapisaniu wartości częstotliwości w oknie *NCO Output Frequency*, wartość okna *Increment start value* jest wyliczana automatycznie.

Po kliknięciu na Generate Code MPLAB Code Confugurator generuje plik **nco1.c** z funkcją **NCO1_Inicjalize()** pokazaną na **listingu 3**. Skonfigurowany poprawnie moduł powinien generować przebieg prostokątny o częstotliwości 1 kHz. Jeżeli w konfiguracji zostanie zaznaczona opcja *Enable Output Pin*, to sygnał wyjściowy z generatora będzie mógł być wyprowadzony na dedykowany pin mikrokontrolera. Aby to było możliwe, trzeba skonfigurować to wyprowadzanie jako wyjście NCO. MPLAB Code Configurator ma specjalne narzędzie do konfigurowania wyprowadzeń – *Pin Manager*. Po kliknięciu na przycisk *Pin Manager* w oknie *Package* trzeba wybrać obudowę mikrokontrolera i zostanie wyświetlony rysunek obudowy z zaznaczonymi na niebiesko wyprowadzeniami, które można przypisać do NCO. Wyprowadzenie RA5 może być wejściem sygnału zegarowego, a jedno z wyprowadzeń RC6, lub RC1 wyjściem sygnału z NCO. Ponieważ ustawiliśmy zliczanie impulsów z HFINTOSC, to wejście NCO nie będzie konfigurowane. Wyjście konfigurujemy klikając prawym klawiszem myszki na wyprowadzenia RC1, lub RC6 (zależnie od potrzeb). Po wybraniu na przykład RC1 i zatwierdzeniu kolor wyprowadzenia zmienia się z niebieskiego na zielony. Kolor zielony oznacza, że wyprowadzenie jest przypisane do modułu, a kolor niebieski, ze wyprowadzenie jest potencjalnie przeznaczone do modułu. – zostało to pokazane na rysunku 10. *Configurator Pin Manager* generuje plik **pin_manager.c** z funkcją **PIN_MANAGER_Initialize()** pokazana na **listingu 4**. Działanie NCO ze skonfigurowanym wyjściem można przetestować w naszym układzie testowym. Program po skompilowaniu należy zapisać za pomocą PICKit3 w pamięci programu mikrokontrolera. Jeżeli wszystko zostało wykonane prawidłowo, to na wyprowadzeniu RC1 powinien pojawić się sygnał prostokątny o wypełnieniu 50% i częstotliwości 1000 Hz. Można poeksperymentować z ustawieniami częstotliwości lub wprowadzić tryb PF i zobaczyć, jak zachowuje się układ. Nawet tak nieskomplikowany program można po odpowiednim rozbudowaniu wykorzystać do realizacji generatora przebiegu prostokątnego o precyzyjnie regulowanej częstotliwości.

W kolejnej części artykuły napiszę o innych, „zaskakujących" zastosowaniach NCO.

Tomasz Jabłoński, EP



Rysunek 8. Inicjalizacja NCO