Nowe peryferia Microchipa (2)

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.

W poprzednim artykule omówiłem bardzo interesujący moduł peryferyjny mikrokontrolerów Microchip – NCO. W przykładzie zastosowaliśmy go do zbudowania generatora o częstotliwości 1 kHz, jednak NCO można wykorzystać do innych, zaskakujących zastosowań. Jednym z nich jest moduł PWM generujący przebieg o dużej rozdzielczości.

Generator przebiegu o dużej rozdzielczości

Wiele mikrokontrolerów ma wbudowany generator przebiegu PWM, który jest charakteryzowany przez (**rysunek 11**):

- Częstotliwość (okres) cyklu
- Współczynnik wypełnienia definiowany jako procentowy udział czasu włączenia zasilania (*duty cycle*) do okresu cyklu.
- Rozdzielczość, czyli krok, z którym może być zmieniany czas włączenia zasilania.

Częstotliwość przebiegu i *duty cycle* nie wymagają szerszego komentarza. Przyjrzyjmy się natomiast problemowi rozdzielczości przebiegu.

Rozdzielczość, to krok, z którym możemy zmieniać czas trwania impulsów. Okres przebiegu PWM oznaczmy Tpwm. W jednym okresie mieści się N cykli o okresie Tsys – na rys. 11 będą to 4 cykle (dla wypełnienia przebiegu 100%). Rozdzielczość bitową przebiegu PWM wylicza się z równania

$$Resolution = log_2(N).$$

Jak to wygląda w praktyce? Mikrokontroler jest taktowany przebiegiem o częstotliwości 16 MHz, więc Tsys=1/16000000=62,5 ns. Jeżeli skonfigurujemy przebieg PWM o Tpwm=200 kHz, to czas Tpwm będzie równy 1/200000=5 µs. W jednym okresie Tpwm zmieści się 80 cykli Tsys, bo 6 µs/62,5 ns=80. Zatem nasze N będzie równe 80, a rozdzielczość bitowa

$$log_2(8) = 6{,}32.$$

Z tych obliczeń wynika też inna zależność. Jeżeli PWM jest taktowany określoną częstotliwością i potrzebujemy określonej rozdzielczości bitowej, to ograniczeniu podlega częstotliwość przebiegu PWM. Na przykład, jeżeli PWM jest taktowane przebiegiem o częstotliwości 16 MHz i potrzebujemy rozdzielczości 10-bitowej, to częstotliwość przebiegu PWM nie może być wyższa niż 15,6 kHz. Maksymalną częstotliwość przebiegu PWM dla zadanej rozdzielczości bitowej wylicza się z równania

$$\frac{Fosc}{\#Steps} = \frac{16\ MHz}{2^{10}} = 15{,}6\ kHz$$

W praktyce, częstotliwość cyklu przebiegu PWM jest wybierana w szerokim zakresie i zależy od rodzaju obciążenia. Jeżeli chcemy sterować jasnością żarówki, to wybieramy wartość z zakresu 100…200 Hz. Przełączanie zasilania z tą częstotliwością w połączeniu z bezwładnością rozgrzanego włókna daje efekt ciągłego świecenia się z jasnością zależną od czasu załączenia zasilania. Silniki elektryczne są zasilane przebiegami PWM o częstotliwości kilku – kilkunastu kHz. Trudno sobie wyobrazić, by silnik elektryczny mógł zatrzymywać się i ruszać kilka tysięcy razy na sekundę. Również w takiej aplikacji charakter obciążenia powoduje, że dostarczana moc jest uśredniana mechaniczną bezwładnością silnika Dioda LED zasilana przebiegiem PWM o częstotliwości kilkuset Hz lub kilku kHz nie będzie uśredniała mocy do niej dostarczonej, ale będzie się zaświecała i gasła w takt przebiegu PWM. Efekt postrzegania uśrednionej zmiany jasności świecenia nastąpi w oku, bo nie może ono zarejestrować tak szybkich zmian.

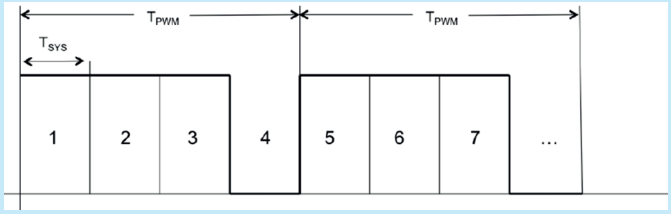
Jak widać w wielu aplikacjach częstotliwość PWM jest o wiele wyższa od tej, z którą sterowany układ może zmieniać swój stan. Żarówka lub silnik elektryczny są swego rodzaju filtrem dolnoprzepustowym uśredniającym sygnał sterujący. W układach elektronicznych stosuje się również typowe filtry dolnoprzepustowe. Przykładem może być przetwornik PWM/napięcie lub szerzej – przetwornik cyfrowo/analogowy. Jeżeli zastosujemy filtr dolnoprzepustowy filtrujący wyższe harmoniczne przebiegu PWM, to otrzymamy napięcie na wyjściu proporcjonalne do współczynnika wypełnienia przebiegu PWM. Co oczywiste, jeżeli rozdzielczość bitowa przebiegu jest mała, to rozdzielczość przetwornika również będzie mała. Zwiększenie precyzji regulacji napięcia można osiągnąć przez zwiększenie rozdzielczości bitowej przebiegu PWM. Z przedstawionych wyżej rozważań wynika, ze w typowym ujęciu zwiększenie rozdzielczości można osiągnąć przez zwiększanie częstotliwości taktowania PWM lub przez zmniejszanie częstotliwości PWM. Oba te sposoby maja ograniczenia. Częstotliwość taktowania nie może być wyższa niż przewidziana dla mikrokontrolera, zaś niskie częstotliwości PWM trudniej się filtruje.

Czas trwania Tsys zależy od częstotliwości taktowania PWM i nie może być dowolnie podzielony. Żeby zmniejszyć Tsys dwukrotnie trzeba zwiększyć dwukrotnie taktowanie. Te ograniczenia mogą być znacznie mniejsze, jeśli do generowania przebiegu PWM użyjemy modułu NCO.

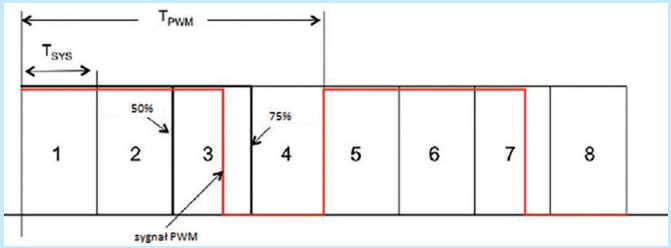
NCO można użyć do zbudowania wyzwalanego generatora monostabilnego generującego impuls o zadanym czasie trwania. Ponieważ wiemy, że NCO może generować częstotliwości z dużą rozdzielczością, to czas trwania impulsów będzie można również ustawić z dużo większą rozdzielczością, niż w metodzie pokazanej na rys. 11.

Popatrzmy teraz na **rysunek 12**. W okresie Tpwm można zmieścić 4 cykle Tsys i przebieg PWM może mieć wypełnienie 25%, 50%, 75% lub 100%. Używając generatora monostabilnego zbudowanego z NCO można wygenerować przebieg PWM, którego opadające zbocze może wypadać z określoną dokładnością wewnątrz okresu sygnału Tsys. Kluczem do osiągnięcia dużej rozdzielczości sygnału PWM jest duża rozdzielczość generowania czasów przez moduł NCO. A jak się okazuje, w określonych warunkach może ona być dużo wyższa, niż możliwa do osiągniecia za pomocą typowej metody stosowanej w standardowych generatorach PWM.

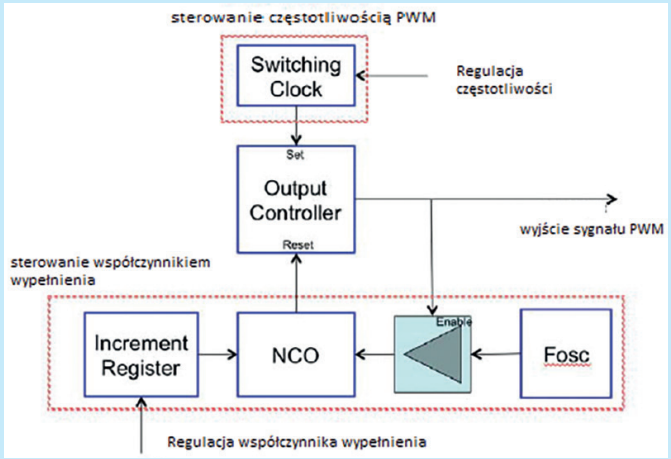
Pozostaje teraz zbudować układ, który będzie wykorzystywał NCO do generowania PWM, bo jak wiadomo z opisu, sam NCO nie może generować PWM, a sygnał



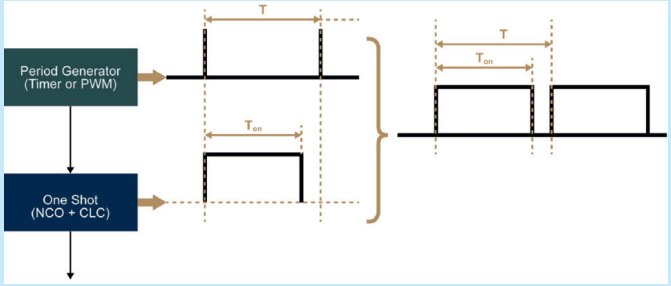
Rysunek 11. Typowy przebieg PWM



Rysunek 12. Generowanie przebiegu PWM za pomocą NCO



Rysunek 13. Idea budowy generatora PWM z użyciem NCO



Rysunek 14. Przebiegi czasowe w trakcie pracy generatora PWM

wyjściowy ma stałe wypełnienie 50%. Jako źródło wyzwalania okresu przebiegu PWM i generatora monostabilnego z układem NCO można użyć, na przykład, konwencjonalnego sygnału PWM lub jednego z timerów mikrokontrolera. Ogólną ideę budowy generatora PWM w użyciem NCO pokazano na **rysunku 13**. Generator *Switching Clock* generuje zbocze co czas Tpwm. Każde takie zbocze wyzwala generator monostabilny zbudowany z NCO, a czas trwania poziomu wysokiego jest określony przez wartość wpisaną do rejestru *Increment* i wynosi *Ton=1/Fnco*, gdzie Fnco jest częstotliwością pracy generatora NCO. Zegar taktujący NCO jest bramkowany sygnałem wyjściowym PWM. Przebiegi czasowe w trakcie pracy układu pokazano na **rysunku 14**.

W teorii wygląda to na łatwe do wykonania, ale trzeba jakoś zaimplementować opisane rozwiązanie w mikrokontrolerze PIC16LF1507. Oprócz NCO do tego celu potrzebne nam będzie kilka bramek. Tu z pomocą przyjdzie nam kolejny nietypowy moduł: **CLC**. Dokładniej zostanie on opisany dalej, ale teraz musimy wiedzieć, że jest to konfigurowalny makro blok **Configurable Logic Cell**, podobny jak umieszczane w układach FPGA. PIC16LF1507 ma dwa takie bloki i obydwa zostaną użyte w naszym generatorze. Praktyczną realizację modułu PWM o dużej rozdzielczości pokazano na **rysunku 15**.

Konfigurowanie modułu NCO rozpoczynamy od ustawienia trybu PF. Przypomnijmy, że w tym trybie po każdym przepełnieniu się licznika – akumulatora wyjście NCOx jest aktywowane i pozostaje w tym stanie przez czas równy zaprogramowanej wielokrotności okresu zliczanego sygnału wejściowego (zliczany przez ripple counter). Czas trwania poziomów aktywnego i nieaktywnego zależy od bitu określającego polaryzację NxPOL i wartości wpisanej do bitów NxPWS rejestru konfiguracyjnego NCOxCON. Działanie układu można opisać następująco:

1. Kiedy układ startuje, wyjście NCO jest wyzerowane i układ zaczyna odliczać impulsy zegarowe aż do przepełnienia. Po przepełnieniu NCO generuje na wyjściu impuls o wysokim poziomie logicznym. Dla naszych celów konfigurujemy wyjście NCO tak, aby jego stan był zanegowany i gdy NCO zlicza impulsy, to jego wyjście jest ustawione, a po zliczeniu wyzerowane. Wyjście NCO jest jednocześnie wyjściem PWM. Poziom wysoki na wyjściu PWM powoduje, że przebieg taktujący Fosc jest podawany na wejście CLG i NCO zlicza impulsy.
2. Po zliczeniu zaprogramowanej liczby impulsów wyjście NCO jest zerowane. Powoduje to zatrzymanie podawania impulsów Fosc na wejście CLK modułu NCO. Wyjście NCO jest nadal wyzerowane, bo nie może zliczyć zaprogramowanej liczby cykli Fosc impulsów, by dokończyć generowania impulsu wyjściowego.
3. Taki stan trwa do momentu, gdy na wejściu *Timing Source* wystąpi poziom wysoki. Na wejście CLK modułu NCO zostanie podanych kilka impulsów Fosc, NCO dokończy zliczanie i wyjście zostanie ustawione rozpoczynając generowanie następnego okresu przebiegu PWM.

Mamy moduł PWM i oczekujemy teraz przebiegu PWM o dużej rozdzielczości. Wiemy już, że częstotliwość generowana przez moduł NCO jest zmieniana liniowo i stała, a zatem okres przebiegu wyniesie *Tpulse=1/Foverflow*. To niestety powoduje, że efektywna rozdzielczość PWM nie jest liniowa (bo funkcja 1/x nie jest liniowa) i zmienia się zależnie od współczynnika wypełnienia. Na **rysunku 16** pokazano graficznie zależność rozdzielczości bitowej od współczynnika wypełnienia dla przebiegu PWM o częstotliwości 3 kHz i przy taktowaniu NCO przebiegiem o częstotliwości 16 MHz. Dla małego współczynnika wypełnienia osiągamy niesamowitą rozdzielczość 21 bitów, by spaść do rozdzielczości 7,5 bitów dla wypełnienia bliskiego 100%. To dość zaskakujący wynik, bo na końcu zakresu konwencjonalne moduły PWM osiągają lepsze rozdzielczości, na przykład 10 bitów. Jak temu zapobiec? Nieoczekiwanie i wbrew intuicji poprzez zmniejszenie częstotliwości taktowania, na przykład z 16 MHz do 1 MHz. Graficznie przedstawiono to na **rysunku 17**. Można też zwiększyć rozdzielczość przez zanegowanie sygnału PWM, kiedy osiągnie wypełnienie 50% (**rysunek 18**). Jeszcze ciekawiej jest, gdy zwiększy się częstotliwość sygnału PWM. Na **rysunku 19** pokazano zależność rozdzielczości od współczynnika wypełnienia dla częstotliwości PWM równej 500 kHz i częstotliwości taktowania 16 MHz. Tu rozdzielczość w całym zakresie zmienia się do 15 bitów do 17 bitów, czyli jest bardzo duża.

Praktyczną implementację modułu PWM zaczniemy od skonfigurowania źródła sygnału *Timing Source*. Generuje ono przebieg, który wyznacza okres sygnału PWM. Dla potrzeb testów wybrałem moduł PWM2. W *MPLAB Code Configurator* z okna zasobów *Sevice Resources* wybieramy i konfigurujemy PWM2, aby częstotliwość jego pracy była równa na przykład 10 kHz i wypełnienie wynosiło 1%. Częstotliwość pracy modułu PWM określa wartość wpisana do rejestru PR2 licznika TMR2. Dlatego przy wyborze PWM2 *Code Configurator* automatycznie dołącza konfigurację

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: ftp://ep.com.pl, user: 60086, pass: sjh7zycq

licznika TMR2. Licznik jest taktowany przebiegiem o częstotliwości 16 MHz/4 =4 MHz. Aby uzyskać 10 kHz, trzeba tę częstotliwość podzielić przez 4, a potem przez 100 (**rysunek 20**). Potem konfigurujemy przebieg PWM o częstotliwości 10 kHz i wypełnieniu 1% (**rysunek 21**). Funkcje konfiguracyjnie wygenerowane przez MPLAB Code Configurator zostały pokazane na **listingach 5** i **6**.

Zgodnie z tym, co napisano, czas trwania *duty cycle* jest określony przez okres sygnału z generatora NCO. Dla PWM o częstotliwości 10 kHz i współczynnika wypełnienia 50% częstotliwość pracy NCO będzie równa 20 kHz. Dla małego wypełnienia częstotliwość NCO musi być dużo większa od 10 kHz, a dla wypełnienia dążącego do 100% będzie się zbliżała do 10 kHz. Jeżeli przyjmiemy współczynnik wypełnienia 50%, to dla 10 kHz Tpwm=100 µs, a Tdc=50 µs. Wynika z tego, że częstotliwość pracy NCO musi być równa 20 kHz i taką ustawiamy. Zgodnie z tym, co napisano, NCO musi być ustawione w tryb pracy PF, a sygnał wyjściowy musi być zanegowany. Konfigurację modułu pokazano na **rysunku 22**, a funkcję konfiguracyjną wygenerowaną przez *MPLAB Code Configurator* na **listingu 7**.

Skonfigurowane moduły PWM i NCO trzeba teraz połączyć zgodnie ze schematem z rys. 15. Do tego celu wykorzystamy moduł makroceli CLC.

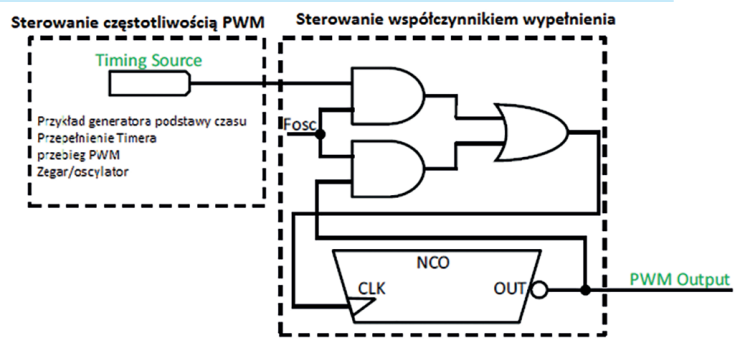
CLC – Configurable Logic Cell

Jeżeli zastanowić się, to opisywany tu moduł NCO lub inne zaawansowane peryferia nie są na tyle uniwersalne, aby same w sobie mogły być stosowane na szeroką skalę. Samo zaimplementowanie nawet najciekawszych modułów może powodować, że mikrokontroler stanie się układem niszowym. Żeby tak nie było, peryferia potrzebują mechanizmu, który pozwoli na wykonywanie pomiędzy nimi elastycznych połączeń. Przykładem jest budowany przez nas generator składający się modułu PWM, modułu NCO i układu bramkującego taktowanie. Takim logicznym „klejem" umożliwiającym łączenie wejść i wyjść modułów pomiędzy sobą, ale też wyprowadzanie sygnałów na wyjścia mikrokontrolera są moduły Configurable Logic Cell – CLC. CLC jest makro celą mogąca pracować w 1 z 8 topologii: AND-OR, OR-XOR, AND, SR, D-Flop, OR-D, JK, D Latch. Na jej wejściu są cztery 4-wejściowe bramki OR z wejściami podłączonymi do selektora sygnałów. Wejścia selektora są programowane i można na nie podawać sygnały z wyjść modułów peryferyjnych, wewnętrzne sygnały zegarowe (na przykład HFINTOSC), poziomy flag przerwań i wyjściowe z innych modułów CLC oraz z doprowadzeń I/O mikrokontrolera.

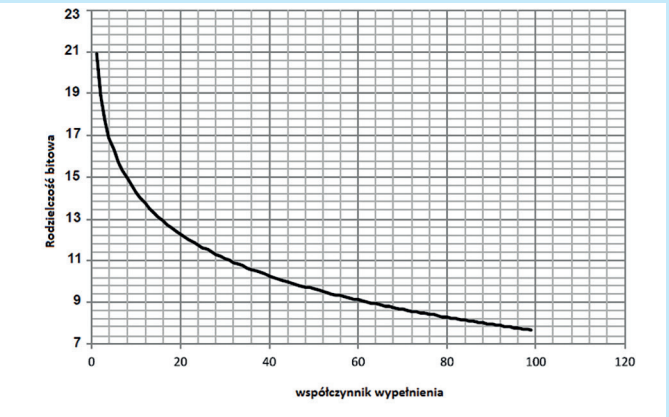
Na **rysunku 23** pokazano topologię AND-OR wyświetlaną w oknie *MPLAB Code Configurator*. Sygnały wejściowe są wybierane z rozwijanego menu i kierowane do wejść bramek OR. Przez kliknięcie na krzyżyk przy wejściu bramki OR sygnał jest dołączany do wejścia. Ponadto, każdy wejście bramki może być zanegowane. Można tez zanegować wyjście każdej z bramki OR (kliknięcie na kwadracik przy wyjściu) oraz zanegować wyjście makroceli. *Code Configurator* umożliwia też konfigurowanie ręczne dostępne po kliknięciu na zakładkę *Manual*. Przy korzystaniu z modułów CLC trzeba pamiętać, że:

- Operacje logiczne wykonywane w bloku CLC są asynchroniczne i niezależne od operacji wykonywanych przez MCU. Szybkość działania nie jest ograniczana przez częstotliwość taktowania mikrokontrolera.
- Konfiguracja każdego z bloków jest kontrolowana przez rejestry SFR umieszczone w pamięci RAM i może być dynamicznie zmieniana w trakcie pracy mikrokontrolera. Oczywiście, po włączeniu zasilania rejestry sterujące CLC musza być zainicjowane przez mikrokontroler.
- Pobór prądu przez układy logiczne jest w stanie statycznym pomijalnie mały, ale rośnie w miarę zwiększania częstotliwości sygnałów wejściowych.

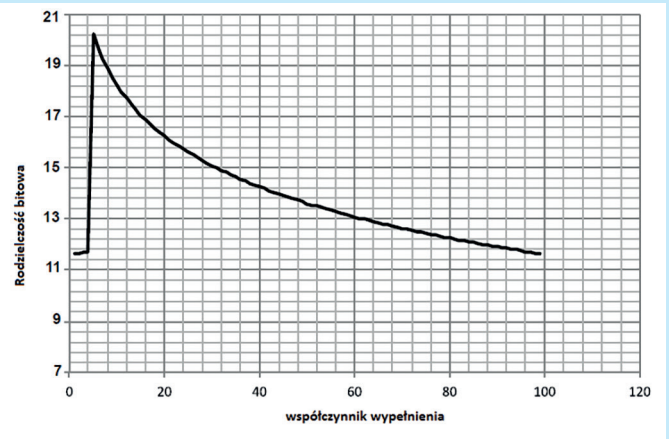
Niewielki pobór prądu pozwala na wykorzystanie CLC w układach bateryjnych, na przykład, do wybudzania mikrokontrolera ze stanu uśpienia.



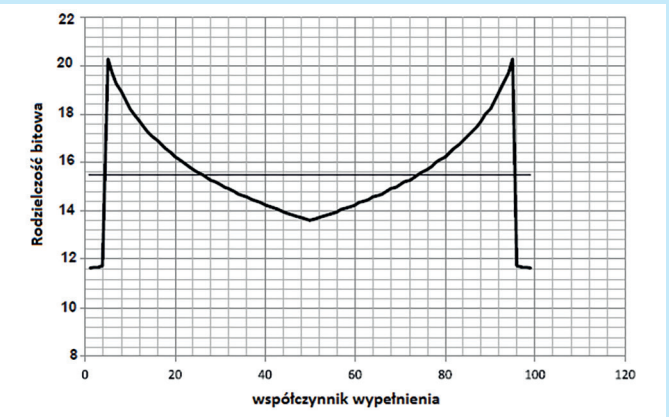
Rysunek 15. Praktyczna realizacja generatora PWM



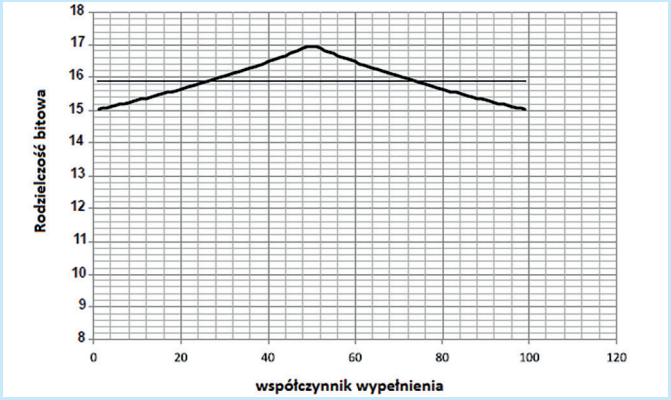
Rysunek 16. Zależność rozdzielczości bitowej od współczynnika wypełnienia dla Fpwm=3 kHz i Fosc=16 MHz



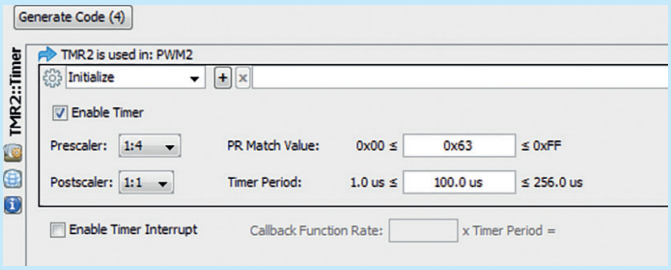
Rysunek 17. Zależność rozdzielczości bitowej od współczynnika wypełnienia dla Fpwm=3 kHz i Fosc=1 MHz



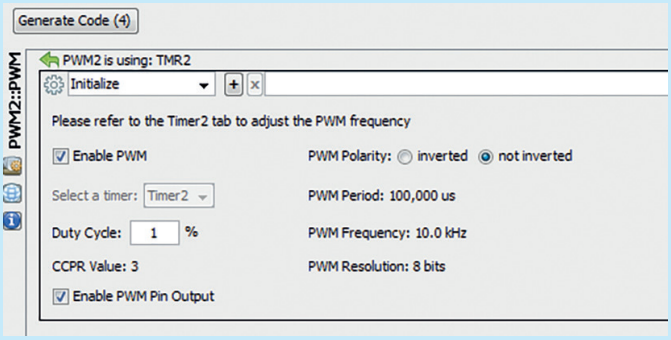
Rysunek 18. Zwiększenie rozdzielczości bitowej przez negowanie sygnału PWM



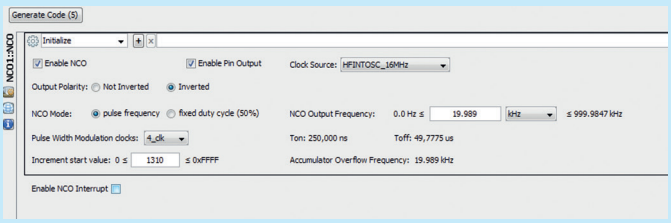
Rysunek 19. Zależność rozdzielczości bitowej od współczynnika wypełnienia dla Fpwm=500 kHz i Fosc=16 MHz



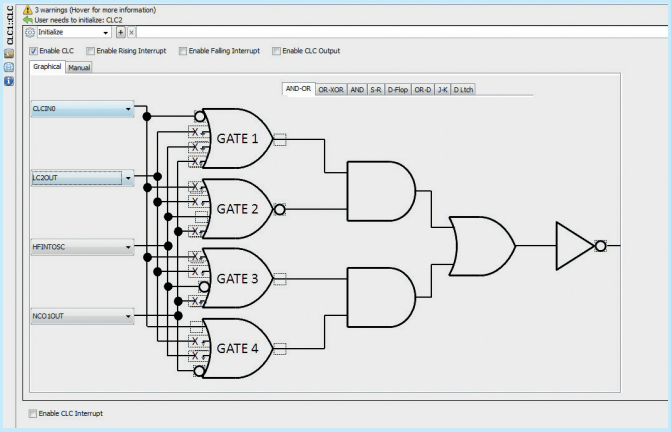
Rysunek 20. Konfiguracja TMR2



Rysunek 21. Konfiguracja modułu PWM2



Rysunek 22. Konfiguracja NCO



Rysunek 23. Konfiguracja CLC makroceli AND-OR

Listing 5. Konfigurowanie timera TMR2

```
void TMR2_Initialize(void) {
    // Set TMR2 to the options selected in the
User Interface
    // TMR2ON off; T2CKPS 1:4; T2OUTPS 1:1;
    T2CON = 0x01;
    // PR2 99;
    PR2 = 0x63;
    // TMR2 0x0;
    TMR2 = 0x00;
    // Clearing IF flag.
    PIR1bits.TMR2IF = 0;
    // Start TMR2
    TMR2_StartTimer();
}
```

Listing 6. Konfigurowanie PWM2

```
void PWM2_Initialize(void) {
    // Set the PWM to the options selected in
the MPLAB® Code Configurator.
    // PWM2POL active_hi; PWM2OE enabled;
PWM2EN enabled;
    PWM2CON = 0xC0;
    // PWM2DCH 0;
    PWM2DCH = 0x00;
    // PWM2DCL 192;
    PWM2DCL = 0xC0;
}
```

Listing 7. Konfigurowanie NCO

```
void NCO1_Initialize(void) {
    // Set the NCO to the options selected in
the GUI
    // N1OUT out_lo; N1PFM PFM_mode; N1POL
active_lo; N1EN disabled; N1OE enabled;
    NCO1CON = 0x51;
    // N1PWS 4_clk; N1CKS HFINTOSC_16MHz;
    NCO1CLK = 0x40;
    // NCOACCU 0;
    NCO1ACCU = 0x00;
    // NCOACCH 0;
    NCO1ACCH = 0x00;
    // NCOACCL 0;
    NCO1ACCL = 0x00;
    // NCO1INCH 5
    NCO1INCH = 5;
    // NCO1INCL 29
    NCO1INCL = 29;
    // Enable the NCO module
    NCO1CONbits.N1EN = 1;
}
```

Wróćmy teraz do projektu modułu PWM. Przebiegi: taktujący o częstotliwości 16 MHz, wejście taktowania i wyjście NCO oraz wyjściowy z PWM2 trzeba połączyć jak na rys. 15. Do tego celu zastosujemy CLC1 w konfiguracji AND-OR. Najpierw Fosc=16 MHz (HFINOSC) łączymy z wejściami bramek GATE2 i GATE3. Do wejścia GATE4 dołączamy wyjście z generatora NCO (NCO1OUT). Pozostaje jeszcze przyłączyć do wejścia bramki GATE1 wyjścia z modułu PWM2. Niestety, konfiguracja wejść CLC nie pozwala na dołączenie wyjścia PWM2OUT do żadnego z wejść. Można wybrać PWM3 lub PWM4, ale wtedy trzy sygnały HFINOSC, NCO1OUT i PWM3OUT można wybrać tylko z dwóch „dolnych" wejść. Rozwiązaniem może być wyprowadzenie sygnału PWM2OUT na wyjście mikrokontrolera, a potem wprowadzenie go za pomocą wejście skonfigurowanego jako CLCIN0. Żeby tego nie robić, użyłem drugiego modułu CLC2 w topologii AND. Skonfigurowałem go tak, aby tylko przenosił sygnał PWM2OUT a jego wyjście LC2OUT wewnętrznie zostało dołączone do wejścia bramki GATE1. Przy takiej konfiguracji nie potrzeba wykonywać żadnych połączeń pomiędzy wyprowadzeniami mikrokontrolera. Na **rysunku 24** pokazano konfigurację CLC1 realizującą połączenie sygnałów modułu PWM o dużej rozdzielczości. LC2OUT jest sygnałem PWM2OUT (**rysunek 25**). Na podstawie opisanej konfiguracji *MPLAB Code Configurator* wygenerował dwie funkcje inicjujące pracę modułów: CLC1 (**listing 8**) i CLC2 (**listing 9**).

Mamy wszystko, by wypróbować generator PWM o dużej rozdzielczości. Do celów testowych, za pomocą menedżera wyprowadzeń dołączyłem do pinów

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: ftp://ep.com.pl, user: 60086, pass: sjh7zycq

mikrokontrolera przebiegi PWM2OUT (LCO2OUT), NCO1CLK (LCO1OUT) i wyjście PWM z modułu (LC1OUT), co pokazano na **rysunku 26**.

Na koniec

Mając do dyspozycji darmowe środowisko **MPLAB IDE** z bezpłatnym kompilatorem **MPLAB XC8** i wtyczką **MPLAB Code Configurator** byliśmy w stanie zaprojektować i wykonać moduł PWM o dużej rozdzielczości. Nie było przy tym konieczności używania układów zewnętrznych i wykonywania połączeń poza mikrokontrolerem. Potrzebne układy peryferyjne są wbudowane w mikrokontroler PIC18LF1507, a wszystko razem zostało „sklejone" przez konfigurowalną makrocelę CLC. Oczywiście, przykład modułu PWM został tak wybrany, by pokazać praktyczne możliwości wykorzystania generatora NCO i układów CLC, ale trudno zaprzeczyć, że zbudowanie modułu PWM pracującego z częstotliwością 500 kHz i rozdzielczością 15 bitów jest sporym osiągnięciem.

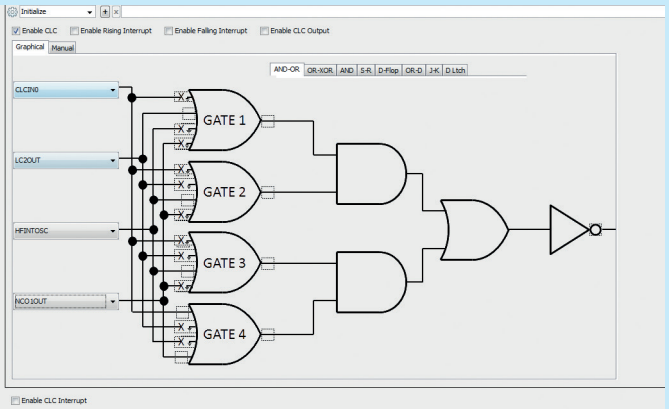
Tomasz Jabłoński, EP

Listing 8. Funkcja inicjalizacji modułu CLC1

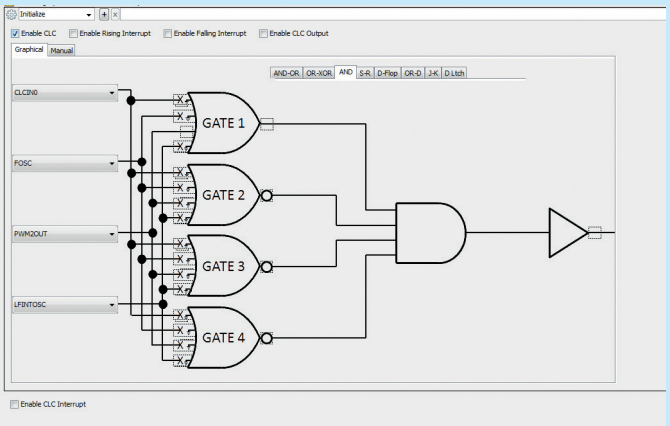
```
void CLC1_Initialize(void) {
    // Set the CLC1 to the options selected in
the User Interface
    // LC1G4POL not_inverted; LC1G1POL not_
inverted; LC1G2POL not_inverted; LC1G3POL not_
inverted; LC1POL inverted;
    CLC1POL = 0x80;
    // LC1D1S CLCIN0; LC1D2S LC2OUT;
    CLC1SEL0 = 0x50;
    // LC1D3S HFINTOSC; LC1D4S NCO1OUT;
    CLC1SEL1 = 0x05;
    // LC1G1D3T disabled; LC1G1D2T enabled;
LC1G1D1T disabled; LC1G1D4N disabled; LC1G1D1N
disabled; LC1G1D2N disabled; LC1G1D3N
disabled; LC1G1D4T disabled;
    CLC1GLS0 = 0x08;
    // LC1G2D4N disabled; LC1G2D3N disabled;
LC1G2D2N disabled; LC1G2D4T disabled; LC1G2D3T
enabled; LC1G2D2T disabled; LC1G2D1N disabled;
LC1G2D1T disabled;
    CLC1GLS1 = 0x20;
    // LC1G3D4N disabled; LC1G3D2N disabled;
LC1G3D1N disabled; LC1G3D3N disabled; LC1G3D3T
enabled; LC1G3D4T disabled; LC1G3D1T disabled;
LC1G3D2T disabled;
    CLC1GLS2 = 0x20;
    // LC1G4D1T enabled; LC1G4D4T enabled;
LC1G4D2N disabled; LC1G4D3N disabled; LC1G4D4N
disabled; LC1G4D1N disabled; LC1G4D3T
disabled; LC1G4D2T disabled;
    CLC1GLS3 = 0x82;
    // LC1MODE AND-OR; LC1EN enabled; LCINTP
disabled; LCINTN disabled; LC1OUT disabled;
LC1OE disabled;
    CLC1CON = 0x80;
}
```

Listing 9. Funkcja inicjująca moduł CLC2

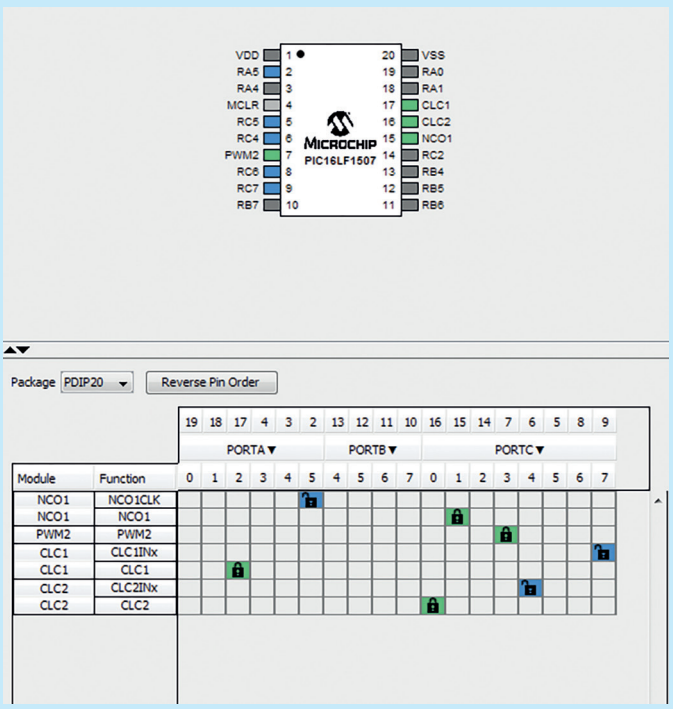
```
void CLC2_Initialize(void) {
    // Set the CLC2 to the options selected in
the User Interface
    // LC2G3POL inverted; LC2POL not_inverted;
LC2G4POL inverted; LC2G1POL not_inverted;
LC2G2POL inverted;
    CLC2POL = 0x0E;
    // LC2D1S CLCIN0; LC2D2S FOSC;
    CLC2SEL0 = 0x00;
    // LC2D4S LFINTOSC; LC2D3S PWM2OUT;
    CLC2SEL1 = 0x07;
    // LC2G1D2N disabled; LC2G1D1N disabled;
LC2G1D4N disabled; LC2G1D3N disabled; LC2G1D4T
disabled; LC2G1D1T disabled; LC2G1D2T
disabled; LC2G1D3T enabled;
    CLC2GLS0 = 0x20;
    // LC2G2D2N disabled; LC2G2D4T disabled;
LC2G2D1N disabled; LC2G2D3T disabled; LC2G2D3N
disabled; LC2G2D4N disabled; LC2G2D1T
disabled; LC2G2D2T disabled;
    CLC2GLS1 = 0x00;
    // LC2G3D4N disabled; LC2G3D1T disabled;
LC2G3D1N disabled; LC2G3D3N disabled; LC2G3D2N
disabled; LC2G3D4T disabled; LC2G3D2T
disabled; LC2G3D3T disabled;
    CLC2GLS2 = 0x00;
    // LC2G4D4T disabled; LC2G4D3T disabled;
LC2G4D2T disabled; LC2G4D1T disabled; LC2G4D1N
disabled; LC2G4D3N disabled; LC2G4D4N
disabled; LC2G4D2N disabled;
    CLC2GLS3 = 0x00;
    // LC2MODE 4-input AND; LCINTP disabled;
LC2OE disabled; LCINTN disabled; LC2OUT
disabled; LC2EN enabled;
    CLC2CON = 0x82;
}
```



Rysunek 24. Połączenie sygnałów modułu PWM o dużej rozdzielczości



Rysunek 25. Dołączenie sygnału PWM2OUT do LC2OUT



Rysunek 26. Sygnały testowe na wyprowadzeniach mikrokontrolera