Kurs programowania mikrokontrolerów PIC (2)

Sterowanie za pomocą PWM

W poprzednim odcinku kursu nauczyliśmy się sposobu wykonania projektu oraz zaświecania i gaszenia diody LED. Spróbujemy teraz „zmusić ją", by zmieniła jasność świecenia. Aby to zrobić, potrzebujemy przebiegu PWM.

Dodatkowe materiały na CD/FTP:
ftp://ep.com.pl, user: 14039, pass: 4p80b5b5
• pierwsza część kursu

PWM to modulacja szerokości impulsu (*Pulse Width Modulation*), która jest techniką stosowaną szeroko do sterowania mocą zasilania urządzeń elektrycznych. Jest też wykorzystana do konwersji C/A na przykład we wzmacniaczach klasy D. Sterowanie mocą odbywa się przez cykliczne przełączanie pomiędzy włączaniem i wyłączaniem napięcia zasilającego o stałej wartości. Zależność pomiędzy czasem załączenia i wyłączenia w cyklu określa moc oddawaną do obciążenia. Im dłuższy jest czas załączenia w cyklu, tym większa jest moc.

Przebieg PWM jest charakteryzowany przez:

– częstotliwość (a tym samym okres) cyklu,
– współczynnik wypełnienia (definiowany jako procentowy udział czasu włączenia zasilania tzw. *duty cycle* do okresu cyklu).

Dodatkowo można zdefiniować rozdzielczość, czyli krok z jakim może być zmieniany czas włączenia zasilania

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 z jasnością zależną od czasu załączenia zasilania. Silniki elektryczne są zasilane przebiegami PWM o częstotliwości kilku kHz. Trudno sobie wyobrazić, aby silnik elektryczny mógł się zatrzymywać i uruchamiać kilka tysięcy razy na sekundę. Również w tym przypadku 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 cyklicznie zaświecała i gasła w takt przebiegu PWM. Efekt postrzegania uśrednionej zmiany jasności świecenia nastąpi w oku, bo nie może ono zarejestrować tak szybkich zmian. Diodę będziemy zasilać z portu mikrokontrolera w takt przebiegu PWM.

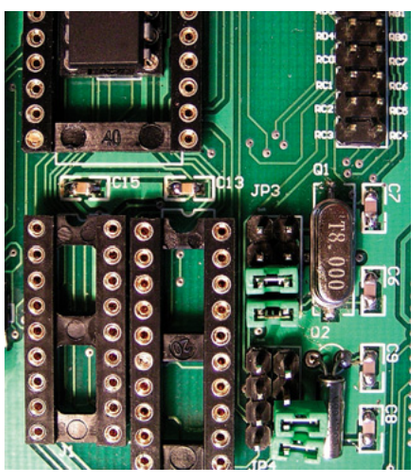
Generowanie PWM z użyciem licznika i przerwań

Sterowanie PWM jest powszechnie stosowane w technice sterowania i wiele mikrokontrolerów ma wbudowane moduły przeznaczone do generowania przebiegu PWM. Mikrokontrolery przeznaczone do sterowania silnikami mają ich nawet kilka. PIC18F2320 ma również wbudowane dwa takie moduły CCP, ale opiszę je później. Najpierw zajmiemy się możliwością generowania przebiegów PWM z zastosowaniem liczników i przerwań.

Załóżmy, że będziemy chcieli generować przebieg o częstotliwości 1 kHz, czyli o okresie 1 ms. Do sterowania jasnością diody wystarczy nam rozdzielczość 100 kroków, czyli zmiana czasu załączenia napięcia zasilania będzie się odbywała z rozdzielczością 1 ms/100=10 µs. Zatem przerwania od przepełnienia licznika powinny się odbywać z częstotliwością 100 kHz. Z tego co już wiemy mikrokontroler taktowany częstotliwością 4 MHz wykonuje rozkazy z częstotliwością 4 MHz/4=1 MHz. To zdecydowanie za mało, by obsługiwać przerwania zgłaszane co 100 kHz, bo pomiędzy kolejnymi przerwaniami mikrokontroler może wykonać tylko 10 rozkazów. Obsługa przerwań zajmowałaby cały czas procesora. Mamy do wyboru: zwiększyć częstotliwość taktowania, zmniejszyć rozdzielczość lub zmniejszyć częstotliwość PWM. Mikrokontroler PIC18F2320 może być taktowany maksymalną częstotliwością 40 MHz przy taktowaniu z oscylatora kwarcowego o częstotliwości fxtal=10 MHz i trybie HSPLL (Fxtal×4). Wtedy rozkazy będą wykonywane z częstotliwością 10 MHz, czyli pomiędzy przerwaniami wykona się ich 100. To już lepiej, ale również może się okazać za mało, jeżeli w programie chcemy wykorzystać inne przerwania lub pewne fragmenty kodu będą musiały być wykonane w reżimie czasowym. Możemy też zmniejszyć rozdzielczość. Do sterowania jasnością diody LED pewnie wystarczy 10 kroków, ale trudno będzie wykorzystać do innych celów PWM pracujący z taką rozdzielczością. Można zmniejszyć częstotliwość PWM do na przykład 100 Hz. Wtedy przerwania dla rozdzielczości 100 kroków będą zgłaszane z częstotliwością 10 kHz. W mikrokontrolerze taktowanym fxtal=40 MHz pomiędzy kolejnymi przerwaniami mikrokontroler wykona 1000 rozkazów, a to już całkiem sporo.

Te wyliczenia pokazują ograniczenia metody generowania PWM w przerwaniu od przepełnienia licznika. Jednak w sytuacji, kiedy będziemy potrzebowali kilku kanałów, PWM może się okazać metodą bardzo przydatną.

Wróćmy do tematu sterowania jasnością diody LED. Załóżmy, że będziemy ją sterować przebiegiem PWM o częstotliwości 200 Hz z rozdzielczością 100 kroków. Na początek zmienimy sposób taktowania



Fotografia 1. Ustawienie zworek JP3 oscylatora kwarcowego

Listing 1. Inicjalizacja licznika i układu przerwań
```
TMR0H=0xFE; //wartość początkowa TMR0
TMR0L=0x6F;
T0CON=0x88; //włącz Timer0, rozdz. 16 bitow, bez preskalera
TMR0IE=1;   //odblokowanie przerwania od przepełnienia Timer0
PEIE=1;     //odblokowanie przerwań od peryferii
GIE=1;      //odblokowanie systemu przerwań
```

Listing 2. Procedura obsługi przerwania z generowaniem PWM i odliczaniem opóźnień
```
volatile unsigned int mSek=0;
volatile unsigned char ppwm, hpwm,
pwm;
static void interrupt int_tmr0(void)
{
  TMR0H=0xFE;//przerwanie co 50us
  TMR0L=0x6F;//fosc=32MHz
  ++pwm;          //generowanie PWM
  if(pwm==ppwm)
  pwm=0;
  if(pwm<=hpwm)//stan wysoki na RA1
  LATA|=2;
  else
  LATA&=~2;       //stan niski na RA1
  if(tms){        //odliczanie
                  //opóźnień
  --mSek;
  if(mSek==0)
  tms=0;}
  TMR0IF=0;
}
```

Listing 3. Prezentacja możliwości sterowania jasnością diody LED
```
while(1){
  for(i=0;i<60;i++){
  hpwm=i*2;       //zmiana
                  //współczynnika
                  //wypełnienia
  delay(10);}
}
```

Listing 4. Obsługa przerwania z dwoma kanałami PWM
```
volatile unsigned char
ppwm,hpwm1,hpwm, pwm;
static void interrupt int_tmr0(void)
{
  TMR0H=0xFE;//przerwanie co 50us
  TMR0L=0x6F;//fosc=32MHz
  ++pwm;          //generowanie PWM
  if(pwm==ppwm) pwm=0;
  if(pwm<=hpwm)//ustawienie RA1
  LATA|=2;
  else
  LATA&=~2;       //zerowanie RA1
  if(pwm<=hpwm1)//ustawienie RA0
  LATA|=1;        //zerowanie RA0
  else
  LATA&=~1;       //stan niski na RA0
  if(tms){
  --mSek;
  if(mSek==0)
  tms=0;}
  TMR0IF=0;
}
```

Listing 5. Prezentacja możliwości sterowania jasnością dwóch diod LED
```
while(1){
  for(i=0;i<50;i++){
  hpwm=i*2;
  hpwm1=100-(i*2);
  delay(10);}
}
```

Listing 6. Zmodyfikowana procedura odliczania opóźnień
```
void delay(unsigned int msek){
unsigned int i;
  mSek=msek*20;   //korekcja, bo
                  //przerwania co
                  //50µs, a nie co
                  //1 ms
  TMR0H=0xFE;
  TMR0L=0x6F;
  tms=1;
  while(tms);
}
```

mikrokontrolera, tak aby można było zwiększyć częstotliwość, wykorzystując oscylator kwarcowy i tryb HSPLL. Najpierw zrezygnujemy z wewnętrznego oscylatora RC na rzecz zewnętrznego oscylatora kwarcowego Q1 o częstotliwości 8 MHz umieszczonego na płytce modułu. Żeby go dołączyć do mikrokontrolera, trzeba odpowiednio ustawić zworki JP3 (**fotografia 1**).

Po włączeniu opcji taktowania z generatora kwarcowego z 4-krotnym powieleniem częstotliwości przez układy PLL (HSPLL) otrzymujemy taktowanie z częstotliwością 4×8 MHz=32 MHz. W makro definiującym pierwsze słowo konfiguracyjne trzeba zmienić rodzaj oscylatora: *__CONFIG(1,IESODIS&FCMDIS&HSPLL);*. Teraz mikrokontroler będzie wykonywał rozkazy z częstotliwością 32 MHz/4=8 MHz. Przyjmijmy, że przepełnienie licznika Timer0 następuje po zliczeniu 400 cykli rozkazowych. Wtedy przerwania będą zgłaszane z częstotliwością 8 MHz/400=20 kHz.

Licznik Timer0 skonfigurujemy do zliczania cykli rozkazowych w trybie 16-bitowym bez preskalera. Fragment inicjalizacji licznika i układu przerwań pokazano na **listingu 1**.

Aby licznik zgłaszał przerwanie co 400 zliczonych cykli rozkazowych, trzeba do niego wpisać 65535-400=65135=0xFE6F po każdym zgłoszeniu przerwania.

Przerwania teraz są zgłaszane z częstotliwością 20 kHz, czyli co 50 µs. Aby uzyskać częstotliwość przebiegu PWM równą 200 Hz, czyli o okresie 5 ms, trzeba po zliczeniu każdych 100 przerwań zmienić poziom linii portu kanału PWM z niskiego na wysoki. Do zliczania przerwań zostanie użyta 8-bitowa zmienna *pwm* inicjowana w programie głównym wartością 0x00. Po każdym zgłoszeniu przerwania jest ona inkrementowana i jeżeli osiągnie wartość równą wartości drugiej 8-bitowej zmiennej *ppwm*, to jest zerowana. W *ppwm* jest zapisane, ile trzeba zgłoszeń przerwania na jeden okres generowanego przebiegu PWM. Wpisujemy tam 100 i mamy 100 zgłoszeń, każde co 50 µs, co daje okres 5 ms. Wprowadzenie dodatkowej zmiennej *ppwm* pozwala na zmianę w pewnym zakresie okresu PWM z programu głównego. Jeżeli na przykład do *ppwm* wpiszemy 200, to okres PWM zmieni się z 5 ms (200 Hz) na 10 ms (100 Hz).

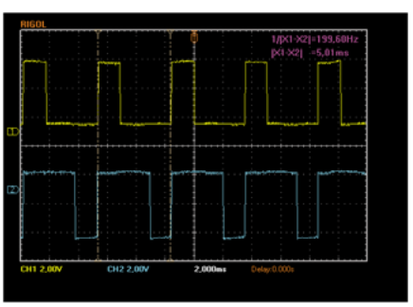
Okres generowania PWM zawsze zaczyna się od ustawienia linii RA1. Czas trwania poziomu wysokiego określa kolejna zmienna – *hpwm*. Wartość *hpwm* nie powinna być większa od *ppwm*. W procedurze obsługi przerwania, po każdej inkrementacji zmiennej *pwm*, jest sprawdzany warunek, czy *pwm* nie jest mniejsze od *hpwm*. Jeżeli tak, to linia RA1 pozostaje ustawiona, inaczej jest zerowana. Wpisując do *ppwm* 100, a do *hpwm* 80, otrzymamy przebieg PWM o częstotliwości 200 Hz, czyli okresie 5 ms, o poziomie wysokim trwającym 4 ms (80×50 µs), czyli o współczynniku wypełnienia 80%.

Na **listingu 2** pokazano procedurę obsługi przerwania od przepełnienia Timer0. Fragment programu wykonującego się w nieskończonej pętli pokazuje możliwości zmieniania jasności diody LED w wyniku modyfikacji współczynnika wypełnienia (**listing 3**). Ten sposób ma ważną zaletę – można łatwo dodawać nowe kanały PWM. Oczywiście, ich liczba jest ograniczona możliwościami obsługi przerwania, która nie może wykonywać się zbyt długo. W naszym przykładzie możemy bez problemu dodać drugi kanał PWM przez powielenie programowych mechanizmów pierwszego kanału. Jeżeli zadowalają nas rozdzielczość i okres, to wystarczy zdefiniować tylko nową zmienną określającą czas trwania stanu wysokiego, na przykład *hpwm1*, i uzupełnić procedurę przerwania o sterowania linią, na przykład RA0 (**listing 4**).

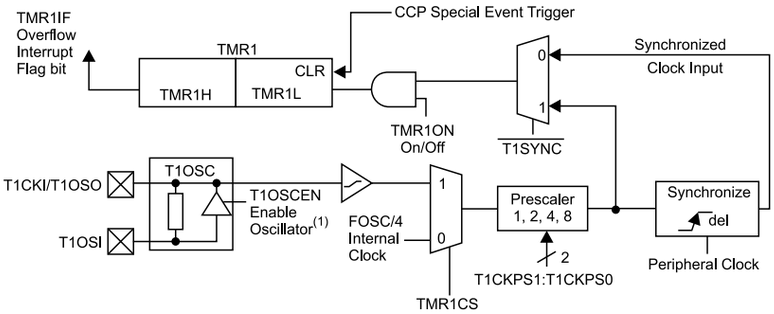
Na **listingu 5** pokazano program demonstrujący możliwości sterowania jasnością świecenia 2 diod podłączonych do linii RA0 i RA1. Diody są tak sterowane, że kiedy jedna zwiększa jasność, to druga ją zmniejsza.

Wyjaśnienia wymaga fragment odliczający opóźnienia. Ponieważ przerwania są zgłaszane co 50 µs, procedura odliczania opóźnień musi być zmodyfikowana (**listing 6**).

Na **rysunku 2** pokazano oscylogram dwu kanałów PWM z różnymi współczynnikami wypełnienia. Przebieg górny został wygenerowany dla *hpwm*=30, a przebieg dolny dla *hpwm1*=70. Takie rozwiązanie może być bardzo przydatne a przykład do sterowania jasnością diod podświetlania wyświetlaczy LCD. Niestety zbyt mała częstotliwość przebiegu powoduje, że niezbyt się nadaje do sterowania silnikami małej mocy. W czasie prób okazało się (zgodnie z przewidywaniami), że silniki tak sterowane wpadają w drgania i pracują nierównomiernie.



Rysunek 2. Oscylogram przebiegów PWM



Rysunek 3. Schemat blokowy licznika Timer1

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
RD16	T1RUN	T1CKPS1	T1CKPS0	T1OSCEN	T1SYNC	TMR1CS	TMR1ON

Bit7 RD16 1 licznik 16-bitowy
0 licznik 8-bitowy
Bit6 T1RUN bit tylko do odczytu
Licznik zlicza impulsy z oscylatora T1OSC
Licznik zlicza Fosc/4
Bit 5:4 T1CKPS1 T1CKPS0 preskaler
11- 1:8
10- 1:4
1:2
1:1
Bit 3 T1OSCEN włączenie oscylatora T1OSC
1 oscylator włączony
0 oscylator wyłączony
Bit2 T1SYNC włączenie synchronizacji zewnętrznego przebiegu z T1OSC lub z RC0 z wewnętrznym zegarem
1 synchronizacja włączona
0 synchronizacja wyłączona
Bit1 TMR1CS źródło zliczania
1 z T1OSC lub RC0
zlicza Fosc/4
bit0 TMR1ON włączenie zliczania
zliczanie włączone
zliczanie wyłączone

Rysunek 4. Rejestr T1CON

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
--	--	DC1B1	DC1B0	CCP1M3	CCP1M2	CCP1M1	CCP1M0

Bit5: DC1B1
bit B1 rejestru określającego czas poziomu wysokiego (*duty cycle*)
Bit4:DC1B0
bit B0 rejestru określającego czas poziomu wysokiego (*duty cycle*)
bity 3:0
11xx tryb PWM

Rysunek 5. Rejestr CCP1CON

Wykorzystując dwa niezależne liczniki i mechanizm przerwań, można wygenerować przebieg PWM o zdecydowanie większym zakresie częstotliwości i dużej rozdzielczości. PIC18F2320 ma wbudowane 4 liczniki, w tym trzy 16-bitowe. Do naszych celów wykorzystamy liczniki Timer1 i Timer3 skonfigurowane jako 16-bitowe. Licznik Timer1 będzie służył do odliczania okresu PWM, a licznik Timer3 do odliczania czasu trwania poziomu wysokiego tzw. *duty cycle*.

Schemat blokowy licznika Timer1 pokazano na rysunku 4. Źródłem zliczanych impulsów może być dedykowany dla Timer1 zewnętrzny oscylator kwarcowy lub wewnętrzny zegar o częstotliwości Fosc/4. Przed rejestrami licznika jest umieszczony programowany preskaler i układ synchronizacji zliczanych impulsów z wewnętrznym zegarem. Zliczanie może być programowo zatrzymywane lub uruchamiane, a sam licznik zerowany zdarzeniem z modułu CCP. W trybie zliczania 16-bitowego przepisanie 8 starszych bitów z rejestru TMR1H do rejestru licznika TMR1 następuje tylko w momencie zapisaniu rejestru TMR1L. Musimy pamiętać, by przed zapisaniem TMR1L w TMR1H była już wpisana właściwa wartość. Przepełnienie licznika ustawia znacznik przerwania TMR1IF i jeżeli układ przerwań jest odblokowany i bit maski przerwania TMR1IE jest ustawiony, to zostanie zgłoszone przerwanie. Licznik jest konfigurowany rejestrem T1CON (**rysunek 4**).

Do naszych celów licznik skonfigurujemy jako 16-bitowy, bez preskalera, zliczający cykle rozkazowe o częstotliwości Fosc/4. Dodatkowo wyłączymy oscylator T1OSC. Do T1CON wpiszemy 0x85.

Załóżmy, że mikrokontroler jest dalej taktowany oscylatorem 8 MHz w trybie HSPLL, czyli nasze Fosc=32 MHz. Licznik zlicza impulsy o częstotliwości 32 MHz/4=8 MHz. Częstotliwość przebiegu PLL będzie wynosiła 4 kHz, czyli z taką częstotliwością ma się przepełniać licznik Timer1. Aby tak było, trzeba zliczyć 8 MHz/4 kHz=2000=0x7D0 impulsów. Licznik T3 zostanie uruchomiany w procedurze przerwania od przepełnienia Timer1.

Generowanie przebiegu PWM wygląda następująco:

- Po zgłoszeniu przerwania od przepełnienia Timer1: jest ładowany Timer1, potem ładowany Timer3, a następnie jest uruchamiane zliczanie Timer3 i ustawiana linia PWM.
- Po odliczeniu zadanej liczby impulsów licznik Timer3 przepełnia się i zgłasza przerwanie. W obsłudze przerwania zatrzymuje swoje zliczanie przez wyzerowanie TMR3ON i ustawia linię PWM (koniec *duty cycle*).
- Następne przerwanie od Timer1 ładuje Timer3 i uruchamia zliczanie przez niego impulsów przez ustawienie TMR3ON i tak dalej.

Żeby cały mechanizm działał prawidłowo, licznik Timer3 musi zliczać do przepełnienia mniej impulsów niż licznik Timer1 (okres PWM musi być dłuższy niż *duty cycle*). Wyliczyliśmy, że dla przebiegu 4 kHz Timer1 musi się przepełniać co 2000 impulsów o częstotliwości Fsoc/4=8 MHz. Aby uzyskać wypełnienie 50%, licznik Timer3 odlicza 1000 impulsów.

Budowa licznika Timer3 jest bardzo podobna do licznika Timer1. Rejestr konfiguracyjny T3CON (dla naszego zastosowania) ma również bardzo podobną strukturę – dokładne dane można znaleźć w dokumentacji.

Na **listingu 7** jest pokazany fragment programu inicjalizującego liczniki Timer1 i Timer3. Po jej wykonaniu liczniki zliczają w trybie 16-bitowym impulsy o częstotliwości Fosc/4. Licznik Timer1 zlicza impulsy (TMR1ON=1), a licznik Timer3 jest zatrzymany (TMR3ON=0).

Po inicjalizacji przerwania pochodzą z trzech źródeł: od przepełnienia liczników Timer0, Timer1 i Timer3. Procedura obsługi przerwania musi rozróżniać źródło przerwania. Robi to przez testowanie stanu bitów znaczników przerwania (**listing 8**).

Ta metoda ma dwie ważne zalety: duży zakres częstotliwości i dużą rozdzielczość. Jednak wymaga wykorzystania aż dwóch liczników. Nie jest to problemem, jeżeli są one do dyspozycji. Gorzej, że dodanie nowego kanału wymagałoby użycia kolejnego licznika do odliczania *duty cycle*. Dlatego, jeżeli potrzebujemy dwóch kanałów PWM, na przykład do sterownia silnikami modelu pojazdu, lepiej jest wykorzystać sprzętowe moduły PWM.

Generowanie PWM za pomocą modułów CCP

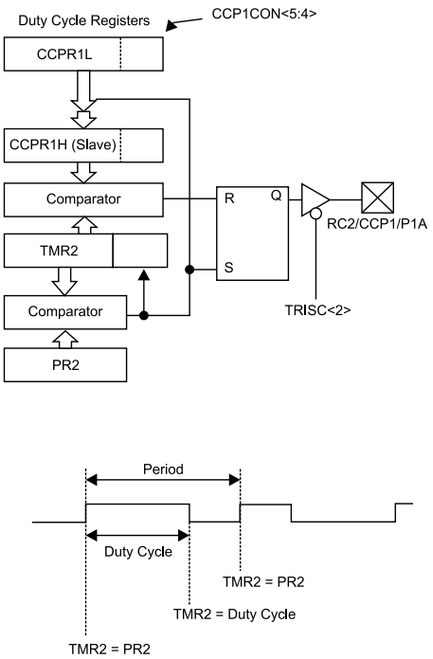
Wspomniałem już, że PIC18F2320 ma wbudowany dwa sprzętowe moduły CCP (*Capture*/*Compare*/*PWM*) przeznaczone m.in. do generowania dwóch przebiegów

Listing 7. Inicjalizacja liczników Timer1 i Timer3

```
TMR1H=0xf8;  // TMR1 dla 4 kHz
TMR1L=0x2F;
TMR3H=0xfc; //odliczanie duty cycle dla 50%, 4 kHz
TMR3L=0x17;
T1CON=0xc5; //włącz Timer1: 16 bitow bez preskalera, zliczanie START
T3CON=0x80; //włącz Timer3: 16 bitów bez preskalera, zliczanie STOP
TMR1IE=1;   //odblokowanie przerwań od przepełnienia TIMER1
TMR3IE=1;   //odblokowanie przerwań od przepełnienia TIMER3
```

Listing 8. Procedura obsługi przerwania

```
static void interrupt int_tmr(void){
  if(TMR0IF){  //przerwanie od licznika Timer0
    TMR0H=0xFe; //przerwanie co 50us
    TMR0L=0x6F; //fosc=32MHz
    ++pwm;  //generowanie PWM
    if(pwm==ppwm)
    pwm=0;
    if(pwm<=hpwm)  //ustawienie RA1
    LATA|=2;
    else
    LATA&=~2;  //zerowanie RA1
    if(pwm<=hpwm1)  //ustawienie RA0
    LATA|=1;
    else
    LATA&=~1;  //zerowanie RA0
    if(tms){
      --mSek;
      if(mSek==0)
      tms=0;}
      TMR0IF=0;
    }
  if(TMR1IF){  //przerwanie od licznika Timer1
    TMR1H=0xf8;  //dla PWM f=4kHz
    TMR1L=0x2F;
    LATA|=4;  //ustawienie RA2
    TMR3H=0xfc;  //odliczanie duty cycle przez Timer3
    TMR3L=0x17;
    TMR3ON=1;  //uruchom zliczanie przez Timer3
    TMR1IF=0;  //zerowanie flagi przerwania od Timer1
  }
  if(TMR3IF){  //przerwanie od licznika Timer3
               //koniec odliczania duty cycle
    LATA&=~4;  //zerowanie linii RA2
    TMR3ON=0;  //zatrzymaj zliczanie Timer3
    TMR3IF=0;  //zerowanie flagi przerwania od Timer3
  }
}
```



Rysunek 6. Schemat blokowy modułu PWM i sygnał wyjściowy

PWM o rozdzielczości 10-bitowej. Moduł CCP może wykonywać różne funkcje, więc trzeba go skonfigurować tak, by pracował jako kanał PWM. Do tego celu jest wykorzystywany rejestr CCPxCON (CCP1CON dla CCP1 i CCP2CON dla CCP2). Na **rysunku 5** pokazano rejestr CCP1CON, struktura CCP2CON jest identyczna. Funkcje bitów zostały opisane tylko dla trybu PWM.

Schemat blokowy modułu zaprogramowanego do pracy jako generator PWM pokazano na **rysunku 6**. Oprócz modułu CCP zatrudniony jest też licznik Timer2 do odliczania okresu przebiegu i czasu *duty cycle*.

Jak to działa? Okres przebiegu jest określany przez rejestr PR2 (wspólny dla obu modułów) według zależności:

$$PWM\ Period = [(PR2) + 1] \times 4 \times T_{osc} \times \times (wartość\ preskalera\ Timer2)$$

T_{osc} to częstotliwość taktowania mikrokontrolera.

Licznik TMR2 zlicza impulsy o częstotliwości Fosc/4 (częstotliwość cyklu maszynowego) podzielone przez wartość preskalera i kiedy jego wartość jest równa wartości wpisanej do PR2, to:

- TMR2 jest zerowany,
- wyprowadzenie CCP1 jest ustawiane,
- rozpoczyna się odliczanie czasu trwania poziomu wysokiego,

Czas trwania poziomu wysokiego jest określony przez 10-bitową liczbę składającą się z 8 starszych bitów rejestru CCPR1L i 2 bitów CCP1CON (bity DC1B1 i DC1B0). Ta wartość jest przepisywana do 10-bitowego rejestru *CCPR1H=CCPR1L:CCP1CON<5:4>*. Zawartość rejestru CCP1H jest porównywana z 8 starszymi bitami rejestru TMR2 i 2 bitami preskalera Timer2. Jeżeli są one sobie równe, to wyprowadzenie CCP1 zostaje wyzerowane.

Mając dany okres przebiegu, wartość wpisywaną do PR2 wyliczamy z zależności:

$$[(PR2) + 1] = \frac{Okres\ PWM}{4 \times T_{osc} \times (wartość\ preskalera\ Timer2)}$$

Dla przebiegu o częstotliwości 4 kHz PWMPeriod= 250 µs, a dla Fosc=32 MHz Tosc=32,25 ps. Zatem do PR2 należy wpisać:

$$\frac{250\ \mu s}{4 \times 32{,}25\ ps} - 1 = 1999$$

Ponieważ PR2 jest 8-bitowy, musimy użyć preskalera. Preskaler TMR2 może podzielić przebieg wejściowy przez 4 lub przez 16. My będziemy potrzebowali podziału przez 16:

$$\frac{250\ \mu s}{4 \times 32{,}25\ ps \times 16} - 1 = 124$$

Do PR2 wpisujemy 124, a preskaler programujemy, ustawiając najmłodsze 2 bity rejestru T2CON.

W ten sposób ustaliliśmy okres przebiegu PWM. W następnym kroku trzeba podać czas poziomu wysokiego. Programuje się go z rozdzielczością 10 bitów, zapisując 8-bitowy rejestr CCPR1L i 2 bity <5:4> rejestru CCP1CON według zależności:

$$PWM\ Duty\ Cycle = CCPR1L{:}CCP1CON < 5{:}4 > \times T_{osc} \times (wartość\ preskalera\ Timer2)$$

Dla preskalera :16 otrzymamy:

$$(CCPR1L{:}CCP1CON < 5{:}4 >) = \frac{PWM\ Duty\ Cycle}{T_{osc} \times 16}$$

Jeżeli okres przebiegu PWM wynosi 250 µs, to na przykład dla wypełnienia 50% PWM *duty cycle* będzie wynosił 125 µs. W takim razie wartość wpisywana do (CCPR1L:CCP1CON<5:4>) będzie równa

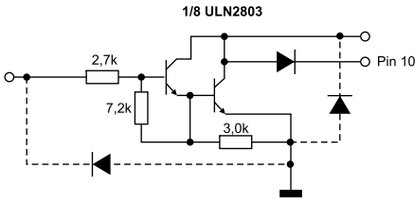
$$(CCPR1L{:}CCP1CON < 5{:}4 >) = \frac{125\ \mu s}{16 \times 32{,}25\ ps} = 250$$

Tak skonfigurowany moduł CCP generuje przebieg prostokątny o częstotliwości 4 kHz i wypełnieniu 50% i nie obciąża CPU. Współczynnik wypełnienia modyfikuje się przez zapisywanie CCPR1L:CCP1CON<5:4>. Można to zrobić w dowolnym momencie, bo przepisanie do CCPR1H następuje na początku cyklu, czyli kiedy TMR2=PR2.

Konfigurację modułu można przeprowadzić w kilku krokach:

- Zaprogramowanie częstotliwości (okresu przebiegu) przez zapisanie rejestru PR2.
- Zaprogramowanie czasu poziomu wysokiego przez zapisanie rejestrów CCPR1L:CCP1CON<5:4>.
- Ustawienie linii CCP jako wyjściowej przez wyzerowanie odpowiedniego bitu rejestru TRISC.
- Zaprogramowanie preskalera TMR2 i uruchomienie licznika przez zapisanie rejestru T2CON.
- Skonfigurowanie modułu CCP do pracy w trybie PWM.

Na **listingu 9** pokazano fragment konfiguracji modułu CCP1. Po skonfigurowaniu na wyjściu CCP1 (RC2) jest generowany przebieg PWM o częstotliwości 4 kHz i wypełnieniu 50%. Po dołączeniu jednej z diod LED do wyjścia CCP1 można obserwować zmianę jasności świecenia w funkcji czasu



Rysunek 7. Pojedynczy wzmacniacz prądowy z ULN2803

Listing 9. Inicjalizacja modułu CCP1 w trybie pracy PWM

```
#define TPWM 124
PR2=TPWM;
CCPR1L=0x3e;//8 starszych bitów duty cycle
CCP1CON=0x20;//2 młodsze bity
TRISC=0;
T2CON=0x07;//preskaler 1/16 TMR2 on
CCP1CON=0x2c;//tryb PWM
```

duty cycle zmienianego zawartością CCPR1L:CCP1CON<5:4>.

Moduł CCP daje możliwość generowania przebiegu PWM w dużym zakresie częstotliwości i o sporej rozdzielczości. Trzeba jednak pamiętać, że wraz ze wzrostem częstotliwości PWM może maleć dostępna rozdzielczość.

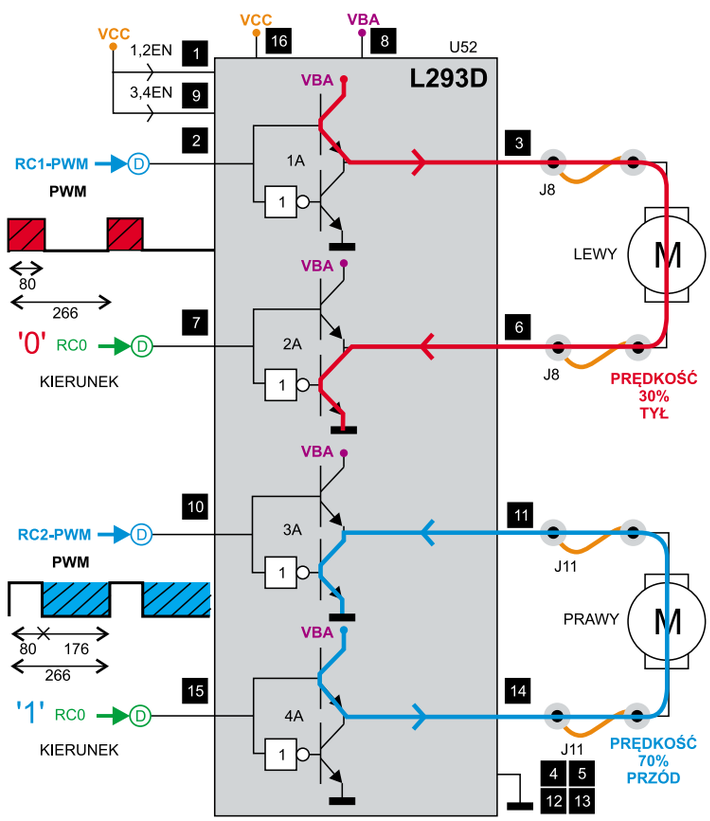
Sterowanie jasnością diod LED jest bardzo użyteczne w regulacji podświetlenia wyświetlaczy LCD. Drugim, często wykorzystywanym zastosowaniem, jest sterowanie silnikami prądu stałego. Takie silniki wykorzystuje się w modelarstwie, automatyce, robotyce itp. Są one zasilane napięciem 3…12 V i pobierają stosunkowo niewielki prąd.

Sterowanie silnikiem małej mocy

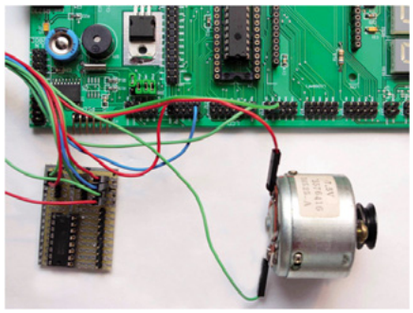
Sterowanie silnikiem polega na zmianie kierunku wirowania i regulacji prędkości obrotowej. Pierwsza próba będzie bardzo prosta. Jeżeli dysponujemy małym silnikiem elektrycznym mogącym pracować przy napięciu +5V, to można wykorzystać napięcie „pokładowe" +5V, a jako wzmacniacz prądu zainstalowany na płytce układ U2 ULN2803. Jeden biegun zasilania silnika podłączamy do +5 V, a drugi do jednego z wyjść złącza J6 – na przykład UOUT1. Sygnał PWM z wyjścia CCP1 łączymy z wejściem UIN1. ULN2803 składa się z 8 wzmacniaczy prądowych (**rysunek 8**).

Doprowadzenie do wejścia napięcia odpowiadającego poziomowi wysokiemu powoduje nasycenie tranzystora i zwarcie wyjścia do masy. W tym czasie silnik jest zasilany napięciem +5 V z drugiego bieguna zasilania. Po podaniu poziomu niskiego tranzystor jest zatkany i silnik nie jest zasilany. Ponieważ tranzystor ULN2803 zwiera wyjście do masy, jest możliwe również zasilanie silnika napięciem wyższym niż napięcie zasilania mikrokontrolera.

Ten stosunkowo prosty sposób sterowania doskonale sprawdza się przy sterowaniu silników obracających się w jedną stronę. Jednak często oprócz regulacji prędkości obrotowej zachodzi potrzeba sterowania zmiany kierunku wirowania wału silnika. W takim przypadku silnik jest sterowany za pomocą mostka umożliwiającego zamianę biegunów napięcia zasilającego silnik. Mostek



Rysunek 8. Sterowanie 2 silników za pomocą L293D

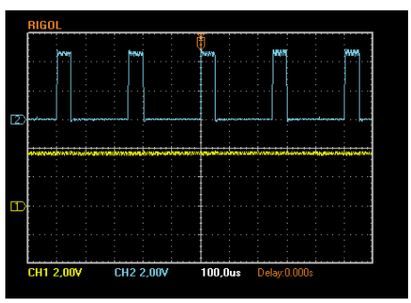


Fotografia 9. Dołączenie sterowania silnikiem przez układ L293D

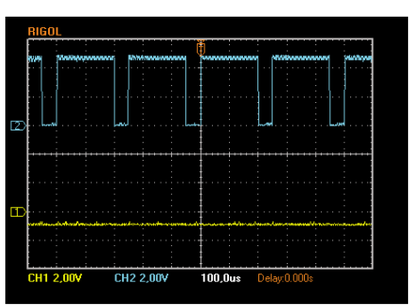
można zbudować z par tranzystorów mocy lub skorzystać ze specjalizowanego układu scalonego. Ja postanowiłem wykorzystać znany i łatwo dostępny układ L293D. Ma on wbudowane cztery pary tranzystorów zasilane napięciem od 4,5 do 36 V. Każda z par może oddawać do obciążenia ciągły prąd 0,6 A. Logiczne układy sterujące akceptują napięcia z zakresu 4,5…7 V. Sposób dołączenia i sterowania dwoma silnikami elektrycznymi małej mocy z wykorzystaniem tego układu pokazano na rysunku 8.

Ponieważ L293D nie jest umieszczony na płytce modułu, do prób zamontowałem układ na płytce uniwersalnej. Wszystkie wyprowadzenia układu L239D są połączone do listew goldpinów, tak aby można było je połączyć z modułem standardowymi kablami. Sposób dołączenia dodatkowej płytki z modułem i silnikiem pokazano na **fotografii 9**. Silnik jest zasilany z zewnętrznego zasilacza napięciem +7,5 V doprowadzonym do nóżki VBA (pin 8 L293D).

Do sterowania wykorzystamy sygnał PWM generowany przez układ CCP1. Do ste-



Rysunek 10. Sterowanie „w lewo", RC0 w stanie wysokim



Rysunek 11. sterowanie „w prawo", RC0 w stanie niskim

Listing 9. Sterowanie silnikiem

```
char MotorCtrl(unsigned char speed,
char dir){
  unsigned char temp;
  if(speed>TPWM) return(-1);
  CCPR1L=speed;//8 starszych bitów
do CCPR1L
  if(dir==LEFT){
    RC0=1;//kierunek w lewo
    CCP1CON=0x0F; //zanegowany
przebieg PWM
  }
  if(dir==RIGHT){
    RC0=0;//kierunek w prawo
    CCP1CON=0x0C;//normalny przebieg
PWM
  }
  return(0);
}
```

Listing 10. Procedura realizująca miękki start silnika

```
char MotorSoftStart(unsigned char
maxspeed ,char dir, unsigned char
del){
unsigned char i;
  if(maxspeed>TPWM)
  return(-1);
  for (i=TPWM/2;i<maxspeed;i++){
    MotorCtrl(i,dir);
    delay(del);
  }
  return(0);
}
```

rowania silnika małej mocy nie jest potrzebna duża, 10-bitowa rozdzielczość. Dlatego będziemy ją ustalać, zapisując tylko 8-bitowy rejestr CCPR1L. Na **listingu 10** umieszczono procedurę o nazwie *MotorCtrl* przeznaczoną do sterowania prędkością i kierunkiem obrotów silnika w sposób pokazany na rysunku 8. Argument *speed* zawiera wartość wpisywaną do CCPR1L i określającą czas trwania *duty cycle*. Argument *dir* określa kierunek wirowania silnika.

Pamiętamy, że wartość wpisana do tego rejestru nie może być większa niż wartość wpisana do PR2. Na początku jest sprawdzany warunek, czy parametr *speed* nie jest większy od stałej TPWM. W czasie inicjalizacji TPWM jest inicjowana wartością 124, a potem ta wartość jest przepisywana do PR2. Jeżeli *speed* jest większy od TPWM, to funkcja kończy działanie i zwraca kod błędu *–1*. Na podstawie wartości *dir* procedura zeruje lub ustawia linię RC0 sterowania kierunkiem obrotów. Kiedy RC0 jest w stanie wysokim, to przebieg PWM musi być zanegowany, bo stan niski PWM włącza zasilanie silnika. Negacja sygnału jest uzyskiwana przez przeprogramowanie rejestru CCP1CON.

Mając do dyspozycji taką procedurę, można zobaczyć w jaki sposób współczynnik wypełnienia wpływa na prędkość silnika. W przypadku mojego silnika (nieobciążonego) prawidłowa praca zaczynała się przy współczynniku wypełnienia równym 50%. Wówczas silnik ruszał i powoli się obracał. Poniżej tej wartości efektywne sterowanie nie było możliwe. Na **rysunku 10** i **rysunku 11** pokazano oscylogramy sterowania silnikiem dla obrotów w lewo i w prawo. Górny przebieg to PWM, a dolny stan na linii sterującej RC0.

W wielu zastosowaniach stosuje się tzw. miękki start silnika. Polega on na stopniowym zwiększaniu obrotów silnika do zadanej wartości maksymalnej.

Na **listingu 10** umieszczono procedurę *MotorSoftStart* z argumentami *maxspeed* (docelowa prędkość), *dir* (kierunek wirowania) i *del* (czas pomiędzy kolejnymi zmianami współczynnika wypełnienia). Na początku jest sprawdzany warunek, czy *maxspeed* nie jest większy niż *TPWM*. Potem w pętli jest zwiększany współczynnik wypełnienia od TPWM/2 do maxspeed. Opóźnienie reguluje czas, w którym silnik będzie rozpędzał się do maksymalnej prędkości. Sterowanie silnikiem realizuje wcześniej opisana procedura *MotorCtrl*.

Podsumowanie

Omówiliśmy trzy metody generowania przebiegu PWM. Znając ich wady i zalety, wymagania aplikacji i dostępne zasoby mikrokontrolera, można wybrać najlepsze rozwiązanie. W kolejnym odcinku zajmiemy się interfejsem użytkownika.

Tomasz Jabłoński, EP