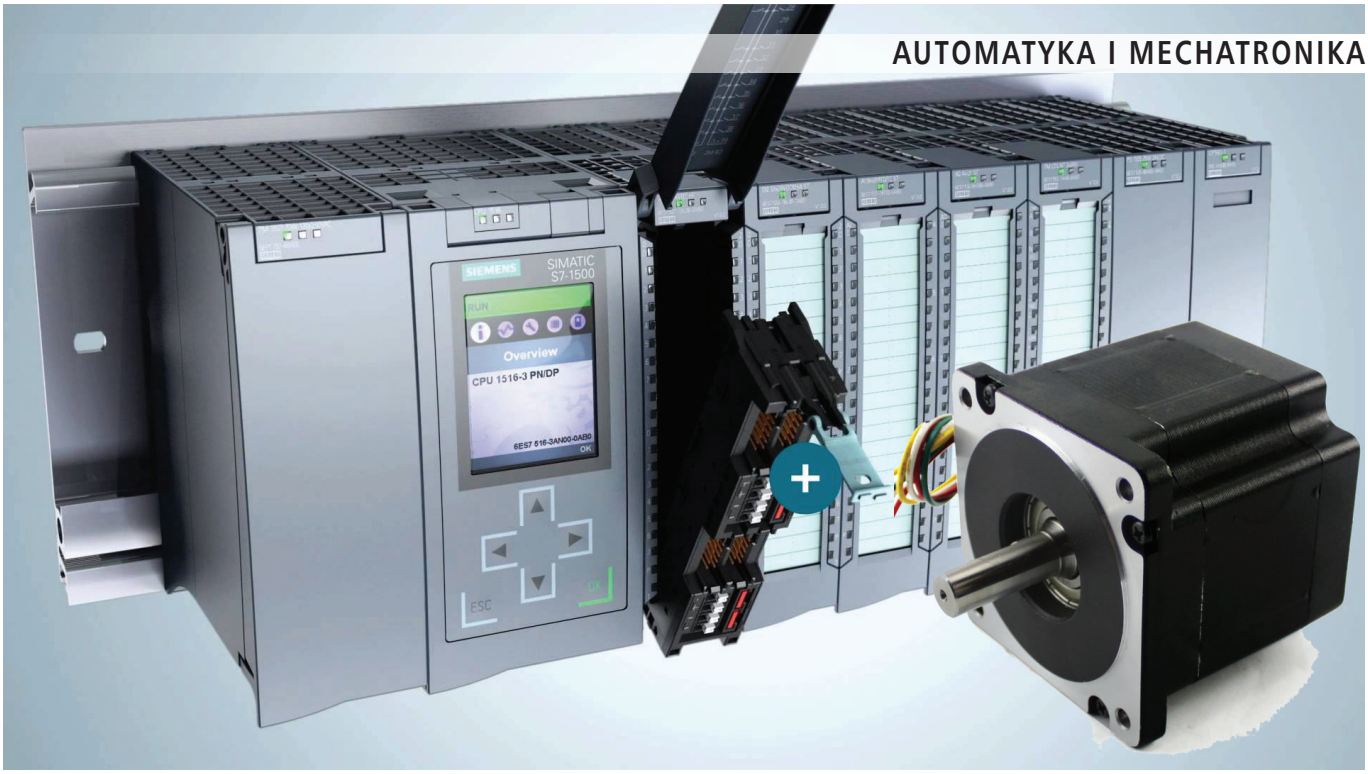
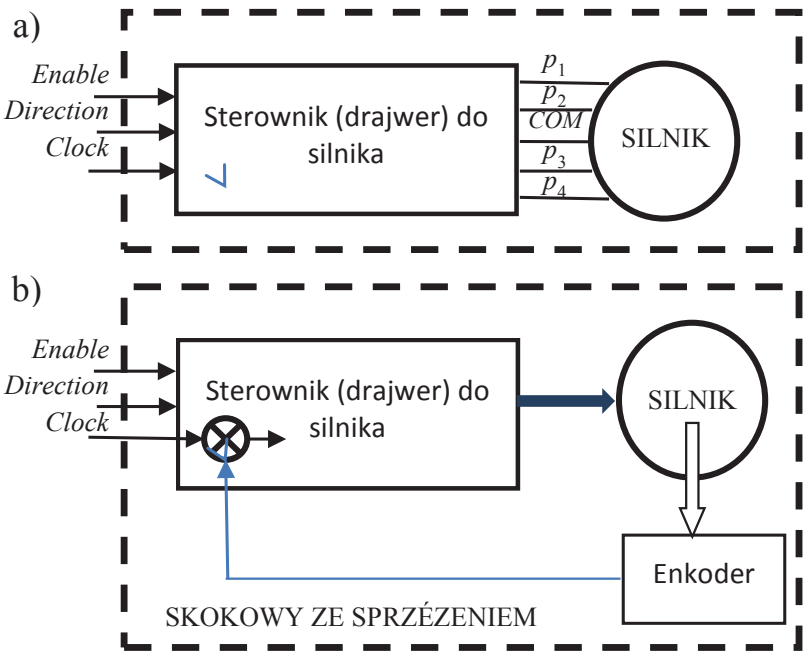



# Sterowanie silnikiem skokowym za pomocą sterownika S7-1500 (1)

*Sterowniki S7-1500 są przystosowane do bezpośredniego sterowania pracą silników skokowych. Silniki takie są szeroko stosowane w urządzeniach, w których wymagany jest precyzyjnie kontrolowany ruch.*




**Rys. 1. Schemat blokowy sterownika silnika skokowego**

Położenie kątowe jest ustawiane w napędach z tymi silnikami w układzie otwartym bez sprzężenia zwrotnego (**rysunek 1**). Tylko w systemach, od których wymaga się dużej niezawodności i występuje duża zmienność obciążenia i prędkości stosuje się regulację z sygnałem w sprzężeniu zwrotnym od enkodera, który generuje impulsy.

Każdy typ silnika może być sterowany na co najmniej cztery sposoby, od których m.in. zależy rozdzielczość ruchu, czyli wartość najmniejszego przemieszczenia k*ątowego* na jeden skok. Dość często towarzyszy temu zmniejszenie dokładności (**rysunek 2**):

- falowy, jednofazowy, T/4 o rozdzielczości często 1,8 °/skok,
- pełnokrokowy, dwufazowy, T/2 o rozdzielczości często 1,8 °/skok,
- półkrokowy, 3T/8 o rozdzielczości często 0,9 °/skok,
- mikrokrokowy, często 1/3÷1/32 kroku.

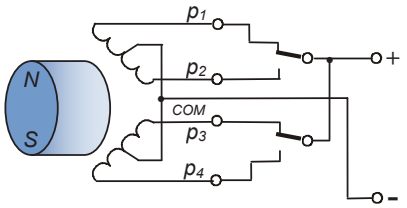
Zmniejszenie dokładności na jeden skok (krok) wynika z tego że silniki jednofazowe mają dokładność około 5%, to dwufazowy algorytm sterowania ma dokładność rzędu 10%, a 32 mikrokrokowy może dać błąd 160%.

Podczas pracy silnika w układzie zamkniętym stosuje tzw. rozdzielczość pozy-

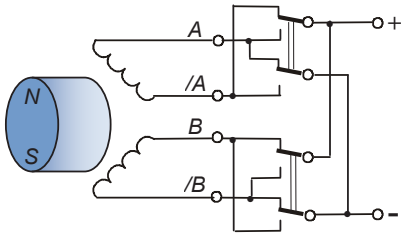
**Rodzaje silników skokowych**

W zależności od rodzaju uzwojeń i sposobu połączeń wyróżniamy następujące silniki skokowe:

- Unipolarne (rysunek poniżej), prąd płynie zawsze w tym samym kierunku tylko w połowie uzwojenia, czyli silnik nie osiąga pełnego momentu. Silnik ma 5 wyprowadzeń albo 4 do biegunów (p1÷p4) i jeden wspólny (COM) bądź 6 wyprowadzeń, po trzy dla każdego z uzwojeń (początek, środek i koniec).




- Bipolarne (rysunek poniżej), sterowanie jest dla nich bardziej rozbudowane, prąd płynie przez całe uzwojenie przy pierwszym impulsie raz w jednym a przy drugim impulsie w przeciwnym kierunku, silnik osiąga pełny moment i ma 4 wyprowadzenia.




- Uniwersalne, 8-przewodowe. Stopnie mocy umożliwiające realizację najbardziej zaawansowanych algorytmów sterowania, do których konieczne jest podłączenie wszystkich wyprowadzeń są jednak rzadko spotykane. W praktyce łączy się odpowiednio wyprowadzenia tak, że silnik taki może pracować zarówno w trybie unipolarnym, jak i bipolarnym szeregowym i równoległym.

cjonowania albo rozdzielczość osi. Aby ją obliczyć niezbędna jest znajomość sposobu zliczania impulsów z enkodera. Przykłady obliczania rozdzielczości liniowej osi $R_{osi}$ z enkoderem:

- Inkrementalny enkoder ma następujące dane:
  - liczba skoków na obrót $R_{enk}$=5000 skoków,
  - odległość pokonana podczas obrotu: $x$=1000 mm,
  - 1 skok=1 impuls (zliczanie kwadraturowe),

$$R_{osi} = \frac{\text{Odległość pokonana podczas obrotu}}{\text{Liczba skoków na obrót}} = \frac{1000\ mm}{5000\ skoków} = 0{,}2\frac{mm}{4\ impulsy} = 0{,}05\frac{mm}{impuls}$$

- Enkoder z interfejsem SSI ma następujące parametry:
  - liczba skoków na obrót $R_{enk}$=4096 skoków,
  - odległość pokonana podczas obrotu $x$=1000 mm,
  - 1 skok=1 impuls (zliczanie jednofazowe),

$$R_{osi} = \frac{\text{Odległość pokonana podczas obrotu}}{\text{Liczba skoków na obrót}} = \frac{1000\ mm}{4096\ skoków} = 0{,}2441\frac{mm}{impuls}$$

Silnik skokowy możemy sterować wykorzystując specjalizowany drajwer (sterownik) albo bezpośrednio ze sterownika, do czego niezbędne są stopnie mocy dla każdego z biegunów w silniku.

Sterowanie silnikiem skokowym lub serwosilnikiem sprowadza się do podania trzech sygnałów: *Enable* EN, *Direction* DIR i *Clock* CLK, co można zrealizować na trzy sposoby:

a) przy użyciu dodatkowo instrukcji normalizacji NORM i skalowania SCALE,
b) wykorzystując jedno narzędzie do konfiguracji osi np. *TO_Axis_PTO z Technology objects w drzewie projektu, które zawiera w zwartej formie podstawowe instrukcje do sterowania ruchem Motion Control* (MC) a następnie napisanie programu,
c) wykorzystując bezpośrednio instrukcje zawarte w bibliotece *Motion Control*.

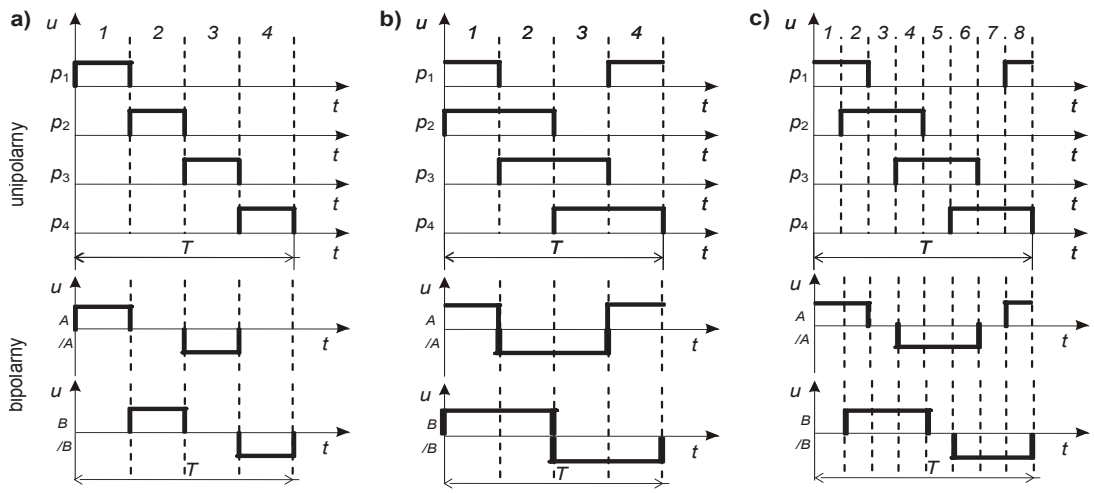
Najprostszy algorytm sterowania obrotami silnika skokowego przedstawiono na **listingu 1**. Trzy sygnały w drajwerze są niezbędne do sterowania ręcznego wartością z potencjometru, którym ustawia się częstotliwość impulsowania wejścia zegarowego CLK. Do instrukcji normalizacji NORM wprowadzamy parametry odczytane z potencjometru dla jego skrajnych położeń a jego sygnał wyjściowy zmieniający się od 1 do zera odejmujemy, od 1,0 aby przywrócić prawoskrętny wzrost wartości z potencjometru. Styki z literką H na końcu *służą do sterowania* silnikiem z ekranu.

Wykorzystanie gotowego drajwera do silnika skokowego daje prostote obsługi tj. wymagane jest jedynie trzy sygnały sterujące, ale niesie za sobą wadę, jaką jest brak wykorzystania w pełni możliwości napędu. Ponadto firmowy drajwer nie umożliwia poznania algorytmów sterujących silnikiem.

Głównym celem poniżej przytoczonego przykładu dla sterownika S7-1500 będzie przybliżenie prostych i skutecznych metod sterowania silnikiem krokowym (krok, półkrok, mikrokrok). Ponadto przy okazji sterowania mikrokrokowego przedstawione zostaną metody programowej generacji sygnału PWM na dowolnym wyjściu cyfrowym sterownika.

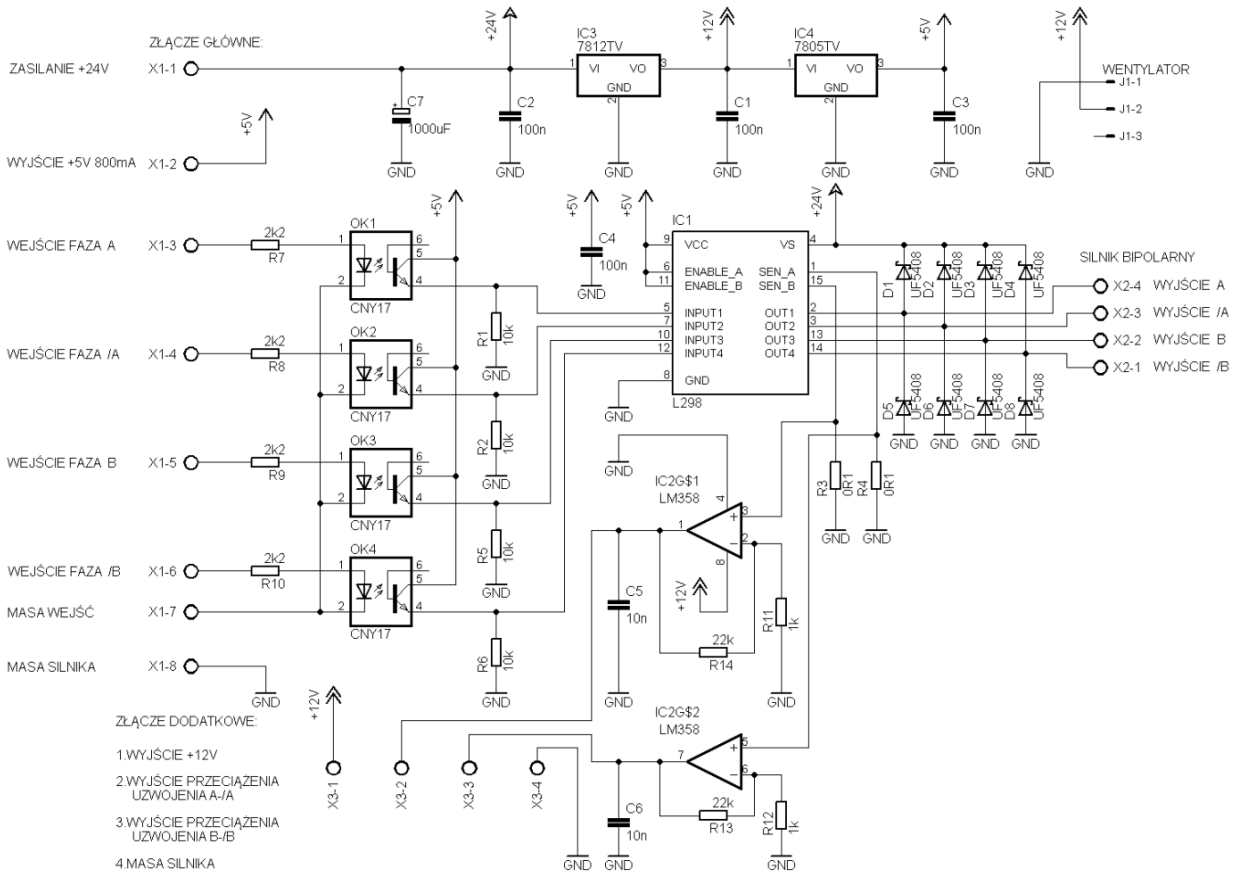
W celach tetsowych zaprojektowano i wykonano stopnie mocy dla silnika unipolarnego i bipolarnego umozliwiajace zrealizwac sterowanie ze sterownika. Na **rysunku 3** przedstawiono schemat stopnia mocy tylko do silnika bipolarnego ze względu na jego większą uniwersalność. Wraz z schematem podane zostały kody kolorów uzwojeń, ułatwiajace podłączania sterownika do stopnia mocy. Jednakże proszę zwrócić uwagę że podane kody są tzw. *standardem de facto*, toteż przed podłączeniem warto sprawdzić omomierzem uzwojenia silnika.

Omawiany stopień mocy został wyposażony również w obwód pomiaru prądu, który można podłączyć wprost do wejścia analogowego (0÷10 V) sterownika. Jednakże ze względu na niską tolerancję wysokowatowych rezystorów nie należy na tym pomiarze polegać. Praktyczne wykorzystanie tego wyjścia ogranicza się do wykrywania przeciążeń – np. spalenia uzwojeń silnika w celu jego awaryjnego wyłączenia.




**Rysunek 2. Sposoby zasilania uzwojeń w kolejnych krokach w unipolarnym i bipolarnym silniku skokowym: a) falowy, T/4, b) pełnokrokowy, T/2, c) półkrokowy, 3T/8**

```
Listing 1. Algorytm „ręcznego" sterowania obrotami silnika skokowego
  "EN":=("Start" OR "EN" OR "ENE") AND NOT "Stop"AND NOT "StopE";
  // Zezwolenie na pracę silnika, ENE równoległy styk pomocniczy do sterowania z ekranu
  "DIR":= ("DirL" OR "DIR"OR "DIRLE") AND NOT "DirR" AND NOT "DIRRE";
  //Wybór kierunku pracy, DIRLE równoległy, DIRRE szeregowy styk do sterowania z ekranu
  "czas":=SCALE_X(MIN:=20, VALUE:=1.0-NORM_X(MIN:=0, VALUE:="pot", MAX:=28000), MAX:=300);
  //Obliczenie okresu miedzy skokami, odjęcie od wartości 1.0 aby odwrócic obroty z potencjometru
  "IEC_Timer_0_DB".TON(IN:=NOT "imp", PT:="czas", Q=>"imp");
  // Najprostszy generator impulsów z ich ustawianą  częstotliwością z potencjometru
  "CLK":="EN" AND "imp";// Zezwolenie zliczania
```




**Rysunek 3. Schemat stopnia mocy silnika bipolarnego**

**Tabela 1. Kolejne stany silnika w bloku DB Kody kroków**

| Pelne_kroki | Array [0 .. 3] of Byte | | | | | | A+; B+; A-; B-; |
|---|---|---|---|---|---|---|---|
| Pelne_kroki[0] | Byte | 16#01 | | | | | A |
| Pelne_kroki[1] | Byte | 16#04 | | | | | B |
| Pelne_kroki[2] | Byte | 16#02 | | | | | /A |
| Pelne_kroki[3] | Byte | 16#08 | | | | | /B |

a) Sterowanie falowe, jednofazowe. Jako że sterownik, do którego został podłączony stopień mocy będzie miał do wykonania wiele innych zadań, obciążenie instrukcjami odwołującymi się do przestrzeni WE/WY powinno być jak najmniejsze. W tym celu w bloku DB *Kody kroków* została utworzona tablica kolejnych stanów silnika (**tabela 1**). Warto już teraz zwrócić uwagę na zalety tablicowania – odwołanie się do kolejnego elementu tablicy praktycznie zawsze będzie szybsze w stosunku do generowania kolejnego kroku na podstawie aktualnego. Przy tym często tablicowanie funkcji matematycznych nie ma sensu, gdyż strata pamięci w porównaniu do zysku ze zwiększenia prędkości wykonania algorytmu jest niepraktycznie duża.

**Tabela 2. Rozszerzona tablica stanów silnika w bloku DB Kody kroków**

| Pol_kroki | Array [0 .. 7] of Byte | | | | | | A+; A+B+; B+; B+A-; A-; A-B-; B-; B-A+; |
|---|---|---|---|---|---|---|---|
| Pol_kroki[0] | Byte | 16#01 | | | | | A |
| Pol_kroki[1] | Byte | 16#05 | | | | | AB |
| Pol_kroki[2] | Byte | 16#04 | | | | | B |
| Pol_kroki[3] | Byte | 16#06 | | | | | B/A |
| Pol_kroki[4] | Byte | 16#02 | | | | | /A |
| Pol_kroki[5] | Byte | 16#0A | | | | | /A/B |
| Pol_kroki[6] | Byte | 16#08 | | | | | /B |
| Pol_kroki[7] | Byte | 16#09 | | | | | /BA |

Jak widać, w kolejnych krokach kolejno załączane są uzwojenia *A, B, /A, /B* analogicznie jak w przypadku silnika unipolarnego $p_1, p_2, p_3, p_4$. Ustawiona jedynka na którymkolwiek bicie jest tożsama z załączeniem odpowiadającego jej uzwojenia. Program sterujący przedstawiono na **listingu 2**.

Warto zwrócić uwagę na to, że zapis do taga *WY_DRIVER* ma miejsce nie tylko wtedy, kiedy pojawia się dodatnie zbocze zegara, a zawsze kiedy wejście *ENZ* jest w stanie wysokim. Służy to temu, aby utrzymać moment statyczny po ponownym włączeniu sygnału *ENZ*.

b) Sterowanie półkrokowe. Każda kolejna próba zmniejszenia kąta o jaki obraca się wał silnika krokowego w pojedyńczym kroku sprowadza się do wprowadzenia stanów pośrednich do już istniejących. W przypadku półkroku, w stanach pośrednich, osiąga się to poprzez załączenie obu uzwojeń, oczywiście nie tej

Listing 2. Program sterowania falowego

```
1  IF "FirstScan" THEN //w pierwszym cyklu obiegu sterownika
2    "aktualny_krok" := 0; //zmienna pomocnicza przechowująca numer
3                          //aktualnego stanu silnika
4    "WY_DRIVER" := 0;   //zapis zera na wyjście bajtowe, do którego podłączony jest driver
5                        //tj. wszystkie cewki silnika zostają wyłączone
6  END_IF;
7
8  "R_TRIG_DB"(CLK:="CLKZ",  //z każdym dodatnim zboczem podanym na wejście CLKZ
9              Q=>"ZBOCZE_DOD_ZEGAR"); //zostaje ustawiona flaga ZBOCZE_DOD_ZEGAR
10
11 IF NOT "ENZ" THEN   //gdy wejście ENZ (Enable) ma stan niski
12   "WY_DRIVER" := 0; //wszystkie cewki silnika wyłączone
13 END_IF;
14
15 IF "ENZ" THEN     //gdy wejście ENZ (Enable) ma stan wysoki
16
17   IF "ZBOCZE_DOD_ZEGAR" THEN  //i gdy pojawiło się dodatnie zbocze od zegara
18
19     IF "DIRZ" = TRUE THEN //gdy DIRZ ma stan wysoki, to silnik powinen obracać się w prawo
20
21         "aktualny_krok" := "aktualny_krok" + 1; //kolejny stan silnika
22
23        IF "aktualny_krok" > 3 THEN  //kolejne stany w tablicy mają numery 0..3
24           "aktualny_krok" := 0;     //powrót do początku
25         END_IF;
26
27     ELSE                 //w przeciwnym wypadku obroty w lewo
28
29       IF "aktualny_krok" = 0 THEN //gdy numer aktualnego stanu jest już równy zeru
30         "aktualny_krok" := 3;
31       ELSE
32         "aktualny_krok" := "aktualny_krok" - 1;
33       END_IF;
34
35     END_IF;
36
37
38   END_IF;
39
40   "WY_DRIVER" := "Kody_krokow".Pelne_kroki["aktualny_krok"];
41   //przepisanie aktualnego stanu silnika na wyjście bajtowe
42
43 END_IF;
```

samej pary! Tzn. przykładowo, zostają załączone uzwojenia */B* i *A*, natomiast nigdy *A* i */A* jednocześnie. W przypadku silnika unipolarnego taka pomyłka nie prowadzi do uszkodzenia drajwea. Sytuacja ta prowadzi do drastycznego spadku momentu, gdyż dochodzi do zjawiska równoważenia się pól magnetycznych w silniku. **Tabela 2** jest rozszerzoną tablicą stanów.

Pewnym udoskonaleniem w programie sterowania półkrokowego jest implementacja obsługi przełączników do regulacji prędkości. Jak widać to w linii 10 programu (**listing 3**), na tag *PRZELACZNIKI* reprezentujący bajt wejścia cyfrowego najpierw zostaje nałożona maska 16#F0. Wynikiem tej operacji jest usunięcie z bajtu czterech najmłodszych bitów. Do czterech najmłodszych bitów zostały podłączone przełączniki zadające sygnały *CLKZ*, *DIRZ* oraz *ENZ*, toteż mogły by one wprowadzać zakłócenia do zadawanej wartości. Zaraz po tym wykonywana jest konwersja, ponieważ zegary w S7-1200 i S7-1500 wymagają typu danych S7Time. Przykładowo, zapisana w przykładzie konwersja zamienia wartości 16...240, na T#16 ms…T#240 ms. Następnie, zostaje wywołana instancja zegara TON, gdzie jako parametr *IN* przyjmowane jest wyrażenie *"ENZ" AND (NOT "WYJ_ZEGAR")*. Jak można łatwo wnioskować, zegar zadziała w następujący sposób – po upłynięciu czasu zadanego przez tag *"CZAS"*, tag *"WYJ_ZEGAR"* przyjmie stan niski. Spowoduje to, że całe wyrażenie parametru *IN* będzie mieć stan niski (na czas jednego cyklu sterownika). Wtedy, w kolejnym cyklu, dojdzie do restartu zegara i rozpoczęcia odmierzania czasu od nowa. Wobec tego tag *"WYJ_ZEGAR"* będzie miał stan niski przez odcinek czasu określony tagiem *"CZAS"* i stan wysoki przez jeden cykl sterownika (typowo 1 ms). Zatem, zbocze dodatnie, na które oczekuje instrukcja *R_TRIG* w linii 24 będzie pojawiać się co *"CZAS"*. Pozostała część programu pozostaje bez zmian poza wartościami przy warunkach IF, ponieważ w półkroku tablica kolejnych stanów silnika zawiera osiem elementów.

c) Mikrokrok ¼. Algorytm dla kroku i półkroku wyczerpują możliwości sterowania silnikiem krokowym jedynie przy użyciu sygnałów *WŁĄCZ*/*WYŁĄCZ*. Dalsza regulacja położenia wirnika między biegunami będzie wymagać różnicowa-

List. 3. Program sterowania półkrokowego

```
IF "FirstScan" THEN //w pierwszym cyklu obiegu sterownika
  "aktualny_krok" := 0; //zmienna pomocnicza przechowująca numer
                        //aktualnego stanu
  "WY_DRIVER" := 0; //zapis zera na wyjście bajtowe, do którego podłączony jest driver
                    //tj. wszystkie cewki silnika zostają wyłączone
END_IF;

//do czterech najstarszych bitów bajtowego wejścia zostały podłączone
//przełączniki odpowiedzialne ze regulację prędkości obrotowej silnika
"CZAS" := BYTE_TO_TIME("PRZELACZNIKI" AND 16#F0);
//tag czas przyjmie wartości od T#16ms do T#240ms,
//wliczając w to T#0ms, gdy wszystkie przełączniki będą wyłączone
IF "CZAS" = T#0ms THEN  //w takim przypadku
  "CZAS" := T#1ms;      //tag czas zostaje ustawiony na T#1ms
END_IF;

//deklaracja timera odmierzającego ustawiony czas
"IEC_Timer_0_DB".TON(IN:="ENZ" AND (NOT "WYJ_ZEGAR"),
                     //zegar jest aktywny, gdy wejście ENZ ma stan wysoki
                     //i na jego wyściu jest stan niski
                     PT:= "CZAS", //ustawiony przełącznikami czas
                     Q=>"WYJ_ZEGAR"); //wyście zegara

"R_TRIG_DB_1"(CLK:="WYJ_ZEGAR", //z każdym dodatnim zboczem na wyjściu zegara
              Q=>"ZBOCZE_DOD_ZEGAR"); //zostaje ustawiona flaga ZBOCZE_DOD_ZEGAR

IF NOT "ENZ" THEN   //gdy wejście ENZ ma stan niski
  "WY_DRIVER" := 0; //wszystkie cewki silnika wyłączone
END_IF;

IF  "ENZ" THEN  //gdy wejście ENZ ma stan wysoki

  IF "ZBOCZE_DOD_ZEGAR" THEN  //i gdy pojawiło się dodatnie zbocze od zegara

    IF "DIRZ" = TRUE THEN //gdy DIRZ ma stan wysoki, to silnik powinien obracać się w prawo

        "aktualny_krok" := "aktualny_krok" + 1; //kolejny stan silnika

        IF "aktualny_krok" > 7 THEN //w przypadku półkroku, kolejne stany mają numery 0..7
          "aktualny_krok" := 0;
        END_IF;

    ELSE  //w przeciwnym wypadku obroty w lewo

      IF "aktualny_krok" = 0 THEN
        "aktualny_krok" := 7;
      ELSE
        "aktualny_krok" := "aktualny_krok" - 1;
      END_IF;

    END_IF;

  END_IF;

  "WY_DRIVER" := "Kody_krokow".Pol_kroki["aktualny_krok"];
  //przepisanie aktualnego stanu silnika na wyjście bajtowe

END_IF;
```

nia natężenia prądu w uzwojeniach silnika. Do tego celu wykorzystany zostanie programowo zaimplementowany generator PWM (**listing 4**).

W takim generatorze, ustawiając minimalną wartość, jaką przyjmuje typ danych S7Time, tj. T#1ms, uzyskuje się sterowanie typu *WŁĄCZ/WYŁĄCZ*. Wtedy implementacja tego kodu nie ma sensu, ponieważ nic nie wnosi. Toteż minimalna sensowna podstawa czasu to T#2 ms (PWM 500 Hz), gdzie uzyskuje się następujące stany, *WYŁĄCZ(0%), 50%, WŁĄCZ(100%)*.

**Tomasz Starak**

*Opracowano na podstawie materiałów z książki „Język tekstu strukturalnego w sterownikach SIMATIC S7-1200 i S7-1500” (autor – Janusz Kwaśniewski, planowana do wydania w 2014 roku przez Wydawnictwo BTC) oraz materiałów firmowych firmy Siemens.*

List. 4. Program generatora PWM

```
 9  //PROGRAMOWY GENERATOR PWM
10 IF "WYJ_ZEGAR" THEN   //Gdy zegar doliczy do 100%, włączane są wszystkie wyjścia
11    "A_PLUS" := TRUE;
12    "A_MINUS" := TRUE;
13    "B_PLUS" := TRUE;
14    "B_MINUS" := TRUE;
15 END_IF;
16
17 "IEC_Timer_0_DB_1".TON(IN:=NOT "WYJ_ZEGAR",
18                         //tak zapisany warunek IN powoduje samoistne resetowanie
19                         //zegara, gdy ten doliczy do 100%
20                         PT:=T#4ms, //podstawa czasu, tutaj PWM 250Hz
21                         Q=>"WYJ_ZEGAR",  //flaga wyjściowa
22                         ET=> "AKTUALNIE_ZEGAR");
23                        //tag AKTUALNIE_ZEGAR przechowuje aktualną wartość czasu
24
25
26 IF "AKTUALNIE_ZEGAR" >= BYTE_TO_TIME("PWM_0") THEN  //Rejestr porównawczy 0
27    //gdy zegar doliczy do wartości większej niż PWM_0
28    "A_PLUS" := FALSE;
29    //to wyjście zmienia stan na niski
30 END_IF;
31
32 IF "AKTUALNIE_ZEGAR" >= BYTE_TO_TIME("PWM_1") THEN  //Rejestr porównawczy 1
33    "A_MINUS" := FALSE;
34 END_IF;
35
36 IF "AKTUALNIE_ZEGAR" >= BYTE_TO_TIME("PWM_2") THEN  //Rejestr porównawczy 2
37    "B_PLUS" := FALSE;
38 END_IF;
39
40 IF "AKTUALNIE_ZEGAR" >= BYTE_TO_TIME("PWM_3") THEN  //Rejestr porównawczy 3
41    "B_MINUS" := FALSE;
42 END_IF;
43
44  //KONIEC PROGRAMOWEGO GENERATORA PWM
```