Sterowanie silnikiem skokowym za pomocą sterownika S7-1500 (2)

Sterowniki S7-1500 są przystosowane do bezpośredniego sterowania pracą silników skokowych. Silniki takie są szeroko stosowane w urządzeniach, w których wymagany jest możliwość precyzyjnego kontrolowania ruchu.

Pierwszą część artykułu zakończyliśmy omówieniem konfiguracji generatora PWM, który działa w następujący sposób:

- na początku każdego cyklu, czyli wtedy, kiedy wyjście zegara przyjmuje stan wysoki na wszystkich wyjściach również ustawiany jest stan wysoki (oczywiście za wyjątkiem pierwszego cyklu sterownika po przejściu do trybu RUN, gdzie wszystkie wyjścia zostaną wyłączone, podobnie jak wyjście zegara),
- rozpoczyna się odliczanie czasu w zegarze, gdzie w każdym cyklu obiegu sterownika uaktualniana jest wartość taga *"AKTUALNIE_ZEGAR"*,
- w każdym cyklu, dla każdego kanału sprawdzany jest warunek, tj. czy aktualna wartość czasu jest większa lub równa od zadanej, jeśli tak, to odpowiadające zadanej wartości wyjście zmienia stan na niski,
- gdy zegar doliczy do 100%(*T#4ms*), to następuje ponowne wpisanie jedynek na wszystkie wyjścia oraz samoczynny reset zegara,
- cykl zaczyna się od początku.

Wprowadzenie sterowania mikrokowego wymaga też bardziej skomplikowanych zmian w tablicy stanów (**rysunek 4**), gdzie teraz oprócz konieczności przechowywania stablicowanych kolejnych stanów silnika należy przechować też procentową wartość prądu, która powinna przepływać przez odpowiednie uzwojenia.

Cztery najmłodsze bity każdego z 16 bajtów nadal niosą informację o tym, które uzwojenia powinny być włączone w danym momencie. Korzystając z założenia, że nigdy nie wystąpi taka kombinacja, która wymagałaby włączenia dwóch uzwojeń jednej pary z różnymi wartościami prądu (a nie wystąpi nigdy, gdyż powód omówiono przy okazji mikrokroku), cztery najstarsze bity

16		Mikrokrok_1_4	Array [0 .. 15] of Byte					
17		Mikrokrok_1_4[0]	Byte	16#69				
18		Mikrokrok_1_4[1]	Byte	16#21				
19		Mikrokrok_1_4[2]	Byte	16#65				
20		Mikrokrok_1_4[3]	Byte	16#A5				
21		Mikrokrok_1_4[4]	Byte	16#95				
22		Mikrokrok_1_4[5]	Byte	16#84				
23		Mikrokrok_1_4[6]	Byte	16#96				
24		Mikrokrok_1_4[7]	Byte	16#A6				
25		Mikrokrok_1_4[8]	Byte	16#66				
26		Mikrokrok_1_4[9]	Byte	16#22				
27		Mikrokrok_1_4[10]	Byte	16#6A				
28		Mikrokrok_1_4[11]	Byte	16#AA				
29		Mikrokrok_1_4[12]	Byte	16#9A				
30		Mikrokrok_1_4[13]	Byte	16#88				
31		Mikrokrok_1_4[14]	Byte	16#99				
32		Mikrokrok_1_4[15]	Byte	16#A9				

Rysunek 4. Zmodyfikowana tablica kolejnych stanów silnika w bloku DB *Kody kroków*

Listing 5. Program sterowania mikrokrokowego ¼

```
IF "FirstScan" THEN //w pierwszym cyklu obiegu sterownika
   "aktualny_krok" := 0; //zmienna pomocnicza przechowująca numer
                         //aktualnego stanu
   "A_PLUS" := FALSE;  //wyzerowanie wszystkich wyjść
   "A_MINUS" := FALSE;
   "B_PLUS" := FALSE;
   "B_MINUS" := FALSE;
END_IF;
 //PROGRAMOWY GENERATOR PWM
IF "WYJ_ZEGAR" THEN   //Gdy zegar doliczy do 100%, włączane są wszystkie wyjścia
   "A_PLUS" := TRUE;
   "A_MINUS" := TRUE;
   "B_PLUS" := TRUE;
   "B_MINUS" := TRUE;
END_IF;
"IEC_Timer_0_DB_1".TON(IN:=NOT "WYJ_ZEGAR",
                       //tak zapisany warunek IN powoduje samoistne resetowanie
                       //zegara, gdy ten doliczy do 100%
                       PT:=T#4ms, //podstawa czasu, tutaj PWM 250Hz
                       Q=>"WYJ_ZEGAR",  //flaga wyjściowa
                       ET=> "AKTUALNIE_ZEGAR");
                      //tag AKTUALNIE_ZEGAR przechowuje aktualną wartość czasu
IF "AKTUALNIE_ZEGAR" >= BYTE_TO_TIME("PWM_0") THEN  //Rejestr porównawczy 0
   //gdy zegar doliczy do wartości większej niż PWM_0
   "A_PLUS" := FALSE;
   //to wyjście zmienia stan na niski
END_IF;
IF "AKTUALNIE_ZEGAR" >= BYTE_TO_TIME("PWM_1") THEN  //Rejestr porównawczy 1
   "A_MINUS" := FALSE;
END_IF;
IF "AKTUALNIE_ZEGAR" >= BYTE_TO_TIME("PWM_2") THEN  //Rejestr porównawczy 2
   "B_PLUS" := FALSE;
END_IF;
IF "AKTUALNIE_ZEGAR" >= BYTE_TO_TIME("PWM_3") THEN  //Rejestr porównawczy 3
   "B_MINUS" := FALSE;
END_IF;
 //KONIEC PROGRAMOWEGO GENERATORA PWM
"R_TRIG_DB_2"(CLK:="CLKZ",  //z każdym dodatnim zboczem
              //zadanym na wejście CLKZ
              Q=>"ZBOCZE_DOD_ZEGAR"); //zostaje ustawiona flaga ZBOCZE_DOD_ZEGAR
IF NOT "ENZ" THEN //gdy wejście EN ma stan niski
   "PWM_0" := 0; //zostają wyzerowane wszystkie wyjścia
   "PWM_1" := 0;
   "PWM_2" := 0;
   "PWM_3" := 0;
END_IF;
```

Listing 6. Generator PWM o zwiększonej częstotliwości

```
 //PROGRAMOWY GENERATOR PWM

CASE BYTE_TO_INT("PWM_0") OF  //instrukcja CASE wymaga typu danych INT
   0: "A_PLUS" := FALSE; //jeżeli PWM_0 = 0%, to wyjście jest ciągle  wyłączone
   2: "A_PLUS" := NOT "A_PLUS"; //jeżeli PWM_0 = 50%, to stan wyjścia
                                //z każdym cyklem jest przełączany na przeciwny
   4: "A_PLUS" := TRUE; //jeżeli PWM_0 = 100%, to wyjście jest ciągle włączone
END_CASE;

CASE BYTE_TO_INT("PWM_1") OF
   0: "A_MINUS" := FALSE;
   2: "A_MINUS" := NOT "A_MINUS";
   4: "A_MINUS" := TRUE;
END_CASE;

CASE BYTE_TO_INT("PWM_2") OF
   0: "B_PLUS" := FALSE;
   2: "B_PLUS" := NOT "B_PLUS";
   4: "B_PLUS" := TRUE;
END_CASE;

CASE BYTE_TO_INT("PWM_3") OF
   0: "B_MINUS" := FALSE;
   2: "B_MINUS" := NOT "B_MINUS";
   4: "B_MINUS" := TRUE;
END_CASE;

 //KONIEC PROGRAMOWEGO GENERATORA PWM
```

wykorzystano do przechowania informacji o udziale prądu. I tak, bity o numerach 7 i 6 przechowują informację, jaki powinien być procentowy udział prądu w uzwojeniu B silnika (B+ lub B-, w zależności od tego, które z nich jest załączone), natomiast 5 i 4 analogicznie w uzwojeniu A. Jak łatwo zauważyć, taka konwencja pozwala bez zwiększenia zapotrzebowania na pamięć, realizację mikrokroku 1/8 – gdzie potrzebne są 4 różne poziomy prądu (bo na dwóch bitach można zapisać 4 różne liczby). Współpracujący z nową tablicą kompletny kod przedstawiony jest na **listingu 5**.

Program wygląda analogicznie do omawianych wcześniej, jedyną zmianą poza oczywiście implementacją generatora jest dodanie sekcji rozpoznającej poziomy prądu. Jednakże po wgraniu programu do sterownika powstaną problemy – mianowicie przy tak niskiej częstotliwości PWM wirnik silnika w położeniach pośrednich najprawdopodobniej będzie drgał. Konieczne jest zwiększenie częstotliwości PWM – co można uczynić na dwa sposoby: zmieniając podstawę czasu na *T#2ms* w podanym kodzie lub korzystając z rozwiązania bardziej sprytnego, pokazanego na **listingu 6**.

Ustawienie wypełnienia na 50% powoduje w każdym cyklu sterownika naprzemienne ustawianie zera i jedynki na bicie wyjściowym. Jako że czas cyklu obiegu sterownika wynosi 1 ms, to na wyjściu generowany jest sygnał *WŁĄCZ* przez 1 ms i *WYŁĄCZ* przez 1 ms. Wynikiem tych operacji jest ciąg prostokątny o wypełnieniu 50% i podstawie czasu 2 ms, tj. 500 Hz. Taki generator nie wymaga stosowania zegara *TON* jak poprzednio, co daje krótszy kod programu. Ponadto, zwiększenie częstotliwości PWM częściowo redukuje wibracje wirnika, jednakże dalej powoduje efekty akustyczne. Częstotliwość PWM ogranicza częstotliwość wejściowa sygnału *CLKZ*, tzn. częstotliwość *CLKZ* powinna być, co najmniej 10 razy mniejsza od częstotliwości PWM. Wibracje te ograniczają zastosowanie tak sterowanego napędu w praktyce. Nie zawsze również jest sens stosowania mikrokroku, gdyż rozdzielczość typowych silników sterowanych pełno krokowo wynosi 0,9°/krok. W każdym razie, w ogólności mikrokrok 1/n wymaga:

- 4*n podkroków elementarnych, tj. tyle powinna wynosić liczba stanów silnika,
- n/2 osiągalnych różnych natężeń prądu w uzwojeniu.

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.