MechaNet (2)

Sterownik silnika krokowego kontrolowany przez Ethernet

AVT 5284

Każdy elektronik hobbysta poszukuje jakichś zastosowań dla swoich projektów. O ile radość z dzieła jest większa, gdy nowe elektrocudo nie jest tylko ozdobnym, świecącym bajerem. Jednym z takich bardziej praktycznych projektów może być układ pozwalający na wykonanie mechanicznej pracy. Ciekawiej może być, jeżeli ta praca będzie sterowana zdalnie z komputera PC poprzez sieć LAN. Proponuję więc ethernetowy sterownik bipolarnego silnika krokowego z protokołem TCP/IP – MechaNet.

Rekomendacje: *ciekawy projekt będący przykładem implementacji i użycia stosu TCP/IP w mikrokontrolerze PIC firmy Microchip; może przydać się w automatyce, robotyce, inteligentnym budynku itd.*

Rozkazy sterownika

Każdy rozkaz ma następującą budowę: *Rozkaz*(*Parametr_1*,*Parametr_2*,...,*Parametr_N*). Każda instrukcja zaczyna się od słowa kluczowego *Rozkaz* definiującego jego rodzaj. Następnie w nawiasach ujęte są parametry danej instrukcji. Liczba i format parametrów są indywidualne dla każdego rozkazu. Poszczególne parametry są rozdzielane znakiem przecinka. Niekiedy parametr można pominąć, zostawiając w jego miejscu puste pole, co powoduje użycie parametru domyślnego. Poszczególne rozkazy są rozdzielane dowolnymi znakami z wyłączeniem znaków alfanumerycznych, nawiasów okrągłych i przecinka.

1) *IVAL()*. Rozkaz powoduje przesłanie informacji o stanie pracy sterownika i stanie wejść cyfrowych (krańcówek). Funkcja jest wykonywana bezzwłocznie.

Odpowiedź: *REIVAL[xyz]*, gdzie:
x – wejście krańcowe_1 (1 – aktywne, 0 – nieaktywne)
y – wejście krańcowe_2 (1 – aktywne, 0 – nieaktywne)
z – stan sterownik silnika skokowego (1 – w ruchu, 0 – nieaktywny)

2) *MOV(Liczba_Kroków, Tryb_Prędkości)*. Rozkaz powoduje rozpoczęcie obracania silnika skokowego. Jest on wykonany w trybie sekwencyjnym.

Liczba_Kroków – parametr określa liczbę kroków (lub mikrokroków) do wykonania. Jest to liczba całkowita z zakresu $\pm(2^{23})$, przy czym znak + można pomijać. Ruch może być wykonany ze stałą prędkością obrotową lub z wykonaniem procedur miękkiego startu i stopu. Procedura może także zwiększać liczbę skoków do wykonania dla aktualnie wykonywanego zdania MOV.

Tryb_Prędkości – pojedynczy znak ASCII; „*C*” – tryb stałej prędkości; „F” – tryb miękkiego rozruchu i hamowania; „+” – dodanie wartości *Liczba_Kroków* do liczby kroków pozostałych do wykonania dla aktualnie wykonywanej instrukcji MOV. Pominięcie pola powoduje wykonanie rozkazu w sposób domyślny. Odpowiedź: *REMOV[R]* – potwierdzenie przyjęcia rozkazu.

Uwaga: W trybie aktualizacji prędkości, jeżeli suma aktualnej liczby kroków do wykonania i wartości aktualizującej będzie większa niż 2^{23}, to wartość zostanie zaktualizowana do 2^{23}, a w przypadku, gdy sterownik silnika krokowego jest nieaktywny lub przeprowadza hamowanie, rozkaz jest odrzucany. W trybie miękkiego rozruchu i hamowania minimalna częstotliwość skoków to 30 Hz (wartość stała).

Przykłady:

```
MOV(-200,F)     //wykonać 200 skoków silnika w kierunku „-” z miękkim rozruchem i hamowaniem
MOV(456789,)    //wykonanie 456789 skoków silnika w kierunku „+” z domyślnym trybem rozruchu
MOV(345,+)      //zwiększenie liczby skoków do wykonania dla aktualnie wykonywanej instrukcji MOV o 345
```

AVT-5284 w ofercie AVT:
AVT-5284A – płytka drukowana

Podstawowe informacje:
- Mikrokontroler PIC18F97J60 z wbudowanym Ethernetem PHY (10BaseT) i sprzętowym kontrolerem MAC
- Układ wykonawczy A3977 firmy Allegro MicroSystems
- Maksymalny prąd obciążenia 2,5 A na cewkę
- Napięcie zasilania 12...32 VDC
- Sterowanie układem odbywa się poprzez Ethernet za pomocą TCP/IP

Dodatkowe materiały na CD/FTP:
ftp://ep.com.pl, user: 12147, pass: 2e7u6a2a
- wzory płytek PCB
- karty katalogowe i noty aplikacyjne elementów oznaczonych w Wykazie elementów kolorem czerwonym

Projekty pokrewne na CD/FTP:
(wymienione artykuły są w całości dostępne na CD)

AVT-1585	Sterownik bipolarnego silnika krokowego (EP 8/2010)
AVT-2933	Sterownik silnika krokowego USB (EdW 2/2010)
AVT-1525	Sterownik unipolarnego silnika krokowego (EP 6/2009)
AVT-5137	Sterownik silnika krokowego z interfejsem MODBUS (EP 6–7/2008)
AVT-1314	Najprostszy sterownik silnika krokowego (EP 8/2001)

3) *REBOOT()*. Wykonanie rozkazu powoduje restart sterownika. Rozkaz jest wykonany w trybie sekwencyjnym. Odpowiedź: Brak.

Wykonanie rozkazu spowoduje zerwanie połączenie TCP i niewykonanie rozkazów pozostałych w kolejce.

4) *ROT(Kierunek,Częstotliwość_Skoków)*. Wykonanie rozkazu powoduje rozpoczęcie obracania silnikiem krokowym. Rozkaz jest wykonany w trybie sekwencyjnym. *Kierunek* – jest znakiem ASCI „+" lub „–" określającym kierunek obrotów silnika. *Częstotliwość_Skoków* – parametr zawierający maksymalną częstotliwość skoków silnika krokowego. Jest to liczba całkowita z zakresu 120...62500, z jednostką 0,1 Hz. Dodatkowo, parametr 0 wskazuje na prędkość domyślną. Należy zwrócić uwagę, że tryb ruchu z rozpędzeniem jest dostępny dla częstotliwości maksymalnej z zakresu 30 Hz...2083,3 Hz. Odpowiedź: *REROT[R]* – potwierdzenie przyjęcia rozkazu.

To, czy przy wykonywaniu funkcji zostanie zastosowany miękki rozruch, zależy od domyślnego ustawienia, które można zmieniać za pomocą *SETMOV*. W trybie miękkiego rozruchu i hamowania minimalna częstotliwość skoków to 30 Hz (wartość stała). Pracę sterownika można zatrzymać funkcją STOP.

Przykłady:

```
ROT(+,6000)     //rotacja w kierunku
„+" z częstotliwością skoków 600 Hz
ROT(-,125)     //rotacja w kierunku
„-" z częstotliwością skoków
12,5 Hz
```

5) *SETACC(Przyspieszenie)*. Polecenie ustawia parametr odpowiedzialny za szybkość zmiany częstotliwości podczas miękkiego rozruchu i hamowania silnika krokowego. Rozkaz jest wykonany w trybie sekwencyjnym.

Przyspieszenie – określa tempo zmiany częstotliwości skoków silnika krokowego podczas miękkiego rozruchu i hamowania. Jest to liczba z zakresu 1...65535 z jednostką 0,1 Hz/s. Odpowiedź: *RESETTACC[R]* – potwierdzenie przyjęcia rozkazu.

Przykład:

```
SETACC(25)     ustawienie
przyspieszenia na 2,5 Hz/s².
```

6) *SETMOV(Częstotliwość_Skoków, Tryb_Prędkości, Krańcówka_1, Krańcówka_2, Relacja_Krańcówek, Mikroskok*). Rozkaz pozwala na zmianę ustawień domyślnych sterownika silnika krokowego. Jest on wykonany w trybie sekwencyjnym. *Częstotliwość_Skoków* – parametr zgodnie ustala domyślną, maksymalną częstotliwość skoków silnika krokowego. Jest to liczba całkowita z zakresu 120...62500, z jednostką 0,1 Hz. Należy zwrócić uwagę, że tryb ruchu z rozpędzeniem jest dostępny dla częstotliwości maksymalnej z zakresu 30 Hz...2083,3 Hz. Pominięcie parametru skutkuje pozostawianiem tego ustawienia bez zmian. *Tryb_Prędkości* – pojedynczy znak alfanumeryczny: *C* – tryb stałej prędkości; F – tryb miękkiego rozruchu i hamowania; pominięcie parametru skutkuje pozostawianiem tego ustawienia bez zmian. *Krańcówka_1* i *Krańcówka_2*: – parametry decydują o tym, jaki poziom na danym dyskretnym wejściu krańcowym jest rozpoznawany jako aktywny. Pojedynczy znak alfanumeryczny, *L* – oznacza, że wejście jest aktywne w stanie niskim; *H* – oznacza, że wejście aktywne w stanie wysokim; *D* – oznacza, że wyłącznik jest nieaktywny (nieużywany). Pominięcie parametru skutkuje pozostawianiem tego ustawienia bez zmian. *Relacja_Krańcówek* – Określa sposób reakcji sterownika na aktywność krańcowych wejść dyskretnych. Jeżeli wynikiem operacji na krańcówkach będzie stan aktywny, to będzie to powodowało zatrzymanie pracy sterownika silnia skokowego. Pojedynczy znak ASCII, | – suma logiczna aktywności krańcówek, *&* – iloczyn logiczny aktywności krańcówek, ^ – suma logiczna modulo 2 (xor) aktywności krańcówek. Pominięcie parametru skutkuje pozostawianiem tego ustawienia bez zmian. *Mikroskok* – Określa tryb pracy mikroskokowej lub pełnoskokowej. Pojedynczy znak numeryczny, *1* – praca pełnoskokowa; 2 – praca półskokowa; 4 – praca ćwierć sokowa; 8 – praca 1/8 skoku. Pominięcie pola skutkuje pozostawianiem tego ustawienia bez zmian. Odpowiedź: *RESETMOV[R]* – komunikat o przyjęciu rozkazu

Przykłady:

```
SETMOV(20000,F,H,H,|,8)     //
ustawienie domyślnej częstotliwości
skoków na 2 kHz, domyślnie
wykonywany miękki rozruch
i hamowanie, obydwie krańcówki
aktywowane stanem wysokim, aktywność
którejkolwiek krańcówki spowoduje
zatrzymanie silnika. Sterownik
pracuje w trybie mikroskokowym
o rozdzielczości 1/8 skoku
SETMOV(2110,,,,,)     //tylko
ustawienie domyślnej częstotliwości
skoków na 211 Hz.
```

7) *SETNET(IP,Brama,Maska,Port*). Rozkaz zmienia ustawienia sieciowe sterownika. Rozkaz jest wykonany w trybie sekwencyjnym. *IP* – adres IP sterownika. *Brama* – brama sieciowa. *Maska* – maska podsieci. *Port* – port nasłuchu serwera kontrolnego sterownika.

Parametry *IP*, *Brama*, *Maska* należy podać w typowej konwencji zapisu adresów IP, czyli jako liczby rozdzielone kropkami (np.: 126.56.7.3). Parametr *Port* należy podać jako liczbę dziesiętną. Pominięcie parametru powoduje pozostawienie danego ustawienia bez zmian.

Odpowiedź: *RESETNET[R]* – potwierdzenie przyjęcia rozkazu.

Nowe ustawienia będą wprowadzone po ponownym uruchomieniu sterownika. Można to zrobić manualnie lub za pomocą rozkazu *REBOOT*. Przedtem jednak należy zapisać zmienione ustawienia w pamięci nieulotnej za pomocą rozkazu *SAVESET*.

Przykład:

```
SETNET (198.162.10.12,198.162.
10.1,255.255.255.0,9760)     //
ustawienie adresu IP sterownika
na 198.162.10.12, bramy sieciowej
na 198.162.10.1, maski podsieci
na 255.255.255.0 i portu nasłuch
serwera kontrolnego na  9760.
SETNET(,,,65535)     //Zmiana tylko
portu nasłuch serwera kontrolnego
na 65535.
```

8) *STOP()*. Rozkaz powoduje zatrzymanie pracy sterownika silnika krokowego. Jest on wykonywany bezzwłocznie. Odpowiedź: *RESTOP[1]* – komunikat o wykonaniu rozkazu ze skutkiem w postaci zatrzymaniu pracy sterownika silnika skokowego. *RESTOP[0]* – komunikat o wykonaniu rozkazu bez zatrzymywania pracy sterownika silnika skokowego (ponieważ nie pracował).

9) MOTPOW(*ON_OFF*). Instrukcja umożliwia włączenie lub wyłączenie zasilania cewek silnika krokowego. Rozkaz jest wykonywany bezzwłocznie. *ON_OFF* – Parametr określa czy instrukcja ma włączać czy wyłączać zasilanie dla cewek silnik krokowego. Pojedynczy znak numeryczny, *1* – włączyć; *0* – wyłączyć. Parametru nie można pominąć. Odpowiedz: *REMOTPOW[R]* – potwierdzenie przyjęcia rozkazu.

Rozkaz służy głównie do wyłączania zasilania silnika krokowego. W przypadku instrukcji *ROT* i *MOV* załączenie zasilania dla silnika skokowego jest samoczynne, ale po zakończeniu pracy sterownika zasilanie nie jest odłączane. Aby wyłączyć zasilanie, można użyć właśnie rozkazu *MOTPOW*.

10) *SAVESET*. Rozkaz powoduje zapisanie bieżących ustawień sterownika do pamięci nieulotnej. Odpowiedź: *RESAVESET[R]* – potwierdzenie przyjęcia rozkazu

Jako pamięć konfiguracyjna funkcjonuje część pamięci programu mikrokontrolera PIC18F97J60. Ma ona trwałość 100 tysięcy cykli zapisu i kasowania.

11) *SETIMAX(I_Max*). Rozkaz ustawia maksymalny prąd w cewkach silnika skokowego. Funkcja jest wykonywana sekwencyjnie. *I_Max* – parametr określa maksymalny prąd płynących przez cewki silnika skokowego, liczba całkowita z zakresu 1...255, z jednostką 1/255 [I_{max}], gdzie I_{max}=1,875 A. Odpowiedź: *RESETIMAX[R]* – potwierdzenie przyjęcia rozkazu.

Należy zwrócić uwagę na to, aby zadawany maksymalny prąd nie przekraczał wartości znamionowej dla danego silnika.

Część wykonawcza

Na podstawie otrzymanych rozkazów funkcja *IdRozkaz* generuje zadania dla części wykonawczej aplikacji. Te zadania są dostarczane w postaci kolejki zadań w przypadku rozkazów sekwencyjnych i jako flagi w przypadku rozkazów bezzwłocznych (struktura *ReciveOrder*). Do każdego zadania jest przyporządkowany wskaźnik do obszaru pamięci zawierającego strukturę danych z parametrami instrukcji (w przypadku

rozkazu bezparametrowego wskaźnik jest nieistotny).

Nadrzędnym procesem wykonawczym jest podprogram *ComandInit* (**listing 2**). Z wyorzystaniem licznika rozkazów *LiczRozk* i flag jest ustalana obecność nowych zadań. Zadania, o ile są, zostają przekazane do wykonania odpowiednim podprogramom. Wyjątkiem jest zadanie dla instrukcji *REBOOT*, które jest wykonywane bezpośrednio (za pomocą wstawki asemblerowej) i bezpośredni kod wykonujący instrukcję *MOTPOW*. Pierwszeństwo w realizacji mają zadania odpowiadające instrukcjom bezzwłocznym.

Za wykonywanie poszczególnych zadań lub inicjację procesu wykonawczego odpowiedzialne są podprogramy:

- *StopExe* – zadanie dla rozkazu STOP,
- *IvalExe* – zadanie dla rozkazu IVAL,
- *MovExe* – zadanie dla rozkazu MOV,
- *RotExe* – zadanie dla rozkazu ROT,
- *SetMovExe* – zadanie dla rozkazu SETMOW,
- *SetNetExe* – zadanie dla rozkazu SETNET,
- *SetAccExe* – zadanie dla rozkazu SETACC,
- *SaveSetExe* – zadanie dla rozkazu SAVESET,
- *SetCurentExe* – zadanie dla rozkazu SETIMAX.

Rzecz jasna każdy podprogram wykonuje przewidziane dla danego rozkazu zadanie.

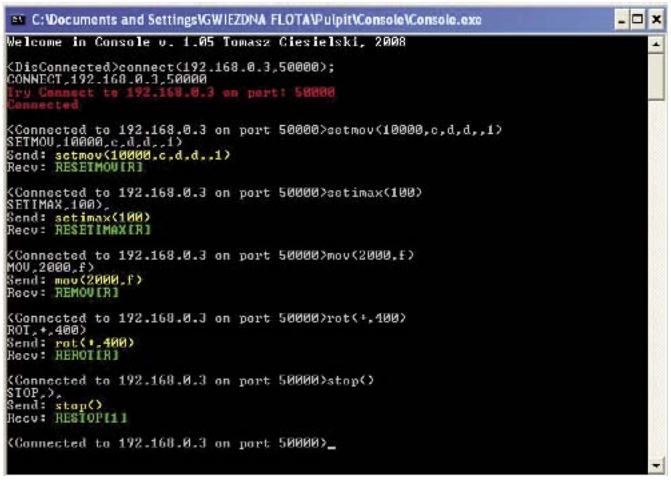
MovExe. W pierwszej kolejności procedura ustala, czy aktualnie odbywa się wykonywanie zadania *MOV* lub *ROT*. Jeżeli jest wykonywane zadanie *MOV*, to aktualnie można co najwyżej zwiększać liczbę zadanych skoków (parametr *Tryb_Prędkości*=+) i to pod warunkiem, że sterownik aktualnie nie realizuje hamowania. Jeżeli sterownik hamuje lub jest wykonywany rozkaz *ROT*, to rozkaz zwiększenia liczby kroków zostaje odrzucony. Instrukcje z pozostałymi trybami muszą czekać na zakończenie obracania silnikiem. Kiedy nastąpi dopuszczenie do wykonania zadania zapoczątkowania ruchu, w zależności od wybranego trybu prędkości okres skoku zostaje ustalony jako wartość stała lub obliczony za pomocą podprogramu *NextSteppCalc*. Wraz z okresem obliczane są też zakresy: rozpędzania, prędkości stałej, hamowania, wszystkie w skali licznika skoków. Program w końcu załącza stopień mocy, moduł Timer0 i kolejno w przerwaniu od modułu (funkcja *MyHighInterupt*, **listing 3**), co okres skoku są generowane impulsy zegarowe dla sterownika A3779 i (jeżeli to konieczne) jest generowane osobne zadanie wyznaczania kolejnych opóźnień. Co ważne, do celu obliczenia opóźnień wykorzystany jest mechanizm odpowiedzialny za jedno z nieużywanych inaczej przerwań. Zapewnia to realizację procesu kalkulacji w priorytecie niższym niż mechanizm odpowiedzialny za zachowanie zależność czasowych (tj. *MyHighInterupt*), ale przed programem głównym. Taki sposób działania jest konieczny z uwagi na to, że czas wykonania procedury *NextSteppCalc* może być potencjalnie dłuższy od połowy okresu generowanych skoków sterownika. Kiedy licznik kroków zostanie wyzerowany, wy-

Listing 2. Podprogram CommandInit

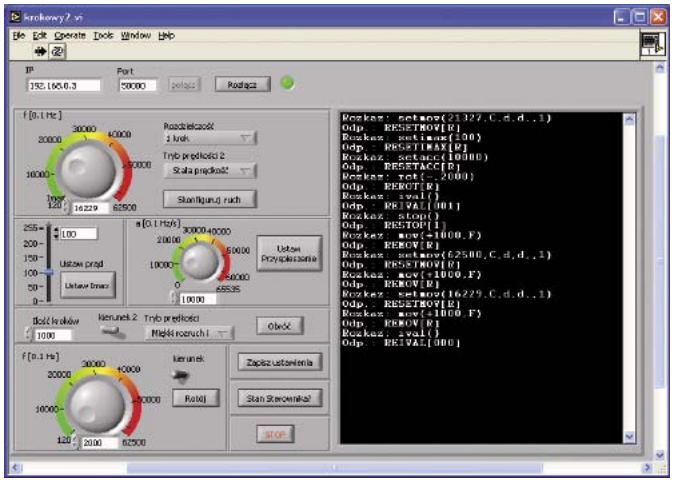
```
extern REC_ORDER ReciveOrder;  //struktura przechowująca rozkazy
extern unsigned char ActualOrder,LiczRozk;  //Numer aktualnego rozkazu i
licznik rozkazów
void ComandInit (void)
{
  unsigned char Znacznik_wykonania,i,j;
  //obsługa rozkazów bezzwłocznych

  if(ReciveOrder.Stop == 1)//STOP
  {
    StopExe();
    ReciveOrder.Stop = 0;
  }
  if(ReciveOrder.Ival == 1)//IVAL
  {
    if (IvalExe() == 0x00) ReciveOrder.Ival = 0;
  }
  if(ReciveOrder.Motpow != 0)//MOTPOW
  {
    if(ReciveOrder.Motpow == 2){Enable3977Output();}
    else if(ReciveOrder.Motpow == 1){Disable3977Output();}
    ReciveOrder.Motpow = 0;
  }
  if(LiczRozk>0)//obsługa rozkazów sekwencyjnych
  {
    switch(ReciveOrder.ORDER[ActualOrder])
    {
      case 0x01:  //MOV
        Znacznik_wykonania = MovExe();
        break;
      case 0x02:  //ROT
        Znacznik_wykonania = RotExe();
        break;
      case 0x04:  //SETMOV
        Znacznik_wykonania = SetMovExe();
        break;
      case 0x07:    //SETNET
        Znacznik_wykonania = SetNetExe();
        break;
      case 0x08: //SETACC
        Znacznik_wykonania = SetAccExe();
        break;
      case 0x09: //SAVESET
        Znacznik_wykonania = SaveSetExe();
        break;
      case 0x0B: //REBOT
        Reset3977();      //sygnał resetu dla A3977
      Nop(); Nop();
        _asm RESET _endasm        //reset programowy
        break;
      case 0x0D: //SETIMAX
        Znacznik_wykonania = SetCurentExe();
      default:
        Znacznik_wykonania=1;
        break;
      }//end_switch(ReciveOrder.ORDER[ActualOrder])
      if (Znacznik_wykonania < 2)//Rozkaz został przetworzony
      {
        if(++ActualOrder>=MAX_WAITING_ORDERS) ActualOrder = 0;
        LiczRozk--;
      }
    }
  }
}
```

Listing 3. Wysokie przerwanie – generator sygnału steep

```
near void MyHighInterupt(void)
{
  if(PIR1bits.TMR1IF&&PIE1bits.TMR1IE)
  {
    if(Allegro3977.HiToLoInt)
    {
      STEP_3977 = 0;  //0 na wyprowadzenie STEEP A3977
      Allegro3977.HiToLoInt=0;  //następne zbocze STEEP narastające
      TMR1H=nTui2.bt[1];  //Aktualizuj opóźnienie
      TMR1L=nTui2.bt[0];
      if((SteepCounter == 0)||( krancowki()!=0))  // Silnik stop?
      {
        PIE1bits.TMR1IE=0;  //wyłącz przerwanie
        T1CONbits.TMR1ON=0;  // wyłącz Timer1
        Flagi_Kroki.wszystkie=0;  //reset flag generatora kroków
        STEP_3977=0;  //0 na wyprowadzenie STEEP A3977
      }
    }
    else
    {
      STEP_3977 = 1;  //1 na wyprowadzenie STEEP A3977
      Allegro3977.HiToLoInt=1;  //następne zbocze STEEP opadające
      TMR1H=nTui.bt[1];  //Aktualizuj opóźnienie
      TMR1L=nTui.bt[0];
      if(Flagi_Kroki.Rotacja == 0x00)  //skończona liczba kroków?
        SteepCounter--;  //aktualizacja liczby kroków do wykonania
      t=t+Tui;  //aktualizacja znacznika czasu
      if(Flagi_Kroki.TrybPrzyspieszania)  //zmienna prędkość?
      {
        nTui2=nTui3;  //Aktualizuj opóźnienie dla następnego przerwania
        PIR1bits.TMR2IF=1;//włączenie NextSteppCalc w LowInterupt
      }
    }
    PIR1bits.TMR1IF = 0;//kasuj flagę przerwania
  }
}
```



Rysunek 4. Widok ekranu klienta TCP „Console"



Rysunek 5. Widok ekranu programu „krokowy_0_2"

konanie instrukcji dobiega końca. Wykonanie instrukcji *MOV* może dobiec końca także z powodu aktywności wejść krańcowych. Ostatnią możliwością zakończenia wykonywania *MOV* jest zatrzymanie przez rozkaz *STOP*. Należy zwrócić uwagę, że w tle egzekucji rozkazu odbywa się cały czas obsługa pozostałych procesów sterownika.

RotExe. Algorytm obsługi tego zadania jest niemal identyczny ze schematem działania dla instrukcji MOV. Różnicą jest brak samoistnego zakończenia wykonania instrukcji i użycie zawsze domyślnego trybu prędkości. Stąd, oprócz części inicjalizującej zadanie, kod odpowiedzialny za jego obsługę jest wspólny z algorytmem wykonywania zadania *MovExe*.

StopExe. Funkcja po prostu wyłącza przerwanie, w którym przetwarzana jest procedura *MyHighInterupt*, co skutkuje wymuszonym zakończeniem zadań MOV i ROT. Ponadto przygotowuje komunikat zwrotny o wykonaniu.

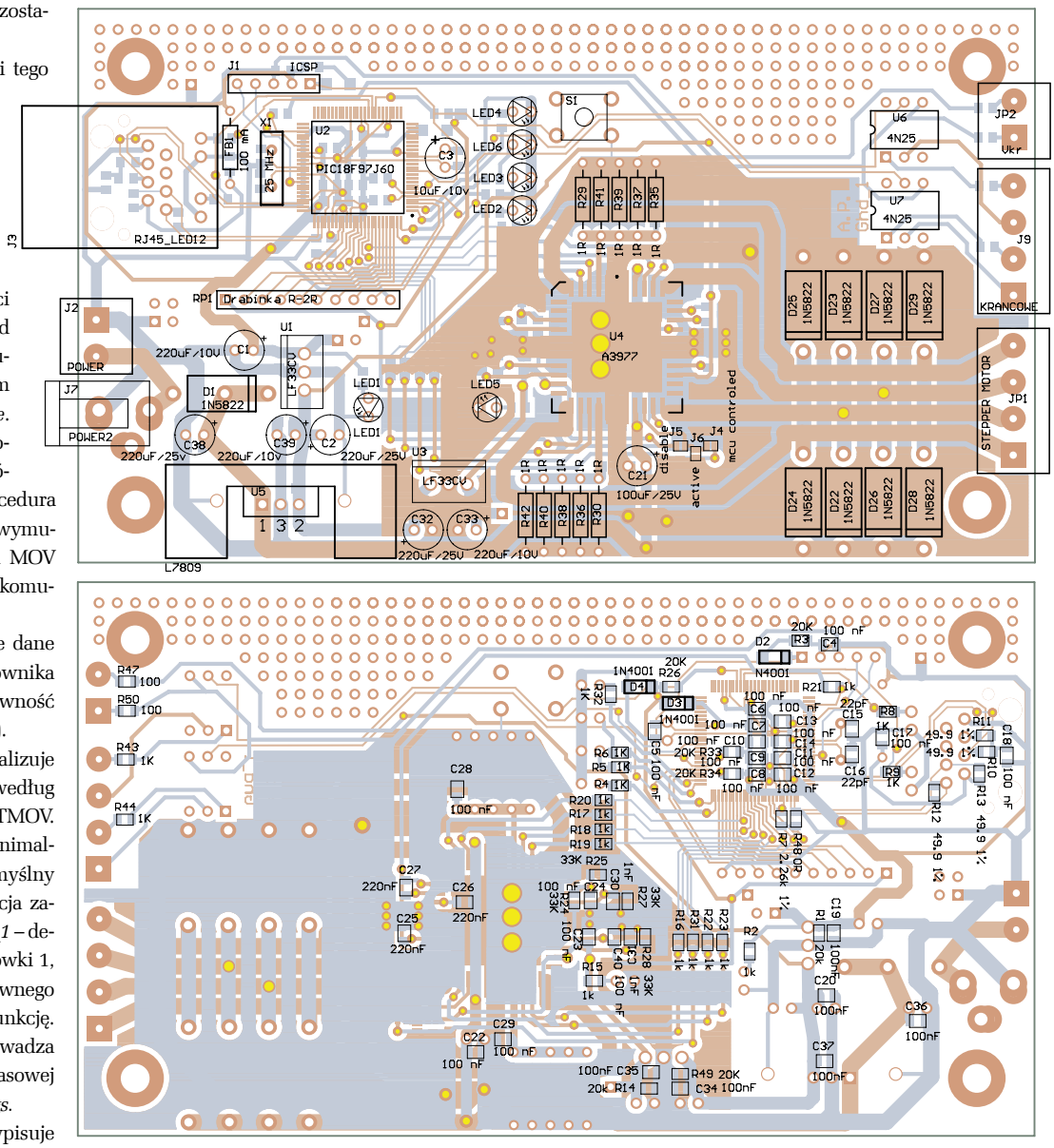
IvalExe. Funkcja generuje dane do komunikatu o stanie sterownika (stan wejść krańcowych i aktywność sterownika silnika skokowego).

SetMovExe. Funkcja aktualizuje ustawienia sterownika ruchu według parametrów instrukcji SETMOV. Zmienia parametry: *minT* – minimalny okres skoku, *DefTryb* – domyślny tryb prędkości, *relacja* – funkcja zatrzymująca krańcówek, *active_1* – defincja stanu aktywnego krańcówki 1, *active_2* – definicja stanu aktywnego krańcówki 2 poprzez podfunkcję. Ponadto, odpowiednio wprowadza kopie zmiennych do tymczasowej struktury ustawień *tempSetings*.

SetNetExe. Funkcja wpisuje do tymczasowej struktury ustawień *tempSetings* nowe ustawienia sieciowe sterownika tj.: *Ip* – adres IP sterownia, *Mask* – maska podsieci, *Geatway* – brama domyslna, *GCP* – port nasłuchu serwer. Wprowadzane ustawienia będą dostępne po zapisaniu ustawień do pamięci nieulotnej i restarcie.

SetAccExe. Funkcja aktualizuje ustawienie szybkość zmiany częstotliwości (parametr ***a*** i kopia *tempSetings*).

SaveSetExe. Funkcja odpowiedzialna jest za zapis ustawień tymczasowych sterownika (struktura *tempSetings*) do pamięci nieulotnej.



Rysunek 6. Schemat montażowy sterownika

Korzysta z zarezerwowanej do tego celu części pamięci FLASH programu. Obszar zarezerwowany najpierw jest kasowany (funkcja *RawEraseFlashBlock*), a następnie kopiuje się do niego strukturę z ustawieniami.

SetCurentExe. Funkcja bezpośrednio ustawia zadaną wartość na port z drabinką R-2R i kopiuje ustawienie prądu do pola *tempSetings.imax*.

Wejścia dyskretne. Za odszumianie wejść dyskretnych odpowiedzialna jest funkcja *KrancowkiDebonce* wykonywana w niskim przerwaniu. Udostępnia ona dane o stanie tych wejść w postaci zmiennej *ukrancowki* innym częściom aplikacji.

Zerowanie nastaw. Za zerowanie ustawień sieciowych jest odpowiedzialna funkcja *ServRebotButton*. Jeżeli wykryje ona naciśnięcie przycisku S2 IP_RESET przez co najmniej 3 s, to wprowadza domyślne ustawianie sieciowe (IP:192.168.0.3, Maska:255.255.255.0, Brama:192.168.0.1, Port:50000) i funkcjonalne (przyspieszenie, prąd etc.) i programowo resetuje sterownik.

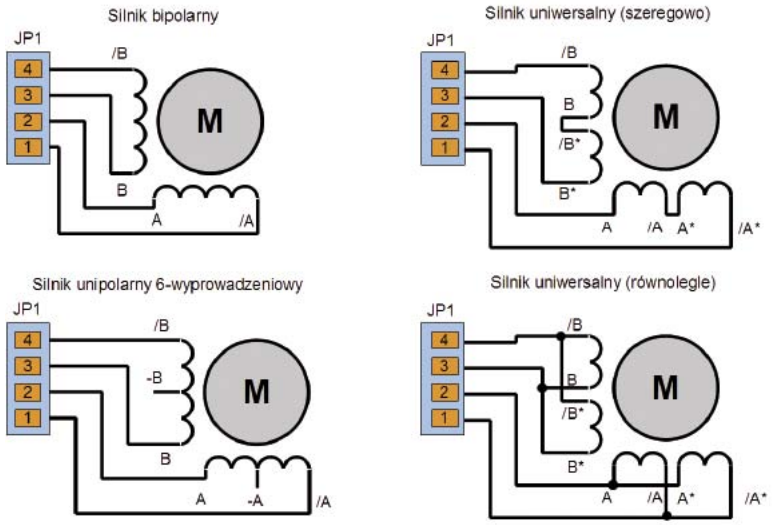
Zdalne oprogramowanie

Do celu uruchamiania sterownika służy zdalne oprogramowanie klienta kontrolującego sterownik „krokowy_0_2" – umożliwia ono wydawanie urządzeniu komend i odbiór odpowiedzi (**rysunek 4**). Ponieważ sterownik przyjmuje rozkazy w postaci ciągów znaków ASCII, do jego kontrolowania może też służyć kliencka tekstowa konsola TCP np. program Console autorstwa Tomasza Ciesielskiego (**rysunek 5**). Do integracji sterownika z indywidualną aplikacją jest wymagane utworzenia adekwatnego oprogramowania po stronie hosta kontrolującego. Zadanie to jest dosyć łatwe z uwagi na obecność standardowych komponentów wspierających komunikację TCP/IP w większości typowo programistycznych narzędzi (np. VisualStudio) oraz dedykowanych do integracji aplikacji kontrolno-pomiarowych (np. LabView).

Na **rysunku 6** pokazano schemat montażowy płytki sterownika. Została ona zaprojektowana raczej jako płytka dydaktyczna. Należy zauważyć, że obwód z przyciskiem S2 został dodany na powierzchni prototypowej, oprócz tego pin MCLR PIC'a został zmostkowany do pinu 12. aby naprawić błąd przypisania pinów do Footprintu układu.

Uruchomienie i testowanie

Poprawność działania mikrokontrolera sprawdzamy, wykrywając go za pomocą programatora lub debuggera (np. ICD2). Mikrokontroler należy zaprogramować. Po włączeniu zaprogramowanego układu działanie sterownika jest sygnalizowane miganiem LED6. Teraz możemy przystąpić do sprawdzenia poprawności działania komunikacji sieciowej. Najpierw jednak musimy odpowiednio skonfigurować ustawienia sieciowe naszego komputera PC. Sterownik ma domyślny adres IP 192.168.0.3 i maskę podsieci 255.255.255.0, więc ustawienia sieciowe komputera PC należy wybrać inne, ale z tej samej podsieci (np.: 192.168.0.1). Kolejnym krokiem jest podłączenie sterownika za pomocą odpowiedniego przewodu. Jeśli sterownik jest podłączany bezpośrednio do karty sieciowej komputera PC, to należy użyć przewodu sieciowego UTP z przeplotem. W przypadku włączenie do switcha należy użyć zwykłego przewodu sieciowego UTP.



Rysunek 7. Podłączenie silnika do sterownika

Poprawność fizycznego dołączenia do sieci LAN lub karty sieciowej urządzenia jest sygnalizowana poprzez świecenie się diod LED w gniazdku ethernetowym sterownika. Ponadto, w przypadku gdy podłączyliśmy urządzenie bezpośrednio do karty sieciowej komputera, w *Panelu Sterowania –> Połączenia sieciowe* przy adekwatnym *Połączeniu Lokalnym* powinna się wyświetlić informacja *Podłączono.*

Prawidłowe działanie interfejsu sieciowego sterownika można sprawdzić, wykorzystując funkcję *ping*. Dla adresu domyślnego będze to *„ping 192.168.0.3"*. O ile jest komunikacja ze sterownikiem, to powinniśmy otrzymać szereg komunikatów *„Odpowiedź z 192.168.0.3: bajtów=32 czas 2 ms TTL..."*.

Kolejnym krokiem jest dołączenie silnika krokowego. Musi być to silnik 2-fazowy z uzwojeniem bipolarnym (4 – wyprowadzenia), silnik uniwersalny (8- wyprowadzeń) lub ewentualnie 6-wyprowadzeniowy silnik unipolarny. Sterownik oryginalnie był testowany z silnikiem 57BYG801 kupionym w firmie Wobit. Sposób podłączenia silnika do sterownika pokazano na **rysunku 7**.

Jeżeli silnik, zamiast kręcić się, „piszczy" lub „buczy", to należy zamienić miejscami końce jednej z par uzwojeń wchodzących do sterownika (w przypadku kiedy nie znamy orientacji końców cewek silnika). Z podłączonym silnikiem przystępujemy do testu układu sterowania silnika krokowego. Mamy do wyboru dwa programy: *krokowy_0_2* lub *Console*. Dla wygody korzystam z tego drugiego. Test wykonuje się następująco:

– Sprawdzić, czy adres IP i Port są wpisane prawidłowo, następnie kliknąć przycisk *Połącz*. Nawiązanie komunikacji jest widoczne poprzez zmianę koloru wirtualnej diody na panelu programu i aktywację pozostałych kontrolek.
– Aby móc wysterować silnik, musimy najpierw skonfigurować parametry ruchu. Dla celu testów dobrze jest ustawić typowe parametry, niezbyt „wyżyłowane". Ustawiamy więc częstotliwość kroków na np. *500 Hz*, wybieramy tryb prędkości *Stała prędkość* i rozdzielczość *1 krok*. Następnie klikamy przycisk *Skonfiguruj ruch*. W wyniku tego w polu testowym obok powinna pojawić się wygenerowana treść rozkazu: *setnow(5000,C,d,d,,1)* i odpowiedź na rozkaz *RESETMOV[R]*.
– Ustawić prąd sterownika na odpowiedni dla danego sinika, np. dla uzyskania natężenia 1 A zadajemy na suwaku 136 i klikamy przycisk *Ustaw Imax*. W polu tekstowym powinna się pojawić treść rozkazu *setimax(136)* i odpowiedz *RESETIMAX[R]*.
– Obok przycisku *Rotuj* ustawiamy częstotliwość kroków (dla rotacji) na *500 Hz* i klikamy przycisk *Rotuj*. W wyniku tego nasz silnik powinien zacząć się równomiernie kręcić, a w konsoli obok pojawić się rozkaz *rot(-,5000)* i odpowiedź *REROT[R]*.

Ostatnim elementem do przetestowania jest para krańcówek. Najprościej je przetestować, sprawdzając poziom wejść niepodłączonych i wejść spolaryzowanych poprzez odpowiednie podłączenie do wyprowadzenia zasilania JP9. Odczytu stanu krańcówek dokonujemy, klikając przycisk *Stan Sterownika?* W wyniku operacji w konsoli powinna pojawić się treść rozkazu *ival()* i odpowiedź np.: *REIVAL[100]*.

Podsumowanie

Sądzę, że elementy oprogramowania tego projektu świetnie nadają się do realizacji podobnych amatorskich sieciowych urządzeń pomiarowych i sterujących. Myślę, że wiele rzeczy dałoby się w nim zrobić lepiej lub dodać jeszcze kolejne możliwości, dlatego zachęcam też innych do wykonywania podobnych projektów.

Andrzej Puzdrowski
a.m.puzdrowski@interia.pl