Sterowanie silnikiem

przy pomocy modułu ARD MOTOR SHIELD

Wstęp

Zapewne niejednokrotnie w przyszłości będziecie potrzebować możliwości sterowania różnymi silnikami jako urządzeniami wykonawczymi. Z sytuacją taką na pewno spotkacie się, jeśli będziecie chcieli wykonać np. prostego robota. Zatem czas wziąć się do pracy i poznać metody sterowania różnych silników – postanowiłem omówić tutaj kwestie dotyczące sterowania:

a) silników prądu stałego
b) serwomechanizmów modelarskich
c) silników krokowych unipolarnych
d) silników krokowych 4-fazowych

Sprzęt

Do sterowania silnikami polecam wykorzystać gotowy moduł Motor Shield zawierający układ L293D mogący sterować silnikami o poborze prądu do 0,6A (w rozruchu impulsowo do 1,2A), będący podwójnym mostkiem H, czyli w wielkim uproszczeniu układem pozwalającym m. in. na kontrolę kierunku oraz prędkości obrotów silników DC (prądu stałego). Dodatkowo do niektórych eksperymentów przyda się nam układ ULN2003/2803, który wykorzystywaliśmy już podczas poprzednich lekcji.

Moduł Motor Shield

Moduł ten, jak już wspomniałem, oparty jest o układ L293D. Ponadto posiada możliwość wyboru do jakich pinów ma zostać podłączony ten układ. Do wyboru mamy 2 opcje:

1. Piny 10, 11, 12, 13:
    a. Pin 10 – kontrola włączenia/wyłączenia (prędkości) silnika 1
    b. Pin 11 – kontrola włączenia/wyłączenia (prędkości) silnika 2
    c. Pin 12 – kontrola kierunku obrotów silnika 1
    d. Pin 13 – kontrola kierunku obrotów silnika 2
2. Piny 4, 5, 6, 7:
    a. Pin 4 – kontrola kierunku obrotów silnika 1
    b. Pin 5 – kontrola włączenia/wyłączenia (prędkości) silnika 1
    c. Pin 6 – kontrola włączenia/wyłączenia (prędkości) silnika 2
    d. Pin 7 – kontrola kierunku obrotów silnika 2

Dodatkowo na płytce zostały umieszczone 2 przyciski zwierające do masy, podłączone do wyprowadzeń A0 i A1, 2 złącza do szybkiego podłączenia serwomechanizmów[1] oraz dodatkowe złącze zasilania[2].

[1] serwomechanizmy możemy podłączyć jednak także bezpośrednio do płytki Arduino, bez konieczności wykorzystywania modułu
[2] złącze (po założeniu odpowiedniej zworki) zasila nie tylko moduł, ale cały zestaw, dlatego należy podać odpowiednie napięcie zasilania ALBO na złącze na module ALBO na złącze na płytce bazowej Arduino. NIGDY NIE NALEŻY PODŁĄCZAĆ RÓWNOCZEŚNIE ZASILANIA DO OBU ZŁĄCZ – STWARZA TO REALNE ZAGROŻENIE USZKODZENIEM ZASILACZY I/LUB INNEGO SPRZĘTU.

Sterowanie silnikami DC

Na początek zajmijmy się bardzo prostym sterowaniem silnikami DC, które podpinamy pod złącza M1 i M2 (2 silniki). Zacznijmy od kontroli 1 silnika[3] na zasadzie włącz/wyłącz/zmień kierunek obrotów. Wykorzystamy od razu do tego 2 przyciski znajdujące się na płytce.

UWAGA! Podczas tego eksperymentu na jedno ze złącz zasilania zewnętrznego podajemy napięcie, jakim mają być zasilone silniki, nie należy przekraczać jednak wartości maksymalnej dopuszczalnej dla samego zestawu Arduino, tj. maksymalnie 35V (przy takim napięciu stabilizator na płytce będzie mocno się nagrzewać, dlatego nie zalecam używania silników zasilanych napięciem wyższym niż 12V).

```
void setup(){
 pinMode(10, OUTPUT);
 pinMode(12, OUTPUT);
 pinMode(A1, INPUT);
 digitalWrite(A1, HIGH);
 pinMode(A2, INPUT);
 digitalWrite(A2, HIGH);
}

byte on_of=0,dir=0;

void loop(){
  if(digitalRead(A1)==0){
    delay(50);
    if(digitalRead(A1)==0){
      on_of=!on_of;
      digitalWrite(10,on_of);
      while(digitalRead(A1)==0);
    }
  }
  if(digitalRead(A2)==0){
    delay(50);
    if(digitalRead(A2)==0){
      dir=!dir;
      digitalWrite(12,dir);
      while(digitalRead(A2)==0);
    }
  }
}
```

Cóż, szczerze mówiąc ja na razie nie wykroczyliśmy poza podstawowe informacje z pierwszych lekcji. Teraz jednak zajmujmy się sterowaniem prędkością obrotów silnika, jak się zapewne domyślacie z połączeń użyjemy do tego celu sygnału PWM. A więc napiszmy prosty program testowy:

```
void setup(){
 Serial.begin(9600);
 pinMode(10, OUTPUT);
 pinMode(12, OUTPUT);
}
```

[3] kontrolę drugiego silnika realizujemy w analogiczny sposób na odpowiednich pinach.

```
int odbierz(){//funkcja odbioru liczb z części 6 przystosowana do odbioru liczb typu int (bez znaku)
  byte odebrany=0;
  int liczba=0;
  do{
    if(Serial.available()>0){
      odebrany=Serial.read();
      if(odebrany!=10&&odebrany!=13){//filtrujemy znaki CR i LF (tak też ustawiamy terminal RS232)
        liczba*=10;
        liczba+=odebrany-'0';
      }
    }
  }while(odebrany!=10);//jeśli odebraliśmy znak LF to wychodzimy z pętli
  return liczba;//zwracamy odebraną liczbę
}

byte dir=0,pwm=0;

void loop(){
  Serial.print("Podaj kierunek: ");
  dir=(odbierz()!=0);
  Serial.println(dir, DEC);
  Serial.print("Podaj predkosc: ");
  pwm=odbierz();
  Serial.println(pwm, DEC);
  digitalWrite(12, dir);
  analogWrite(10, pwm);
  Serial.println("Ustawiono parametry:");
  Serial.print("Kierunek: ");
  Serial.println(dir, DEC);
  Serial.print("Predkosc: ");
  Serial.println(pwm, DEC);
}
```

I możemy kontrolować prędkość obrotów silnika, jednak… Nie w pełnym zakresie. Otóż z powodu zasady działania wszystkich silników DC przy niskich wartościach PWM (u mnie poniżej 70-80) nie będą one w ogóle pracować, a jedynie będą wydawały z siebie piski – jest to związane właśnie z tym iż sygnał PWM nie jest stałą wartością napięcia, a jedynie przebiegiem prostokątnym o określonej częstotliwości. Dlatego też przed wykorzystaniem wszystkich tych informacji w praktyce warto, abyście sprawdzili jakie wartości PWM „akceptuje" wasz silnik.

Serwomechanizmy modelarskie

Serwomechanizmy modelarskie to specjalne układy składające się w części mechanicznej z silnika i zestawu przekładni, natomiast sterowane są bezpośrednio z układu cyfrowego (np. naszej płytki Arduino), gdyż zawierają już w sobie odpowiedni sterownik. Mogą one poruszać się zwykle w zakresie kątów do 180° lub nieco większym i przeważnie są zasilane napięciem 5V. Są (jak sama nazwa wskazuje) wykorzystywane przede wszystkim w modelarstwie do sterowania np. skrętem kół samochodu, pozycją gaźnika lub lotkami w samolotach. Warto abyśmy poznali możliwość sterowania serwami, abyśmy mieli niezbędne podstawy, jeśli zamarzy nam się kiedyś wykonanie jakiegoś modelu w oparciu o platformę Arduino.

Serwomechanizmy podłączamy za pomocą 3 przewodów:

a) czarnego/niebieskiego – masa zasilania (GND)
b) czerwonego (środkowy) – zasilanie +5V
c) białego/żółtego – sygnał z mikrokontrolera

Możemy podpiąć nasze serwa bezpośrednio do zasilania naszej płytki (nie jest konieczne korzystanie z zewnętrznego zasilania), a przewód sygnałowy do dowolnego pinu naszej płytki, możemy także skorzystać ze specjalnych złącz na module sterownika silników, które są podłączone do wybranych zworką pinów odpowiedzialnych za kontrolę prędkości normalnych sil ników.

Sterowanie serwomechanizmami odbywa się za pomocą podawania na odpowiednie złącze sygnału PWM o określonej częstotliwości i wypełnieniu odpowiadającemu żądanemu kątowi wychylenia serwa. Na szczęście nie będziemy musieli się wgłębiać w ten temat, gdyż i tu z pomocą przyszli nam Twórcy Arduino:

```
#include <Servo.h>

Servo servo1;//tworzymy 2 obiekty do sterowania serwomechanizmami
Servo servo2;

int odbierz(){//funkcja odbioru liczb z części 6 przystosowana do odbioru
liczb typu int (bez znaku)
  byte odebrany=0;
  int liczba=0;
  do{
    if(Serial.available()>0){
      odebrany=Serial.read();
      if(odebrany!=10&&odebrany!=13){//filtrujemy znaki CR i LF (tak też
ustawiamy terminal RS232)
        liczba*=10;
        liczba+=odebrany-'0';
      }
    }
  }while(odebrany!=10);//jeśli odebraliśmy znak LF to wychodzimy z pętli
  return liczba;//zwracamy odebraną liczbę
}


void setup()
{
  Serial.begin(9600);
  servo1.attach(10);//przypisujemy określone obiekty do dowolnych
wyprowadzeń procesora
  servo2.attach(11);
}

byte kat;

void loop()
{
  Serial.print("Servo 1 (0-180): ");
  kat=odbierz();
  Serial.println(kat, DEC);
  servo1.write(kat);//ustawiamy podłączone serwo w zadanej pozycji
```

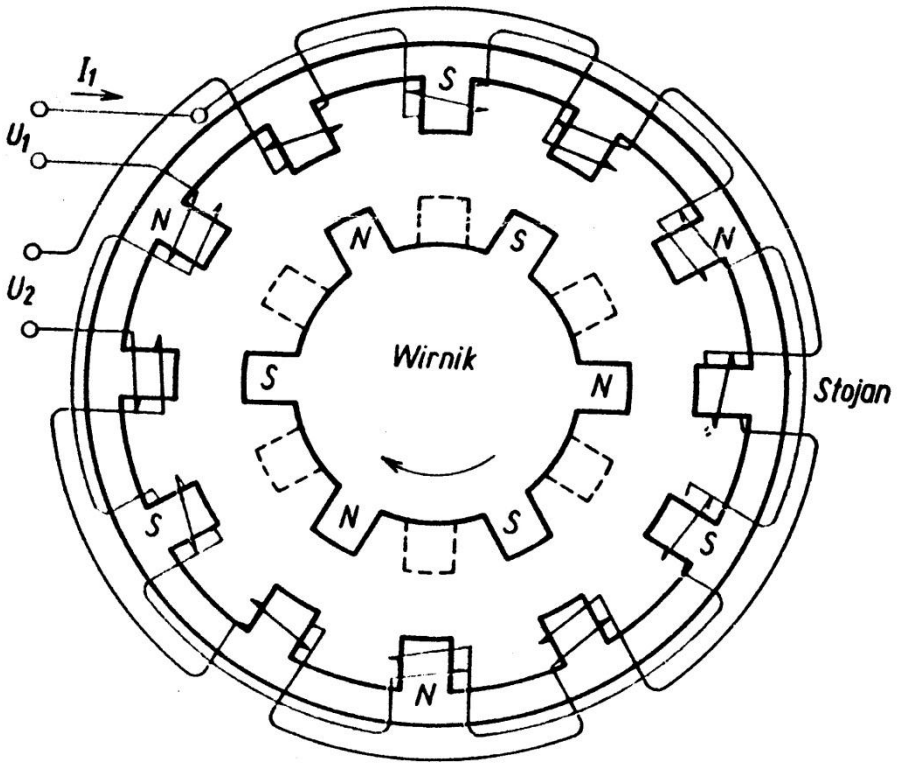
```
  Serial.print("Servo 2 (0-180): ");
  kat=odbierz();
  Serial.println(kat, DEC);
  servo2.write(kat);
}
```

Nic prostszego, po podaniu odpowiedniego kąta serwa powinny się odpowiednio wychylać. Jeśli w skrajnych położeniach (0 i 180) nie osiągają one pożądanych wychyleń, lub chcemy aby wychylenia były większe (gdyż pozwala na to ograniczenie mechaniczne w naszym modelu serwa), możemy przy inicjalizacji ręcznie skalibrować minimalny i maksymalny czas trwania impulsu sterującego (domyślne wartości to 544 i 2400). Jeśli chcemy zwiększyć wychylenie w pozycji 0 musimy zmniejszyć dolną granicę, a jeśli chcemy to wychylenie zmniejszyć granicę tę zwiększamy. Odwrotnie dla maksymalnego wychylenia w pozycji 180 – w celu zwiększenia wychylenia zwiększamy wartość, a w celu zmniejszenia maksymalnego wychylenia w pozycji 180 wartość górną zmniejszamy. Przykład kalibracji przeprowadzonej doświadczalnie na moich 2 serwach, zmiany obejmują jedynie 2 linijki programu w funkcji ustawień:

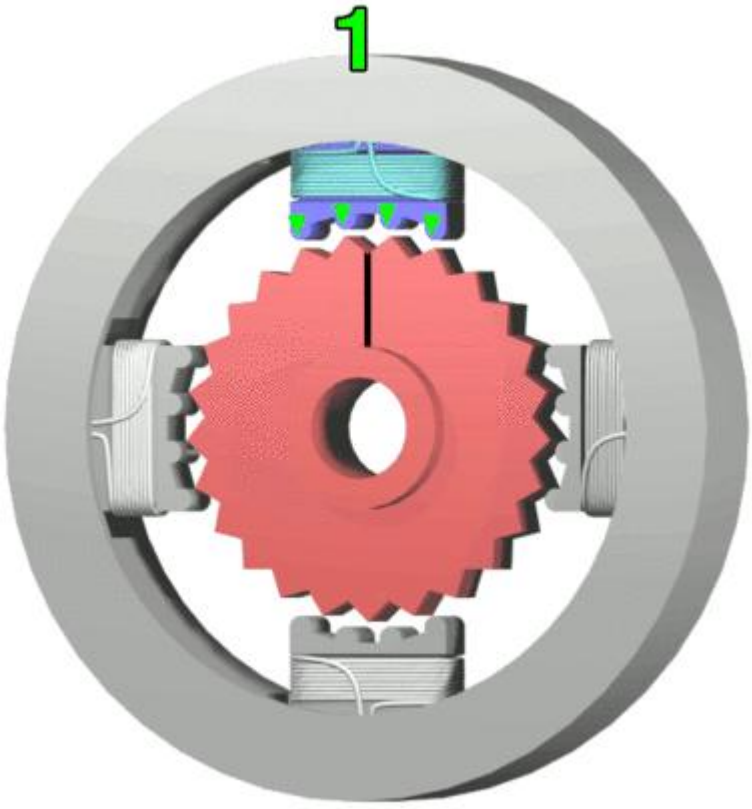
```
void setup()
{
  Serial.begin(9600);
  servo1.attach(10, 544, 2400);//przypisujemy określone obiekty do
dowolnych wyprowadzeń procesora, oraz ustawiamy czasy maksymalnego i
minimalnego impulsu
  servo2.attach(11, 544, 2600);//w moim przypadku zwiększyłem nieco
wychylenie w pozycji 180 dla serwa 2
}
```

Silniki krokowe

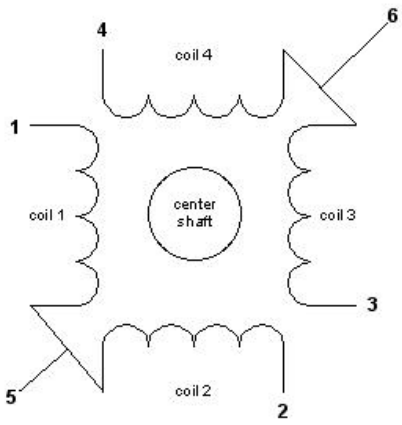
Teraz kolej na nieco bardziej skomplikowane zastosowanie naszego sterownika silników, a mianowicie sterowanie silnikami krokowymi. Ogólnie zasada działania silnika krokowego polega (w uproszczeniu) na obecności wielobiegunowego magnesu stałego oraz specjalnie nawiniętych cewek. Te umożliwiają przyciąganie biegunów (których jest zwykle 200 w silnikach 200-krokowych o kroku 1,8°) na przemian, tak, że jest możliwy ruch silnika o bardzo mały kąt. Ogólnie wyróżniamy (także w dużym uogólnieniu) silniki 2- i 4- fazowe. Silniki 2 fazowe posiadają 2 uzwojenia, w których musimy na przemian zmieniać kierunek przepływu prądu w celu naprzemiennego odpychania i przyciągania biegunów magnesu stałego połączonego z osią. Silniki 4-fazowe posiadają 4 uzwojenia w których musimy powodować kolejno przepływ prądu, tak aby bieguny były przyciągane do kolejnych uzwojeń. Uproszczony schemat silnika 2-fazowego przedstawiono poniżej:



Tak zaś wygląda uproszczony schemat silnika 4-fazowego:

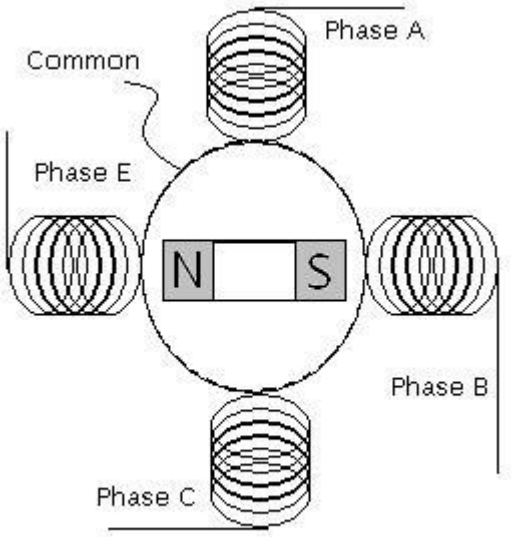


Silniki 2-fazowe[4] mają 4 lub 6 przewodów, które są połączone w następującej konfiguracji:



Silniki 4-ro przewodowe nie posiadają wyprowadzeń oznaczonych na rysunku numerami 5 i 6. Jako iż każda cewka posiada swoją rezystancję przyporządkowanie poszczególnych przewodów do cewek możemy sprawdzić korzystając z omomierza.

Z kolei silnik 4-ro fazowy posiada 5 wyprowadzeń – po jednym od każdej z 4 cewek i dodatkowo jedno wspólne zakończenie wszystkich cewek połączone razem. Tu także wyprowadzenia możemy zidentyfikować omomierzem.



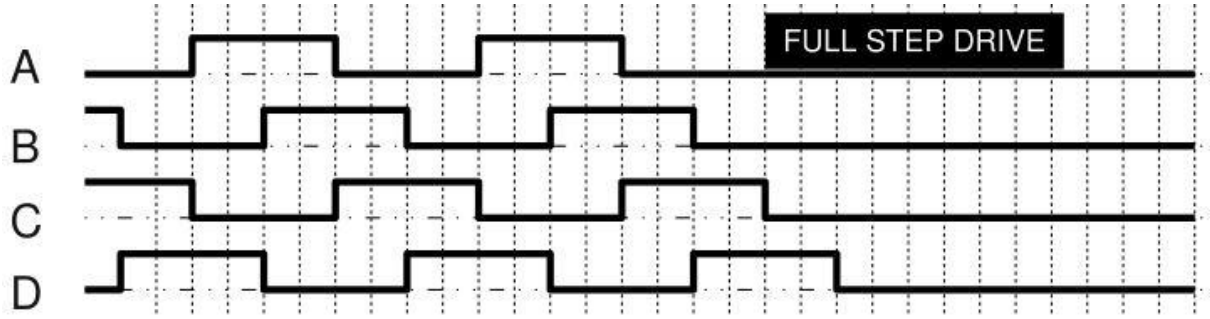
Sterowanie silnikiem 2-fazowym

Na początek „pomęczmy" jeszcze moduł sterownika silników i układ L293D. Na początek sprawdźmy czy prąd znamionowy naszego silnika nie przekracza możliwości sterownika, a także jakie napięcie musimy podać na wejście zasilania. Napiszmy więc bardzo prosty program który jedynie wprawi w ruch nasz silnik w trybie pełno-krokowym. Końcówki uzwojeń podpinamy odpowiednio do złącz M1 i M2, tak aby każde uzwojenie było podpięte jakby zwykły silnik DC. A teraz bardzo prosty kod:

[4] dalej używam pewnego uproszczenia w nazewnictwie i uogólniam

```
void setup(){
  pinMode(10, OUTPUT);
  pinMode(11, OUTPUT);
  pinMode(12, OUTPUT);
  pinMode(13, OUTPUT);//wszystkie piny do kontroli silnika jako wyjścia
  digitalWrite(10, HIGH);//aktywujemy oba uzwojenia
  digitalWrite(11, HIGH);

  pinMode(A1, INPUT);//przyciski do zmiany kierunku
  digitalWrite(A1, HIGH);
  pinMode(A2, INPUT);
  digitalWrite(A2, HIGH);
}

void full_step(byte krok){
  switch(krok%4){//mamy do dyspozycji 4 kolejne kombinacje stanów -
kierunków przepływu prądu przez uzwojenia
    case 0:
       digitalWrite(12, LOW);
       digitalWrite(13, LOW);
    break;
    case 1:
       digitalWrite(12, HIGH);
       digitalWrite(13, LOW);
    break;
    case 2:
       digitalWrite(12, HIGH);
       digitalWrite(13, HIGH);
    break;
    case 3:
       digitalWrite(12, LOW);
       digitalWrite(13, HIGH);
    break;
  }
}


byte licz=0;
boolean dir=0;
void loop(){
  if(dir){//w zależnmości od kierunku
    full_step(licz++);
  }else{
    full_step(licz--);
  }
  if(digitalRead(A1)==0)dir=0;
  if(digitalRead(A2)==0)dir=1;
  delay(250);//czekamy moment, aby silnik nie kręcił się za szybko
}
```

Jak widać właściwie nie robimy nic ponad przełączanie kierunku przepływu prądu przez uzwojenia. W wyniku działania programu na wyjściach otrzymujemy taki przebieg:



Jednak jeśli już korzystamy z silników krokowych zwykle chodzi nam nie o ciągłe obroty, ale o wykonanie z góry założonej ilości kroków w danym kierunku (aby np. przesunąć ramię obrabiarki o określoną odległość). Dodatkowo nie mam sensu marnować czasu działania procesora na zwykłe oczekiwanie – i tu z pomocą przyjdzie nam timer. Poniżej przedstawiam prosty program wykonujący określoną liczbę kroków podaną z komputera:

```
#include <TimerOne.h>

//Zmienne globalne
volatile int steps=0;
volatile boolean dir=0;

void setup(){
  pinMode(10, OUTPUT);
  pinMode(11, OUTPUT);
  pinMode(12, OUTPUT);
  pinMode(13, OUTPUT);//wszystkie piny do kontroli silnika jako wyjścia

  digitalWrite(10, HIGH);//aktywujemy oba uzwojenia
  digitalWrite(11, HIGH);

  Serial.begin(9600);

  Timer1.initialize();
  Timer1.setPeriod(10000);//częstotliwość pracy silnika[5] = 100Hz (1/T,
gdzie T=10 000 us)
  Timer1.attachInterrupt(silnik);
}

int odbierz(){//funkcja odbioru liczb z części 6 przystosowana do odbioru
liczb typu int (bez znaku)
  byte odebrany=0;
  int liczba=0;
  do{
    if(Serial.available()>0){
      odebrany=Serial.read();
      if(odebrany!=10&&odebrany!=13){//filtrujemy znaki CR i LF (tak też
ustawiamy terminal RS232)
        liczba*=10;
        liczba+=odebrany-'0';
      }
    }
  }while(odebrany!=10);//jeśli odebraliśmy znak LF to wychodzimy z pętli
  return liczba;//zwracamy odebraną liczbę
}
```

[5] Jeśli nasz silnik ma mniej kroków na obrót konieczne może okazać się zmniejszenie częstotliwości jego pracy

```
void full_step(byte krok){
  switch(krok%4){//mamy do dyspozycji 4 kolejne kombinacje stanów - kierunków przepływu prądu przez uzwojenia
    case 0:
       digitalWrite(12, LOW);
       digitalWrite(13, LOW);
    break;
    case 1:
       digitalWrite(12, HIGH);
       digitalWrite(13, LOW);
    break;
    case 2:
       digitalWrite(12, HIGH);
       digitalWrite(13, HIGH);
    break;
    case 3:
       digitalWrite(12, LOW);
       digitalWrite(13, HIGH);
    break;
  }
}

void silnik(){
  static byte licz=0;
  if(steps){//jeśli silnik ma wykonać jakieś kroki
    licz+=1-2*(!dir);//mała sztuczka na szybkie zmienianie wskazania aktualnego kroku w zależności od kierunku
    full_step(licz);
    steps--;
  }
}

void kroki(int ile, byte kier){//ustawiamy zmienne
  dir=kier;
  steps=ile;
}

boolean gotowy(){//sprawdzamy czy poprzednie polecenie zostało wykonane
  return (steps==0);
}

void loop(){
  int l_krokow;
  byte kierunek;
  if(gotowy()){
    Serial.print("Podaj liczbe krokow: ");
    l_krokow=odbierz();
    Serial.println(l_krokow, DEC);
    Serial.print("Podaj kierunek: ");
    kierunek=odbierz();
    Serial.println(kierunek, DEC);
    kroki(l_krokow, kierunek);
    Serial.println();
  }
}
```

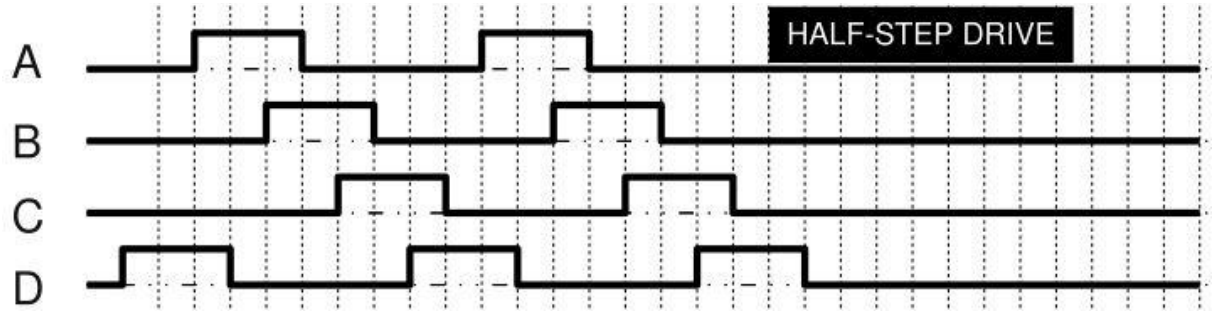
Jeśli mimo wszystko (w przypadku naszego silnika) dokładność 1,8° byłaby niewystarczająca możemy łatwo ją dwukrotnie zwiększyć korzystając z trybu pracy pół-krokowej. Konieczne wtedy będzie dodatkowe operowanie załączaniem i wyłączaniem poszczególnych uzwojeń. W naszych programach

wystarczy tylko podmienić jedną funkcję i pamiętać, że teraz na pełny obrót potrzebujemy 400 półkroków.

```
void half_step(byte krok){
  switch(krok%8){//mamy do dyspozycji 8 kombinacji stanów w trybie
półkrokowym
    case 0:
       digitalWrite(10, HIGH);
       digitalWrite(12, LOW);

       digitalWrite(11, HIGH);
       digitalWrite(13, LOW);
    break;
    case 1:
       digitalWrite(10, LOW);
       digitalWrite(12, LOW);

       digitalWrite(11, HIGH);
       digitalWrite(13, LOW);
    break;
    case 2:
       digitalWrite(10, HIGH);
       digitalWrite(12, HIGH);

       digitalWrite(11, HIGH);
       digitalWrite(13, LOW);
    break;
    case 3:
       digitalWrite(10, HIGH);
       digitalWrite(12, HIGH);

       digitalWrite(11, LOW);
       digitalWrite(13, LOW);
    break;
    case 4:
       digitalWrite(10, HIGH);
       digitalWrite(12, HIGH);

       digitalWrite(11, HIGH);
       digitalWrite(13, HIGH);
    break;
    case 5:
       digitalWrite(10, LOW);
       digitalWrite(12, HIGH);

       digitalWrite(11, HIGH);
       digitalWrite(13, HIGH);
    break;
    case 6:
       digitalWrite(10, HIGH);
       digitalWrite(12, LOW);

       digitalWrite(11, HIGH);
       digitalWrite(13, HIGH);
    break;
    case 7:
       digitalWrite(10, HIGH);
       digitalWrite(12, LOW);

       digitalWrite(11, LOW);
```

```
      digitalWrite(13, HIGH);
    break;
  }
}
```

Oprócz tego możemy zwiększyć częstotliwość pracy silnika do 200Hz, aby nadal obracał się z taką samą szybkością jak poprzednio. Poniżej zamieszczam jeszcze wygląd przebiegów na wyjściach sterownika w trybie sterowania pół-krokowego:



Sterowanie silnikiem 4-fazowym

Jak już wcześniej wspomniałem do sterowania silnika 4-fazowego wykorzystamy układ ULN2003/2803. W takim układzie podłączamy wspólne wyprowadzenie cewek silnika do zasilania (zgodnie z danymi znamionowymi), natomiast poszczególne zakończenia cewek podpinamy do układu ULN2803/2003. Tym razem nie musimy zmieniać kierunku przepływu prądu – wystarczy tylko powodować przepływ prądu w kolejnych cewkach. Możemy wykorzystać większość programu z poprzedniego ćwiczenia, musimy jednak podmienić funkcję odpowiadającą za ustawienie odpowiedniej kombinacji stanów dla danego kroku. Pamiętajmy też o doborze częstotliwości – ja w swoich testach wykorzystałem silnik 4-fazowy, który miał znacznie mniej kroków niż 400, bo zaledwie 50. Poniżej przedstawiam zmienioną procedurę do sterowania silnikiem 4-fazowym w trybie pełno-krokowym. Kolejne uzwojenia są podpięte (poprzez układ ULN2003/2803) do pinów 13, 12, 11 i 10.

```
void full_step(byte krok){
  switch(krok%4){//mamy do dyspozycji 4 kolejne kombinacje stanów -
kierunków przepływu prądu przez uzwojenia
    case 0:
       digitalWrite(13,HIGH);
       digitalWrite(12,LOW);
       digitalWrite(11,LOW);
       digitalWrite(10,LOW);
    break;
    case 1:
       digitalWrite(13,LOW);
       digitalWrite(12,HIGH);
       digitalWrite(11,LOW);
       digitalWrite(10,LOW);
    break;
    case 2:
       digitalWrite(13,LOW);
       digitalWrite(12,LOW);
       digitalWrite(11,HIGH);
       digitalWrite(10,LOW);
    break;
    case 3:
       digitalWrite(13,LOW);
       digitalWrite(12,LOW);
```

```
      digitalWrite(11,LOW);
      digitalWrite(10,HIGH);
    break;
  }
}
```

W podobny sposób jak poprzednio możemy też sterować silnikami w trybie pół-krokowym, wystarczy podmienić odpowiednią procedurę:

```
void half_step(byte krok){
  switch(krok%8){//mamy do dyspozycji 8 kroków
    case 0:
      digitalWrite(13,HIGH);
      digitalWrite(12,LOW);
      digitalWrite(11,LOW);
      digitalWrite(10,LOW);
    break;
    case 1:
      digitalWrite(13,HIGH);
      digitalWrite(12,HIGH);
      digitalWrite(11,LOW);
      digitalWrite(10,LOW);
    break;
    case 2:
      digitalWrite(13,LOW);
      digitalWrite(12,HIGH);
      digitalWrite(11,LOW);
      digitalWrite(10,LOW);
    break;
    case 3:
      digitalWrite(13,LOW);
      digitalWrite(12,HIGH);
      digitalWrite(11,HIGH);
      digitalWrite(10,LOW);
    break;
    case 4:
      digitalWrite(13,LOW);
      digitalWrite(12,LOW);
      digitalWrite(11,HIGH);
      digitalWrite(10,LOW);
    break;
    case 5:
      digitalWrite(13,LOW);
      digitalWrite(12,LOW);
      digitalWrite(11,HIGH);
      digitalWrite(10,HIGH);
    break;
    case 6:
      digitalWrite(13,LOW);
      digitalWrite(12,LOW);
      digitalWrite(11,LOW);
      digitalWrite(10,HIGH);
    break;
    case 7:
      digitalWrite(13,HIGH);
      digitalWrite(12,LOW);
      digitalWrite(11,LOW);
      digitalWrite(10,HIGH);
    break;
  }
```

```
}
```

Cóż, na tym przykładzie zakończymy nasze dzisiejsze spotkanie. Jeśli macie dostęp do większej liczby silników krokowych i posiadacie warsztat możecie dzięki poznanym informacjom zbudować np. prosty ploter sterowany z naszego zestawu Arduino.