

Controlling a small unipolar stepping motor

Introduction

For many applications, stepping motors are preferable to conventional DC motors or servomotors.

Typical advantages of stepping motors are:

- precise positioning without position sensor feedback
- small rotations without gear reduction
- high torque especially for small speed
- controllable speed (especially low speed)
- possibility to hold the positioning after moving
- instantaneous start and stop

In this Application Note we demonstrate how a small stepping motor can be directly connected to the ADB I/O through Port A.

What is a stepping motor

A stepping motor is an electrical motor which has no brush but windings (2 or 4) on the stator only.

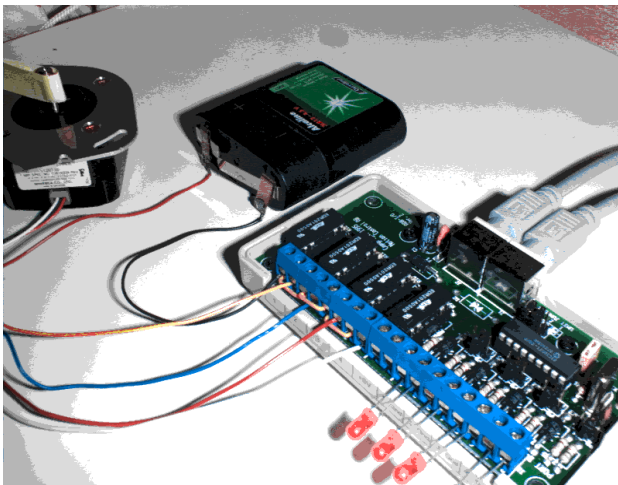


Fig. 1 The stepping motor connected to port A of ADB I/O. The orange LEDs connected to Port B monitor the currents of the motor windings

It turns of one step each time that at least one winding is connected or disconnected from the supply.

The number of step per revolution is variable but the most common values are 100 or 200 steps/turn meaning an angular resolution of 3.6 or 1.8 degrees.

In principle a stepping motor can turn continuously if the windings are successively powered in a well defined sequence. This is why it is often driven by an electronic interface which controls the windings.

In this Application the electronic interface is completely replaced by the Mac and the ADB I/O.

As can be seen on Fig. 1, the only parts are the ADB I/O, The motor and a battery.

How to recognise a unipolar stepping motor

There are two types of stepping motors:

- unipolar stepping motors
- bipolar stepping motors

Since this application works only with unipolar stepping motors, we give the following rules to recognise a unipolar stepping motor:

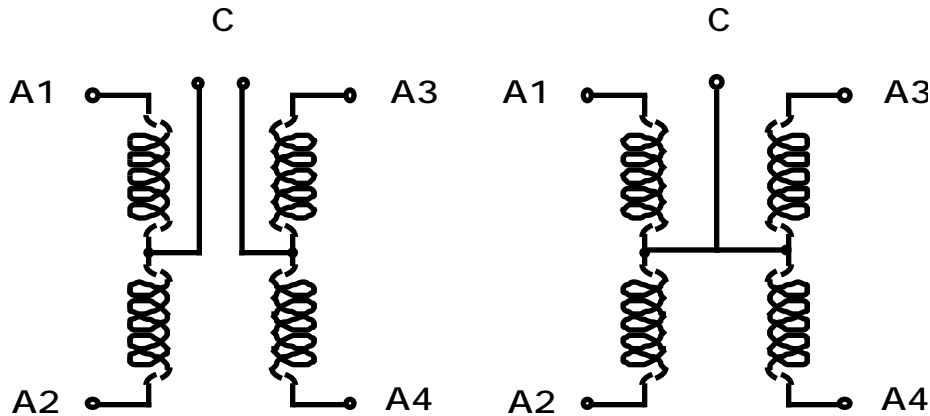


Fig. 2 Connections of 6 wires and 5 wires unipolar stepping motors

- A typical unipolar stepping motor has 5 wires.
- in some cases it may have 6 wires.
- a stepping motor which has only 4 wires is definitely a bipolar one.

The windings of a unipolar motor are connected as shown on Fig.2.

They are made of two independent windings with a middle point wire on each (6 wires unipolar). Possibly the two middle points may be internally connected to a single wire called Common (C)(5 wires unipolar).

In the present application the middle points are always connected together whatever this is done inside or outside the motor, so that Fig. 2.b always applies.

What type of stepping motor to choose

Since the motor windings are driven by the ADB I/O relays, one must choose a motor compatible with the relays characteristics (max. 100 V, 500 mA).

Taking into account possible spikes due to switching-off the windings, it is recommended not to go beyond 12 V 200 mA. The motor shown on Fig. 1 has 4 windings of 75 Ohms (12 V - 160 mA/phase) and works fine when powered by a 4.5 V battery.

How to identify the wires

The first task when starting with a stepping motor is to identify the wires.

This can be done with an Ohm-Meter. Each time you find a resistance R, you are between C and one of the A terminals. If you are measuring 2R you are between two A terminals.

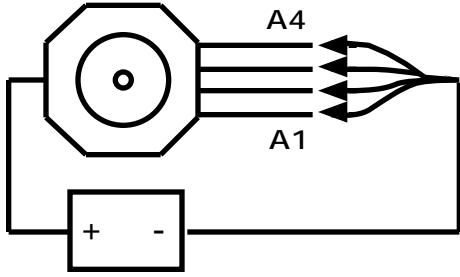


Fig. 3 Determination of the good colour sequence

Now all A wires are not equivalent, this is why they have a different colour. Unfortunately the colours are not universal and you will have to determine what is the "good colour sequence". The "good sequence" is the one which will make the motor to run continuously and smoothly, not going back and forth. To do that, connect the C wire to one (say +) of the pole of a small battery (4.5 V is OK). Then connect successively the 4 A wires to the other pole (Fig. 3). If the motor has always turned in the same direction you won: write quickly that the "good sequence" was brown, blue, green, red!

But if you noticed the motor going in the wrong direction while connecting the 3rd wire, this means that the 3rd wire is wrong in the sequence. Repeat the beginning of the sequence but change the 3rd colour.

Doing like this, you should quickly determine what is the "good sequence".

Note that a sequence is cyclic, which means that if:

	brown	blue	green	red	is a good sequence
then	blue	green	red	brown	
or	green	red	brown	blue	
or	red	brown	blue	green	

are also good sequences.

This also means that no matter the colour you are starting from to test a given sequence.

Connecting the stepping motor to ADB I/O

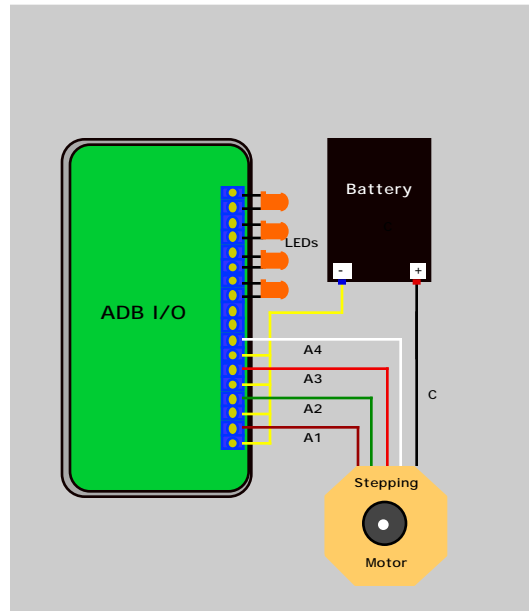
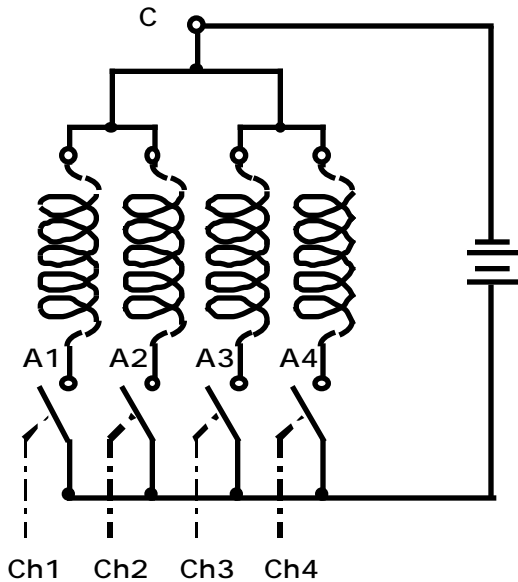


Fig. 4 Connection of the ADB I/O to the motor

As shown on Fig.4 a, the wiring requires a common "ground" wire connected to one side of each relay screw terminal. This is done by the yellow wire directly screwed on blocks (Fig.1 and Fig. 4 b) which connects all the ADB I/O right output screws. Once the good sequence has been determined just connect the first motor wire of the motor (say brown) to channel 1, the second to channel 2 ...

Then connect the common (or the two common) wire to the + battery.

Finally connect the - battery to the yellow ground through one of the block screws.

That's all for the hardware...

You can now test that the motor moves a bit each time you set one of the channel of Port A to HIGH with the ADB I/O standard controller.

In the second part of this Note, we give some examples of HyperTalk Scripts, and briefly describe an HyperCard Applications used to run the motor.

The Software

The HyperCard Application

The software is an HyperCard stack made of 2 Cards with HyperTalk scripts.

! If you want to create your own stack, remind to install first the XCMDs for ADB I/O (see ADB I/O Manual P. 21)

The first card only contains one script to configure ADB ports A and B, and to define the sequence numbers (listN). This script is executed by clicking in the card. At the end of the script, the second card is opened.

The first card also has one field to report on errors returned by ConfigureADBIO.

The second card has 5 buttons and 6 fields.

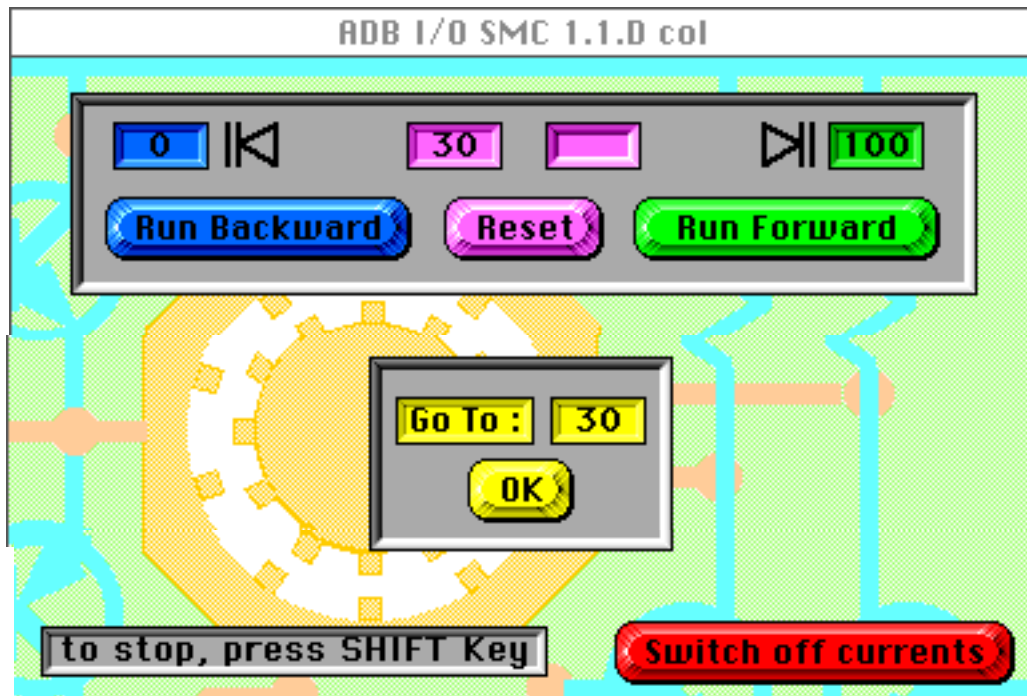


Fig. 5 The HyperCard interface for the Stepping motor controller

Buttons

The "Forward" button is used to make the motor run forward until the shift key is pressed or the upper limit is reached.

The "Backward" button is used to make the motor run backward until the shift key is pressed or the lower limit is reached.

The "Go To" button is used to make the motor reach the value (target) entered in the field just on the right (card field 6).

The "switch off current" button opens all 4 relays.

The "Reset position counter" makes the counter (just on the left) to reset at the present position. This is to be used at the beginning or to set a new origin for angles.

Fields

The upper left field (card field 4) is a field where you enter the lower limit for the motor position. When you close the field (i.e. pressing the Enter key while the cursor is in the field, or clicking outside the field) this lower limit is put into the global variable "downstop" to be used by the button scripts.

The upper right field (card field 5) do the same for the upper limit (upstop).

The left-centre field (card field 2) is the step counter. It is cumulative for successive operations (Forward, Backward, Go To).

The value of the step counter is stored in the field even after closing the stack or shutting down the Mac.

The right-centre field (card field 3) displays the number of steps made since the present movement has started, while the motor is running.

The field on the right of the "Go To button" (card field 6) is the field where you enter the target to be reached. The target is first checked to be within the limits defined in the above fields.

The bottom left field (card field 1) is an information field (not for input).

The logic for the motor motion.

The principle to move the motor is simply to make the current to flow in the windings according to the following sequence:

step #	1	2	3	4	----->
channel 1	1	0	0	1	----->
channel 2	1	1	0	0	----->
channel 3	0	1	1	0	----->
channel 4	0	0	1	1	----->
decimal N	3	6	12	9	

- In the above table, "step #" in the first line represent four successive steps in the currents setting.
- Each step correspond to a SetADBIO output command which is defined in the corresponding column.
- Numbers 1 and 0 in the table means current ON an OFF.
- The channel setting at a given time is given by the 4 bit number in the corresponding column.
- "decimal N" is just the decimal value of the corresponding binary number in the column above, the Least Significant Bit being on the top.(ex.: for step #3 the currents (0011) are represented by $N = 0+0+2**2+2**3 = 12$).

Example HyperTalk Scripts

This four steps sequence can be executed by the following script:

```

on mouseUp
  put "3,6,12,9" into listN
  repeat with k=1 to 4
    put item k of listN into N
    SetADBIO 1,A,N
    put N into card field 1
    wait 60 -- wait 1 second
  end repeat
end mouseUp

```

Here is a script to make an arbitrary number of steps.

It uses the (j mod 4) function which returns the remainder of the division of j by 4. Note the correspondence:

j	0	1	2	3	4	5	6	7
(j mod 4)	0	1	2	3	0	1	2	3
k	1	2	3	4	1	2	3	4

since item 0 is invalid and (j mod 4) never reaches the value 4.

```

on mouseUp
  put "3,6,12,9" into listN
  put 10 into jmax
  repeat with j=0 to jmax-1
    put ((j mod 4)+1) into k
  end repeat
end mouseUp

```

```
    put item k of listN into N
    SetADBIO 1,A,N
    put N into card field 1
    wait 60
end repeat
end mouseUp
```

Here is a script including a real time interrupt (by the shift key):

```
on mouseUp
    put "3,6,12,9" into listN
    put 20 into jmax
    put 0 into j
    repeat while ((the shiftkey is up) and (j < jmax))
        put ((j mod 4)+1) into k
        put item k of listN into N
        SetADBIO 1,A,N
        put j+1 into j
        put N into card field 1
        put j into card field 2
        wait 60
    end repeat
end mouseUp
```

Scripts of the HyperCard Application

Here are the main scripts of the HyperCard Application shown on Fig. 4.

Note that they include commands to Ports A and B.

Port B is used to drive four LEDs shown on Fig.1, which monitor the currents in the windings.

This, together with the “wait 30” commands, is for making easier the development of the software

1st Card

Card Script

```
on mouseUp
  global listN
  put false into err
  put "" into card field 1
  put "" into str
  ConfigureADBIO 1, A, "digital out", "digital out", "digital out", "digital out"
  if the result is not empty then
    put the result && " ( Port A)" into card field 1
    put " Port A + " into str
    put true into err
    wait 30
    beep
  end if
  ConfigureADBIO 1, B, "digital out", "digital out", "digital out", "digital out"
  if the result is not empty then
    put the result & " (" & str & " Port B)" into card field 1
    put true into err
    wait 30
    beep
  end if
  if err is false then
    put "3,6,12,9" into listN
    put "to stop, press on SHIFT Key" into card field 1 of card 2
    set the lockMessages to true
    visual effect venetian blinds slow
    go next card
  end if
end mouseUp
```

2nd Card

Button Script "Forward"

```
on mouseDown
  put card field 5 into upstop
```

```
do goForw upstop
end mouseDown
```

Button Script "Backward"

```
on mouseDown
  put card field 4 into downstop
  do goBack downstop
end mouseDown
```

Button Script "Go to"

```
on mouseUp
  put card field 4 into downstop
  put card field 5 into upstop
  put card field 6 into target
  put card field 2 into presentpos
  if target < downstop or target < 0 or target > upstop then beep 3
  else if target > presentpos then goForw target
  else if target < presentpos then goBack target
end mouseUp
```

Button Script "Reset"

```
on mouseUp
  global listN
  put 0 into card field 2
  put item 4 of listN into N
  SetADBIO 1,A,N
  SetADBIO 1,B,N
end mouseUp
```

Button Script "Switch off currents"

```
on mouseUp
  SetADBIO 1,B,0
  SetADBIO 1,A,0
end mouseUp
```

Field Script "upper limit"

```
on closeField
  global upstop
  put card field 5 into upstop
end closeField
```

Field Script "lower limit"

```
on closeField
  put card field 4 into downstop
  if downstop < 0 then
    repeat 2
```

```

    beep 3
    set hilite of button 3 to true
    wait 25
    set hilite of button 3 to false
    wait 25
end repeat
end if
end closeField

```

Field Script "Go to"

```

on closeField
    put card field 4 into downstop
    put card field 5 into upstop
    put card field 6 into target
    if (target<0 or target<downstop or target>upstop) then
        repeat 2
            beep 3
            set hilite of button 3 to true
            wait 25
            set hilite of button 3 to false
            wait 25
        end repeat
    end if
end closeField

```

Card 2 Scripts

```

on goForw ATarget
    global listN
    put 0 into i      -- i is the current number of steps of the present run
    put card field 2 into j -- j is the cumulated number of steps since the last reset
    repeat while ((the shiftkey is up) and (j < ATarget))
        put ((j mod 4)+1) into k
        put item k of listN into N -- N is the number which binary value represents the
        --channels states
        --SetADBIO 1,B,N
        SetADBIO 1,A,N
        put i+1 into i
        put j+1 into j
        put i into card field 3
        put j into card field 2
        --wait 60
    end repeat
    if j>=ATarget then beep
    put "" into card field 3
    put j into card field 2 -- j is stored for future use
end goForw

on goBack ATarget

```

```
global listN,downstop,upstop
put card field 4 into downstop
put 0 into i      -- i is the current number of steps
put card field 2 into j -- j is the cumulated number of steps
repeat while ((the shiftkey is up) and (j>0) and (j>ATarget))
  put ((j-2) mod 4)+1 into k
  put item k of listN into N -- N is the number which binary value
  -- represents the channels states
  --SetADBIO 1,B,N
  SetADBIO 1,A,N
  put i+1 into i
  put j-1 into j
  put i into card field 3
  put j into card field 2
  --wait 60
end repeat
if j<=ATarget then beep
put "" into card field 3
put j into card field 2 -- j is stored for future use
end goBack
```