AVR446: Linear speed control of stepper motor

Features

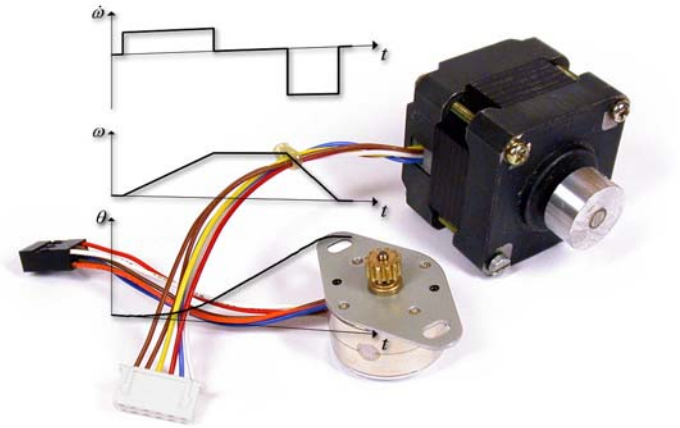
- **Linear speed control of stepper motor**
  - **Control of acceleration, deceleration, max speed and number of steps to move**
- **Driven by one timer interrupt**
- **Full- or half-stepping driving mode**
- **Supports all AVR® devices with 16bit timer**
- **Demo application for ATmega48 running on 3,68MHz, with serial interface on 19200 8/N/1.**

1 Introduction

This application note describes how to implement an exact linear speed controller for stepper motors. The stepper motor is an electromagnetic device that converts digital pulses into mechanical shaft rotation. Many advantages are achieved using this kind of motors, such as higher simplicity, since no brushes or contacts are present, low cost, high reliability, high torque at low speeds, and high accuracy of motion. Many systems with stepper motors need to control the acceleration/ deceleration when changing the speed. This application note presents a driver with a demo application, capable of controlling acceleration as well as position and speed.

This linear speed controller is based on an algorithm presented in ‘Embedded Systems Programming’ January 2005, ‘Generate stepper-motor speed profiles in real time’ an article by D. Austin. This algorithm allows parameterization and calculation in real time, using only simple fixed-point arithmetic operations and no data tables.

Figure 1-1. Stepper motors



8-bit AVR® Microcontrollers

Application Note

Rev. 8017A-AVR-06/06

2 Theory

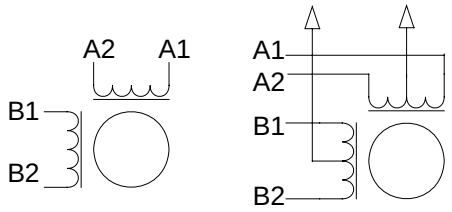
2.1 Stepper motor

This application note covers the theory about linear speed ramp stepper motor control as well as the realization of the controller itself. It is assumed that the reader is familiar with basic stepper motor operation, but a summary of the most relevant topics will be given. Further details about stepper motors can be found in D. W. Jones, Control of Stepper Motors.

2.1.1 Bipolar vs. Unipolar stepper motors

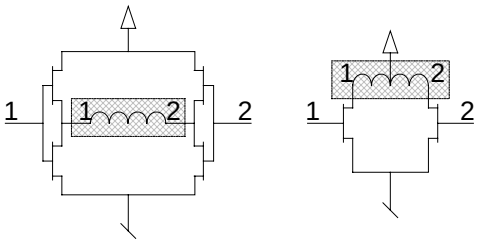
The two common types of stepper motors are the bipolar motor and the unipolar motor. The bipolar and unipolar motors are similar, except that the unipolar has a center tap on each winding as shown in Figure 2-1.

Figure 2-1. Bipolar and Unipolar stepper motors



The bipolar motor needs current to be driven in both directions through the windings, and a full bridge driver is needed as shown in Figure 2-2. The center tap on the unipolar motor allows a simpler driving circuit, also shown in Figure 2-2, limiting the current flow to one direction. The main drawback with the unipolar motor is the limited capability to energize all windings at any time, resulting in a lower torque compared to the bipolar motor. The unipolar stepper motor can be used as a bipolar motor by disconnecting the center tap.

Figure 2-2. Bipolar and Unipolar drivers with MOS transistors



2.1.2 Full vs. half stepping

Stepper motors used in full-stepping mode powers one winding at a time. This way, four different settings (positions) is possible, shown in the ‘Full-stepping’ row of Table 2-1. By powering both windings simultaneous, the stepper motor is trapped between the positions obtained when full-stepping, also known as half-stepping. This gives eight positions as shown in the ‘Half-stepping’ row of Table 2-1. When powering both windings, the torque is approximately 1.4 times higher than when powering only one winding, but at the cost of twice the power consumption. The electrical cycle parts in

Table 2-1are all part of one electrical cycle. One mechanical cycle (revolution) usually consists of several electrical cycles.

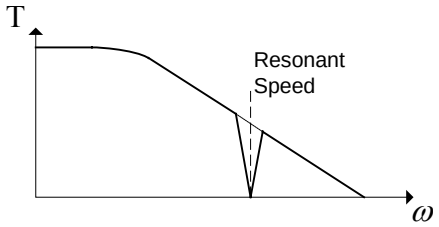
Table 2-1. Full-stepping and half-stepping

Electric polarity	Winding A	+	+		-	-	-		+
	Winding B		+	+	+		-	-	-
Electrical cycle part	Full-stepping	**1**		**2**		**3**		**4**	
	Half-stepping	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**

2.1.3 Speed properties

One drawback with the stepper motor is the limited torque capabilities at high speeds, since the torque of a stepper motor will decrease with increasing speed. The torque also drops at the resonant speed, as shown in Figure 2-3. The resonant speed will depend on the driving scheme of the stepper motor and the load.

Figure 2-3. Torque vs. speed



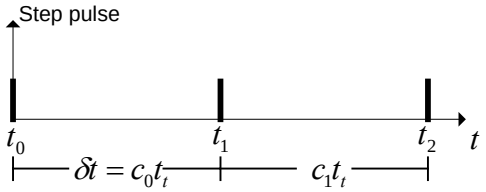
Maximum torque is achieved at low speeds, and this is an advantageous in many applications.

2.2 Fundamental stepper motor equations

To create rotational motion in a stepper motor, the current thru the windings must change in the correct order. This is obtained using a driver that gives the correct output sequence when subjected to a pulse (‘stepper motor pulse’) and a direction signal.

To rotate the stepper motor at a constant speed, pulses must be generated at a steady rate, shown in Figure 2-4.

Figure 2-4. Stepper motor pulses



A counter generates these pulses, running at the frequency f_t [Hz].

The delay δt programmed by the counter c is

$$\delta t = c t_t = \frac{c}{f_t} \text{ [s]}$$

The motor step angle α , position θ , and speed ω are given by

$$\alpha = \frac{2\pi}{spr} \text{ [rad]} \qquad \theta = n\alpha \text{ [rad]} \qquad \omega = \frac{\alpha}{\delta t} \text{ [rad/sec]}$$

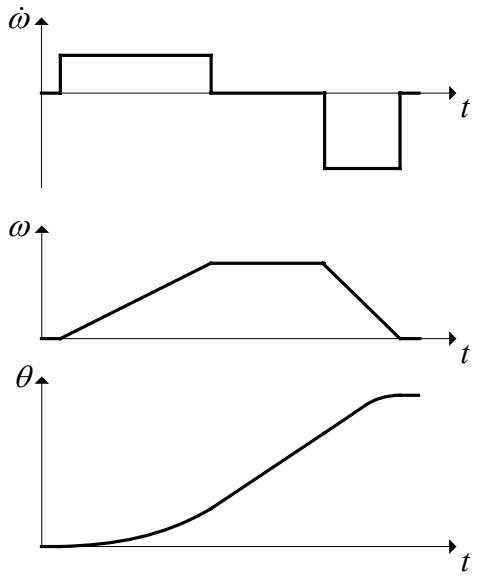
where spr is the number of steps per round, n is the number of steps, and

1 rad/sec = 9,55 rpm

2.3 Linear speed ramp

To start and stop the stepper motor in a smooth way, control of the acceleration and deceleration is needed. Figure 2-5 shows the relation between acceleration, speed and position. Using a constant acceleration/deceleration gives a linear speed profile.

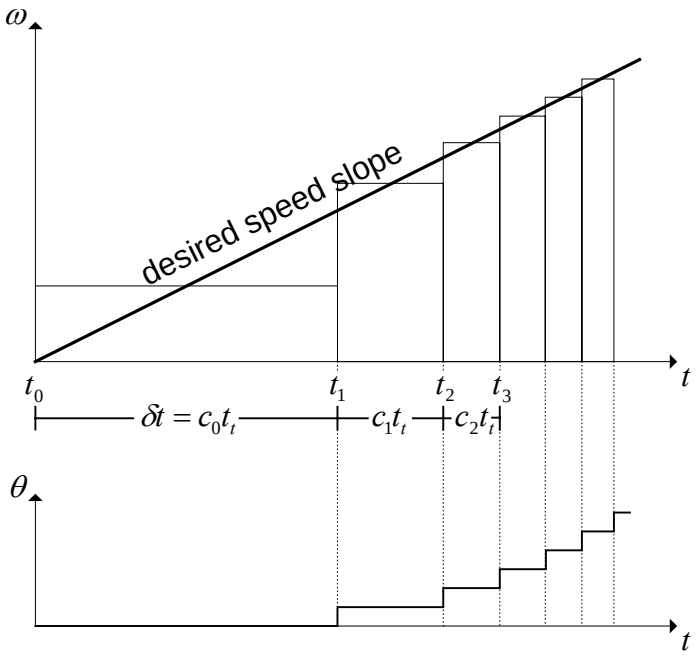
Figure 2-5. Acceleration $(\dot{\omega})$, speed (ω) and position (θ)



The time delay δt between the stepper motor pulses controls the speed. These time delays must be calculated in order to make the speed of the stepper motor follow the speed ramp as closely as possible.

Discrete steps control the stepper motor motion, and the resolution of the time delay between these steps is given by the frequency of the timer.

Figure 2-6. Speed profile vs. stepper motor pulses/speed



2.3.1 Exact calculations of the inter-step delay

The first counter delay c_0 as well as succeeding counter delays c_n, are given by (see appendix for details):

$$c_0 = \frac{1}{t_t}\sqrt{\frac{2\alpha}{\dot{\omega}}} \qquad c_n = c_0\left(\sqrt{n+1} - \sqrt{n}\right)$$

The computational power of a microcontroller is limited, and calculating two square roots is time consuming. Therefore an approximation with less computational complexity is considered.

The counter value at the time n, using Taylor series approximation for the inter-step delay (see appendix for details) is given by:

$$c_n = c_{n-1} - \frac{2c_{n-1}}{4n+1}$$

This calculation is much faster than the double square root, but introduces an error of 0.44 at n =1. A way to compensate for this error is by multiplying c_0 with 0,676.

2.3.2 Change in acceleration

As shown in the appendix, the acceleration is given by c_0 and n. If a change in acceleration (or deceleration) is done, a new n must be calculated.

The time t_n and n as a function of the motor acceleration, speed and step angle are given by

$$t_n = \frac{\omega_n}{\dot{\omega}} \qquad n = \frac{\dot{\omega} t_n^2}{2\alpha}$$

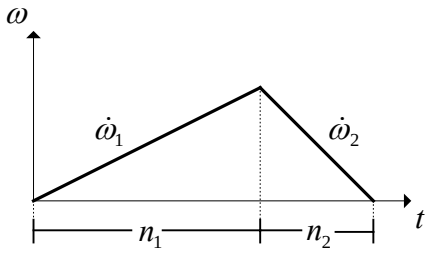
Merging these equation gives the relationship

$$n\dot{\omega} = \frac{\omega^2}{2\alpha}$$

This shows that the number of steps needed to reach a given speed is inversely proportional to the acceleration: $n_1\dot{\omega}_1 = n_2\dot{\omega}_2$

This means that changing the acceleration from $\dot{\omega}_1$ to $\dot{\omega}_2$ is done by changing n. This is shown in Figure 2-7

Figure 2-7. Up/down speed ramp



Moving a given number of steps, deceleration must start at the right step to end at zero speed. The following equation is used to find n_1:

$$n_1 = \frac{(n_2 + n_1)\dot{\omega}_2}{(\dot{\omega}_1 + \dot{\omega}_2)}$$

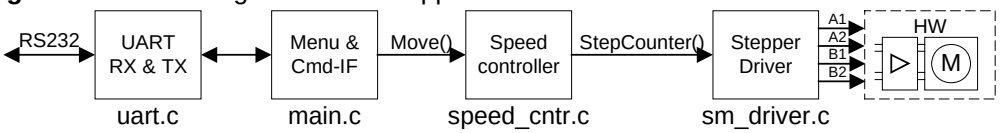
3 Implementation

A working implementation written in C is included with this application note. Full documentation of the source code and compilation information is found by opening the ‘readme.html’ file included with the source code.

The demo application demonstrates linear speed control of a stepper motor. The user can control the stepper motor speed profile by issuing different commands using the serial port, and the AVR will drive the connected stepper motor accordingly.

The demo application is divided in three major blocks, as shown in the block diagram in Figure 3-1. There is one file for each block and also a file for UART routines used by the main routine.

Figure 3-1. Block diagram of demo application



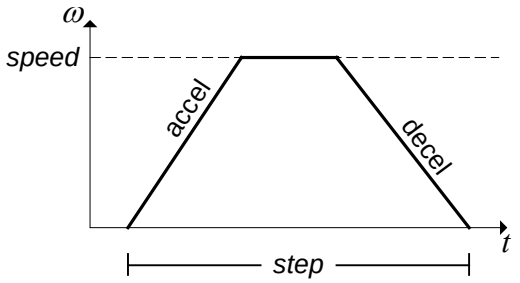
`main.c` has a menu and a command interface, giving the user control of the stepper motor by a terminal connected to the serial line.

`speed_cntr.c` calculates the needed data and generates step pulses to make the stepper motor follow the desired speed profile.

`sm_driver.c` counts the steps and outputs the correct signals to control the stepper motor.

To control the stepper motor, four parameters describing a speed profile are needed. The speed profile starts at zero speed and accelerates up to the given speed. This speed is held constant until deceleration starts. Finally the motor decelerates to zero speed at the given number of steps. A speed profile is shown in Figure 3-2.

Figure 3-2. Speed profile



The parameters describing the speed profile is:

`step` - Number of steps to move.

`accel` - Acceleration to use.

`decel` - Deceleration to use.

`speed` - (Maximum) speed to use.

3.1 Menu and command interface

To use the demo application the user must connect a terminal to the serial port of the AVR. The UART setting is 19200 baud, 8 data bit, none parity and 1 stop bits. Any terminal emulation program should work. Using the terminal the user can give different commands to control the stepper motor and get information back from the demo application.

The UART RX interrupt routine (found in `uart.c`) stores received characters in the receiver buffer and handles backspace. When <enter> (ascii code 13) is received the main routine reads the buffer and executes the given command.

When starting, and on the '?' command, this help screen is shown:

```
------------------------------------------------------------
Atmel AVR446 - Linear speed control of stepper motor

?        - Show help
a [data] - Set acceleration (range: 71 - 32000)
d [data] - Set deceleration (range: 71 - 32000)
s [data] - Set speed (range: 12 - motor limit)
m [data] - Move [data] steps (range: -64000 - 64000)
move [steps] [accel] [decel] [speed]
         - Move with all parameters given
<enter>  - Repeat last move

    acc/dec data given in 0.01*rad/sec^2 (100 = 1 rad/sec^2)
    speed data given in 0.01*rad/sec (100 = 1 rad/sec)
------------------------------------------------------------
```

After the menu is shown or a command is executed the info line is shown:

```
Motor pos: 0    a:4000  d:4000  s:2000  m:400
```

The demo application gives the current motor position, acceleration, deceleration, and speed settings, as well as the number of steps to move.

There are three different ways to make the stepper motor move:

- Pressing <enter>

  The stepper motor runs as specified by the settings given by the application.
- m [data]

  The stepper motor moves [data] steps, with the given settings.
- move [steps] [accel] [decel] [speed]

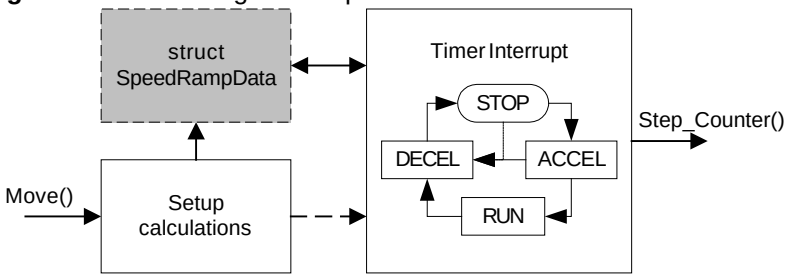
  The application moves [steps] steps, with settings [accel] [decel] [speed].

When the stepper motor starts running '`Running...`' will show. As long as the motor is running, new commands are blocked. After it has stopped '`OK`' shows and new commands are accepted.

3.2 Speed controller

The speed controller calculates and generates the speed profile. The block diagram for the speed controller is shown in Figure 3-3. To run the stepper motor, the speed controller is set up by calling the function Move().

Figure 3-3. Block diagram of speed controller



The function Move() first calculates all the parameters needed and stores them in the speed ramp data struct, then it enables the timer interrupt. The timer generates interrupts according to the desired speed ramp, and calls the function Step_Counter() on each interrupt to move the stepper motor.

3.2.1 Setup calculations

In the demo application parameters for the speed profile is calculated for every command, and a small delay from the call is made to the stepper motor starts moving is introduced. In a real application this may not be necessary if only a limited change in speed profile is needed. In this case the parameters may be calculated in advance and setup calculations skipped.

Use of floating point arithmetic is avoided to make the code fast, therefore scaling of the variables is important to keep the accuracy. Precalculated compiler constants are also used to simplify the arithmetic and can be found in the `smdriver.h` header file:

Find speed:

$$A\_T\_x100 = \alpha f_t \cdot 100$$

$$min\_delay = c = \frac{A\_T\_x100}{speed}$$

Find acceleration:

$$T1\_FREQ\_148 = 0.676 f_t / 100$$

$$A\_SQ = 2\alpha \cdot 10000000000$$

$$step\_delay = c_0 = T1\_FREQ\_148 \sqrt{\frac{A\_SQ}{accel}} / 100$$

There are two different scenarios for calculating the speed profile:

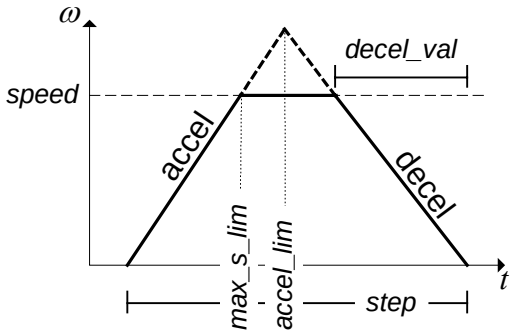
1. Acceleration continues until the desired speed is reached, or
2. Deceleration starts before desired speed is reached.

The scenario depends on all four variables describing the speed profile.

3.2.1.1 Acceleration continues until desired speed is reached

In Figure 3-4 a speed ramp where the desired speed is reached before deceleration starts is shown.

Figure 3-4. Speed ramp limited by desired speed value



- max_s_lim is the number of steps needed to accelerate to the desired speed.

$$max\_s\_\lim = n = \frac{speed^2}{2\alpha \cdot accel \cdot 100}$$

- accel_lim is the number of steps before deceleration starts (disregarding desired speed).

$$accel\_\lim = n_1 = \frac{step \cdot decel}{accel + decel}$$

If $max\_s\_\lim < accel\_\lim$ the acceleration is limited by reaching desired speed.

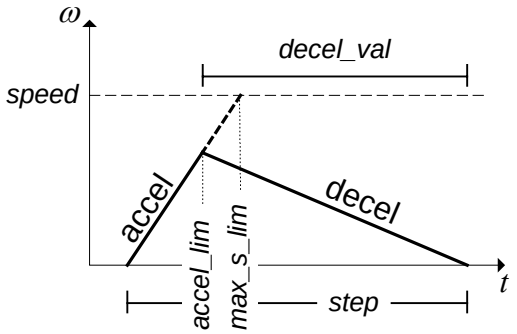
The deceleration depends on this, and in this case decal_val is found by:

$$decel\_val = -max\_s\_lim \cdot \frac{accel}{decel}$$

3.2.1.2 Deceleration starts before desired speed is reached

In Figure 3-5 a speed ramp where deceleration must start before the desired speed is reached is shown.

Figure 3-5. Speed ramp with deceleration start before desired speed reached



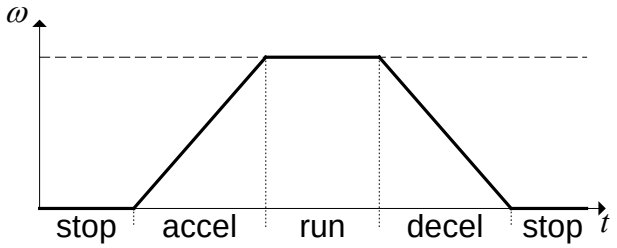
If $max\_s\_\lim > accel\_\lim$ the acceleration is limited by deceleration start, decal_val is then found by:

$$decal\_val = -(step - accel\_lim)$$

3.2.2 Timer interrupt

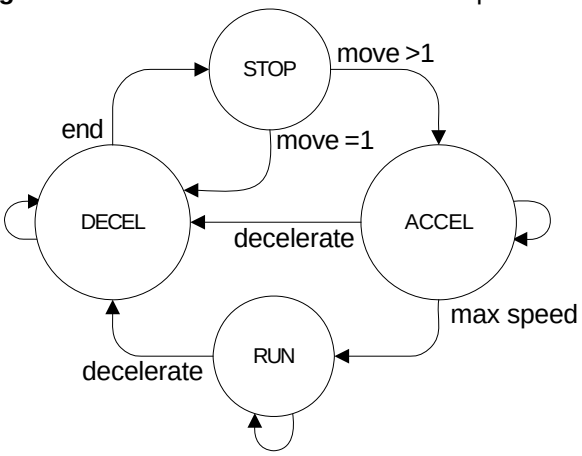
The timer interrupt generates the ‘step pulses’ (calls the function StepCounter()) and is only running when the stepper motor is moving. The timer interrupt will operate in four different states according to the speed profile, shown in Figure 3-6.

Figure 3-6. Operating states for different speed profile parts



This behavior is realized with a state machine in the timer interrupt, shown in Figure 3-7.

Figure 3-7. State machine for timer interrupt



When the application starts or when the stepper motor is stopped the state-machine remains in the state STOP. When setup calculations are done, a new state is set and the timer interrupt is enabled. When moving more than one step the state-machine goes to ACCEL. If moving only 1 step, the state is changed to DECEL.

When the state is changed to ACCEL, the application accelerates the stepper motor until either

the desired speed is reached and the state is changed to RUN, or

deceleration must start, changing the state to DECEL.

When the state is set to RUN, the stepper motor is kept at constant speed until deceleration must start, then the state is changed to DECEL.

It will stay in DECEL and decelerate until the speed reaches zero desired number of steps. The state is then changed to STOP.

3.2.2.1 Calculations and counters

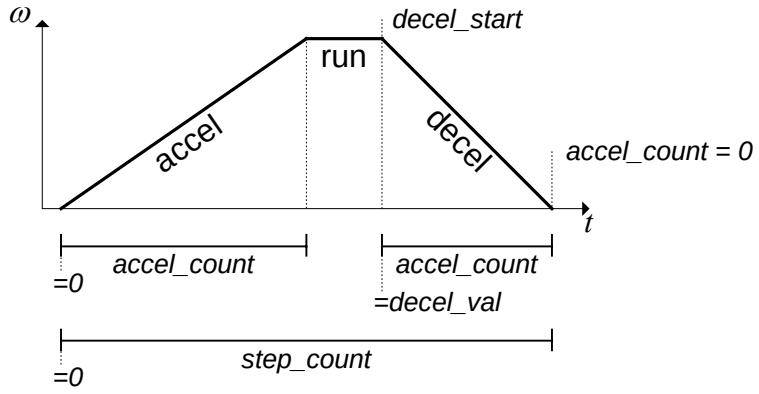
For each step during acceleration and deceleration a new time delay must be calculated. This calculation includes a division giving a remainder, and to improve accuracy this remainder is kept and included in the next calculation.

$$new\_step\_delay = step\_delay - \frac{2 \cdot step\_delay + rest}{4 \cdot accel\_count + 1}$$

$$new\_rest = (2 \cdot step\_delay + rest)(\mathrm{mod}(4 \cdot accel\_count + 1))$$

To keep track of the position and when to change state some counting variables is needed. In Figure 3-8 the use of these are illustrated.

Figure 3-8. Counting variables in time delay

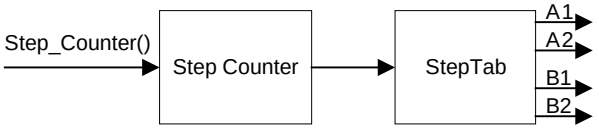


- `step_count` counts steps, starting at zero when ACCEL starts and have the same value as the commanded steps after DECEL is finished.
- `accel_count` is used to control the acceleration/deceleration. In ACCEL it starts at zero and is increased each step until ACCEL ends. When DECEL starts it is set to decel_val, which is negative, and increased each step. When it reaches zero the move is finished, and state set to STOP.
- `decel_start` tells when deceleration starts. When step_count is equal to decel_start state is set to DECEL.

3.3 Stepper motor driver

The stepper motor driver generates the correct sequence of signals to move the stepper motor in the commanded direction. Block schematic for the stepper motor driver is shown in Figure 3-9.

Figure 3-9. Block schematic stepper motor driver



The step counter increments or decrements every time the function Step_Counter() is called. When using fullsteps, the counter value goes from 0 to 3, when using halfsteps, the value goes from 0 to 7. This value equals the different position in one electrical cycle in the stepper motor. The step counter value is used as an index for the step table, and correct signals is given to the stepper motor driver.

3.4 Code size and speed

The complete demo code uses 4k of code memory, the speed controller and stepper motor driver uses 1,5k of this. Removing setup calculations and using precalculated parameters will reduce the code size further.

When calling Move(), setup calculations is done before the timer interrupt is started. This gives a delay of approximately 1,5ms from the call is made to the stepper motor starts. The timer interrupt performs calculations during acceleration and deceleration and approximately 200us are used in one timer interrupt. When running at constant speed less time is need and approximately 35us is sufficient. The maximum output speed is then limited by acceleration/deceleration calculations. For a stepper motor with 400 steps per round maximum output speed is:

$$\omega_{\max} = \frac{2\pi}{200\text{us} \cdot 400} = 78{,}5\ \text{rad/sec}\ \ (= 750\ \text{rpm})$$

When implementing this code in other applications, other interrupts must be taken in consideration. If the timer interrupt for the stepper motor is blocked by another ISR, this will make the stepper motor speed vary (not constant acceleration). It might not be critical in the given application, but making the code as robust and deterministic as possible is always a good thing.

4 Literature references

D. Austin, Generate stepper-motor speed profiles in real time, article in Embedded Systems Programming' January 2005.
http://www.embedded.com//showArticle.jhtml?articleID=56800129

D. W. Jones, Control of Stepper Motors, sections 5.2.10, 10.8, 10.9 and 10.10 of the *Handbook of Small Electric Motors* edited by W. H. Yeadon and A. W. Yeadon, McGraw-Hill, 2001. http://www.cs.uiowa.edu/~jones/step/

5 Appendix

5.1 Counter delay

The speed at a given time is:

$$\omega(t) = \int_{\tau=0}^{t} \dot{\omega} d\tau = \dot{\omega} t$$

The position is given by:

$$\theta(t) = \int_{\tau=0}^{t} \omega(\tau) d\tau = \frac{1}{2}\dot{\omega} t^2 = n\alpha$$

The n'th step pulse at the shaft angle $\theta = n\alpha$ is:

$$t_n = \sqrt{\frac{2n\alpha}{\dot{\omega}}}$$

and the time delay between two steps is:

$$c_n t_t = t_{n+1} - t_n = \sqrt{\frac{2\alpha}{\dot{\omega}}}\left(\sqrt{n+1} - \sqrt{n}\right)$$

Finally the expression for the counter delay is found:

$$c_n = \frac{1}{t_t}\sqrt{\frac{2\alpha}{\dot{\omega}}}\left(\sqrt{n+1} - \sqrt{n}\right)$$

This leads to the expressions for the first and the n'th counter delay:

$$c_0 = \frac{1}{t_t}\sqrt{\frac{2\alpha}{\dot{\omega}}} \qquad c_n = c_0\left(\sqrt{n+1} - \sqrt{n}\right)$$

5.2 Inter-step delay

Using the Taylor series approximation

$$\sqrt{1 \pm \frac{1}{n}} = 1 \pm \frac{1}{2n} - \frac{1}{8n^2} + O\left(\frac{1}{n^3}\right)$$

leads to

$$\frac{c_n}{c_{n-1}} = \frac{c_0\left(\sqrt{n+1} - \sqrt{n}\right)}{c_0\left(\sqrt{n} - \sqrt{n-1}\right)} = \frac{1 + \frac{1}{2n} - \frac{1}{8n^2} + O\left(\frac{1}{n^3}\right) - 1}{1 - \left(1 - \frac{1}{2n} - \frac{1}{8n^2} + O\left(\frac{1}{n^3}\right)\right)} = \frac{4n-1}{4n+1}$$

Finally, the expression for the counter delay can be approximated as:

$$c_n = c_{n-1} - \frac{2c_{n-1}}{4n+1}$$

Atmel Corporation

2325 Orchard Parkway
San Jose, CA 95131, USA
Tel: 1(408) 441-0311
Fax: 1(408) 487-2600

Regional Headquarters

Europe
Atmel Sarl
Route des Arsenaux 41
Case Postale 80
CH-1705 Fribourg
Switzerland
Tel: (41) 26-426-5555
Fax: (41) 26-426-5500

Asia
Room 1219
Chinachem Golden Plaza
77 Mody Road Tsimshatsui
East Kowloon
Hong Kong
Tel: (852) 2721-9778
Fax: (852) 2722-1369

Japan
9F, Tonetsu Shinkawa Bldg.
1-24-8 Shinkawa
Chuo-ku, Tokyo 104-0033
Japan
Tel: (81) 3-3523-3551
Fax: (81) 3-3523-7581

Atmel Operations

Memory
2325 Orchard Parkway
San Jose, CA 95131, USA
Tel: 1(408) 441-0311
Fax: 1(408) 436-4314

Microcontrollers
2325 Orchard Parkway
San Jose, CA 95131, USA
Tel: 1(408) 441-0311
Fax: 1(408) 436-4314

La Chantrerie
BP 70602
44306 Nantes Cedex 3, France
Tel: (33) 2-40-18-18-18
Fax: (33) 2-40-18-19-60

ASIC/ASSP/Smart Cards
Zone Industrielle
13106 Rousset Cedex, France
Tel: (33) 4-42-53-60-00
Fax: (33) 4-42-53-60-01

1150 East Cheyenne Mtn. Blvd.
Colorado Springs, CO 80906, USA
Tel: 1(719) 576-3300
Fax: 1(719) 540-1759

Scottish Enterprise Technology Park
Maxwell Building
East Kilbride G75 0QR, Scotland
Tel: (44) 1355-803-000
Fax: (44) 1355-242-743

RF/Automotive
Theresienstrasse 2
Postfach 3535
74025 Heilbronn, Germany
Tel: (49) 71-31-67-0
Fax: (49) 71-31-67-2340

1150 East Cheyenne Mtn. Blvd.
Colorado Springs, CO 80906, USA
Tel: 1(719) 576-3300
Fax: 1(719) 540-1759

Biometrics/Imaging/Hi-Rel MPU/
High Speed Converters/RF Datacom
Avenue de Rochepleine
BP 123
38521 Saint-Egreve Cedex, France
Tel: (33) 4-76-58-30-00
Fax: (33) 4-76-58-34-80

Literature Requests
www.atmel.com/literature

8017A-AVR-06/06