Quick Start for Beginners to Drive a Stepper Motor

by: Matthew Grant
16-Bit Automotive Applications
Microcontroller Division

Introduction

This application note is for novices who want a general quick-start guide showing how to control a stepper motor. Because stepper motors can be used in a variety of ways and are driven by a variety of devices, there is a great deal of information available about how these motors work and how to use them. To reduce confusion, the focus of this application note is on stepper motors that can be driven by microcontrollers. This document includes basic information needed to get started quickly, and includes a practical example that is simple and easy to implement.

What is a Stepper Motor?

A stepper motor is an electrically powered motor that creates rotation from electrical current driven into the motor. Physically, stepper motors can be large but are often small enough to be driven by current on the order of milliampere. Current pulses are applied to the motor, and this generates discrete rotation of the motor shaft. This is unlike a DC motor that exhibits continuous rotation. Although it is possible to drive a stepper motor in a manner where it has near continuous rotation, doing so requires more finesse of the input waveform that drives the stepper motor. Figure 1 illustrates some basic differences in stepper and DC motor rotation.
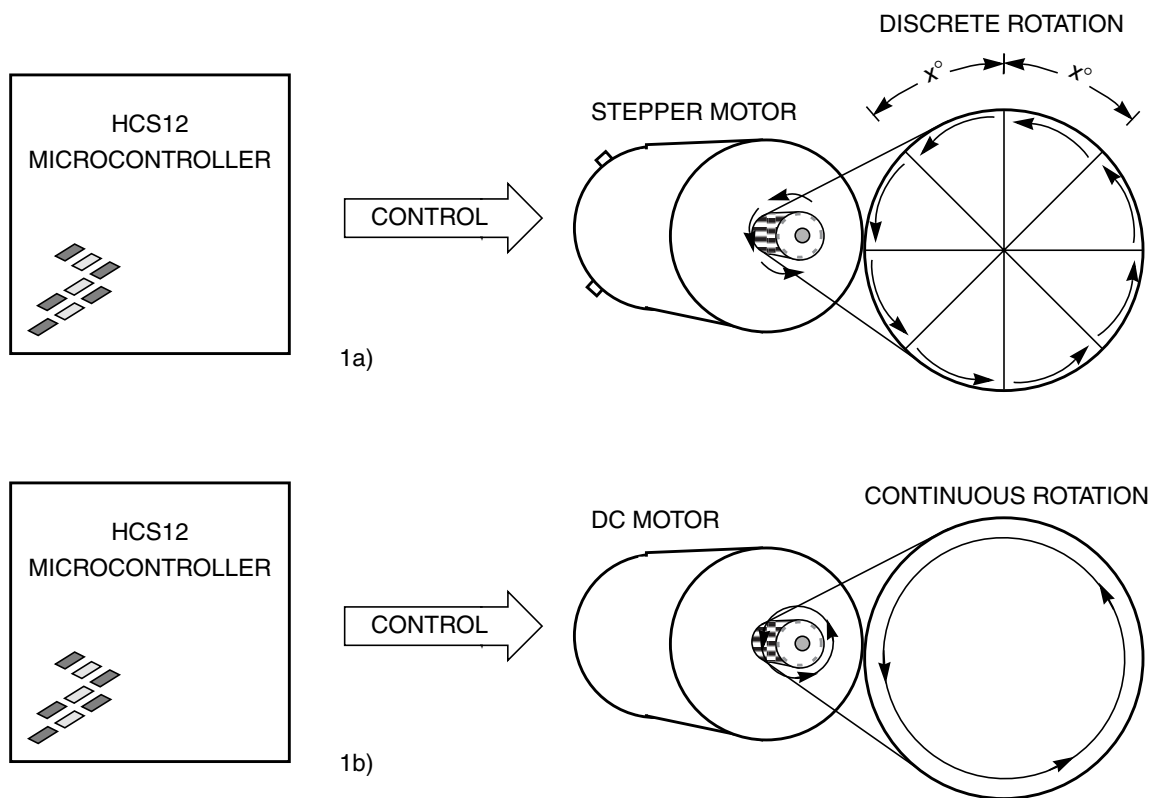
Figure 1. Stepper vs. DC Motor Rotation

Types of Stepper Motors

There are a variety of stepper motors available, but most of them can be separated into two groups:

- **Permanent-magnet (PM) stepper motor** — This kind of motor creates rotation by using the forces between a permanent magnet and an electromagnet created by electrical current. An interesting characteristic of this motor is that even when it is not powered, the motor exhibits some magnetic resistance to turning.

- **Variable-reluctance (VR) stepper motor** — Unlike the PM stepper motor, the VR stepper motor does not have a permanent-magnet and creates rotation entirely with electromagnetic forces. This motor does not exhibit magnetic resistance to turning when the motor is not powered.

What is Inside?

Generally, a stepper motor consists of a stator, a rotor with a shaft, and coil windings. The stator is a surrounding casing that remains stationary and is part of the motor housing, while the rotor is a central shaft within the motor that actually spins during use. The characteristics of these components and how they are arranged determines whether the stepper motor is a PM or VR stepper motor. Figure 2 and Figure 3 show an example of these internal components.
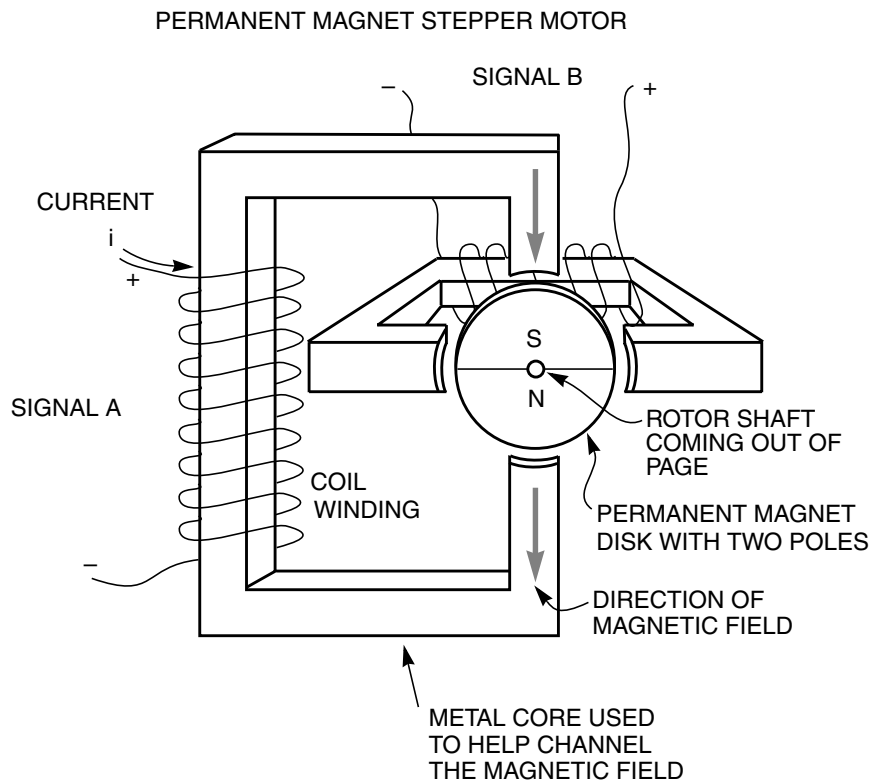

PERMANENT MAGNET STEPPER MOTOR

Figure 2. Permanent Magnet (PM) Stepper Motor

Taking a closer look, the rotor in PM stepper motors is actually a permanent-magnet. In some cases, the permanent magnet is in the shape of a disk surrounding the rotor shaft. One arrangement is a magnetic disk which consists of north and south magnetic poles interlaced together. The number of poles on the magnetic disk varies from motor to motor. Some simple PM stepper motors such as the one in Figure 2 only have two poles on the disk, while others may have many poles. The stator usually has two or more coil windings, with each winding around a soft metallic core.

When electrical current flows through the coil windings, a magnetic field is generated within the coil. The metallic core is placed within the coil windings to help channel the electromagnetic field perpendicular to the outer perimeter of the magnetic disk.

What is Inside?

Depending upon the polarity of the electromagnetic field generated in the coil (north pole, out of the coil, or south pole, into the coil) and the closest permanent magnetic field on the disk, an attraction or repulsion force will exist. This causes the rotor to spin in a direction that allows an opposite pole on the perimeter of the magnetic disk to align itself with the electromagnetic field generated by the coil. When the nearest opposite pole on the disk aligns itself with the electromagnetic field generated by the coil, the rotor will come to a stop and remain fixed in this alignment as long as the electromagnetic field from the coil is not changed.

VR stepper motors work in a very similar fashion. Figure 3 shows some of the physical details that characterize its operation. In a VR stepper motor, the surrounding coils that are physically located opposite of each other are energized to create opposite magnetic fields. For example, in Figure 3a), coil C produces a south-pole magnetic field, and coil $\overline{C}$ produces a north-pole magnetic field. The magnetic fields produced by the coils pass through the air gap and through the metallic rotor. Because the magnetic fields attract each other, the metallic rotor spins in a direction that brings the nearest edges (2 and 4) of the rotor as close as possible to the pair of energized coils (C and $\overline{C}$). Like the PM stepper rotor, the VR stepper rotor will remain aligned to the coils as long as coils C and $\overline{C}$ are energized and the magnetic fields are not changed. To move to the next state and continue this rotation, coils C and $\overline{C}$ must be de-energized, while coils A and $\overline{A}$ must be oppositely energized to attract rotor edges 1 and 3 respectively. The same process occurs with coils B and $\overline{B}$ to attract rotor edges 2 and 4 respectively, and so on. Figure 3 shows how the rotor spins as the coils are energized and de-energized. This is an example of a 3-phase VR motor.

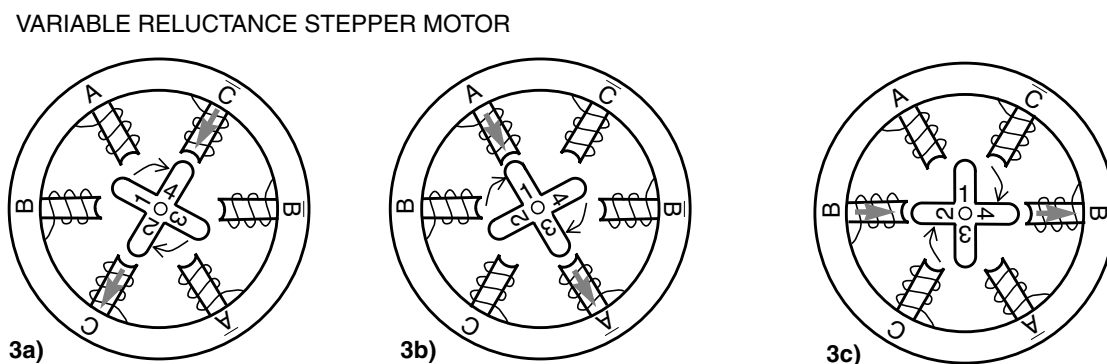VARIABLE RELUCTANCE STEPPER MOTOR

Figure 3. How the Variable Reluctance (VR) Rotor Spins

From the examples discussed earlier, we can see that if the electromagnetic fields in both the PM and VR stepper motors are turned on, off, and reversed in the proper sequence, the rotor can be turned in a specific direction. Each time an electromagnetic field combination is changed, the rotor may turn a fixed number of degrees. As these state changes in electromagnetic fields take place more rapidly, on the order of milliseconds, the rotor can rotate faster, smoother, and sometimes more quietly. Because of the mechanical limitations of the system, the rotor can only rotate effectively up to certain speeds.

An external device, such as an HCS12 microcontroller (or, MCU), is very good for controlling the electromagnetic sequences by directing the flow of current through the coil windings. To do this, software can be written and loaded into an HCS12 MCU.

Waveforms that can Drive a Stepper Motor

Stepper motors have input pins or contacts that allow current from a supply source (in this application note, a microcontroller) into the coil windings of the motor. Pulsed waveforms in the correct pattern can be used to create the electromagnetic fields needed to drive the motor. Depending on the design and characteristics of the stepper motor and the motor performance desired, some waveforms work better than others. Although there are a few options to choose from when selecting a waveform to drive a two-phase PM stepper motor, such as full-stepping or micro-stepping, this application note focuses on one called half-stepping. A graph of the waveform is given in Figure 4.

In Figure 4a), four signals are shown. These signals can be produced by a dedicated stepper driver or a microcontroller. Each signal (a, $\overline{a}$, b, $\overline{b}$) is applied to a coil terminal. Because each coil has two terminals, two signals must work together to drive a single coil. If we consider terminal a as a positive reference, then the combination of signals a and $\overline{a}$ cause the coil to see an effective signal A, shown in Figure 4b). Likewise, signal B in Figure 4b) is produced by combining signals b and $\overline{b}$ from Figure 4a).

It is worth noting that the individual waveforms (a, $\overline{a}$, b, $\overline{b}$) directly from the microcontroller pins to the coil terminals only vary from 0 V to +5 V. However, the effective signal (A, B) applied to the coil varies from −5 V to +5 V, and has positive and negative duty cycles. Two of these effective waveforms shown in Figure 4b), 90 degrees out of phase can be used to drive the PM stepper motor. Both waveforms are applied to the motor simultaneously. Each transition in one of the waveforms corresponds to a state change (movement) in the motor. Altogether, Figure 4a) and b) show eight different states for half-stepping. A step by step description of how these particular waveforms work together to move the motor shaft follows.

When coil signal A is positive and coil signal B is zero, current flows into coil A through terminal a and out of terminal $\overline{a}$. This generates a north-pole electromagnetic field toward the magnetic disk, which repels the nearest north-pole section on the disk and attracts the nearest south-pole section. These forces cause the motor to rotate in a direction that will align opposite poles. Coil B is not energized.

NOTE

The orientation of the rotor prior to energizing a single coil may be unknown. It is possible that, for example, the rotor could be positioned, as shown in Figure 7c), when attempting to align itself, as in Figure 7a). Figure 7c) is the worst case starting position for the desired alignment, shown in Figure 7a). It is even possible that initially the rotor may not turn because the magnetic forces of the coil could be equally divided over pushing and pulling the north and south pole of the PM disk. If this happens, then moving to the next sequential step by energizing both coils should help jolt the rotor free.
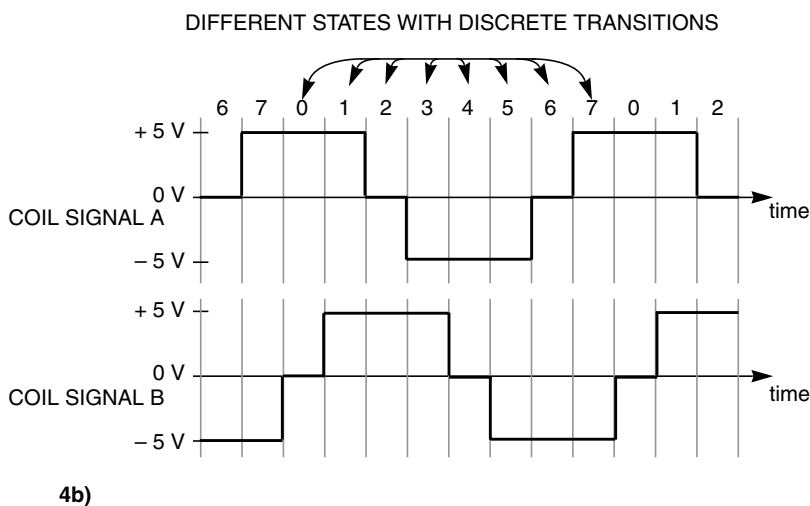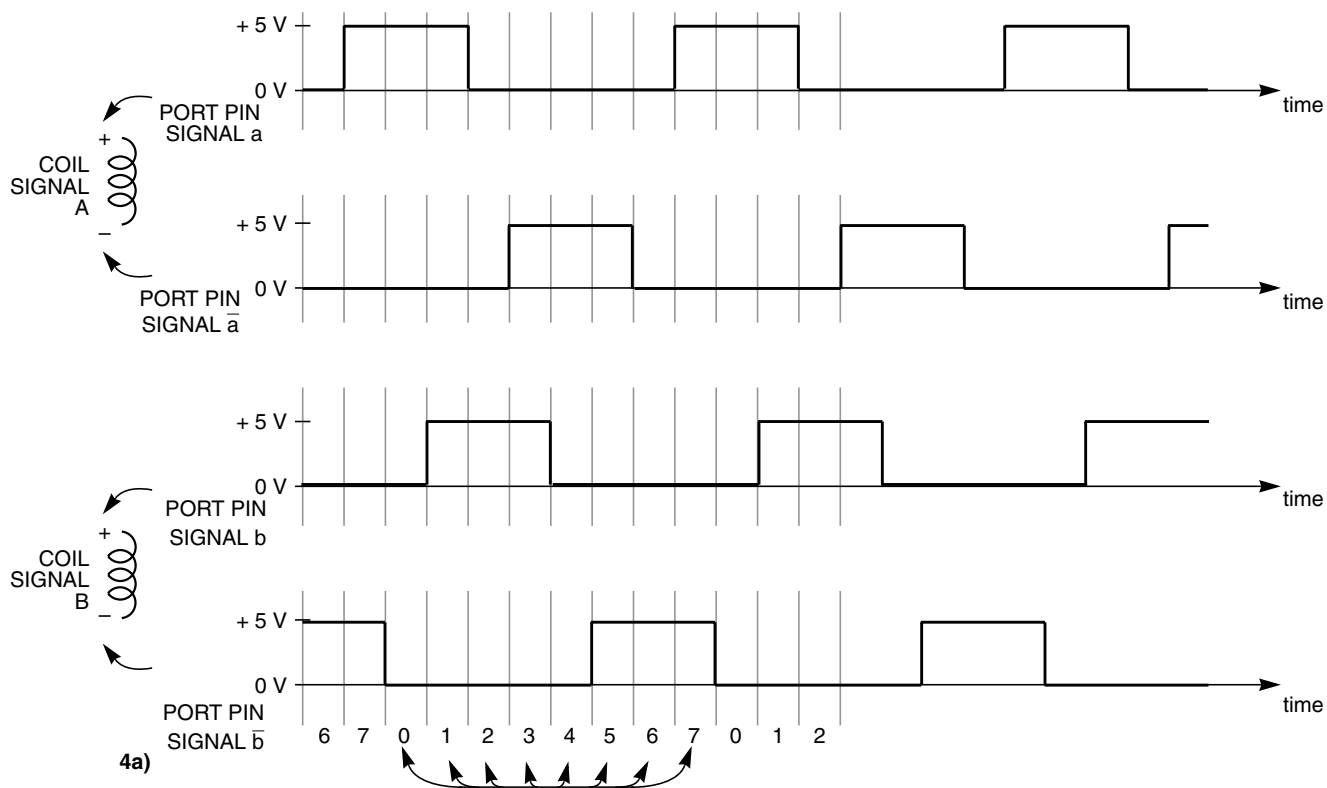
Figure 4. Discrete Transitions

While coil signal A is positively energized, the next transition occurs in coil signal B. Coil signal B rises and positively energizes coil B, creating its own electromagnetic field. Electric current flows into terminal b and out of terminal $\overline{b}$. The north-pole of both coils now share an attraction for the south-pole of the disk, causing the disk to realign (rotate) itself between shared attractions. The same action takes place with the south-pole of the coils and north-pole of the PM disk.

For the next transition, coil signal A falls to zero, leaving the signal in coil B to dominate the alignment of the PM disk.

In summary, coils A and B take turns controlling the PM disk. Before one coil releases full control of the disk, it shares control of the disk with the other coil. This temporary sharing creates a half-step in the transition of control from one coil to the next (half-stepping) and allows smaller, discrete turns to be taken by the motor. Although stepper motors are often used for their ability to make discrete movements, they can also be used for smooth movements. In an ideal case, the waveforms that would allow the smallest incremental change would actually be sinusoidal to ensure the smoothest transition between full steps. In such a case, the distinction between states and specific steps become blurred. This implementation may be well suited for applications that seek to reduce or eliminate the discrete movement of the motor, which also reduces noise and vibration. This technique is often referred to as microstepping. Although the digital waveforms in this example are not sinusoidal, their similarities to a sinusoidal waveform can still be noted by comparing Figure 4 and Figure 5. A series of electromagnet changes over the period of both signals continue to work together in this fashion to rotate the PM disk.
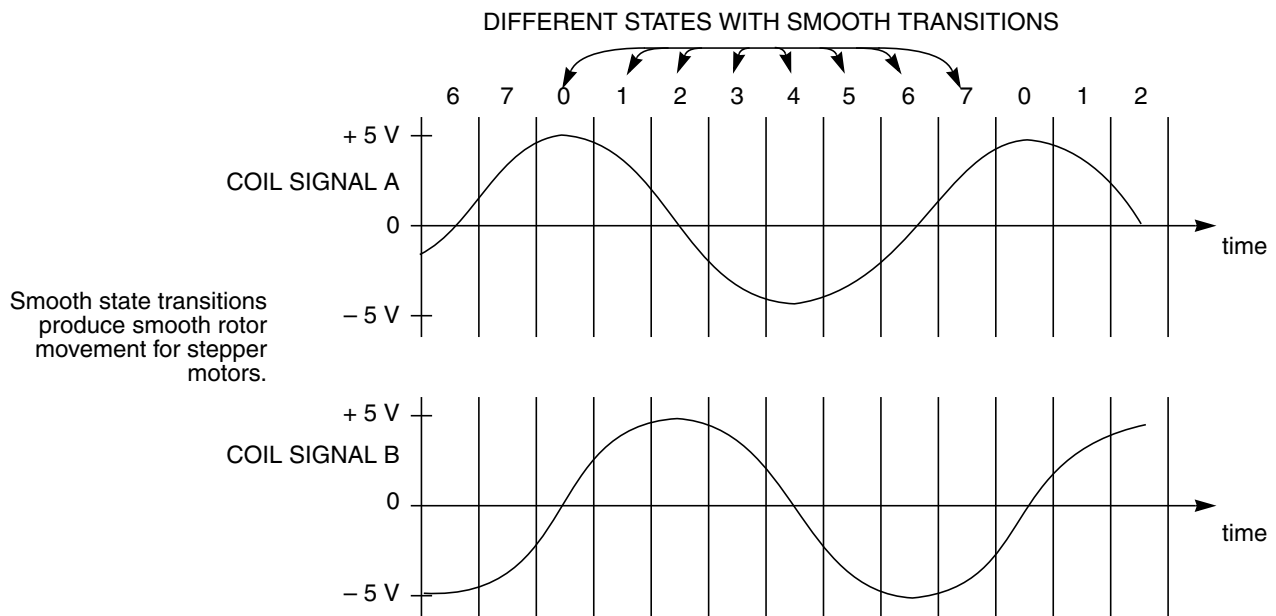

Figure 5. Smooth Transitions

How to Use an HCS12 Microcontroller to Drive the PM Stepper Motor

HCS12 microcontrollers are good devices for driving stepper motors because they are fast, compatible with the discrete movements of steppers, and can be easily programmed to work with steppers of different types. Some examples of use are precision movements, multi-axis control, sophisticated velocity profiling, and increased fault tolerance. In some instances, a microcontroller can provide multiple solutions in a single system because of their ability to be programmed to communicate with other systems while controlling a stepper motor. This is especially advantageous over a dedicated stepper driver that is more difficult to modify and not likely to have full communication capabilities. Microcontrollers can also generate the waveforms needed to produce movement in a stepper motor. Because the desired performance of a stepper motor may vary, the algorithm used by a microcontroller to drive a stepper motor is likely to vary as well. Some of these algorithms can become involved and require intimate understanding of the motor, in addition to very organized use of the microcontroller resources. To soften the approach for beginners, this section gives a general description of how to use the port pins on an HCS12 microcontroller to create basic, step-like movement in a PM stepper motor. To proceed, some general assumptions about the motor and microcontroller have to be made.

The stepper motor is assumed to be a 4-pin, two-phase PM stepper motor with two poles on the PM disk. An internal diagram of what such a motor might look like is shown in Figure 2. The input voltage of the motor is assumed to be about ±5 V, with a typical current somewhere between 1–20 milliampere. A motor of this size could weigh a few ounces and be 3–5 centimeters wide. This is one of the simpler types of motors and will be the subject of example for the remainder of the application note.

To control the four pins of the motor, the microcontroller needs four output pins capable of driving and sinking somewhere between 1–20 milliampere out of each pin. Port pins on an HCS12 microcontroller are suitable for this effort.

Most microcontrollers have registers that can be used to control logic levels of an I/O or port pin. We can select four control bits from any HCS12 I/O register that is available. Let it be assumed that there is a register called register U, and the port corresponding to this register is called port U. For simplicity, we can use the lower nibble of register U, U[3:0], to control port U pins U0, U1, U2, and U3 of the microcontroller. Pins U3 and U2 can be used to control the current in coil B, and pins U1 and U0 can be used to control the current in coil A. A connection should be made from pins U3 and U2 to the contacts of coil B. A connection should also be made from pins U1 and U0 to the contacts of coil A. Current that flows out of the U3 pin will flow into U2, and vice versa. The same condition applies to pins U1 and U0.
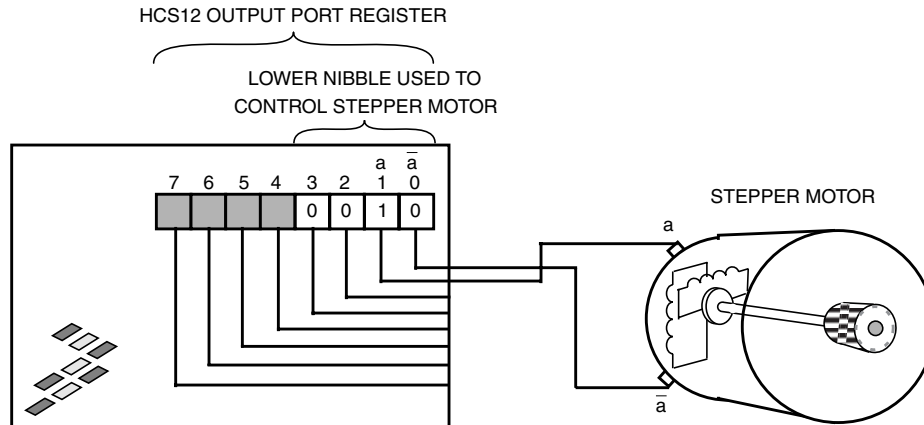
Figure 6. Using an HCS12 MCU to Control the Stepper Motor

With an appropriate algorithm, we can use pins U[3:0] of the HCS12 to produce the waveforms needed to drive a stepper motor. The general flow of the algorithm can be similar to the flow of a state machine, which is to set the bits in register U to a particular state or configuration, wait a discrete amount of time, and set the bits in register U to the next state. For each change in the microcontroller register state, a change is produced in the waveform that causes the motor to rotate a fixed amount. The period of time required between register states will vary depending upon the motor and the performance desired, but is usually on the order of milliseconds. If the delay between changes to the microcontroller register states is too short, the motor will not physically be able to move fast enough to keep up with the register state changes. A delay that is too long could create a motor response with noticeably rigid movements and choppy noises with each step. However, for the purpose of this application note, it may be helpful to have a long delay between register states because it allows easy observation of the motor response and movement due to microcontroller register changes.

An easy way to begin driving the motor is to focus on getting the motor to move a single step at a time, in the direction desired instead of many steps at once. Tracing through an algorithm with a software debugger, if a debugger is available, is a way of slowing the algorithm down so the response of the motor can be observed. After motor movement has been achieved, direction reversal can be accomplished by switching the microcontroller connections to one of the motor coils.

Figure 7 illustrates example microcontroller register contents from state 0 to state 3, It also shows the matching PM stepper motor configuration that might occur in that state. Figure 7 also corresponds with the graph in Figure 4 and the drawing in Figure 6.
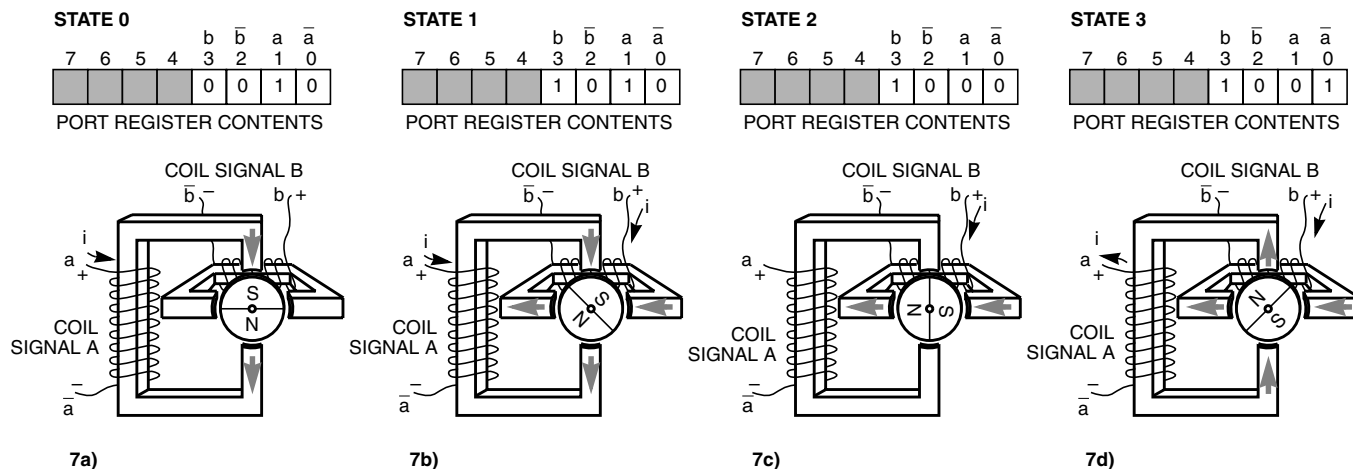
Figure 7. HCS12 MCU Register Contents from State 0 to State 3

Below is an example of a program that performs half-stepping and can be used to drive a stepper motor. The code turns the motor a number of steps (100 half-steps) in one direction, and then turns the motor back the same number of steps in the opposite direction. One of the advantages of the code below is that it can be easily modified to keep track of a motor's position. It also has the advantage of having the port states stored in sequential order in an array. Simply cycling through the states sequentially and placing the state values on port pins will cause a stepper motor to move. This is written in C.

```
#define NUM_OF_STATES 8 //There are 8 different states in this particular example.
#define DELAY_MAX 2000  //The maximum # of counts used to create a time delay.
void main(void)
{
/*******************CREATE VARIABLES*******************/
int i; //Used in a for loop

//This array actually contains the state values that will be placed on Port U.
//State #0 corresponds to a value of 0x06, state #1 corresponds to a value of 0x02, etc.
char state_array[NUM_OF_STATES] = {0x06, 0x02, 0x0A, 0x08, 0x09, 0x01, 0x05, 0x04};

int steps_to_move; //The # of rotational steps the motor will make.
char next_state;   //Used to select the next state to put in register U.
/*******************SET UP PORT U*******************/
DDRU = 0xFF;            //Writing 0xFF to DDRU sets all bits of Port U to act as output.

PTU = 0;                //Init Port U by writing a value of zero to Port U.
/*****************************************************/
steps_to_move = 100;  //Set the # of steps to move. An arbitrary positive # can be used.

next_state = 0;       //Init next_state to state 0. next_state can start from any state
                      //within the range of possible states in this example, 0-7.

PTU = state_array[next_state]; //Init Port U to the starting state. In this example,
                               //since only 4 pins are needed to control the motor, only
                               //the lower nibble of Port U is being used. This line
                               //selects state 0 and places the corresponding value
                               //(0x06) in the lower nibble of Port U.
```

```
for(i = 0; i < DELAY_MAX; i++)
{
  //Wait here for a while.
}
while (steps_to_move > 0)
{
  if (next_state  > (NUM_OF_STATES - 1))  //If next_state is greater than the highest
                                          //available state, 7, then cycle back to 0
  {
    next_state = 0;
  }
  PTU = state_array[next_state]; //Place new value in Port U. Rotation may be observed
  for(i = 0; i < DELAY_MAX; i++)
  {
    //Wait here for a while.
  }
  next_state++;        //Increment next_state. Cycling though the states causes rotation
                       //in one direction. Decrementing states causes opposite rotation.

  steps_to_move--;   //Subtract 1 from the total # of steps remaining to be moved.
}

//The following code rotates the motor back in the opposite direction.
steps_to_move = 100;
while (steps_to_move  > 0)
{
  if (next_state  < 0)
  {
    next_state = (NUM_OF_STATES - 1);
  }
  PTU = state_array[next_state];
  for(i = 0; i < DELAY_MAX; i++)
  {
    //Wait here for a while.
  next_state--;  }
  steps_to_move--;
}
} //End of Main
```

How are Stepper Motors Used?

Stepper motors have found their way into many different areas of control systems. The wide popularity of these motors can be attributed in part to the various ways the motor can be driven and because of its compatibility with digital systems. In particular, stepper motors are ideal for control systems that require discrete, easily repeatable movements at moderate to low frequencies. Steppers are most commonly used in open-loop position control applications. Figure 8 below shows an example block diagram of a system with microcontroller, stepper motor, and feedback. In the case of stepper motors, the feedback is not always needed but can still be provided for precision assistance. In contrast, DC motors need feedback because they have a harder time making precision movements and require a circuit that can compensate for the risk of drifting or overshooting a target position. The feedback circuitry for the position of a motor is likely to be more complicated for dc motors than for stepper motors. Stepper motors have worked well in factories and assembly environments, in applications such as robotic arms and precision assembly controls. They can be found in printers, disk drives, toys, cars, and a host of other applications and products.
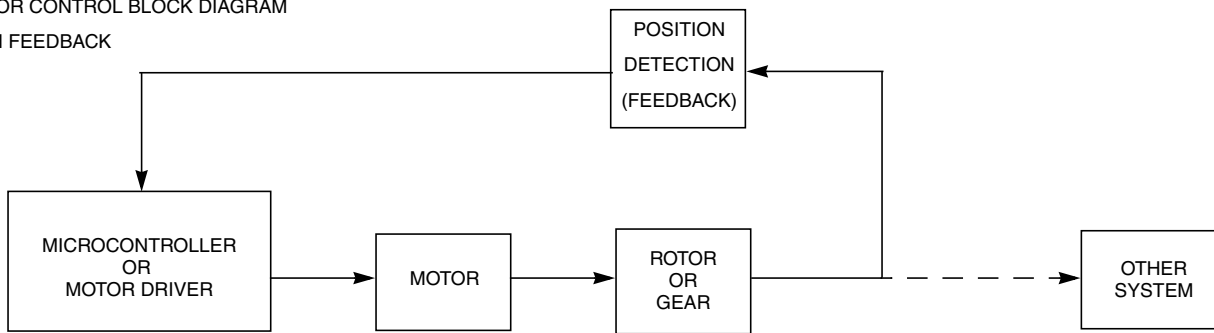
Figure 8. Example System with an MCU, Stepper Motor, and Feedback

Efficient Motor Control with an HCS12 Microcontroller

Actual control of a stepper motor in real applications is not trivial. Often, the motor is a single component within a system of other devices that must all work in unison for successful operation. A microcontroller responsible for driving the motor can also handle other tasks or service other devices within the system, but writing linear software to handle complex motor control can leave little bandwidth for the microcontroller to tend to other matters. In the simple example code given, the microcontroller wastes much of its computing power stuck in a delay loop before performing any other meaningful task. More efficient use of the microcontroller can be obtained by using an HCS12 with a motor control module. The interrupt capability of the motor control module allows the microcontroller to run sequentially through software until the motor needs to be serviced. After a motor interrupt occurs, software can make quick register adjustments to characteristics like polarity, period length, and duty cycle, before returning to normal flow. For more details about applications like motor control and HCS12 microcontrollers, refer to http://www.freescale.com.

This page intentionally left blank.

This page intentionally left blank.

This page intentionally left blank.

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com