A simple algorithm for microstepping a bipolar stepper motor

Jose Quinones, Texas Instruments - July 11, 2011

Here's a simple algorithm that uses conventional microcontroller blocks to control commercially available H-bridges to properly commutate a bipolar stepper motor through a microstepping profile.

Bipolar stepper motors offer a simple way of achieving position control and accurate speed actuation without the need to close the loop through shaft encoders or similar means. To improve performance, we can employ a technique known as microstepping in which a sine wave current wave shape is embedded into the typical full-step commutation wave form.

This article details a simple algorithm utilizing conventional microcontroller blocks to control commercially available H-bridges to properly commutate a bipolar stepper motor through a microstepping profile.

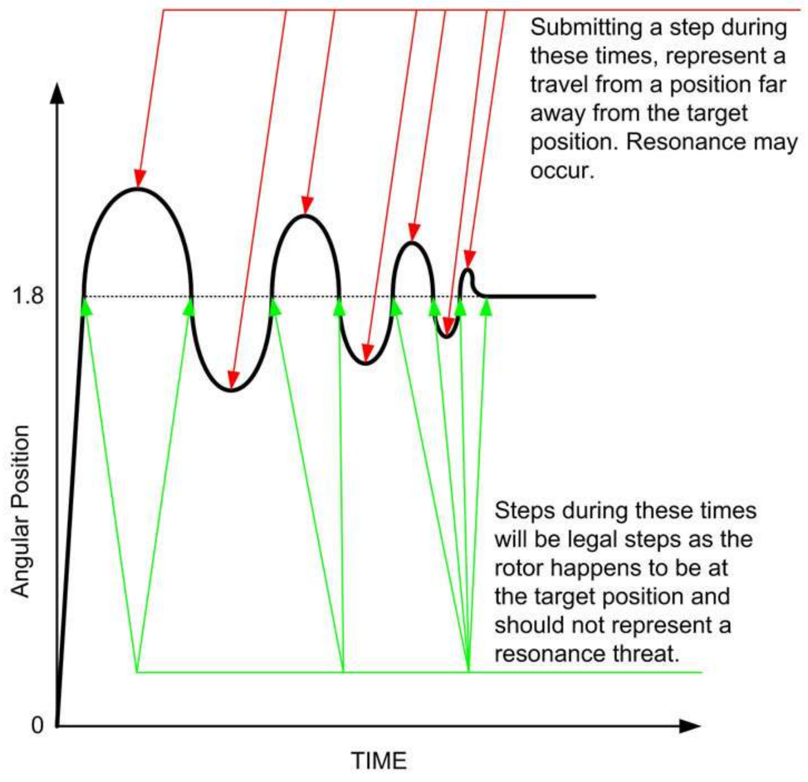
Stepper motors are an excellent motion actuator because they move in steps. This gives us two inherent advantages: 1) position can be easily obtained and maintained by moving a number of steps and then stopping; and 2) an accurate speed can be obtained by properly scheduling the steps in a timely manner.

As a result, steppers can stop at a given angular position and hold that position against external load changes, and at the same time the motor speed can be maintained even when the system undergoes changes in power supply voltage. Whereas other motor topologies could not achieve any of these two feats without the proper amount of closed loop control, the stepper excels at both without the need for any form of closed loop.

However, stepper motors are not perfect, and there are areas in which their performance is severely affected. The most crucial of these inefficacies is resonance, or a vibration induced by the generation of subsequent steps at a time in which further motion is exacerbated. **Figure 1** below illustrates what happens to the angular position as a full-step is generated. As the rotor is scheduled to land on the next step, 1.8 degrees away from the current position, it actually oscillates around this angular region before settling at the target.

What if we scheduled a step when the position is farther away or closer to the next step position? When this happens, the distance traveled by the rotor will be much more or much less than what it would have been had the rotor started from the goal position.

Actuation at these speeds is what causes the motor to vibrate and loose torque. It is very easy to see where the vibration-inducing speeds lie on a particular motor, if you slowly accelerate the motor from a slow speed to a higher speed. You will notice the regions where vibrations increase and decrease as the speed is ramped up.



Click on image to enlarge.

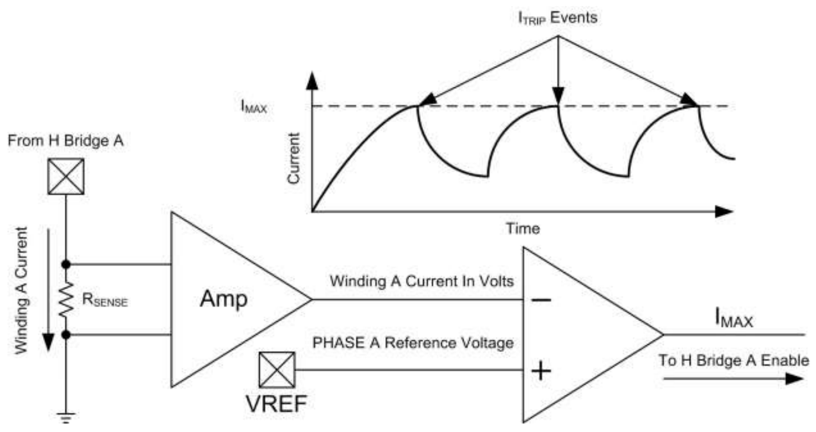
Figure 1. A commanded full-step and its angular position oscillations before settling. If a step is issued in a spot in time such that the position is too far off, resonance effects can be observed.

Both vibration and loss of torque are highly undesirable traits for any motor actuator. Therefore, when operating a stepper motor, it is crucial to eliminate both scenarios from the design. One option is to limit the current to such an amount that it reduces the vibrations considerably. Unfortunately, if this current is not dynamically modulated with load changes, the system suffers from step loss, an even worse threat, as the speed and position accuracy are heavily compromised.

A better solution is to eliminate the vibrations by decreasing the distance the rotor must cover on a step-by-step basis. Motors are built to accommodate a step resolution. A 200-step motor moves 1.8 degrees per generated step. If somehow we can divide each step into several microsteps, then the distance traveled is less than 1.8 degrees. With smaller step motions, we need less energy to reach the target position and the vibrations should be minimized. **Page 2**

Current regulation

To induce multiple microsteps embedded into a single full-step, we must have the capability to regulate current. The great majority of integrated H-bridges commercially available have some means to achieve this goal. Current regulation is then easily obtained by measuring current flowing through a SENSE resistor as shown in **Figure 2**.



Click on image to enlarge.

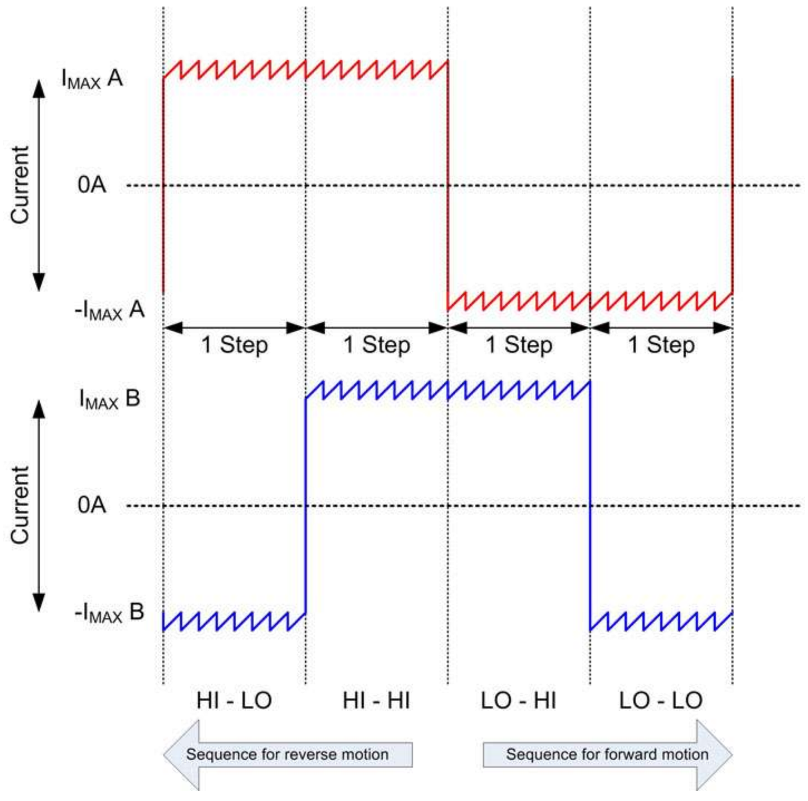
Figure 2. A SENSE resistor in series with the motor winding gives off the dropped voltage, which is directly proportional to the winding current.

The SENSE resistor is in series with the motor winding, so we are measuring the actual winding current. The voltage drop across this resistor is amplified by some known gain. We need to amplify the voltage, as the resistor is fairly small, in order to minimize losses. The amplified version is then compared against a reference voltage (VREF). When the winding current gives off a voltage such that it is larger than the supplied VREF, the H-bridge is disabled for a fixed amount of time. After this time elapses, the H-bridge is enabled again. The process to disable the H-bridge as current reaches an ITRIP target, is repeated ad infinitum, giving us a regulated current.

This reference voltage is provided by the application. If VREF is modulated, the winding current is then also modulated. This is how we achieve microstepping. If the current magnitude changes, so will the stator magnetic field change. By controlling the winding current, we can then control the stator magnetic field strength, which in turn controls the rotor position.

For example, if a 200 step stepper is commutated with full current, then each step is 1.8 degrees. But if the same motor is commutated with full current and half current, then each step is 0.9 degrees. We can keep subdividing the current by as much as we want and we will always obtain even smaller step sizes or larger resolutions.

Microstepping commutation
Bipolar stepper motors are often commutated with full-steps by coordinating each winding phase current in one of the four possible pattern combinations, as shown in **Figure 3**. These combinations are: HI-LO, HI-HI, LO-HI and LO-LO, where HI implies current is regulated to IMAX and LO implies current is regulated to –IMAX. If we follow this sequence, the motor moves in one direction. If we now reverse the same sequence, the motor moves in the opposing direction.



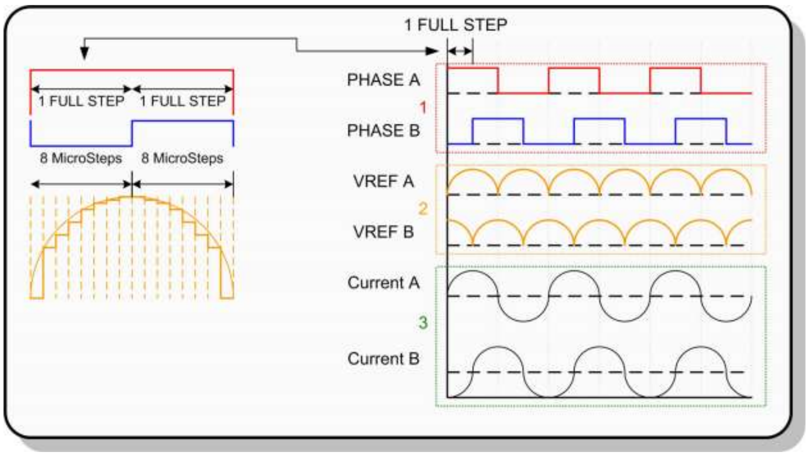
Click on image to enlarge.

Figure 3. The typical quadrature pattern used to full-step commutate a bipolar stepper motor.

Rotor position is controlled by counting the number of steps being issued with regards to a known starting point. To set up the stepping speed, you would then coordinate the amount of time in between steps. Speed is then the inverse of this time with the unit of measurement being the step per second (SPS).

A way of generating these steps is to use an internal timer resource configured to count time intervals. Its interrupt service routine (ISR) can then be used to update the phases according to the desired direction of rotation. For example, if the current step is at quadrature position HI-HI, you would issue a commutation polarity of LO-HI to move the motor one step forward, or HI-LO to move it backwards.

By adding the VREF magnitude modulation to the H-bridge circuitry, we add a current component on top of the full-step commutation, which gives us microstepping commutation. **Figure 4** shows this mechanism.



Click on image to enlarge.

Figure 4. VREF modulation information is embedded on top of the full-step commutation, resulting in microstepping commutation being applied to the bipolar stepper motor windings.

Notice the PHASE information is always a positive value. However, when PHASE is HI, the motor winding current is positive, whereas when PHASE is LO, motor current is negative. Hence, the digital signal PHASE (also called DIRECTION on some H-bridges) gives us current direction, not current magnitude. Current magnitude is obtained by modulating VREF.

In the following example, we embedded half of a sine wave shape into the VREF terminal. In essence, you can use any continuous wave shape as long as it offers soft motion. Sine waves are industry standard, but they are not a requirement. Designers are encouraged to try any other wave shape that can offer good results. This level of flexibility is one of the main advantages of using a microcontroller or digital signal processor (DSP) to achieve microstepping.

The result of embedding VREF information on top of phase information is an AC signal, in this case a sine wave used to commutate each stepper motor winding. So how do we generate the VREF information?

To generate VREF we use some form of DAC module. Since bipolar stepper motors are made of two windings, two DAC channels are required. If a real DAC is not available, a high-speed pulse-width modulation (PWM) output with a low-pass filter can be used to create a crude, but equally useful, programmable analog voltage. The analog magnitudes into this analog output come from an internal lookup table storing the wave shape we have determined to be appropriate for our application. Every time a step is issued, said value is fetched from the lookup table and into the DAC register.

There are a few key notes you must have in mind when it comes to generating and using this lookup table. First is the lookup table depth. This table needs to hold as many elements as twice the number of current settings. That is, if you want to divide each full-step eight times (eight microsteps), you need eight current settings, and the table will be 16 steps wide. This takes care of 180 degrees worth of information. The other 180 degrees comes from reusing the table in its entirety, but for the alternate polarity. That is, we use the table for positive current and then the same table for negative current.

Page 3

In actuality, to control any bipolar stepper we need two signals as there are two phases: PHASE A and PHASE B. Luckily, if PHASE A is a sine wave, PHASE B is the same sine wave, but with 90 degrees worth of phasing. In other words, if PHASE A is a sine, PHASE B is the cosine. It may seem like we need two tables, but the very same table can be reused.

As the 16-step deep table is used to generate the PHASE A sine wave, we can offset the lookup table index by eight and fetch the values that generate the cosine wave shape. For example, it looks like this in pseudocode:

```
#define TABLE_DEPTH     16
VREF_PHASEA = LOOKUPTABLE[INDEX]
VREF_PHASEB = LOOKUPTABLE[(INDEX + TABLE_DEPTH / 2) & (TABLE_DEPTH-1)]
Increase INDEX
```

Notice that for the PHASE B index we must normalize it to the table size. If the table value being fetched for PHASE A is the element #15, we do not want to fetch element #23 for PHASE B, as this would be outside of the table space. We want to fetch element #7. Anding the lookup table index by the table size minus one results in the correct normalized value. I often refer to this anding element as the INDEX_MASK.

We have discussed how to fetch the current magnitude information from the lookup table, but what about the phase information? Should we store this in the table as well? If so, then the table space needs to be four times as large as the number of current settings we want per step, right? Actually, this is not necessary as the PHASE information can be derived from the INDEX variable itself.

The INDEX value goes from 0 to the TABLE_DEPTH – 1 value. But, if we let the INDEX go from 0 to twice the TABLE_DEPTH, we can then use the resulting number's most significant bit information as the PHASE polarity. Take for example our eight degrees of microstepping scenario. In this case, TABLE_DEPTH is equal to 16, but we will let index run from zero to 31. This changes the index masking strategy as we need a mask for the phase information (PHASE_MASK) and another mask for the lookup table fetching procedure (INDEX_MASK). The pseudo code now looks like this:

```
#define         PHASE_MASK              0x10
PHASEA = INDEX & PHASE_MASK
PHASEB = (INDEX + TABLE_DEPTH / 2) & PHASE_MASK
```

Notice that we don't care about wrap up on the INDEX value when deriving PHASE information as the meaningful most significant bit (MSB) will always be toggling. In other words, bits above the MSB are completely ignored, whereas bits below the MSB are used as lookup table index to extract current magnitude information.

```
#define         TABLE_DEPTH     16
#define         INDEX_MASK      TABLE_DEPTH - 1
VREF_PHASEA = LOOKUPTABLE[INDEX & INDEX_MASK]
VREF_PHASEB = LOOKUPTABLE[(INDEX + TABLE_DEPTH / 2) & (INDEX_MASK)]
INDEX = INDEX + IndexIncrement
```

We changed the way in which the index is incremented. The truth is there are times when we will not want to increment the index, but actually want to decrement it. The reason for this is that by walking through the lookup table we can also derive direction of rotation information. In other words, if we walk forward through the table, then the motor moves clockwise, but if we walk backwards through the table, the motor then moves counterclockwise.

Here we describe the logistics behind the code we will implement to use the lookup table. A hardware ISR is used to acknowledge the step command. It can be a timer input capture or a general purpose input output (GPIO) configured to interrupt. You must choose whether you want the step command to be recognized on a rising edge, falling edge, or both. Every time such a transition is registered, the code specified above will be executed.

A second ISR takes care of the rotation direction request. Again, it can be any form of hardware input which grants an interrupt on a transition, except in this case it must react to both rising and falling edges. If a rising edge is registered, the index increment is set to +1. If the falling edge is registered, then the index increment is set to –1.

As you can see, the level of simplicity to turn your stepper driver into a microstepping commutator is considerably simple, with the amount of code executed per microstep being fairly small.

Simple fix

Using a microcontroller or a DSP to embed microstepping information on the conventional full-step commutation algorithm is a simple mechanism to fix the resonance problem. Because microstepping is achieved with software rather than hardware, the levels of flexibility are appealing to this kind of implementation. You can use any wave shape the design requires. Degrees of microstepping can grow from small resolution to very high resolution, with up to thousands degrees of microstepping being used on applications requiring a very soft motion profile.

Jose Quinones *is a motor control applications engineer at Texas Instruments where he is involved in aiding developers in implementing design based on power devices for implementing microstepping such as the DRV8812/8813 and DRV8824/25. He received a B.S. in electrical engineering from the University of Puerto Rico and spent five years developing motion control embedded systems for Xerox Corporation (for which he has two patents) before joining TI. Jose can be reached at ti_josequinones@list.ti.com.*