# AN857

# Brushless DC Motor Control Made Easy

Author: Ward Brown
Microchip Technology Inc.

## INTRODUCTION

This application note discusses the steps of developing several controllers for brushless motors. We cover sensored, sensorless, open loop, and closed loop design. There is even a controller with independent voltage and speed controls so you can discover your motor's characteristics empirically.

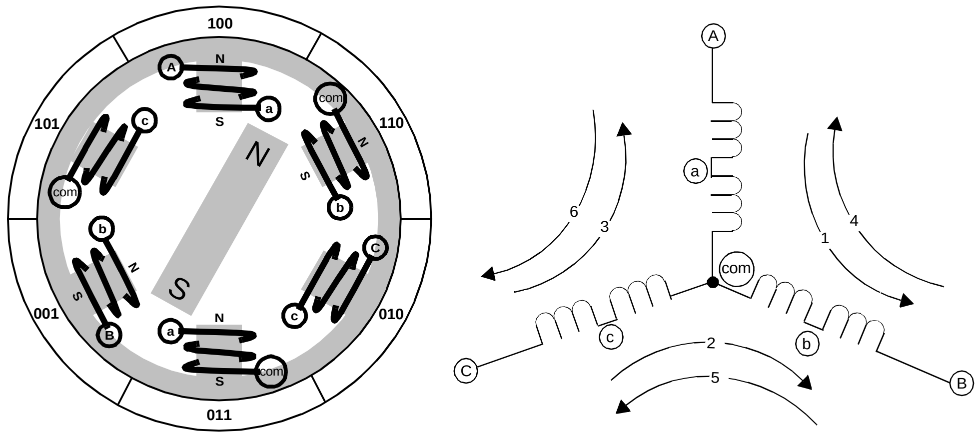
The code in this application note was developed with the Microchip PIC16F877 PICmicro® Microcontroller, in conjuction with the In-Circuit Debugger (ICD). This combination was chosen because the ICD is inexpensive, and code can be debugged in the prototype hardware without need for an extra programmer or emulator. As the design develops, we program the target device and exercise the code directly from the MPLAB® environment. The final code can then be ported to one of the smaller, less expensive, PICmicro microcontrollers. The porting takes minimal effort because the instruction set is identical for all PICmicro 14-bit core devices.

It should also be noted that the code was bench tested and optimized for a Pittman N2311A011 brushless DC motor. Other motors were also tested to assure that the code was generally useful.

### Anatomy of a BLDC

Figure 1 is a simplified illustration of BLDC motor construction. A brushless motor is constructed with a permanent magnet rotor and wire wound stator poles. Electrical energy is converted to mechanical energy by the magnetic attractive forces between the permanent magnet rotor and a rotating magnetic field induced in the wound stator poles.

**FIGURE 1: SIMPLIFIED BLDC MOTOR DIAGRAMS**



In this example there are three electromagnetic circuits connected at a common point. Each electromagnetic circuit is split in the center, thereby permitting the permanent magnet rotor to move in the middle of the induced magnetic field. Most BLDC motors have a three-phase winding topology with star connection. A motor with this topology is driven by energizing 2 phases at a time. The static alignment shown in Figure 2, is that which would be realized by creating an electric current flow from terminal A to B, noted as path 1 on the schematic in Figure 1. The rotor can be made to rotate clockwise 60 degrees from the A to B alignment by changing the current path to flow from terminal C to B, noted as path 2 on the schematic. The suggested magnetic alignment is used only for illustration purposes because it is easy to visualize. In practice, maximum torque is obtained when the permanent magnet rotor is 90 degrees away from alignment with the stator magnetic field.

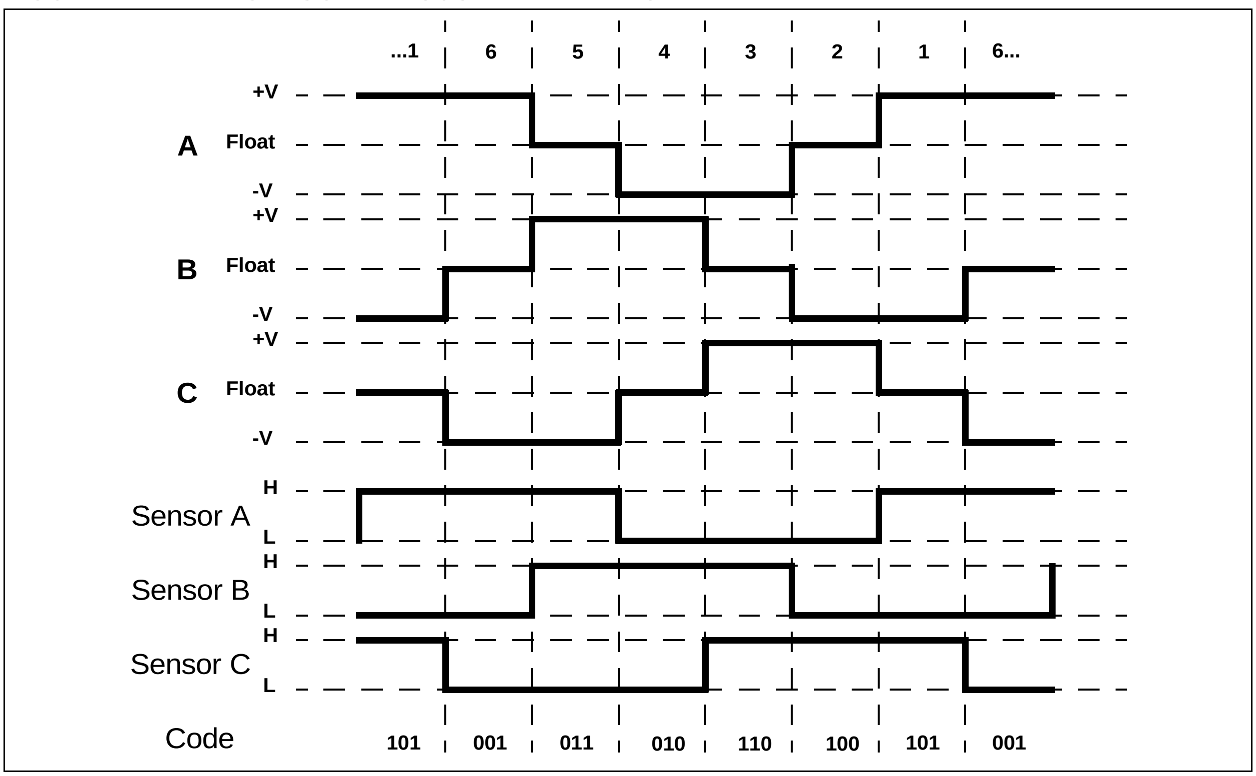
The key to BLDC commutation is to sense the rotor position, then energize the phases that will produce the most amount of torque. The rotor travels 60 electrical degrees per commutation step. The appropriate stator current path is activated when the rotor is 120 degrees from alignment with the corresponding stator magnetic field, and then deactivated when the rotor is 60 degrees from alignment, at which time the next circuit is activated and the process repeats. Commutation for the rotor position, shown in Figure 1, would be at the completion of current path 2 and the beginning of current path 3 for clockwise rotation. Commutating the electrical connections through the six possible combinations, numbered 1 through 6, at precisely the right moments will pull the rotor through one electrical revolution.

In the simplified motor of Figure 1, one electrical revolution is the same as one mechanical revolution. In actual practice, BLDC motors have more than one of the electrical circuits shown, wired in parallel to each other, and a corresponding multi-pole permanent magnetic rotor. For two circuits there are two electrical revolutions per mechanical revolution, so for a two circuit motor, each electrical commutation phase would cover 30 degrees of mechanical rotation.

## Sensored Commutation

The easiest way to know the correct moment to commutate the winding currents is by means of a position sensor. Many BLDC motor manufacturers supply motors with a three-element Hall effect position sensor. Each sensor element outputs a digital high level for 180 electrical degrees of electrical rotation, and a low level for the other 180 electrical degrees. The three sensors are offset from each other by 60 electrical degrees so that each sensor output is in alignment with one of the electromagnetic circuits. A timing diagram showing the relationship between the sensor outputs and the required motor drive voltages is shown in Figure 2.

**FIGURE 2: SENSOR VERSUS DRIVE TIMING**



The numbers at the top of Figure 2 correspond to the current phases shown in Figure 1. It is apparent from Figure 2 that the three sensor outputs overlap in such a way as to create six unique three-bit codes corresponding to each of the drive phases. The numbers shown around the peripheral of the motor diagram in Figure 1 represent the sensor position code. The north pole of the rotor points to the code that is output at that rotor position. The numbers are the sensor logic levels where the Most Significant bit is sensor C and the Least Significant bit is sensor A.

Each drive phase consists of one motor terminal driven high, one motor terminal driven low, and one motor terminal left floating. A simplified drive circuit is shown in Figure 3. Individual drive controls for the high and low drivers permit high drive, low drive, and floating drive at each motor terminal. One precaution that must be taken with this type of driver circuit is that both high side and low side drivers must never be activated at the same time. Pull-up and pull-down resistors must be placed at the driver inputs to ensure that the drivers are off immediately after a microcontoller RESET, when the microcontroller outputs are configured as high impedance inputs.

Another precaution against both drivers being active at the same time is called dead time control. When an output transitions from the high drive state to the low drive state, the proper amount of time for the high side driver to turn off must be allowed to elapse before the low side driver is activated. Drivers take more time to turn off than to turn on, so extra time must be allowed to elapse so that both drivers are not conducting at the same time. Notice in Figure 3 that the high drive period and low drive period of each output, is separated by a floating drive phase period. This dead time is inherent to the three phase BLDC drive scenario, so special timing for dead time control is not necessary. The BLDC commutation sequence will never switch the high-side device and the low-side device in a phase, at the same time.

At this point we are ready to start building the motor commutation control code. Commutation consists of linking the input sensor state with the corresponding drive state. This is best accomplished with a state table and a table offset pointer. The sensor inputs will form the table offset pointer, and the list of possible output drive codes will form the state table. Code development will be performed with a PIC16F877 in an ICD. I have arbitrarily assigned PORTC as the motor drive port and PORTE as the sensor input port. PORTC was chosen as the driver port because the ICD demo board also has LED indicators on that port so we can watch the slow speed commutation drive signals without any external test equipment.

Each driver requires two pins, one for high drive and one for low drive, so six pins of PORTC will be used to control the six motor drive MOSFETS. Each sensor requires one pin, so three pins of PORTE will be used to read the current state of the motor's three-output sensor. The sensor state will be linked to the drive state by using the sensor input code as a binary offset to the drive table index. The sensor states and motor drive states from Figure 2 are tabulated in Table 1.

**FIGURE 3: THREE PHASE BRIDGE**

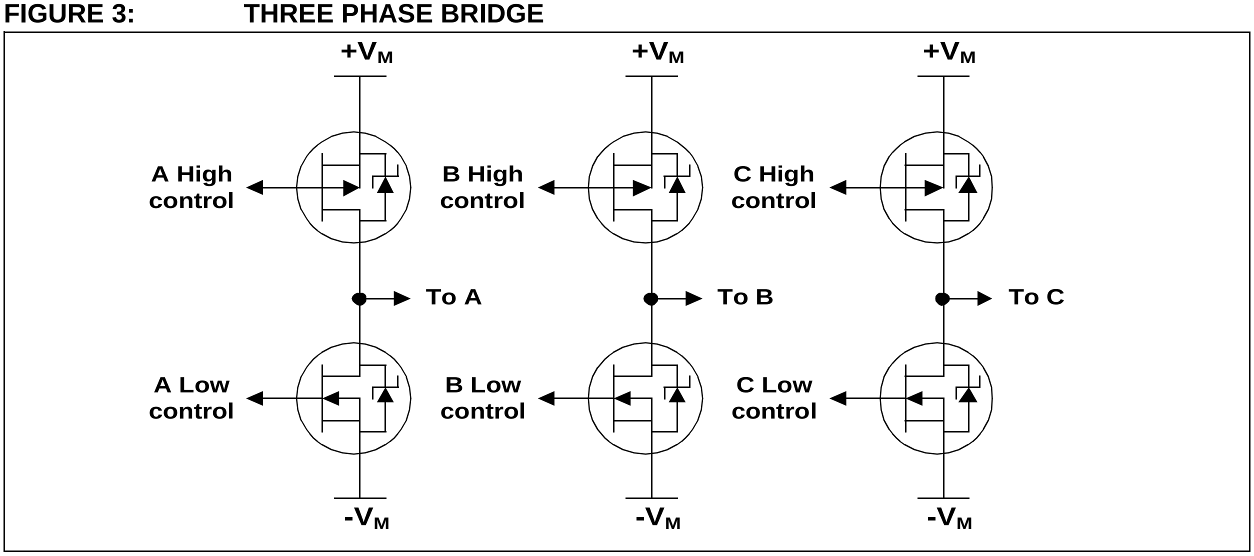


**TABLE 1: CW SENSOR AND DRIVE BITS BY PHASE ORDER**

| Pin | RE2 | RE1 | RE0 | RC5 | RC4 | RC3 | RC2 | RC1 | RC0 |
|---|---|---|---|---|---|---|---|---|---|
| Phase | Sensor C | Sensor B | Sensor A | C High Drive | C Low Drive | B High Drive | B Low Drive | A High Drive | A Low Drive |
| 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 |
| 2 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| 3 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| 4 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| 5 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 |
| 6 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 |

Sorting Table 1 by sensor code binary weight results in Table 2. Activating the motor drivers, according to a state table built from Table 2, will cause the motor of Figure 1 to rotate clockwise.

**TABLE 2: CW SENSOR AND DRIVE BITS BY SENSOR ORDER**

| Pin | RE2 | RE1 | RE0 | RC5 | RC4 | RC3 | RC2 | RC1 | RC0 |
|---|---|---|---|---|---|---|---|---|---|
| Phase | Sensor C | Sensor B | Sensor A | C High Drive | C Low Drive | B High Drive | B Low Drive | A High Drive | A Low Drive |
| 6 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 |
| 4 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| 5 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 |
| 2 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 |
| 3 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |

Counter clockwise rotation is accomplished by driving current through the motor coils in the direction opposite of that for clockwise rotation. Table 3 was constructed by swapping all the high and low drives of Table 2. Activating the motor coils, according to a state table built from Table 3, will cause the motor to rotate counter clockwise. Phase numbers in Table 3 are preceded by a slash denoting that the EMF is opposite that of the phases in Table 2.

**TABLE 3: CCW SENSOR AND DRIVE BITS**

| Pin | RE2 | RE1 | RE0 | RC5 | RC4 | RC3 | RC2 | RC1 | RC0 |
|---|---|---|---|---|---|---|---|---|---|
| Phase | Sensor C | Sensor B | Sensor A | C High Drive | C Low Drive | B High Drive | B Low Drive | A High Drive | A Low Drive |
| /6 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 |
| /4 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| /5 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 |
| /2 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| /1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 |
| /3 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |

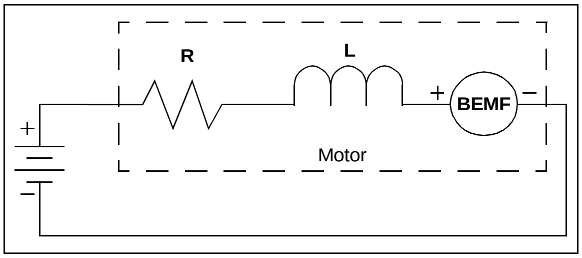
The code segment for determining the appropriate drive word from the sensor inputs is shown in Figure 4.

**FIGURE 4: COMMUTATION CODE SEGMENT**

```
#define        DrivePort              PORTC
#define        SensorMask             B'00000111'
#define        SensorPort             PORTE
#define        DirectionBit           PORTA, 1

Commutate
       movlw   SensorMask             ;retain only the sensor bits
       andwf   SensorPort             ;get sensor data
       xorwf   LastSensor, w          ;test if motion sensed
       btfsc   STATUS, Z              ;zero if no change
       return                         ;no change - return

       xorwf   LastSensor, f          ;replace last sensor data with current
       btfss   DirectionBit           ;test direction bit
       goto    FwdCom                 ;bit is zero - do forward commutation

                                      ;reverse commutation
       movlw   HIGH RevTable          ;get MS byte to table
       movwf   PCLATH                 ;prepare for computed GOTO
       movlw   LOW RevTable           ;get LS byte of table
       goto    Com2
FwdCom                                ;forward commutation
       movlw   HIGH FwdTable          ;get MS byte of table
       movwf   PCLATH                 ;prepare for computed GOTO
       movlw   LOW FwdTable           ;get LS byte of table
Com2
       addwf   LastSensor, w          ;add sensor offset
       btfsc   STATUS, C              ;page change in table?
       incf    PCLATH, f              ;yes - adjust MS byte

       call    GetDrive               ;get drive word from table
       movwf   DriveWord              ;save as current drive word
       return

GetDrive
       movwf   PCL

FwdTable
       retlw B'00000000'    ;invalid
       retlw B'00010010'    ;phase 6
       retlw B'00001001'    ;phase 4
       retlw B'00011000'    ;phase 5
       retlw B'00100100'    ;phase 2
       retlw B'00000110'    ;phase 1
       retlw B'00100001'    ;phase 3
       retlw B'00000000'    ;invalid
RevTable
       retlw B'00000000'    ;invalid
       retlw B'00100001'    ;phase /6
       retlw B'00000110'    ;phase /4
       retlw B'00100100'    ;phase /5
       retlw B'00011000'    ;phase /2
       retlw B'00001001'    ;phase /1
       retlw B'00010010'    ;phase /3
       retlw B'00000000'    ;invalid
```

Before we try the commutation code with our motor, lets consider what happens when a voltage is applied to a DC motor. A greatly simplified electrical model of a DC motor is shown in Figure 5.

**FIGURE 5: DC MOTOR EQUIVALENT CIRCUIT**




When the rotor is stationary, the only resistance to current flow is the impedance of the electromagnetic coils. The impedance is comprised of the parasitic resistance of the copper in the windings, and the parasitic inductance of the windings themselves. The resistance and inductance are very small by design, so start-up currents would be very large, if not limited.

When the motor is spinning, the permanent magnet rotor moving past the stator coils induces an electrical potential in the coils called Back Electromotive Force, or BEMF. BEMF is directly proportional to the motor speed and is determined from the motor voltage constant $K_V$.

**EQUATION 1:**

$$RPM = K_V \text{ x Volts}$$
$$BEMF = RPM / K_V$$

In an ideal motor, R and L are zero, and the motor will spin at a rate such that the BEMF exactly equals the applied voltage.

The current that a motor draws is directly proportional to the torque load on the motor shaft. Motor current is determined from the motor torque constant $K_T$.

**EQUATION 2:**

$$Torque = K_T \text{ x Amps}$$

An interesting fact about $K_T$ and $K_V$ is that their product is the same for all motors. Volts and Amps are expressed in MKS units, so if we also express $K_T$ in MKS units, that is N-M/Rad/Sec, then the product of $K_V$ and $K_T$ is 1.

**EQUATION 3:**

$$K_V * K_T = 1$$

This is not surprising when you consider that the units of the product are [1/(V*A)]*[(N*M)/(Rad/Sec)], which is the same as mechanical power divided by electrical power.

If voltage were to be applied to an ideal motor from an ideal voltage source, it would draw an infinite amount of current and accelerate instantly to the speed dictated by the applied voltage and $K_V$. Of course no motor is ideal, and the start-up current will be limited by the parasitic resistance and inductance of the motor windings, as well as the current capacity of the power source. Two detrimental effects of unlimited start-up current and voltage are excessive torque and excessive current. Excessive torque can cause gears to strip, shaft couplings to slip, and other undesirable mechanical problems. Excessive current can cause driver MOSFETS to blow out and circuitry to burn.

We can minimize the effects of excessive current and torque by limiting the applied voltage at start-up with pulse width modulation (PWM). Pulse width modulation is effective and fairly simple to do. Two things to consider with PWM are, the MOSFET losses due to switching, and the effect that the PWM rate has on the motor. Higher PWM frequencies mean higher switching losses, but too low of a PWM frequency will mean that the current to the motor will be a series of high current pulses instead of the desired average of the voltage waveform. Averaging is easier to attain at lower frequencies if the parasitic motor inductance is relatively high, but high inductance is an undesirable motor characteristic. The ideal frequency is dependent on the characteristics of your motor and power switches. For this application, the PWM frequency will be approximately 10 kHz.

We are using PWM to control start-up current, so why not use it as a speed control also? We will use the analog-to-digital converter (ADC), of the PIC16F877 to read a potentiometer and use the voltage reading as the relative speed control input. Only 8 bits of the ADC are used, so our speed control will have 256 levels. We want the relative speed to correspond to the relative potentiometer position. Motor speed is directly proportional to applied voltage, so varying the PWM duty cycle linearly from 0% to 100% will result in a linear speed control from 0% to 100% of maximum RPM. Pulse width is determined by continuously adding the ADC result to the free running Timer0 count to determine when the drivers should be on or off. If the addition results in an overflow, then the drivers are on, otherwise they are off. An 8-bit timer is used so that the ADC to timer additions need no scaling to cover the full range. To obtain a PWM frequency of 10 kHz Timer0 must be running at 256 times that rate, or 2.56 MHz. The minimum prescale value for Timer0 is 1:2, so we need an input frequency of 5.12 MHz. The input to Timer0 is FOSC/4. This requires an FOSC of 20.48 MHz. That is an odd frequency, and 20 MHz is close enough, so we will use 20 MHz resulting in a PWM frequency of 9.77 kHz.

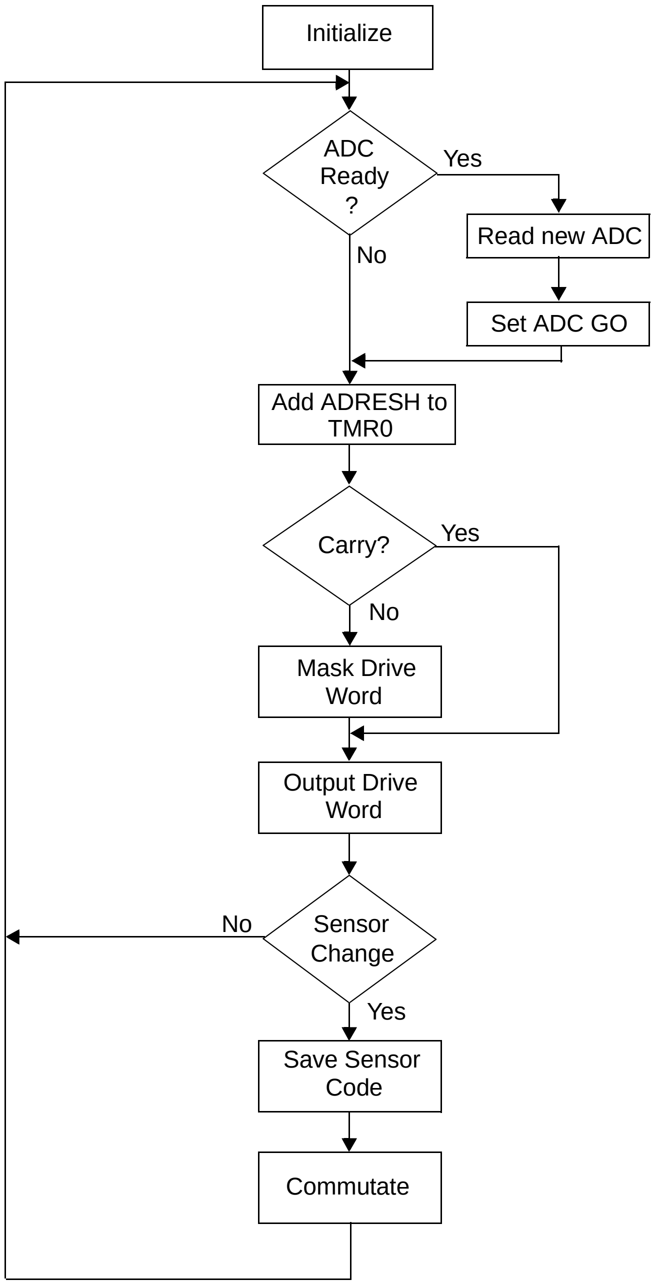
There are several ways to modulate the motor drivers. We could switch the high and low side drivers together, or just the high or low driver while leaving the other driver on. Some high side MOSFET drivers use a capacitor charge pump to boost the gate drive above the drain voltage. The charge pump charges when the driver is off and discharges into the MOSFET gate when the driver is on. It makes sense then to switch the high side driver to keep the charge pump refreshed. Even though this application does not use the charge pump type drivers, we will modulate the high side driver while leaving the low side driver on. There are three high side drivers, any one of which could be active depending on the position of the rotor. The motor drive word is 6-bits wide, so if we logically AND the drive word with zeros in the high driver bit positions, and 1's in the low driver bit positions, we will turn off the active high driver regardless which one of the three it is.

We have now identified 4 tasks of the control loop:

- Read the sensor inputs
- Commutate the motor drive connections
- Read the speed control ADC
- PWM the motor drivers using the ADC and Timer0 addition results

At 20 MHz clock rate, control latency, caused by the loop time, is not significant so we will construct a simple polled task loop. The control loop flow chart is shown in Figure 6 and code listings are in Appendix B.

**FIGURE 6: SENSORED DRIVE FLOWCHART**



## Sensorless Motor Control

It is possible to determine when to commutate the motor drive voltages by sensing the back EMF voltage on an undriven motor terminal during one of the drive phases. The obvious cost advantage of sensorless control is the elimination of the Hall position sensors. There are several disadvantages to sensorless control:

- The motor must be moving at a minimum rate to generate sufficient back EMF to be sensed
- Abrupt changes to the motor load can cause the BEMF drive loop to go out of lock
- The BEMF voltage can be measured only when the motor speed is within a limited range of the ideal commutation rate for the applied voltage
- Commutation at rates faster than the ideal rate will result in a discontinuous motor response

If low cost is a primary concern and low speed motor operation is not a requirement and the motor load is not expected to change rapidly then sensorless control may be the better choice for your application.

## Determining the BEMF

The BEMF, relative to the coil common connection point, generated by each of the motor coils, can be expressed as shown in Equation 4 through Equation 6.

**EQUATION 4:**

$$B_{BEMF} = \sin(\alpha)$$

**EQUATION 5:**

$$C_{BEMF} = \sin\left(\alpha - \frac{2\pi}{3}\right)$$

**EQUATION 6:**

$$A_{BEMF} = \sin\left(\alpha - \frac{4\pi}{3}\right)$$

**FIGURE 7: BEMF EQUIVALENT CIRCUIT**

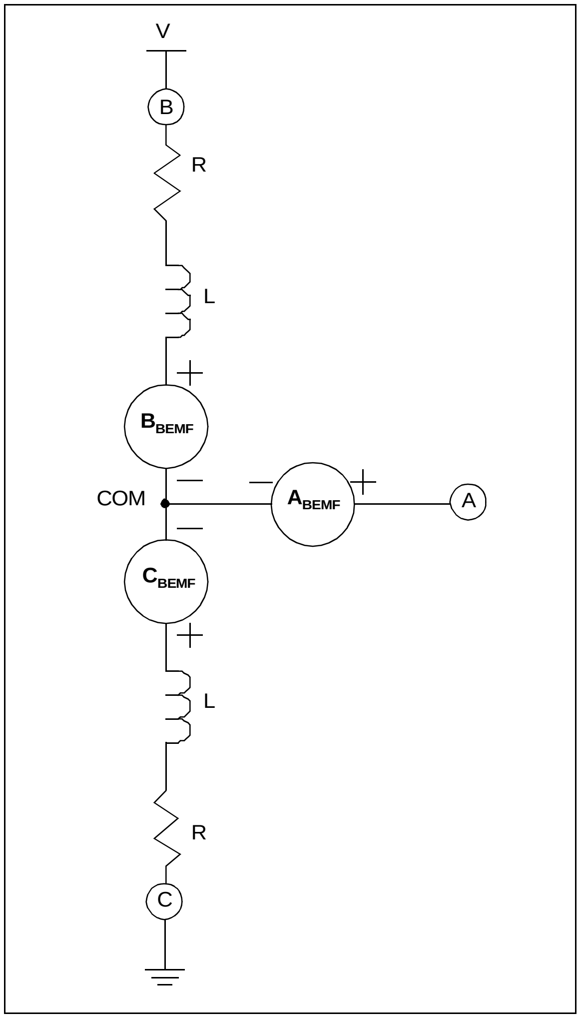



Figure 7 shows the equivalent circuit of the motor with coils B and C driven while coil A is undriven and available for BEMF measurement. At the commutation frequency the L's are negligible. The R's are assumed to be equal. The L and R components are not shown in the A branch since no significant current flows in this part of the circuit so those components can be ignored.

The BEMF generated by the B and C coils in tandem, as shown in Figure 7, can be expressed as shown in Equation 7.

**EQUATION 7:**

$$BEMF_{BC} = B_{BEMF} - C_{BEMF}$$

The sign reversal of $C_{BEMF}$ is due to moving the reference point from the common connection to ground.

Recall that there are six drive phases in one electrical revolution. Each drive phase occurs +/- 30 degrees around the peak back EMF of the two motor windings being driven during that phase. At full speed the applied DC voltage is equivalent to the RMS BEMF voltage in that 60 degree range. In terms of the peak BEMF generated by any one winding, the RMS BEMF voltage across two of the windings can be expressed as shown in Equation 8.

**EQUATION 8:**

$$BEMF_{RMS} = \sqrt{\frac{3}{\pi}\int_{\frac{\pi}{6}}^{\frac{\pi}{2}}\left(\sin(\alpha) - \sin\left(\alpha - \frac{2\pi}{3}\right)\right)^2 d\alpha}$$

$$BEMF_{RMS} = \sqrt{\frac{3}{\pi}\left(\frac{\pi}{2} + \frac{3\pi}{4}\right)}$$

$$BEMF_{RMS} = 1.6554$$

We will use this result to normalize the BEMF diagrams presented later, but first lets consider the expected BEMF at the undriven motor terminal.

Since the applied voltage is pulse width modulated, the drive alternates between on and off throughout the phase time. The BEMF, relative to ground, seen at the A terminal when the drive is on, can be expressed as shown in Equation 9.

**EQUATION 9:**

$$BEMF_A = \frac{[V - (B_{BEMF} - C_{BEMF})]R}{2R} - C_{BEMF} + A_{BEMF}$$

$$BEMF_A = \frac{V - B_{BEMF} + C_{BEMF}}{2} - C_{BEMF} + A_{BEMF}$$

Notice that the winding resistance cancels out, so resistive voltage drop, due to motor torque load, is not a factor when measuring BEMF.

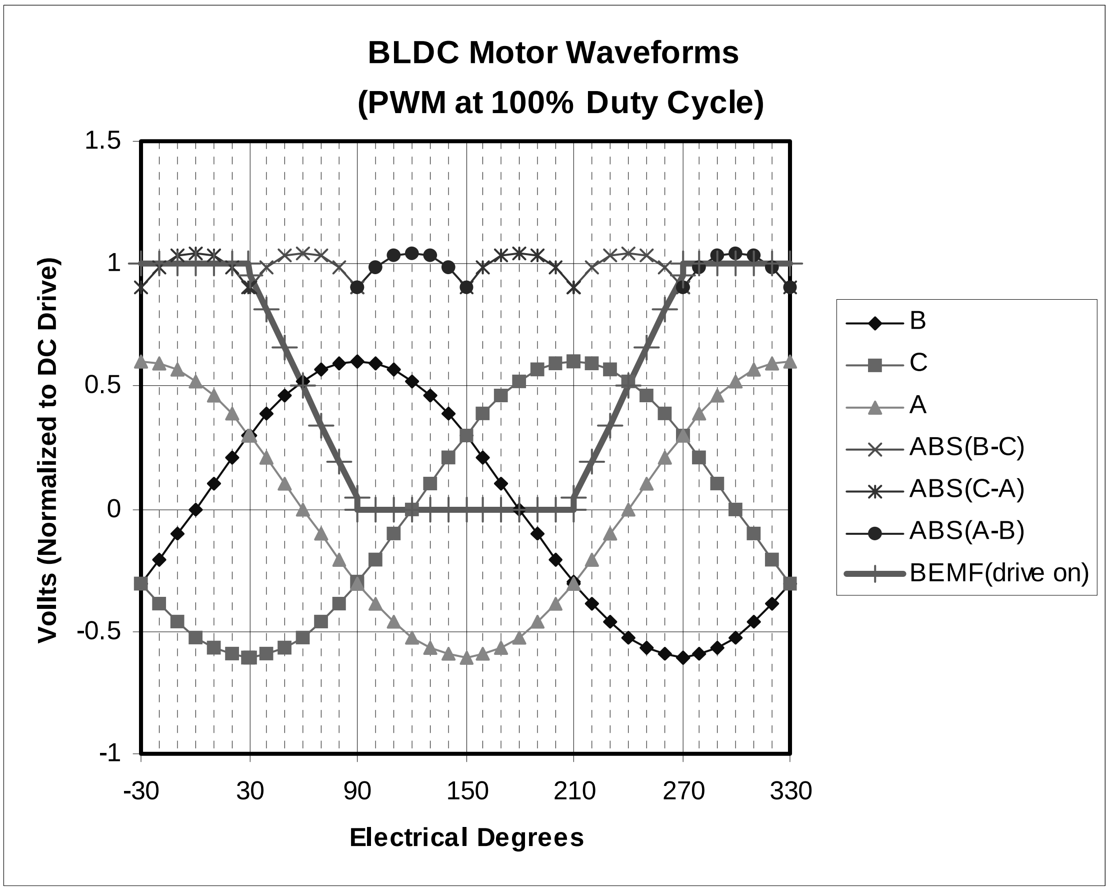
The BEMF, relative to ground, seen at the A terminal when the drive is off can be expressed as shown in Equation 10.

**EQUATION 10:**

$$BEMF_A = A_{BEMF} - C_{BEMF}$$

Figure 8 is a graphical representation of the BEMF formulas computed over one electrical revolution. To avoid clutter, only the terminal A waveform, as would be observed on a oscilloscope is displayed and is denoted as BEMF(drive on). The terminal A waveform is flattened at the top and bottom because at those points the terminal is connected to the drive voltage or ground. The sinusoidal waveforms are the individual coil BEMFs relative to the coil common connection point. The 60 degree sinusoidal humps are the BEMFs of the driven coil pairs relative to ground. The entire graph has been normalized to the RMS value of the coil pair BEMFs.

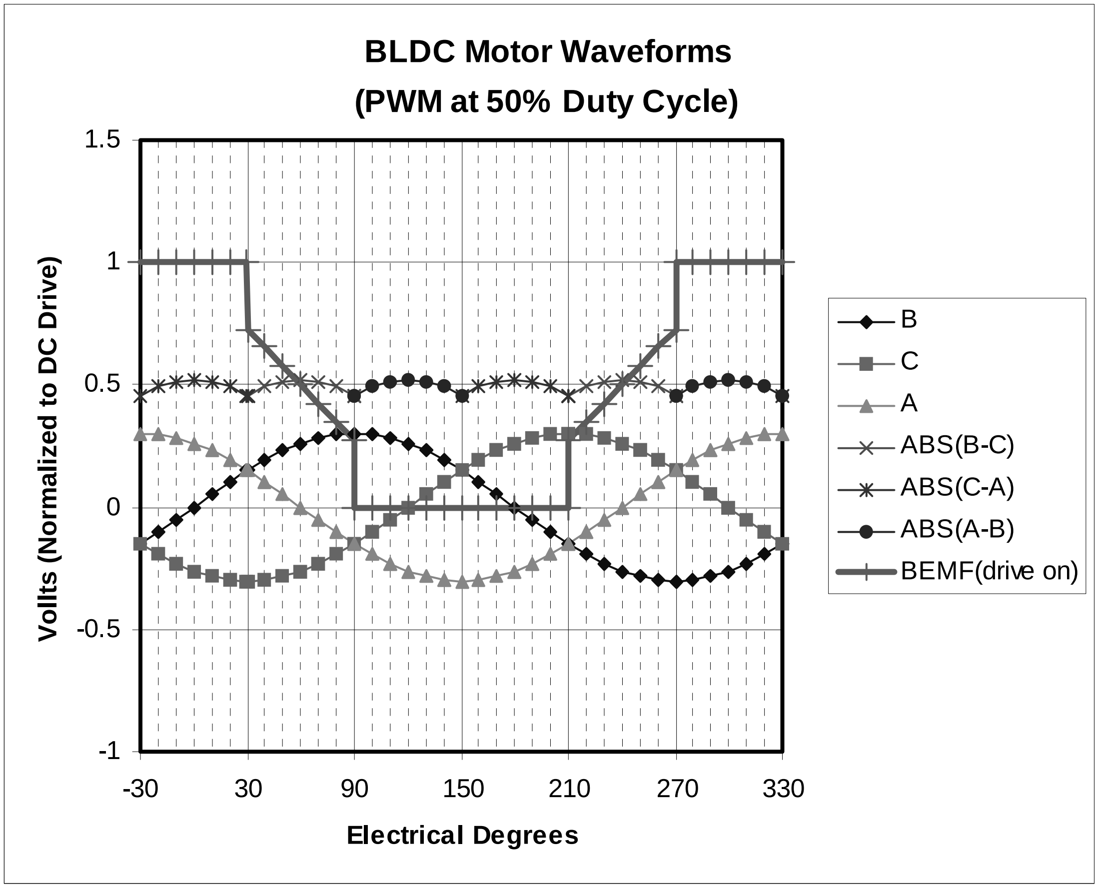
**FIGURE 8: BEMF AT 100% DRIVE**




Notice that the BEMF(drive on) waveform is fairly linear and passes through a voltage that is exactly half of the applied voltage at precisely 60 degrees which coincides with the zero crossing of the coil A BEMF waveform. This implies that we can determine the rotor electrical position by detecting when the open terminal voltage equals half the applied voltage.

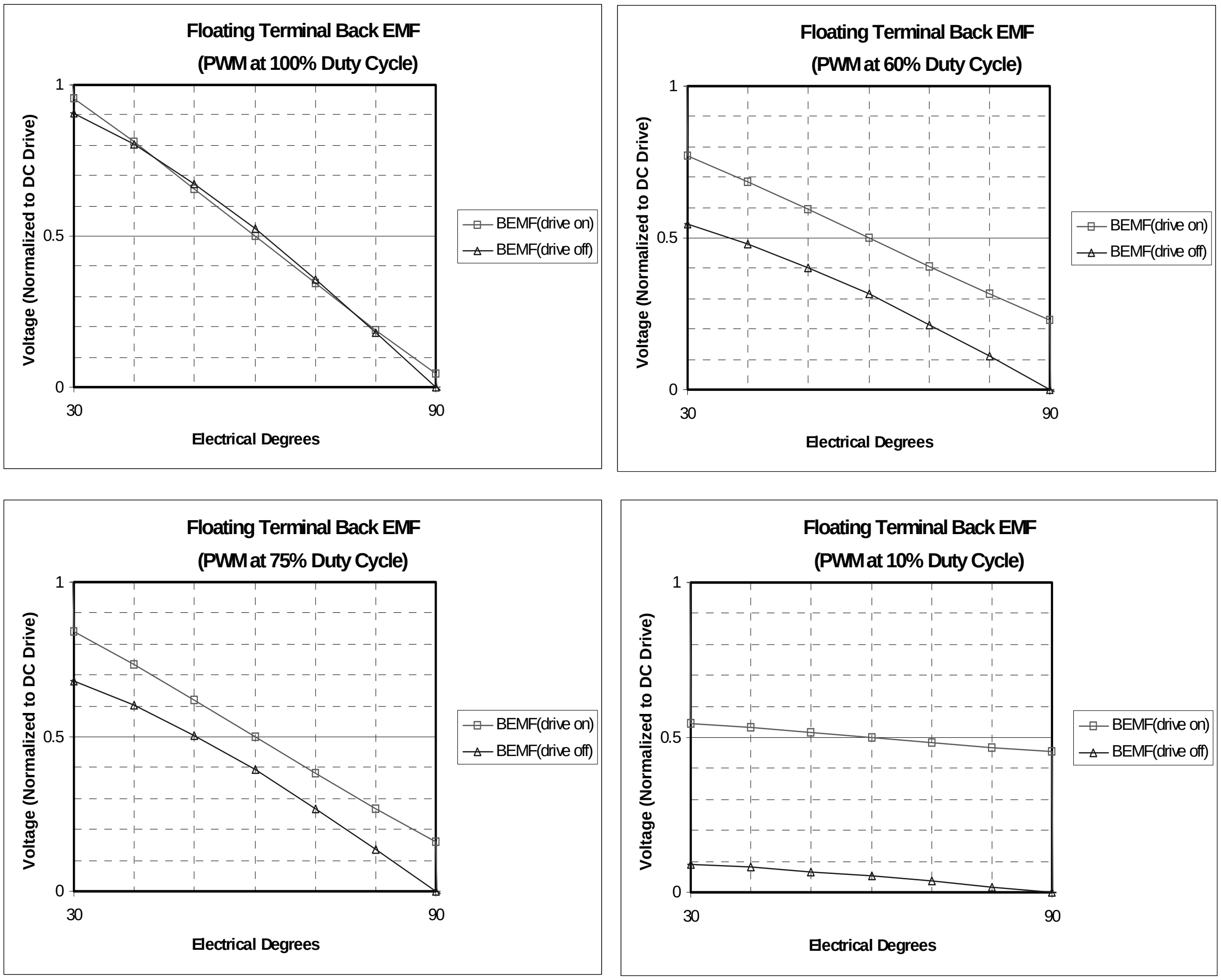
What happens when the PWM duty cycle is less than 100%? Figure 9 is a graphical representation of the BEMF formulas computed over one electrical revolution when the effective applied voltage is 50% of that shown in Figure 8. The entire graph has been normalized to the peak applied voltage.

**FIGURE 9: BEMF AT 50% DRIVE**



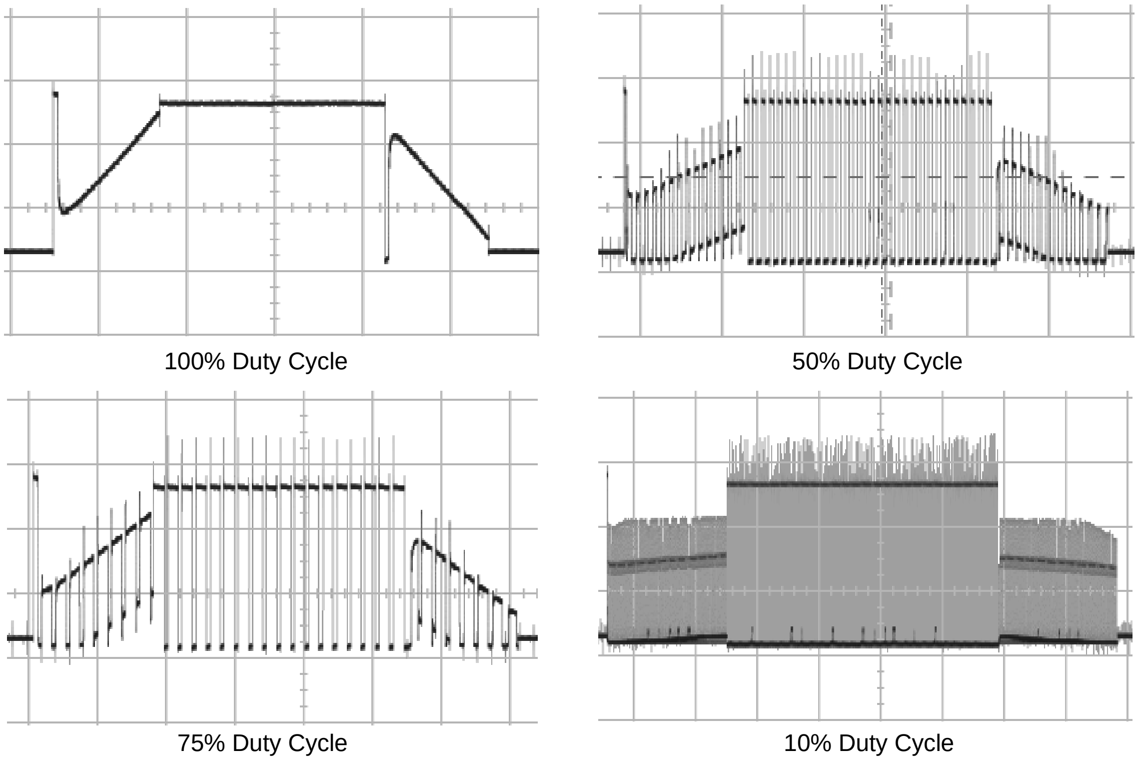

As expected the BEMF waveforms are all reduced proportionally but notice that the BEMF on the open terminal still equals half the applied voltage midway through the 60 degree drive phase. This occurs only when the drive voltage is on. Figure 10 shows a detail of the open terminal BEMF when the drive voltage is on and when the drive voltage is off. At various duty cycles, notice that the drive on curve always equals half the applied voltage at 60 degrees.

**FIGURE 10: DRIVE ON VS. DRIVE OFF BEMF**




How well do the predictions match an actual motor? Figure 11 is shows the waveforms present on terminal A of a Pittman N2311A011 brushless motor at various PWM duty cycle configurations. The large transients, especially prevalent in the 100% duty cycle waveform, are due to flyback currents caused by the motor winding inductance.

**FIGURE 11: PITTMAN BEMF WAVEFORMS**




The rotor position can be determined by measuring the voltage on the open terminal when the drive voltage is applied and then comparing the result to one half of the applied voltage.

Recall that motor speed is proportional to the applied voltage. The formulas and graphs presented so far represent motor operation when commutation rate coincides with the effective applied voltage. When the commutation rate is too fast then commutation occurs early and the zero crossing point occurs later in the drive phase. When the commutation rate is too slow then commutation occurs late and the zero crossing point occurs earlier in the drive phase. We can sense and use this shift in zero crossing to adjust the commutation rate to keep the motor running at the ideal speed for the applied voltage and load torque.

## Open Loop Speed Control

An interesting property of brushless DC motors is that they will operate synchronously to a certain extent. This means that for a given load, applied voltage, and commutation rate the motor will maintain open loop lock with the commutation rate provided that these three variables do not deviate from the ideal by a significant amount. The ideal is determined by the motor voltage and torque constants. How does this work? Consider that when the commutation rate is too slow for an applied voltage, the BEMF will be too low resulting in more motor current. The motor will react by accelerating to the next phase position then slow down waiting for the next commutation. In the extreme case the motor will snap to each position like a stepper motor until the next commutation occurs. Since the motor is able to accelerate faster than the commutation rate, rates much slower than the ideal can be tolerated without losing lock but at the expense of excessive current.

Now consider what happens when commutation is too fast. When commutation occurs early the BEMF has not reached peak resulting in more motor current and a greater rate of acceleration to the next phase but it will arrive there too late. The motor tries to keep up with the commutation but at the expense of excessive current. If the commutation arrives so early that the motor can not accelerate fast enough to catch the next commutation, lock is lost and the motor spins down. This happens abruptly not very far from the ideal rate. The abrupt loss of lock looks like a discontinuity in the motor response which makes closed loop control difficult. An alternative to closed loop control is to adjust the commutation rate until self locking open loop control is achieved. This is the method we will use in our application.

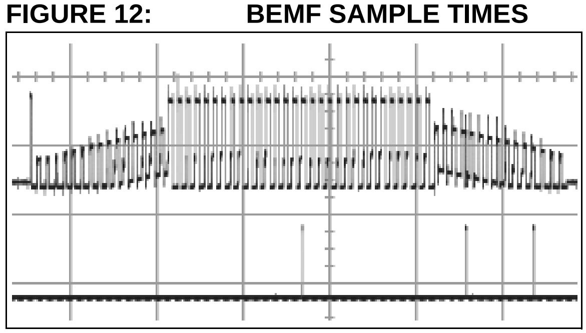
When the load on a motor is constant over it's operating range then the response curve of motor speed relative to applied voltage is linear. If the supply voltage is well regulated, in addition to a constant torque load, then the motor can be operated open loop over it's entire speed range. Consider that with pulse width modulation the effective voltage is linearly proportional to the PWM duty cycle. An open loop controller can be made by linking the PWM duty cycle to a table of motor speed values stored as the time of commutation for each drive phase. We need a table because revolutions per unit time is linear, but we need time per revolution which is not linear. Looking up the time values in a table is much faster than computing them repeatedly.

The program that we use to run the motor open loop is the same program we will use to automatically adjust the commutation rate in response to variations in the torque load. The program uses two potentiometers as speed control inputs. One potentiometer, we'll call it the PWM potentiometer, is directly linked to both the PWM duty cycle and the commutation time lookup table. The second potentiometer, we'll call this the Offset potentiometer, is used to provide an offset to the PWM duty cycle determined by the PWM potentiometer. An analog-to-digital conversion of the PWM potentiometer produces a number between 0 and 255. The PWM duty cycle is generated by adding the PWM potentiometer reading to a free running 8-bit timer. When the addition results in a carry the drive state is on, otherwise it is off. The PWM potentiometer reading is also used to access the 256 location commutation time lookup table. The Offset potentiometer also produces a number between 0 and 255. The Most Significant bit of this number is inverted making it a signed number between -128 and 127. This offset result, when added to the PWM potentiometer, becomes the PWM duty cycle threshold, and controls the drive on and off states described previously.

## Closed Loop Speed Control

Closed loop speed control is achieved by unlinking the commutation time table index from the PWM duty cycle number. The PWM potentiometer is added to a fixed manual threshold number between 0 and 255. When this addition results in a carry, the mode is switched to automatic. On entering Automatic mode the commutation index is initially set to the PWM potentiometer reading. Thereafter, as long as Automatic mode is still in effect, the commutation table index is automatically adjusted up or down according to voltages read at motor terminal A at specific times. Three voltage readings are taken.

**FIGURE 12: BEMF SAMPLE TIMES**



The first reading is taken during drive phase 4 when terminal A is actively driven high. This is the applied voltage. The next two readings are taken during drive phase 5 when terminal A is floating. The first reading is taken when ¼ of the commutation time has elapsed and the second reading is taken when ¾ of the commutation time has elapsed. We'll call these readings 1 and 2 respectively. The commutation table index is adjusted according to the following relationship between the applied voltage reading and readings 1 and 2:

- Index is unchanged if Reading 1 > Applied Voltage/2 and Reading 2 < Applied Voltage/2
- Index is increased if Reading 1 < Applied Voltage/2
- Index is decreased if Reading 1 > Applied Voltage/2 and Reading 2 > Applied Voltage/2

The motor rotor and everything it is connected to has a certain amount of inertia. The inertia delays the motor response to changes in voltage load and commutation time. Updates to the commutation time table index are delayed to compensate for the mechanical delay and allow the motor to catch up.

### Acceleration and Deceleration Delay

The inertia of the motor and what it is driving, tends to delay motor response to changes in the drive voltage. We need to compensate for this delay by adding a matching delay to the control loop. The control loop delay requires two time constants, a relatively slow one for acceleration, and a relatively fast one for deceleration.

Consider what happens in the control loop when the voltage to the motor suddenly rises, or the motor load is suddenly reduced. The control senses that the motor rotation is too slow and attempts to adjust by making the commutation time shorter. Without delay in the control loop, the next speed measurement will be taken before the motor has reacted to the adjustment, and another speed adjustment will be made. Adjustments continue to be made ahead of the motor response until eventually, the commutation time is too short for the applied voltage, and the motor goes out of lock. The acceleration timer delay prevents this runaway condition. Since the motor can tolerate commutation times that are too long, but not commutation times that are too short, the acceleration time delay can be longer than required without serious detrimental effect.

Consider what happens in the control loop when the voltage to the motor suddenly falls, or the motor load is suddenly increased. If the change is sufficiently large, commutation time will immediately be running too short for the motor conditions. The motor cannot tolerate this, and loss of lock will occur. To prevent loss of lock, the loop deceleration timer delay must be short enough for the control loop to track, or precede the changing motor condition. If the time delay is too short, then the control loop will continue to lengthen the commutation time ahead of the motor response resulting in over compensation. The motor will eventually slow to a speed that will indicate to the BEMF sensor that the speed is too slow for the applied voltage. At that point, commutation deceleration will cease, and the commutation change will adjust in the opposite direction governed by the acceleration time delay. Over compensation during deceleration will not result in loss of lock, but will cause increased levels of torque ripple and motor current until the ideal commutation time is eventually reached.

### Determining The Commutation Time Table Values

The assembler supplied with MPLAB performs all calculations as 32-bit integers. To avoid the rounding errors that would be caused by integer math, we will use a spreadsheet, such as Excel, to compute the table entries then cut and paste the results to an include file. The spreadsheet is setup as shown in Table 4.

**TABLE 4: COMMUTATION TIME TABLE VALUES**

| Variable Name | Number or Formula | Description |
|---|---|---|
| Phases | 12 | Number of commutation phase changes in one mechanical revolution. |
| FOSC | 20 MHz | Microcontroller clock frequency |
| FOSC_4 | FOSC/4 | Microcontroller timers source clock |
| Prescale | 4 | Timer 1 prescale |
| MaxRPM | 8000 | Maximum expected speed of the motor at full applied voltage |
| MinRPM | (60*FOSC_4)/Phases*Prescale*65535)+1 | Limitation of 16-bit timer |
| Offset | -345 | This is the zero voltage intercept on the RPM axis. A property normalized to the 8-bit A to D converter. |
| Slope | (MaxRPM-Offset)/255 | Slope of the RPM to voltage input response curve normalized to the 8-bit A to D converter. |

The body of the spreadsheet starts arbitrarily at row 13. Row 12 contains the column headings. The body of the spreadsheet is constructed as follows:

- Column A is the commutation table index number N. The numbers in column A are integers from 0 to 255.
- Column B is the RPM that will result by using the counter values at index number N. The formula in column B is: =IF(Offset+A13*Slope>MinRPM,Offset+A13*Slope,MinRPM).
- Column C is the duration of each commutation phase expressed in seconds. The formula for column C is: =60/(Phases*B13).
- Column D is the duration of each commutation phase expressed in timer counts. The formula for column D is: =C13*FOSC_4/Prescale.

The range of commutation phase times at a reasonable resolution requires a 16-bit timer. The timer counts from 0 to a compare value then automatically resets to 0. The compare values are stored in the commutation time table. Since the comparison is 16 bits and tables can only handle 8 bits the commutation times will be stored in two tables accessed by the same index.

- Column E is the most significant byte of the 16-bit timer compare value. The formula for column E is: =CONCATENATE("retlw high D'",INT(D13),"").
- Column F is the least significant byte of the 16-bit timer compare value. The formula for column F is: =CONCATENATE("retlw low D'",INT(D13),"").

When all spreadsheet formulas have been entered in row 13, the formulas can be dragged down to row 268 to expand the table to the required 256 entries. Columns E and F will have the table entries in assembler ready format. An example of the table spreadsheet is shown in Figure 13.

**FIGURE 13: PWM LOOKUP TABLE GENERATOR**

BLDC Table Generator.xls

| | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| 1 | Phases/Rev | 12 | | | | |
| 2 | Fosc | 2.00E+07 | | | | |
| 3 | Fosc/4 | 5.00E+06 | | | | |
| 4 | Prescale | 4 | | | | |
| 5 | MaxRPM | 8000 | | | | |
| 6 | MinRPM | 96 | | | | |
| 7 | Offset | -345.00 | | | | |
| 8 | Slope | 32.73 | | | | |
| 9 | | | | | | |
| 10 | | | | | | |
| 11 | | | | | | |
| 12 | N | RPM | Sec per Transition | Timer Counts | MS Byte Code | LS Byte Code |
| 13 | 0 | 96 | 5.19E-02 | 64855 | retlw high D'64854' | retlw low D'64854' |
| 14 | 1 | 96 | 5.19E-02 | 64855 | retlw high D'64854' | retlw low D'64854' |
| 15 | 2 | 96 | 5.19E-02 | 64855 | retlw high D'64854' | retlw low D'64854' |
| 16 | 3 | 96 | 5.19E-02 | 64855 | retlw high D'64854' | retlw low D'64854' |
| 17 | 4 | 96 | 5.19E-02 | 64855 | retlw high D'64854' | retlw low D'64854' |
| 18 | 5 | 96 | 5.19E-02 | 64855 | retlw high D'64854' | retlw low D'64854' |
| 19 | 6 | 96 | 5.19E-02 | 64855 | retlw high D'64854' | retlw low D'64854' |
| 20 | 7 | 96 | 5.19E-02 | 64855 | retlw high D'64854' | retlw low D'64854' |
| 21 | 8 | 96 | 5.19E-02 | 64855 | retlw high D'64854' | retlw low D'64854' |
| 22 | 9 | 96 | 5.19E-02 | 64855 | retlw high D'64854' | retlw low D'64854' |
| 23 | 10 | 96 | 5.19E-02 | 64855 | retlw high D'64854' | retlw low D'64854' |
| 24 | 11 | 96 | 5.19E-02 | 64855 | retlw high D'64854' | retlw low D'64854' |
| 25 | 12 | 96 | 5.19E-02 | 64855 | retlw high D'64854' | retlw low D'64854' |
| 26 | 13 | 96 | 5.19E-02 | 64855 | retlw high D'64854' | retlw low D'64854' |
| 27 | 14 | 113 | 4.42E-02 | 55233 | retlw high D'55233' | retlw low D'55233' |
| 28 | 15 | 146 | 3.43E-02 | 42843 | retlw high D'42842' | retlw low D'42842' |

Lookup Table / Open Loop

## Using Open Loop Control to Determine Motor Characteristics

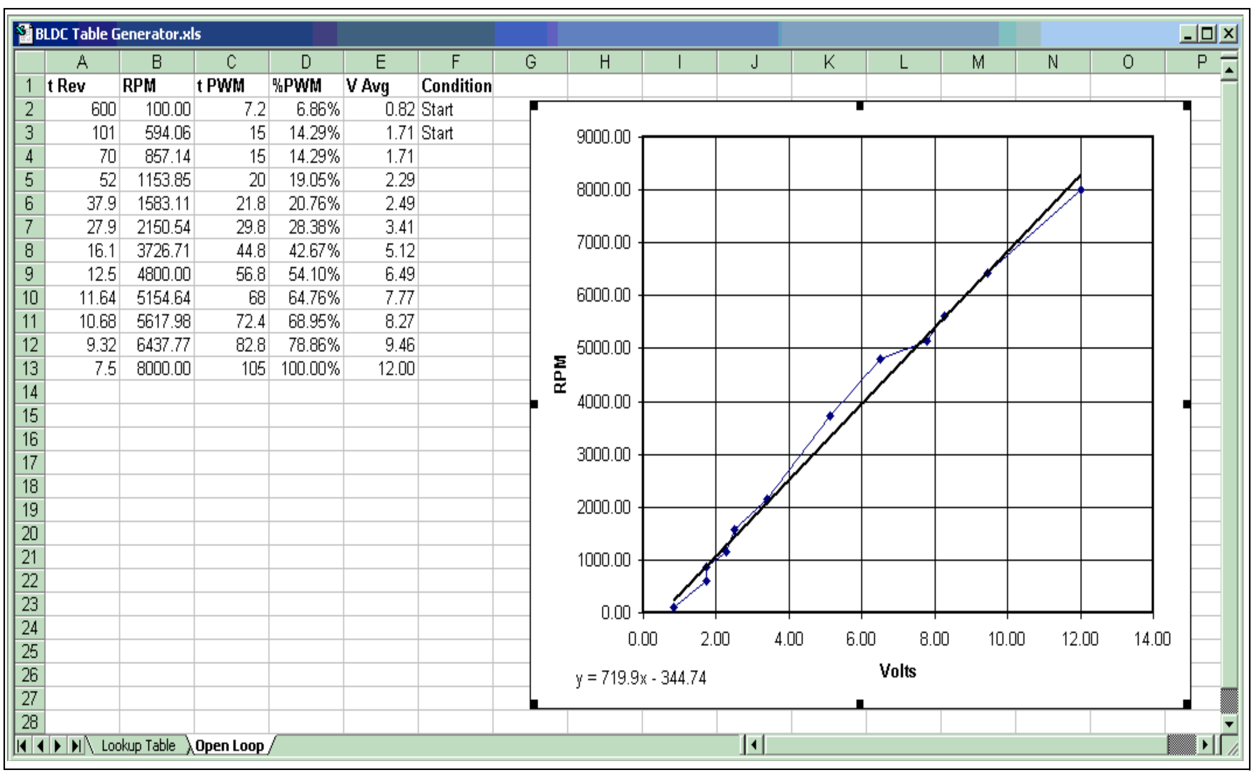
You can measure the motor characteristics by operating the motor in Open Loop mode, and measuring the motor current at several applied voltages. You can then chart the response curve in a spreadsheet, such as Excel, to determine the slope and offset numbers. Finally, plug the maximum RPM and offset numbers back into the table generator spreadsheet to regenerate the RPM tables.

To operate the motor in Open Loop mode:

- Set the manual threshold number (`ManThresh`) to 0xFF. This will prevent the Auto mode from taking over.
- When operating the motor in Open Loop mode, start by adjusting the offset control until the motor starts to move. You may also need to adjust the PWM control slightly above minimum.
- After the motor starts, you can increase the PWM control to increase the motor speed. The RPM and voltage will track, but you will need to adjust the offset frequently to optimize the voltage for the selected RPM.
- Optimize the voltage by adjusting the offset for minimum current.

To obtain the response offset with Excel®, enter the voltage (left column), and RPM (right column) pairs in adjacent columns of the spreadsheet. Use the chart wizard to make an X-Y scatter chart. When the chart is finished, right click on the response curve and select the pop-up menu “add trendline. . .” option. Choose the linear regression type and, in the Options tab, check the “display equation on chart” option. An example of the spreadsheet is shown in Figure 14.

**FIGURE 14: MOTOR RESPONSE SCOPE DETERMINATION**



| t Rev | RPM | t PWM | %PWM | V Avg | Condition |
|---|---|---|---|---|---|
| 600 | 100.00 | 7.2 | 6.86% | 0.82 | Start |
| 101 | 594.06 | 15 | 14.29% | 1.71 | Start |
| 70 | 857.14 | 15 | 14.29% | 1.71 | |
| 52 | 1153.85 | 20 | 19.05% | 2.29 | |
| 37.9 | 1583.11 | 21.8 | 20.76% | 2.49 | |
| 27.9 | 2150.54 | 29.8 | 28.38% | 3.41 | |
| 16.1 | 3726.71 | 44.8 | 42.67% | 5.12 | |
| 12.5 | 4800.00 | 56.8 | 54.10% | 6.49 | |
| 11.64 | 5154.64 | 68 | 64.76% | 7.77 | |
| 10.68 | 5617.98 | 72.4 | 68.95% | 8.27 | |
| 9.32 | 6437.77 | 82.8 | 78.86% | 9.46 | |
| 7.5 | 8000.00 | 105 | 100.00% | 12.00 | |



## Constructing The Sensorless Control Code

At this point we have all the pieces required to control a sensorless motor. We can measure BEMF and the applied voltage then compare them to each other to determine rotor position. We can vary the effective applied voltage with PWM and control the speed of the motor by timing the commutation phases. Some measurement events must be precisely timed. Other measurement events need not to interfere with each other. The ADC must be switched from one source to another and allow for sufficient acquisition time. Some events must happen rapidly with minimum latency. These include PWM and commutation.

We can accomplish everything with a short main loop that calls a state table. The main loop will handle PWM and commutation and the state table will schedule reading the two potentiometers, the peak applied voltage and the BEMF voltages at two times when the attached motor terminal is floating. Figure A-1 through Figure A-10, in Appendix A, is the resulting flow chart of sensorless motor control. Code listings are in Appendix C and Appendix D.

## APPENDIX A: SENSORLESS CONTROL FLOWCHART

### FIGURE A-1: MAIN LOOP

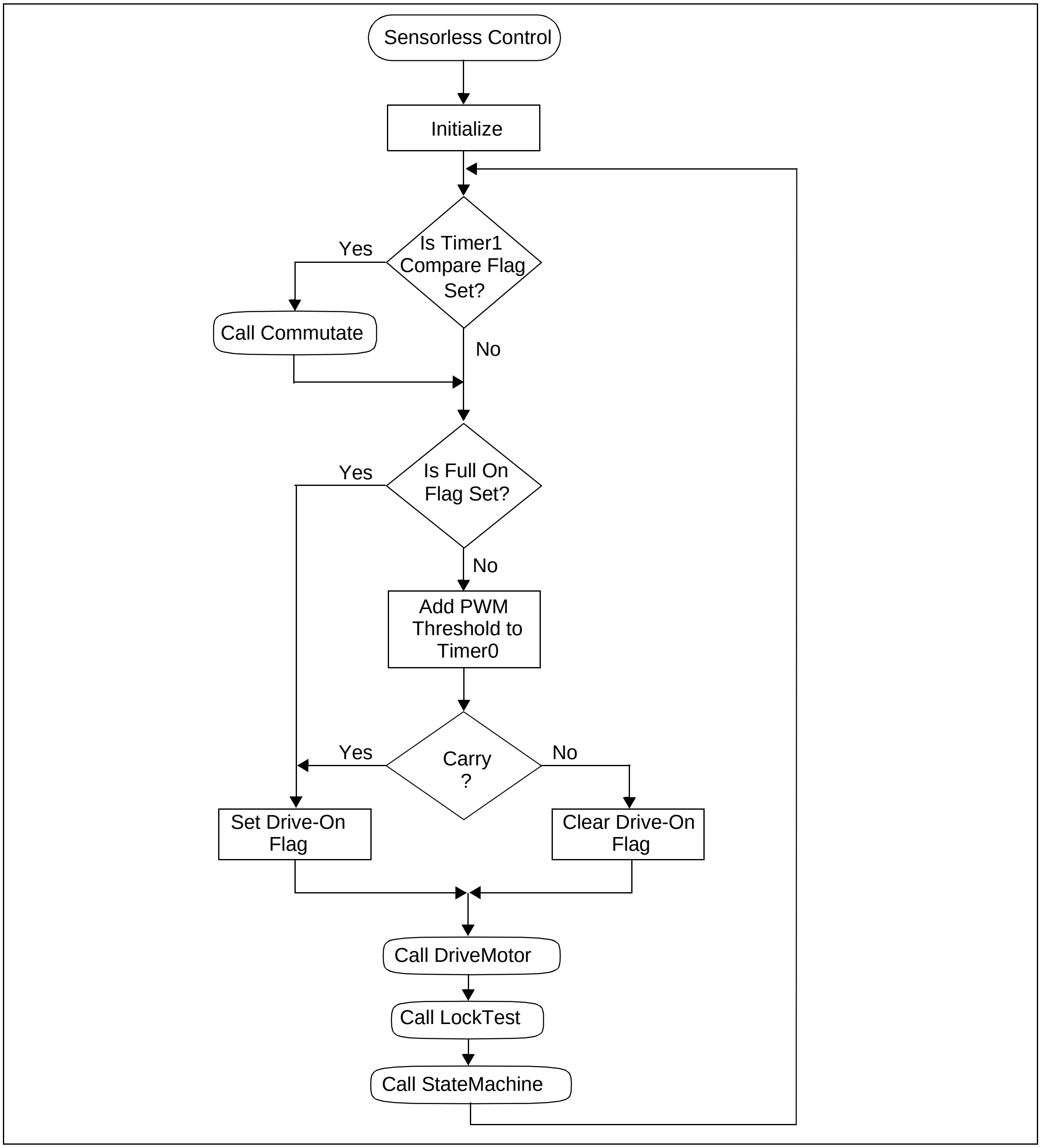


**FIGURE A-2: MOTOR COMMUTATION**

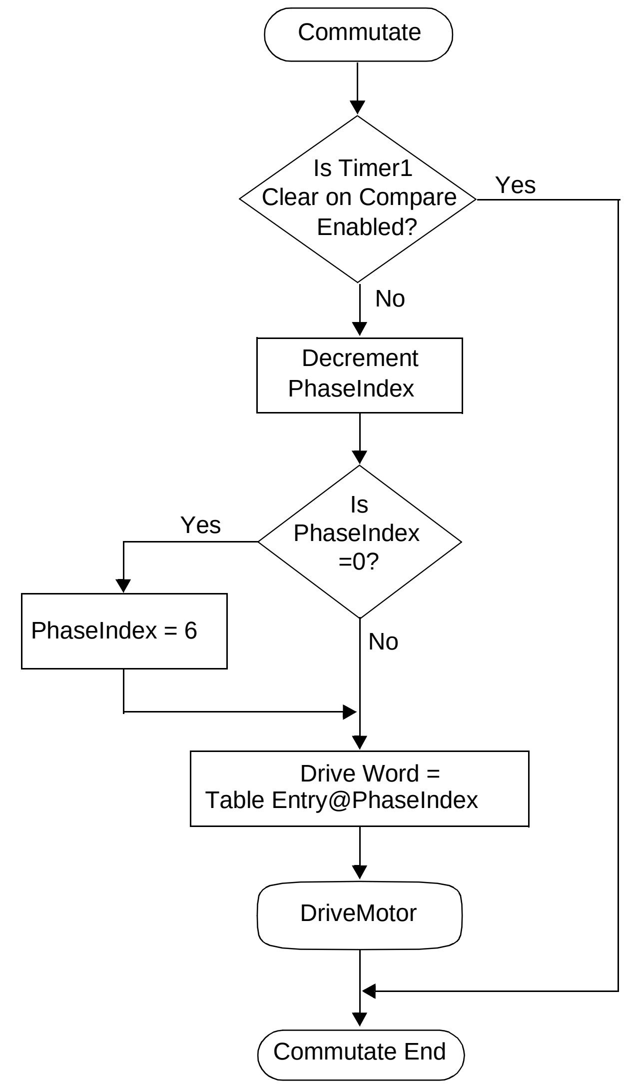


**FIGURE A-3: MOTOR DRIVER CONTROL**

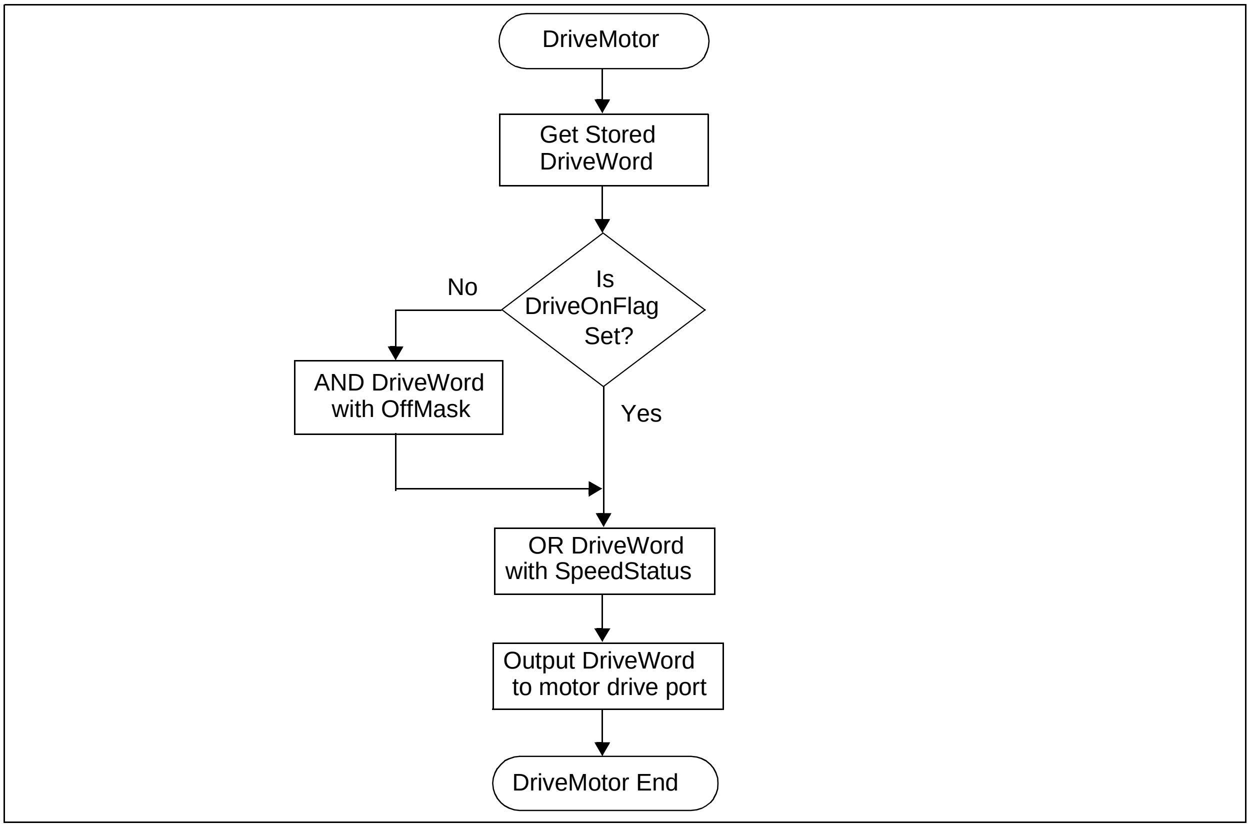



**FIGURE A-4: PHASE DRIVE PERIOD**

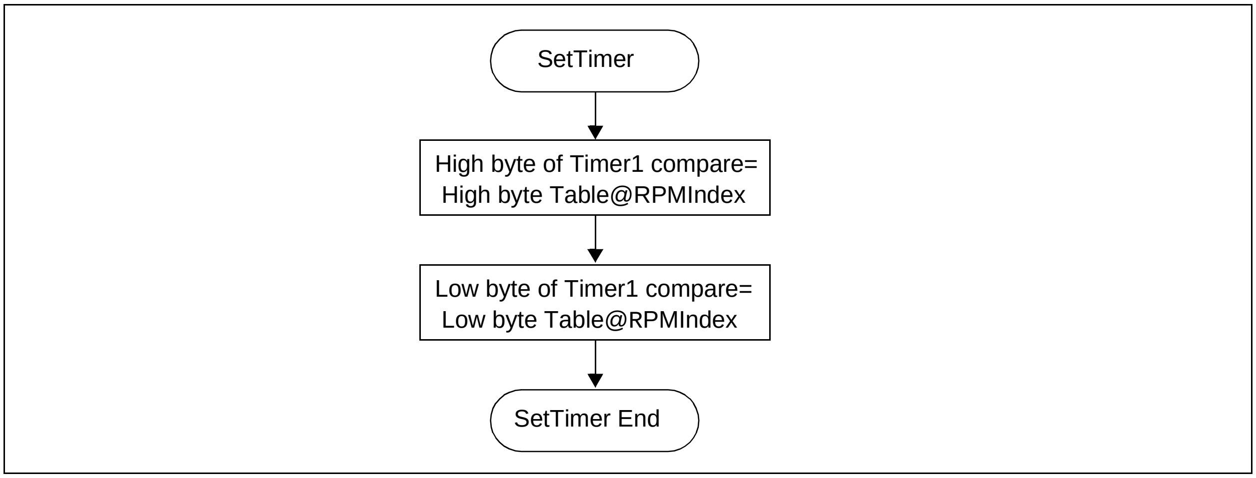


**FIGURE A-5: MOTOR SPEED LOCKED WITH COMMUTATION RATE**

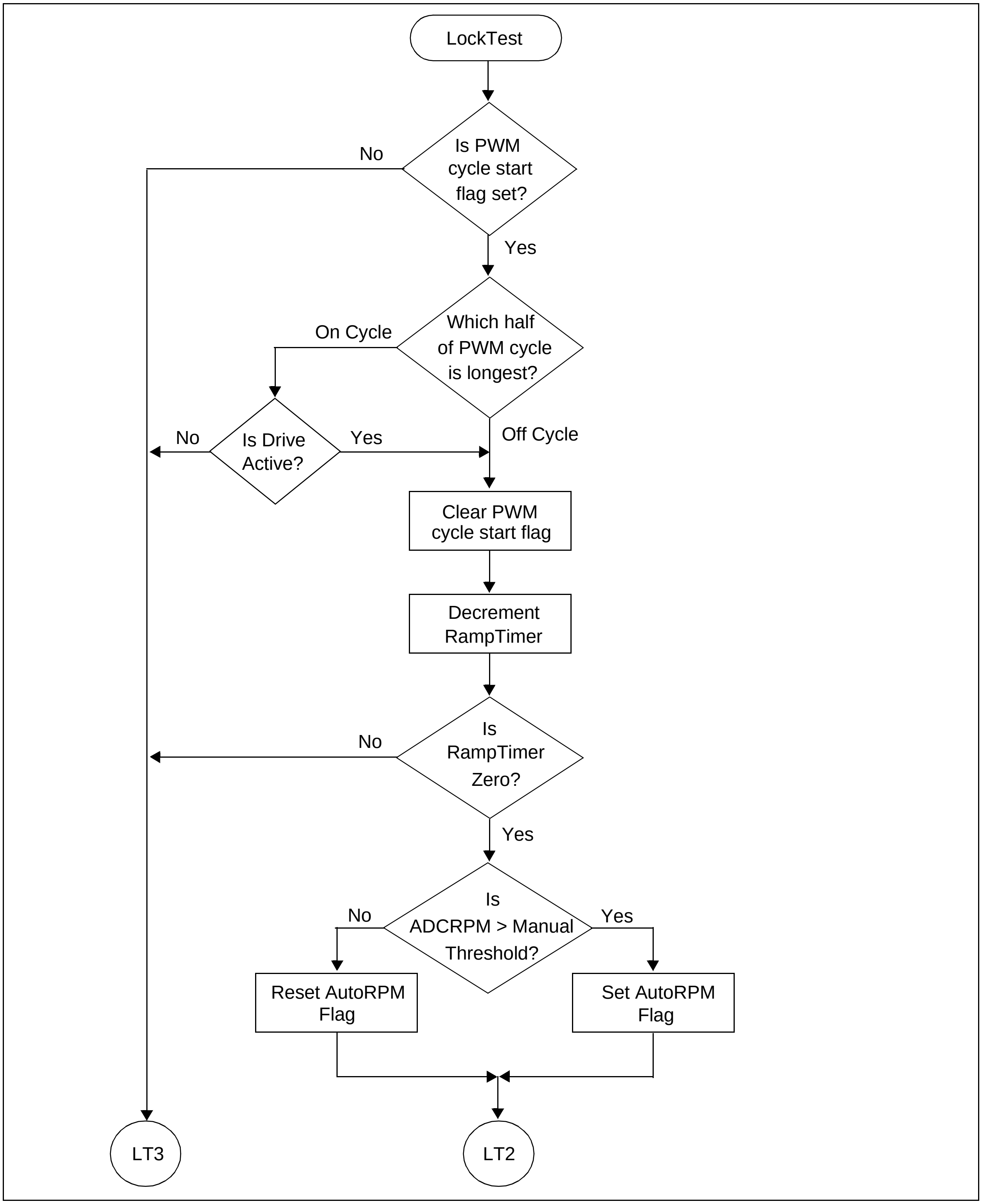


**FIGURE A-6: MOTOR SPEED LOCKED WITH COMMUTATION RATE (CONT.)**

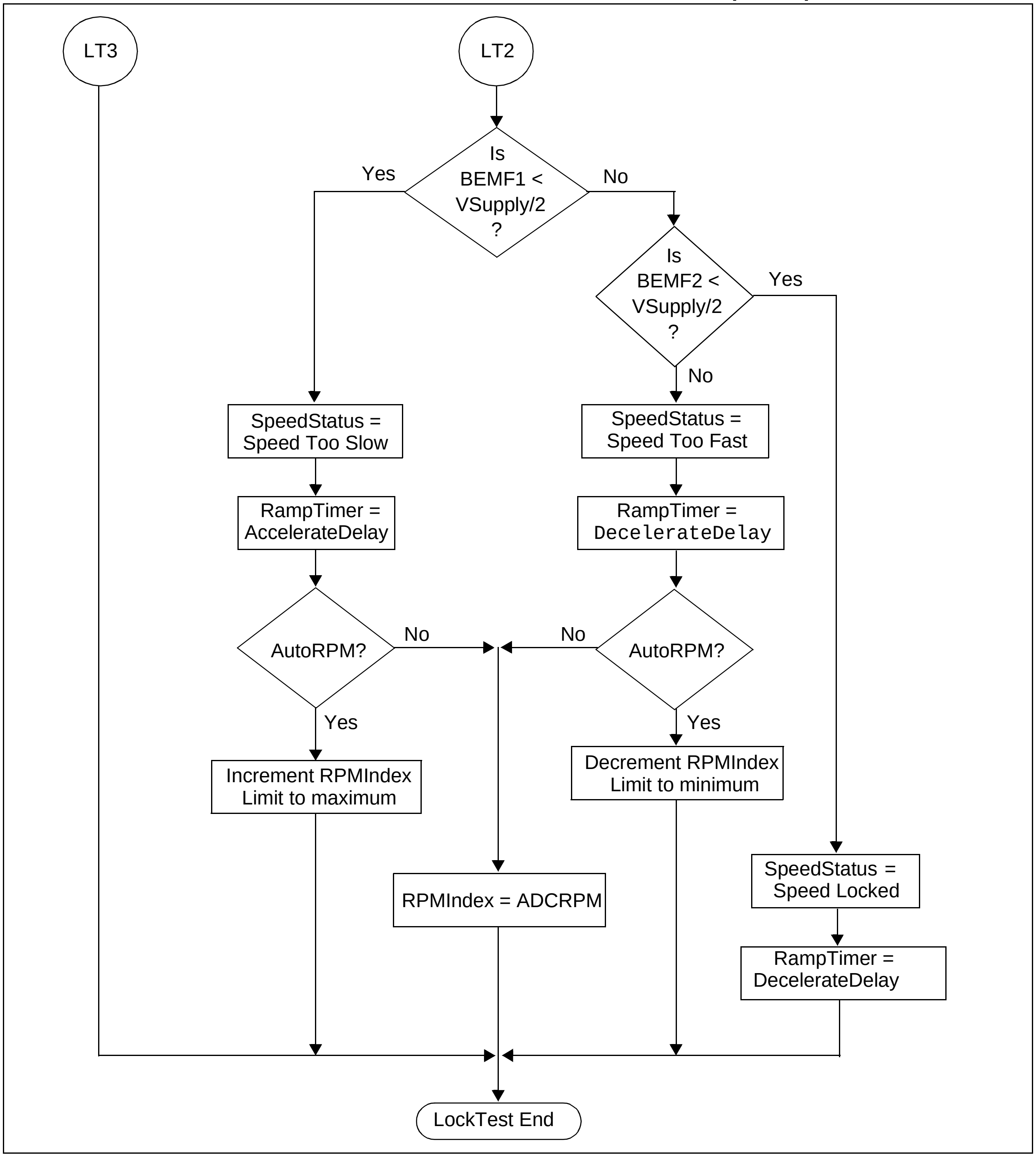


**FIGURE A-7: MOTOR CONTROL STATE MACHINE**

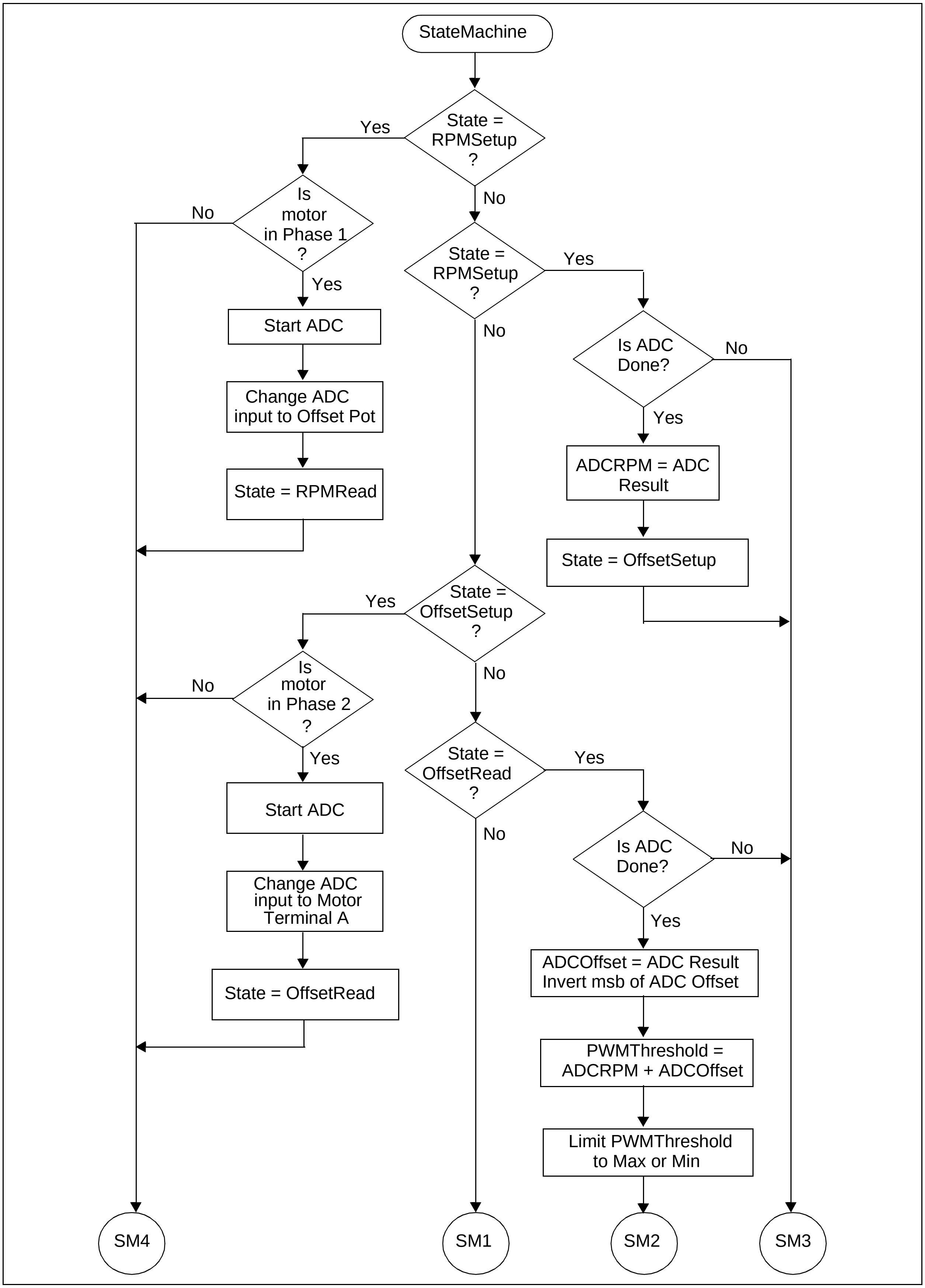


**FIGURE A-8: MOTOR CONTROL STATE MACHINE (CONT.)**

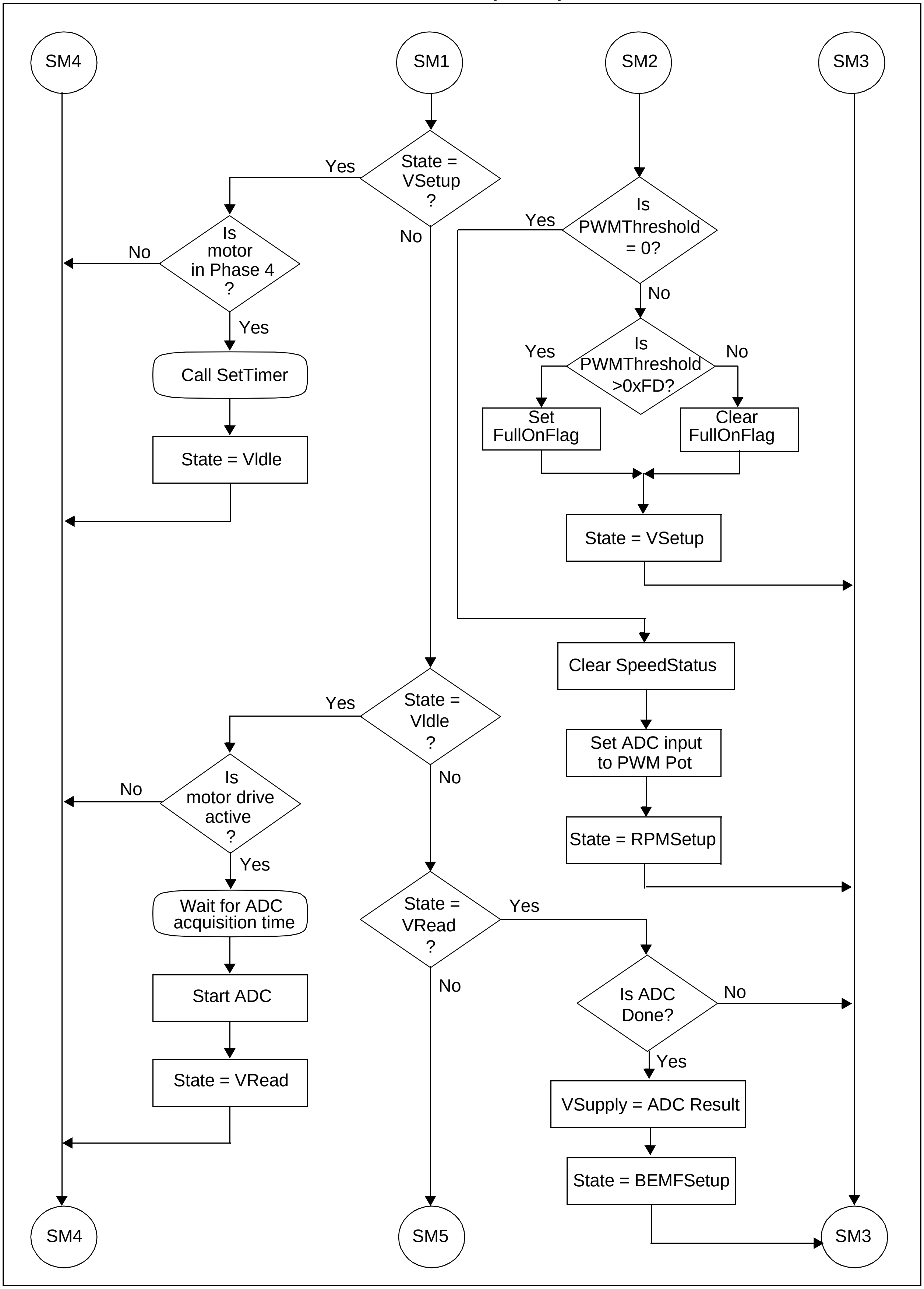


**FIGURE A-9: MOTOR CONTROL STATE MACHINE (CONT.)**

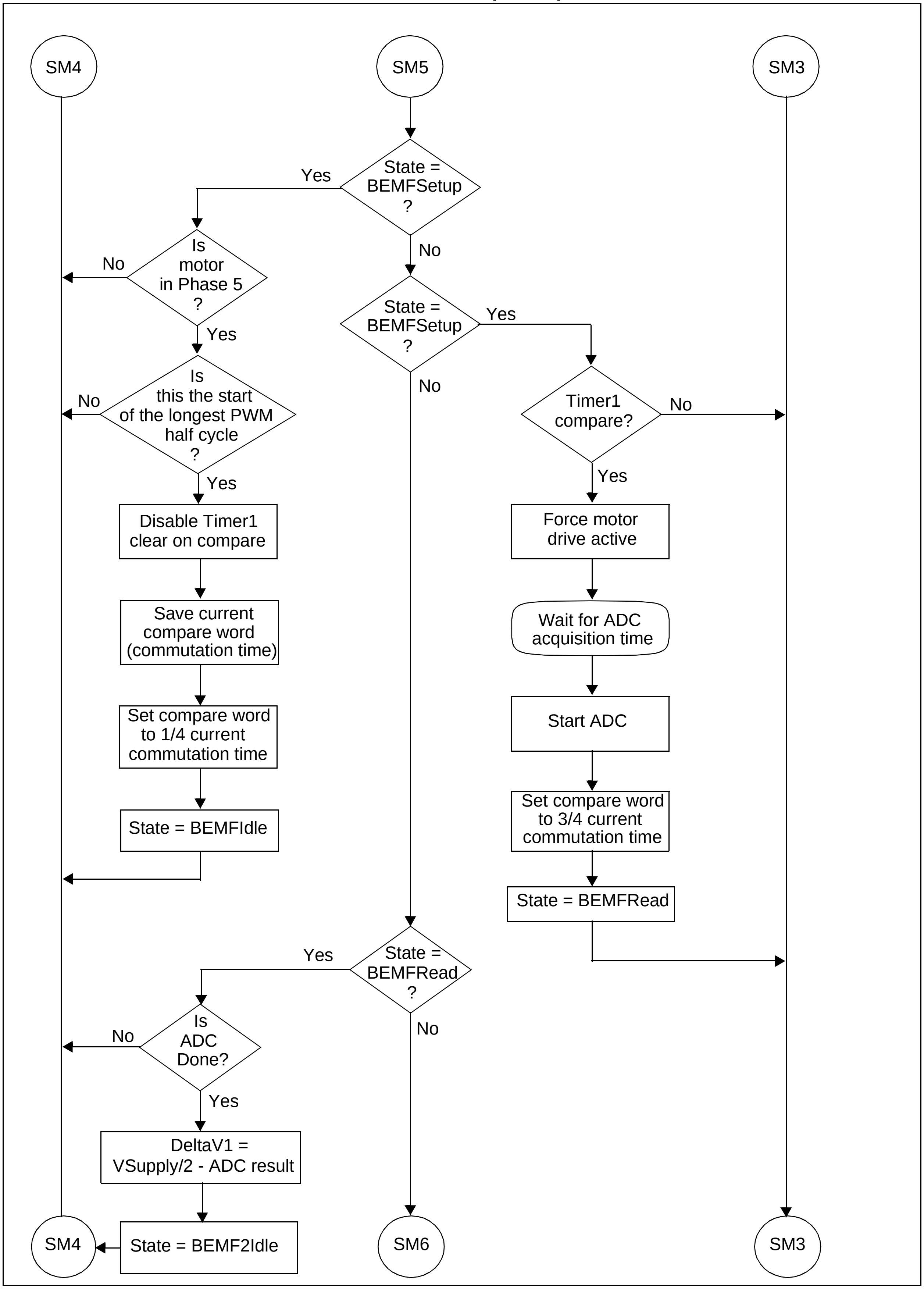
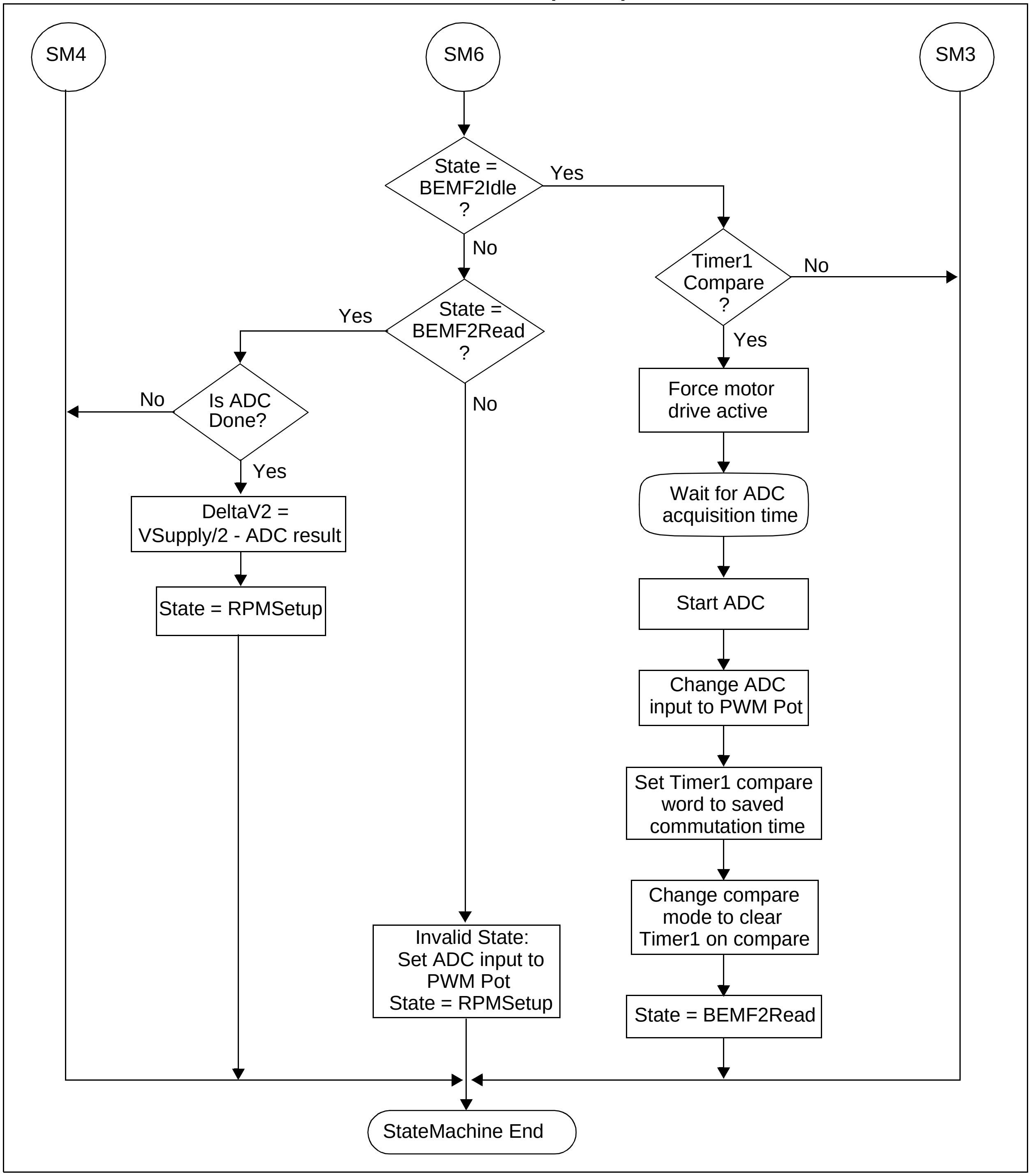


**FIGURE A-10: MOTOR CONTROL STATE MACHINE (CONT.)**



## APPENDIX B: SCHEMATICS

**FIGURE B-1: SCHEMATIC A - MOTOR DRIVERS**

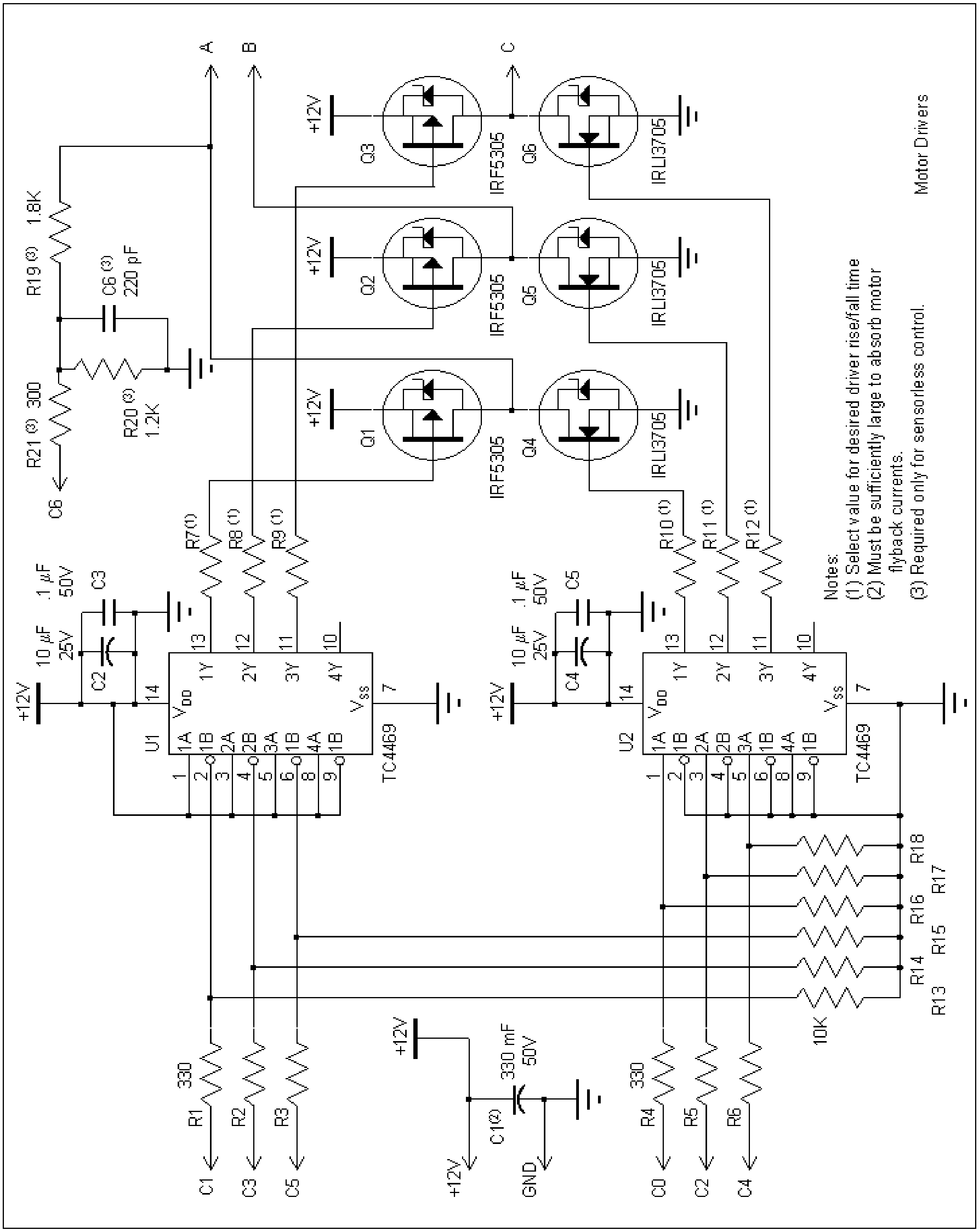
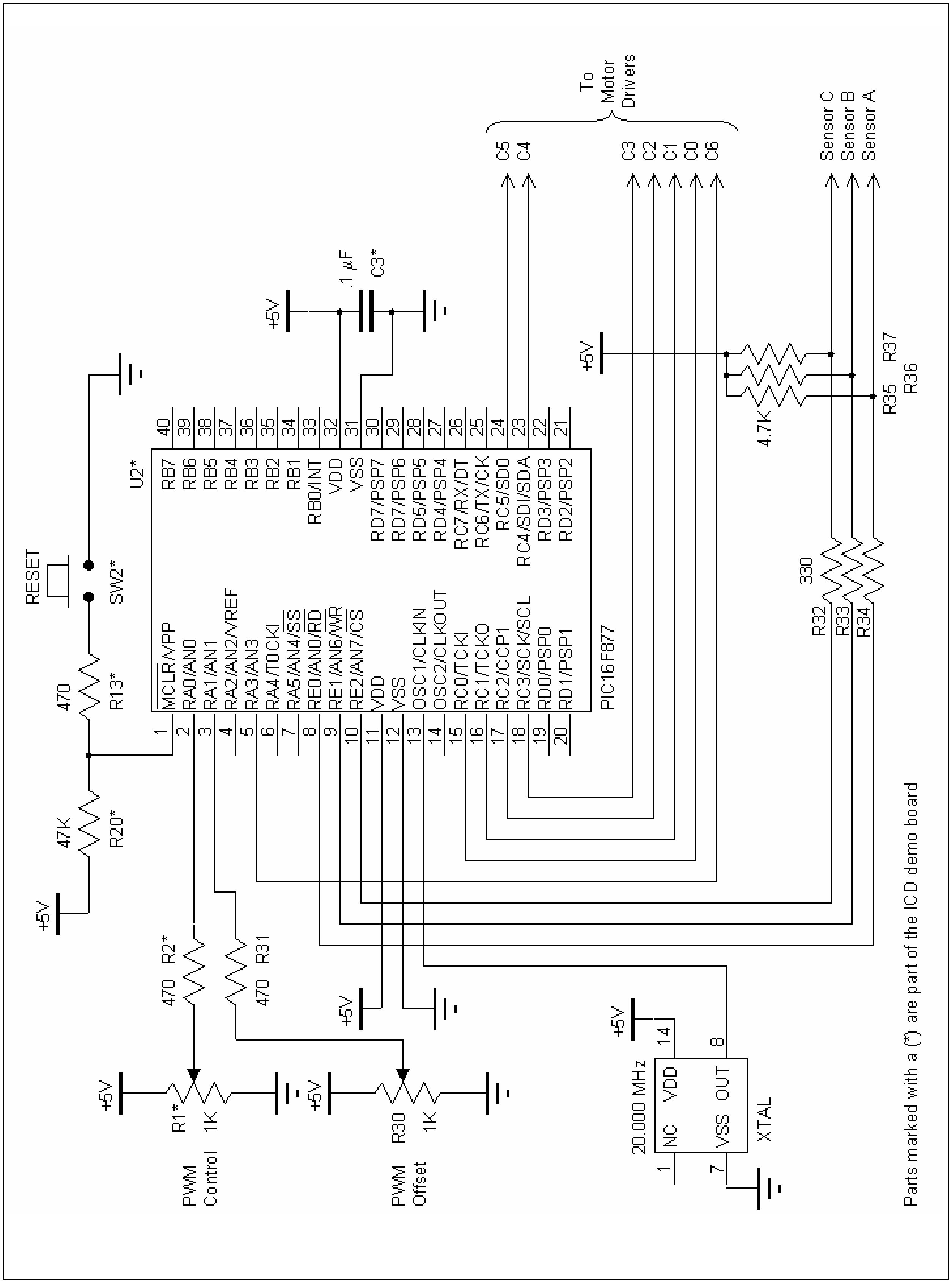


**FIGURE B-2: SCHEMATIC B - CONTROLLER**



## APPENDIX C: SENSORED CODE

```
;**********************************************************************
;                                                                     *
;   Filename:          sensored.asm                                   *
;   Date:              11 Feb. 2002                                   *
;   File Version:      1.0                                            *
;                                                                     *
;   Author:            W.R. Brown                                     *
;   Company:           Microchip Technology Incorporated              *
;                                                                     *
;                                                                     *
;**********************************************************************
;                                                                     *
;   Files required:    p16f877.inc                                    *
;                                                                     *
;                                                                     *
;                                                                     *
;**********************************************************************
;                                                                     *
;   Notes: Sensored brushless motor control Main loop uses 3-bit      *
;   sensor input as index for drive word output. PWM based on         *
;   Timer0 controls average motor voltage. PWM level is determined    *
;   PWM level is determined from ADC reading of potentiometer.        *
;                                                                     *
;**********************************************************************


    list        p=16f877             ; list directive to define processor
    #include    <p16f877.inc>        ; processor specific variable definitions

    __CONFIG _CP_OFF & _WDT_OFF & _BODEN_ON & _PWRTE_ON & _HS_OSC & _WRT_ENABLE_OFF & _LVP_ON &
_DEBUG_OFF & _CPD_OFF

;**********************************************************************
;*
;* Define variable storage
;*
    CBLOCK 0x20

    ADC             ; PWM threshold is ADC result
    LastSensor      ; last read motor sensor data
    DriveWord       ; six bit motor drive data

    ENDC
```

```
;**********************************************************************
;*
;* Define I/O
;*

#define    OffMask            B'11010101'
#define    DrivePort          PORTC
#define    DrivePortTris      TRISC
#define    SensorMask         B'00000111'
#define    SensorPort         PORTE
#define    DirectionBit       PORTA,1

;**********************************************************************
           org      0x000              ; startup vector
           nop                         ; required for ICD operation
           clrf     PCLATH             ; ensure page bits are cleared
           goto     Initialize         ; go to beginning of program


           ORG      0x004              ; interrupt vector location
           retfie                      ; return from interrupt

;**********************************************************************
;*
;* Initialize I/O ports and peripherals
;*

Initialize
           clrf     DrivePort          ; all drivers off

           banksel TRISA
; setup I/O
           clrf     DrivePortTris      ; set motor drivers as outputs
           movlw    B'00000011'        ; A/D on RA0, Direction on RA1, Motor sensors on RE<2:0>
           movwf    TRISA              ;
; setup Timer0
           movlw    B'11010000'        ; Timer0: Fosc, 1:2
           movwf    OPTION_REG
; Setup ADC (bank1)
           movlw    B'00001110'        ; ADC left justified, AN0 only
           movwf    ADCON1

           banksel  ADCON0
; setup ADC (bank0)
           movlw    B'11000001'        ; ADC clock from int RC, AN0, ADC on
           movwf    ADCON0

           bsf      ADCON0,GO          ; start ADC
           clrf     LastSensor         ; initialize last sensor reading
           call     Commutate          ; determine present motor position
           clrf     ADC                ; start speed control threshold at zero until first ADC
reading

;**********************************************************************
;*
;* Main control loop
;*
Loop
           call     ReadADC            ; get the speed control from the ADC
           incfsz   ADC,w              ; if ADC is 0xFF we're at full speed - skip timer add
           goto     PWM                ; add Timer0 to ADC for PWM

           movf     DriveWord,w        ; force on condition
           goto     Drive              ; continue
PWM
```

```
        movf      ADC,w               ; restore ADC reading
        addwf     TMR0,w              ; add it to current Timer0
        movf      DriveWord,w         ; restore commutation drive data
        btfss     STATUS,C            ; test if ADC + Timer0 resulted in carry
        andlw     OffMask             ; no carry - suppress high drivers
Drive
        movwf     DrivePort           ; enable motor drivers
        call      Commutate           ; test for commutation change
        goto      Loop                ; repeat loop

ReadADC
;**********************************************************************
;*
;* If the ADC is ready then read the speed control potentiometer
;* and start the next reading
;*
        btfsc     ADCON0,NOT_DONE     ; is ADC ready?
        return                        ; no - return

        movf      ADRESH,w            ; get ADC result
        bsf       ADCON0,GO           ; restart ADC
        movwf     ADC                 ; save result in speed control threshold
        return                        ;

;**********************************************************************
;*
;* Read the sensor inputs and if a change is sensed then get the
;* corresponding drive word from the drive table
;*
Commutate
        movlw     SensorMask          ; retain only the sensor bits
        andwf     SensorPort,w        ; get sensor data
        xorwf     LastSensor,w        ; test if motion sensed
        btfsc     STATUS,Z            ; zero if no change
        return                        ; no change - back to the PWM loop

        xorwf     LastSensor,f        ; replace last sensor data with current
        btfss     DirectionBit        ; test direction bit
        goto      FwdCom              ; bit is zero - do forward commutation

                                      ; reverse commutation
        movlw     HIGH RevTable       ; get MS byte of table
        movwf     PCLATH              ; prepare for computed GOTO
        movlw     LOW RevTable        ; get LS byte of table
        goto      Com2
FwdCom                                ; forward commutation
        movlw     HIGH FwdTable       ; get MS byte of table
        movwf     PCLATH              ; prepare for computed GOTO
        movlw     LOW FwdTable        ; get LS byte of table
Com2
        addwf     LastSensor,w        ; add sensor offset
        btfsc     STATUS,C            ; page change in table?
        incf      PCLATH,f            ; yes - adjust MS byte

        call      GetDrive            ; get drive word from table
        movwf     DriveWord           ; save as current drive word
        return

GetDrive
        movwf     PCL
```

```
;**********************************************************************
;*
;* The drive tables are built based on the following assumptions:
;* 1) There are six drivers in three pairs of two
;* 2) Each driver pair consists of a high side (+V to motor) and low side (motor to ground) drive
;* 3) A 1 in the drive word will turn the corresponding driver on
;* 4) The three driver pairs correspond to the three motor windings: A, B and C
;* 5) Winding A is driven by bits <1> and <0> where <1> is A's high side drive
;* 6) Winding B is driven by bits <3> and <2> where <3> is B's high side drive
;* 7) Winding C is driven by bits <5> and <4> where <5> is C's high side drive
;* 8) Three sensor bits constitute the address offset to the drive table
;* 9) A sensor bit transitions from a 0 to 1 at the moment that the corresponding
;*    winding's high side forward drive begins.
;* 10) Sensor bit <0> corresponds to winding A
;* 11) Sensor bit <1> corresponds to winding B
;* 12) Sensor bit <2> corresponds to winding C
;*
FwdTable
           retlw    B'00000000'        ; invalid
           retlw    B'00010010'        ; phase 6
           retlw    B'00001001'        ; phase 4
           retlw    B'00011000'        ; phase 5
           retlw    B'00100100'        ; phase 2
           retlw    B'00000110'        ; phase 1
           retlw    B'00100001'        ; phase 3
           retlw    B'00000000'        ; invalid
RevTable
           retlw    B'00000000'        ; invalid
           retlw    B'00100001'        ; phase /6
           retlw    B'00000110'        ; phase /4
           retlw    B'00100100'        ; phase /5
           retlw    B'00011000'        ; phase /2
           retlw    B'00001001'        ; phase /1
           retlw    B'00010010'        ; phase /3
           retlw    B'00000000'        ; invalid

           END                         ; directive 'end of program'
```

## APPENDIX D: SENSORLESS CODE

```
;**********************************************************************
;                                                                     *
;   Filename:          snsrless.asm                                   *
;   Date:              14 Jan. 2002                                   *
;   File Version:      1.0                                            *
;                                                                     *
;   Author:            W.R. Brown                                     *
;   Company:           Microchip Technology Incorporated              *
;                                                                     *
;                                                                     *
;**********************************************************************
;                                                                     *
;   Files required:    p16f877.inc                                    *
;                                                                     *
;                                                                     *
;                                                                     *
;**********************************************************************
;                                                                     *
;   Notes: Sensorless brushless motor control                         *
;                                                                     *
;   Closed loop 3 phase brushless DC motor control.                   *
;   Two potentiometers control operation. One potentiometer (A0)      *
;   controls PWM (voltage) and RPM (from table). The other            *
;   potentiometer (A1) provides a PWM offset to the PWM derived       *
;   from A0. Phase A motor terminal is connected via voltage          *
;   divider to A3. This is read while the drive is on during          *
;   phase 4. The result is the peak applied voltage (Vsupply).        *
;   A3 is also read while the drive is on at two times during         *
;   phase 5. The result is the BEMF voltage. The BEMF voltage is      *
;   read at the quarter (t1) and mid (t2) points of the phase 5       *
;   period. BEMF is compared to VSupply/2. If BEMF is above           *
;   VSupply/2 at t1 and below VSupply/2w at t2 then no speed          *
;   adjustment is made. If BEMF is high at both t1 and t2 then        *
;   the speed is reduced. If BEMF is low at t1 and t2 then the        *
;   speed is increased.                                               *
;                                                                     *
;**********************************************************************
;
    list P = PIC16F877
    include "p16f877.inc"
    __CONFIG _CP_OFF & _WRT_ENABLE_OFF & _HS_OSC & _WDT_OFF & _PWRTE_ON & _BODEN_ON

; Acceleration/Deceleration Time = RampRate * 256 * 256 * Timer0Timer0 prescale / Fosc

#define     AccelDelay      D'100'            ; determines full range acceleration time
#define     DecelDelay      D'10'             ; determines full range deceleration time

#define     ManThresh       0x3f              ; Manual threshold is the PWM potentiomenter
                                              ; reading above which RPM is adjusted automatically
#define     AutoThresh      0x100-ManThresh
```

```
OffMask     equu            B'11010101'       ; PWM off kills the high drives
Invalid     equ             B'00000000'       ; invalid
Phase1      equ             B'00100001'       ; phase 1 C high, A low
Phase2      equ             B'00100100'       ; phase 2 C high, B low
Phase3      equ             B'00000110'       ; phase 3 A high, B low
Phase4      equ             B'00010010'       ; phase 4 A high, C low
Phase5      equ             B'00011000'       ; phase 5 B high, C low
Phase6      equ             B'00001001'       ; phase 6 B high, A low

#define     CARRY           STATUS,C
#define     ZERO            STATUS,Z
#define     subwl           sublw

;**********************************************************************************
;*
;*  Define I/O Ports
;*

#define         ReadIndicator    PORTB,0        ; diagnostic scope trigger for BEMF readings
#define         DrivePort        PORTC          ; motor drive and lock status

;**********************************************************************************
;*
;*              Define RAM variables
;*

                CBLOCK 0x20

                STATE            ; Machine state
                PWMThresh        ; PWM threshold
                PhaseIndx        ; Current motor phase index
                Drive            ; Motor drive word
                RPMIndex         ; RPM Index workspace
                ADCRPM           ; ADC RPM value
                ADCOffset        ; Delta offset to ADC PWM threshold
                PresetHi         ; speed control timer compare MS byte
                PresetLo         ; speed control timer compare LS byte
                Flags            ; general purpose flags
                Vsupply          ; Supply voltage ADC reading
                DeltaV1          ; Difference between expected and actual BEMF at T/4
                DeltaV2          ; Difference between expected and actual BEMF at T/2
                CCPSaveH         ; Storage for phase time when finding DeltaV
                CCPSaveL         ; Storage for phase time when finding DeltaV
                CCPT2H           ; Workspace for determining T/2 and T/4
                CCPT2L           ; Workspace for determining T/2 and T/4
                RampTimer        ; Timer0 post scaler for accel/decel ramp rate
                xCount           ; general purpose counter workspace
                Status           ; relative speed indicator status

                ENDC
```

```
;*********************************************************************************
;*
;*         Define Flags
;*

#define  DriveOnFlag    Flags,0          ; Flag for invoking drive disable mask when clear
#define  AutoRPM        Flags,1          ; RPM timer is adjusted automatically
;                       Flags,3          ; Undefined
#define  FullOnFlag     Flags,4          ; PWM threshold is set to maximum drive
#define  Tmr0Ovf        Flags,5          ; Timer0 overflow flag
#define  Tmr0Sync       Flags,6          ; Second Timer0 overflow flag
;                       Flags,7          ; undefined

#define  BEMF1Low       DeltaV1,7        ; BEMF1 is low if DeltaV1 is negative
#define  BEMF2Low       DeltaV2,7        ; BEMF2 is low if DeltaV2 is negative

;*********************************************************************************
;*
;*  Define State machine states and index numbers
;*

sRPMSetup      equu      D'0'             ; Wait for Phase1, Set ADC GO, RA1->ADC
sRPMRead       equ       sRPMSetup+1      ; Wait for ADC nDONE, Read ADC->RPM
sOffsetSetup   equ       sRPMRead+1       ; Wait for Phase2, Set ADC GO, RA3->ADC
sOffsetRead    equ       sOffsetSetup+1   ; Wait for ADC nDONE, Read ADC->ADCOffset
sVSetup        equ       sOffsetRead+1    ; Wait for Phase4, Drive On, wait 9 uSec, Set ADC GO
sVIdle         equ       sVSetup+1        ; Wait for Drive On, wait Tacq, set ADC GO
sVRead       equ      sVIdle+1        ; Wait for ADC nDONE, Read ADC->Vsupply
sBEMFSetup   equ      sVRead+1        ; Wait for Phase5, set Timer1 compare to half phase time
sBEMFIdle    equ      sBEMFSetup+1    ; Wait for Timer1 compare, Force Drive on and wait 9 uSec,
                                      ; Set ADC GO, RA0->ADC
sBEMFRead    equ      sBEMFIdle+1     ; Wait for ADC nDONE, Read ADC->Vbemf
sBEMF2Idle   equ      sBEMFRead+1     ; Wait for Timer1 compare, Force Drive on and wait 9 uSec,
                                      ; Set ADC GO, RA0->ADC
sBEMF2Read   equ      sBEMF2Idle+1    ; Wait for ADC nDONE, Read ADC->Vbemf

;*********************************************************************************
;*
;*  The ADC input is changed depending on the STATE
;*  Each STATE assumes a previous input selection and changes the selection
;*  by XORing the control register with the appropriate ADC input change mask
;*  defined here:
;*

ADC0to1      equ     B'00001000'        ; changes ADCON0<5:3> from 000 to 001
ADC1to3      equ     B'00010000'        ; changes ADCON0<5:3> from 001 to 011
ADC3to0      equ     B'00011000'        ; changes ADCON0<5:3> from 011 to 000

;*********************************************************************************
;*************************** PROGRAM STARTS HERE *********************************
;*********************************************************************************

       org        0x000
       nop
       goto       Initialize

       org        0x004
       bsf        Tmr0Ovf          ; Timer0 overflow flag used by accel/decel timer
       bsf        Tmr0Sync         ; Timer0 overflow flag used to synchronize code execution
       bcf        INTCON,T0IF
       retfie                      ;

Initialize
       clrf       PORTC            ; all drivers off
       clrf       PORTB
```

```
       banksel TRISA
; setup I/O
       clrf       TRISC            ; motor drivers on PORTC
       movlw      B'00001011'      ; A/D on RA0 (PWM), RA1 (Speed) and RA3 (BEMF)
       movwf      TRISA            ;
       movlw      B'11111110'      ; RB0 is locked indicator
       movwf      TRISB
; setup Timer0
       movlw      B'11010000'      ; Timer0: Fosc, 1:2
       movwf      OPTION_REG
       bsf        INTCON,T0IE      ; enable Timer0 interrupts
; Setup ADC
       movlw      B'00000100'      ; ADC left justified, AN0, AN1
       movwf      ADCON1

       banksel    PORTA
       movlw      B'10000001'      ; ADC clk = Fosc/32, AN0, ADC on
       movwf      ADCON0
; setup Timer 1
       movlw      B'00100001'      ; 1:4 prescale, internal clock, timer on
       movwf      T1CON
; setup Timer 1 compare
       movlw      0xFF             ; set compare to maximum count
       movwf      CCPR1L           ; LS compare register
       movwf      CCPR1H           ; MS compare register
       movlw      B'00001011'      ; Timer 1 compare mode, special event - clears timer1
       movwf      CCP1CON

; initialize RAM

           clrf      PWMThresh
           movlw     D'6'
           movwf     PhaseIndx
           clrf      Flags
           clrf      Status        ;
           clrf      STATE         ; LoopIdle->STATE
           bcf       INTCON,T0IF   ; ensure Timer0 overflow flag is cleared
           bsf       INTCON,GIE    ; enable interrupts

MainLoop
;*****************************************************************
;
;   PWM, Commutation, State machine loop
;
;*****************************************************************

           btfsc     PIR1,CCP1IF   ; time for phase change?
           call      Commutate     ; yes - change motor drive
PWM
           bsf       DriveOnFlag   ; pre-set flag
           btfsc     FullOnFlag    ; is PWM level at maximum?
           goto      PWM02         ; yes - only commutation is necessary

           movf      PWMThresh,w   ; get PWM threshold
           addwf     TMR0,w        ; compare to Timer0
           btfss     CARRY         ; drive is on if carry is set
           bcf       DriveOnFlag   ; timer has not reached threshold, disable drive

           call      DriveMotor    ; output drive word
PWM02
           call      LockTest
           call      StateMachine  ; service state machine
           goto      MainLoop      ; repeat loop
```

```
StateMachine
           movlw    SMTableEnd-SMTable-1  ; STATE table must have 2^n entries
           andwf    STATE,f         ; limit STATE index to state table
           movlw    high SMTable    ; get high byte of table address
           movwf    PCLATH          ; prepare for computed goto
           movlw    low SMTable     ; get low byte of table address
           addwf    STATE,w         ; add STATE index to table root
           btfsc    CARRY           ; test for page change in table
           incf     PCLATH,f        ; page change adjust
           movwf    PCL             ; jump into table

SMTable                             ; number of STATE table entries MUST be evenly divisible by 2
           goto     RPMSetup        ; Wait for Phase1, Set ADC GO, RA1->ADC, clear Timer0 overflow
           goto     RPMRead         ; Wait for ADC nDONE, Read ADC->RPM
           goto     OffsetSetup     ; Wait for Phase2, Set ADC GO, RA3->ADC
           goto     OffsetRead      ; Wait for ADC nDONE, Read ADC->ADCOffset
           goto     VSetup          ; Wait for Phase4
           goto     VIdle           ; Wait for Drive On, wait Tacq, set ADC GO
           goto     VRead           ; Wait for ADC nDONE, Read ADC->Vsupply
           goto     BEMFSetup       ; Wait for Phase5, set Timer1 compare to half phase time
           goto     BEMFIdle        ; When Timer1 compares force Drive on, Set ADC GO after Tacq,
RA0->ADC
           goto     BEMFRead        ; Wait for ADC nDONE, Read ADC->Vbemf
           goto     BEMF2Idle       ; When Timer1 compares force Drive on, Set ADC GO after Tacq,
RA0->ADC
           goto     BEMF2Read       ; Wait for ADC nDONE, Read ADC->Vbemf

; fill out table with InvalidStates to make number of table entries evenly divisible by 2

       goto     InvalidState   ; invalid state - reset state machine
       goto     InvalidState   ; invalid state - reset state machine
       goto     InvalidState   ; invalid state - reset state machine
       goto     InvalidState   ; invalid state - reset state machine
SMTableEnd

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
RPMSetup                       ; Wait for Phase1, Set ADC GO, RA1->ADC, clear Timer0 overflow

       movlw    Phase1         ; compare Phase1 word...
       xorwf    Drive,w        ; ...with current drive word
       btfss    ZERO           ; ZERO if equal
       return                  ; not Phase1 - remain in current STATE

       bsf      ADCON0,GO      ; start ADC
       movlw    ADC0to1        ; prepare to change ADC input
       xorwf    ADCON0,f       ; change from AN0 to AN1
       incf     STATE,f        ; next STATE
       bcf      Tmr0Sync       ; clear Timer0 overflow
       return                  ; back to Main Loop

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
RPMRead                        ; Wait for ADC nDONE, Read ADC->RPM

       btfsc    ADCON0,GO      ; is ADC conversion finished?
       return                  ; no - remain in current STATE

       movf     ADRESH,w       ; get ADC result
       movwf    ADCRPM         ; save in RPM

       incf     STATE,f        ; next STATE
       return                  ; back to Main Loop

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
OffsetSetup                     ; Wait for Phase2, Set ADC GO, RA3->ADC

        movlw    Phase2         ; compare Phase2 word...
        xorwf    Drive,w        ; ...with current drive word
        btfss    ZERO           ; ZERO if equal
        return                  ; not Phase2 - remain in current STATE

        bsf      ADCON0,GO      ; start ADC
        movlw    ADC1to3        ; prepare to change ADC input
        xorwf    ADCON0,f       ; change from AN1 to AN3
        incf     STATE,f        ; next STATE
        return                  ; back to Main Loop

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
OffsetRead                      ; Wait for ADC nDONE, Read ADC->ADCOffset

        btfsc    ADCON0,GO      ; is ADC conversion finished?
        return                  ; no - remain in current STATE

        movf     ADRESH,w       ; get ADC result
        xorlw    H'80'          ; complement MSB for +/- offset
        movwf    ADCOffset      ; save in offset
        addwf    ADCRPM,w       ; add offset to PWM result
        btfss    ADCOffset,7    ; is offset a negative number?
        goto     OverflowTest   ; no - test for overflow

        btfss    CARRY          ; underflow?
        andlw    H'00'          ; yes - force minimum
        goto     Threshold      ;

OverflowTest
            btfsc    CARRY          ; overflow?
            movlw    H'ff'          ; yes - force maximum

Threshold
            movwf    PWMThresh      ; PWM threshold is RPM result plus offset
            btfsc    ZERO           ; is drive off?
            goto     DriveOff       ; yes - skip voltage measurements

            bcf      FullOnFlag     ; pre-clear flag in preparation of compare
            sublw    0xFD           ; full on threshold
            btfss    CARRY          ; CY = 0 if PWMThresh > FullOn
            bsf      FullOnFlag     ; set full on flag
            incf     STATE,f        ; next STATE
            return                  ; back to Main Loop

DriveOff
            clrf     Status         ; clear speed indicators
            movlw    B'11000111'    ; reset ADC input to AN0
            andwf    ADCON0,f       ;
            clrf     STATE          ; reset state machine
            return

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
VSetup                              ; Wait for Phase4

            movlw    Phase4         ; compare Phase4 word...
            xorwf    Drive,w        ; ...with current Phase drive word
            btfss    ZERO           ; ZERO if equal
            return                  ; not Phase4 - remain in current STATE

            call     SetTimer       ; set timer value from RPM table
            incf     STATE,f        ; next STATE
            return                  ; back to Main Loop

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
VIdle                               ; Wait for Drive On, wait Tacq, set ADC GO

           btfss    DriveOnFlag     ; is Drive active?
           return                   ; no - remain in current STATE

           call     Tacq            ; motor Drive is active - wait ADC Tacq time
           bsf      ADCON0,GO       ; start ADC
           incf     STATE,f         ; next STATE
           return                   ; back to Main Loop

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
VRead                               ; Wait for ADC nDONE, Read ADC->Vsupply

           btfsc    ADCON0,GO       ; is ADC conversion finished?
           return                   ; no - remain in current STATE

           movf     ADRESH,w        ; get ADC result
           movwf    Vsupply         ; save as supply voltage
           incf     STATE,f         ; next STATE
           bcf      Tmr0Sync        ; clear Timer0 overflow
           return                   ; back to Main Loop

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
BEMFSetup                           ; Wait for Phase5, set Timer1 compare to half phase time

           movlw    Phase5          ; compare Phase5 word...
           xorwf    Drive,w         ; ...with current drive word
           btfss    ZERO            ; ZERO if equal
       return                   ; not Phase5 - remain in current STATE

       btfss    Tmr0Sync        ; synchronize with Timer0
       return                   ;

       btfss    PWMThresh,7     ; if PWMThresh > 0x80 then ON is longer than OFF
       goto     BEMFS1          ; OFF is longer and motor is currently off - compute now

       btfss    DriveOnFlag     ; ON is longer - wait for drive cycle to start
       return                   ; not started - wait

BEMFS1
       bcf      CCP1CON,0       ; disable special event on compare
       movf     CCPR1H,w        ; save current capture compare state
       movwf    CCPSaveH        ;
       movwf    CCPT2H          ; save copy in workspace
       movf     CCPR1L,w        ; low byte
       movwf    CCPSaveL        ; save
       movwf    CCPT2L          ; and save copy
       bcf      CARRY           ; pre-clear carry for rotate
       rrf      CCPT2H,f        ; divide phase time by 2
       rrf      CCPT2L,f        ;
       bcf      CARRY           ; pre-clear carry
       rrf      CCPT2H,w        ; divide phase time by another 2
       movwf    CCPR1H          ; first BEMF reading at phase T/4
       rrf      CCPT2L,w        ;
       movwf    CCPR1L          ;

       incf     STATE,f         ; next STATE
       return                   ; back to Main Loop

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
BEMFIdle                        ; When Timer1 compares force Drive on, Set ADC GO after Tacq, RA0-
>ADC

       btfss    PIR1,CCP1IF    ; timer compare?
       return                  ; no - remain in current STATE

       bsf      DriveOnFlag    ; force drive on for BEMF reading
       call     DriveMotor     ; activate motor drive
       bsf      ReadIndicator  ; Diagnostic
       call     Tacq           ; wait ADC acquisition time
       bsf      ADCON0,GO      ; start ADC
       bcf      ReadIndicator  ; Diagnostic

; setup to capture BEMF at phase 3/4 T

       movf     CCPT2H,w
       addwf    CCPR1H,f       ; next compare at phase 3/4 T
       movf     CCPT2L,w       ;
       addwf    CCPR1L,f       ; set T/2 lsb
       btfsc    CARRY          ; test for carry into MSb
       incf     CCPR1H,f       ; perform carry
       bcf      PIR1,CCP1IF    ; clear timer compare interrupt flag
       incf     STATE,f        ; next STATE
       return                  ; back to Main Loop

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
BEMFRead                        ; Wait for ADC nDONE, Read ADC->Vbemf

       btfsc    ADCON0,GO      ; is ADC conversion finished?
       return                  ; no - remain in current STATE

       rrf      Vsupply,w      ; divide supply voltage by 2
       subwf    ADRESH,w       ; Vbemf - Vsupply/2

       movwf    DeltaV1        ; save error voltage
       incf     STATE,f        ; next STATE
       return                  ; back to Main Loop

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
BEMF2Idle                       ; When Timer1 compares force Drive on, Set ADC GO after Tacq, RA0-
>ADC

       btfss    PIR1,CCP1IF    ; timer compare?
       return                  ; no - remain in current STATE

       bsf      DriveOnFlag    ; force drive on for BEMF reading
       call     DriveMotor     ; activate motor drive
       bsf      ReadIndicator  ; Diagnostic
       call     Tacq           ; wait ADC acquisition time
       bsf      ADCON0,GO      ; start ADC
       bcf      ReadIndicator  ; Diagnostic
       movlw    ADC3to0        ; prepare to change ADC input
       xorwf    ADCON0,f       ; change from AN3 to AN0

; restore Timer1 phase time and special event compare mode

       movf     CCPSaveH,w
       movwf    CCPR1H         ; next compare at phase T
       movf     CCPSaveL,w     ;
       movwf    CCPR1L         ; set T lsb
       bcf      PIR1,CCP1IF    ; clear timer compare interrupt flag
       bsf      CCP1CON,0      ; enable special event on compare
       incf     STATE,f        ; next STATE
       return                  ; back to Main Loop

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
BEMF2Read                         ; Wait for ADC nDONE, Read ADC->Vbemf

        btfsc   ADCON0,GO         ; is ADC conversion finished?
        return                    ; no - remain in current STATE

        rrf     Vsupply,w         ; divide supply voltage by 2
        subwf   ADRESH,w          ; Vbemf - Vsupply/2

        movwf   DeltaV2           ; save error voltage

        clrf    STATE             ; reset state machine to beginning
        return                    ; back to Main Loop

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
InvalidState                      ; trap for invalid STATE index
        movlw   B'11000111'       ; reset ADC input to AN0
        andwf   ADCON0,f          ;
        clrf    STATE
        return
;____________________________________________________________________________________________________

Tacq
;******************************************************************
;
;       Software delay for ADC acquisition time
;       Delay time = Tosc*(3+3*xCount)
;
;******************************************************************

        movlw   D'14              ; 14 equates to approx 9 uSec delay
        movwf   xCount            ;
        decfsz  xCount,f          ;
        goto    $-1               ; loop here until time complete
        return

LockTest
;******************************************************************
;
;       T is the commutation phase period. Back EMF is measured on the
;       floating motor terminal at two times during T to determine
;       the approximate zero crossing of the BEMF. BEMF low means that
;       the measured BEMF is below (supply voltage)/2.
;       If BEMF is low at 1/4 T then accelerate.
;       If BEMF is high at 1/4 T and low at 3/4 T then speed is OK.
;       If BEMF is high at 1/4 T and 3/4 T then decelerate.
;
;       Lock test computation is synchronized to the PWM clock such
;       that the computation is performed during the PWM ON or OFF
;       time whichever is longer.
;
;******************************************************************

; synchronize test with start of Timer0

        btfss   Tmr0Ovf           ; has Timer0 wrapped around?
        return                    ; no - skip lock test

        btfss   PWMThresh,7       ; if PWMThresh > 0x80 then ON is longer than OFF
        goto    LT05              ; OFF is longer and motor is currently off - compute now

        btfss   DriveOnFlag       ; ON is longer - wait for drive cycle to start
        return                    ; not started - wait
```

```
LT05
        bcf       Tmr0Ovf          ; clear synchronization flag
        decfsz    RampTimer,f      ; RampTimer controls the acceleration/deceleration rate
        return

; use lock results to control RPM only if not manual mode

        bsf       AutoRPM          ; preset flag
        movf      ADCRPM,w         ; compare RPM potentiometer...
        addlw     AutoThresh       ; ...to the auto control threshold
        btfss     CARRY            ; CARRY is set if RPM is > auto threshold
        bcf       AutoRPM          ; not in auto range - reset flag

        btfss     BEMF1Low         ; is first BEMF below Supply/2
        goto      LT20             ; no - test second BEMF

LT10
; accelerate if BEMF at 1/4 T is below Supply/2

        movlw     B'10000000'      ; indicate lock test results
        movwf     Status           ; status is OR'd with drive word later
        movlw     AccelDelay       ; set the timer for acceleration delay
        movwf     RampTimer        ;

        btfss     AutoRPM          ; is RPM in auto range?
        goto      ManControl       ; no - skip RPM adjustment

        incfsz    RPMIndex,f       ; increment the RPM table index
        return                     ; return if Index didn't wrap around

        decf      RPMIndex,f       ; top limit is 0xFF
        return
LT20
        btfsc     BEMF2Low         ; BEMF1 was high...
        goto      ShowLocked       ; ... and BEMF2 is low - show locked

; decelerate if BEMF at 3/4 T is above Supply/2

        movlw     B'01000000'      ; indicate lock test results
        movwf     Status           ; status is OR'd with drive word later
        movlw     DecelDelay       ; set the timer for deceleration delay
        movwf     RampTimer        ;

        btfss     AutoRPM          ; is RPM in auto range?
        goto      ManControl       ; no - skip RPM adjustment

        decfsz    RPMIndex,f       ; set next lower RPM table index
        return                     ; return if index didn't wrap around

        incf      RPMIndex,f       ; bottom limit is 0x01
        return

ShowLocked
        movlw     B'11000000'      ; indicate lock test results
        movwf     Status           ; status is OR'd with drive word later
        movlw     DecelDelay       ; set the timer for deceleration delay
        movwf     RampTimer        ;

        btfsc     AutoRPM          ; was RPM set automatically?
        return                     ; yes - we're done
```

```
ManControl
        movf      ADCRPM,w        ; get RPM potentiometer reading...
        movwf     RPMIndex        ; ...and set table index directly
        return

Commutate
;*****************************************************************
;
;       Commutation is triggered by PIR1<CCP1IF> flag.
;       This flag is set when timer1 equals the compare register.
;       When BEMF measurement is active the compare time is not
;       cleared automatically (special event trigger is off).
;        Ignore the PIR1<CCP1IF> flag when special trigger is off
;       because the flag is for BEMF measurement.
;       If BEMF measurement is not active then decrement phase table
;       index and get the drive word from the table. Save the
;       drive word in a global variable and output to motor drivers.
;
;*****************************************************************

        btfss     CCP1CON,0       ; is special event on compare enabled?
        return                    ; no - this is a BEMF measurement, let state machine handle this

        bcf       PIR1,CCP1IF     ; clear interrupt flag

        movlw     high OnTable    ; set upper program counter bits
        movwf     PCLATH
        decfsz    PhaseIndx,w     ; decrement to next phase
        goto      $+2             ; skip reset if not zero
        movlw     D'6'            ; phase counts 6 to 1
        movwf     PhaseIndx       ; save the phase index
        addlw     LOW OnTable
        btfsc     CARRY           ; test for possible page boundary
        incf      PCLATH,f        ; page boundary adjust
        call      GetDrive
        movwf     Drive           ; save motor drive word
DriveMotor
        movf      Drive,w         ; restore motor drive word
        btfss     DriveOnFlag     ; test drive enable flag
        andlw     OffMask         ; kill high drive if PWM is off
        iorwf     Status,w        ; show speed indicators
        movwf     DrivePort       ; output to motor drivers
        return

GetDrive
        movwf     PCL             ; computed goto
OnTable
        retlw     Invalid
        retlw     Phase6
        retlw     Phase5
        retlw     Phase4
        retlw     Phase3
        retlw     Phase2
        retlw     Phase1
        retlw     Invalid

SetTimer
```

```
;****************************************************************
;
;       This sets the CCP module compare registers for timer 1.
;       The motor phase period is the time it takes timer 1
;       to count from 0 to the compare value. The CCP module
;       is configured to clear timer 1 when the compare occurs.
;       Get the timer1 compare variable from two lookup tables, one
;       for the compare high byte and the other for the low byte.
;
;****************************************************************

        call     SetTimerHigh
        movwf    CCPR1H              ; Timer1 High byte preset
        call     SetTimerLow
        movwf    CCPR1L              ; Timer1 Low byte preset
        return

SetTimerHigh
        movlw    high T1HighTable    ; lookup preset values
        movwf    PCLATH              ; high bytes first
        movlw    low T1HighTable     ;
        addwf    RPMIndex,w          ; add table index
        btfsc    STATUS,C            ; test for table page crossing
        incf     PCLATH,f            ;
        movwf    PCL                 ; lookup - result returned in W
SetTimerLow
        movlw    high T1LowTable     ; repeat for lower byte
        movwf    PCLATH              ;
        movlw    low T1LowTable      ;
        addwf    RPMIndex,w          ; add table index
        btfsc    STATUS,C            ; test for table page crossing
        incf     PCLATH,f            ;
        movwf    PCL                 ; lookup - result returned in W

#include "BLDCspd4.inc"

        end
```

**Note the following details of the code protection feature on PICmicro® MCUs.**

- The PICmicro family meets the specifications contained in the Microchip Data Sheet.
- Microchip believes that its family of PICmicro microcontrollers is one of the most secure products of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the PICmicro microcontroller in a manner outside the operating specifications contained in the data sheet. The person doing so may be engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as “unbreakable”.
- Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our product.

If you have any further questions about this matter, please contact the local sales office nearest to you.



DNV Certification, Inc. USA
ANSI-RAB QMS ACCREDITED
DNV MSC The Netherlands Accredited by the RvA
ISO 9001 / QS-9000 REGISTERED FIRM

*Microchip received QS-9000 quality system certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona in July 1999 and Mountain View, California in March 2002. The Company’s quality system processes and procedures are QS-9000 compliant for its PICmicro® 8-bit MCUs, KEELOQ® code hopping devices, Serial EEPROMs, microperipherals, non-volatile memory and analog products. In addition, Microchip’s quality system for the design and manufacture of development systems is ISO 9001 certified.*

# WORLDWIDE SALES AND SERVICE

## AMERICAS

**Corporate Office**
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200 Fax: 480-792-7277
Technical Support: 480-792-7627
Web Address: http://www.microchip.com

**Rocky Mountain**
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7966 Fax: 480-792-4338

**Atlanta**
500 Sugar Mill Road, Suite 200B
Atlanta, GA 30350
Tel: 770-640-0034 Fax: 770-640-0307

**Boston**
2 Lan Drive, Suite 120
Westford, MA 01886
Tel: 978-692-3848 Fax: 978-692-3821

**Chicago**
333 Pierce Road, Suite 180
Itasca, IL 60143
Tel: 630-285-0071 Fax: 630-285-0075

**Dallas**
4570 Westgrove Drive, Suite 160
Addison, TX 75001
Tel: 972-818-7423 Fax: 972-818-2924

**Detroit**
Tri-Atria Office Building
32255 Northwestern Highway, Suite 190
Farmington Hills, MI 48334
Tel: 248-538-2250 Fax: 248-538-2260

**Kokomo**
2767 S. Albright Road
Kokomo, Indiana 46902
Tel: 765-864-8360 Fax: 765-864-8387

**Los Angeles**
18201 Von Karman, Suite 1090
Irvine, CA 92612
Tel: 949-263-1888 Fax: 949-263-1338

**New York**
150 Motor Parkway, Suite 202
Hauppauge, NY 11788
Tel: 631-273-5305 Fax: 631-273-5335

**San Jose**
Microchip Technology Inc.
2107 North First Street, Suite 590
San Jose, CA 95131
Tel: 408-436-7950 Fax: 408-436-7955

**Toronto**
6285 Northam Drive, Suite 108
Mississauga, Ontario L4V 1X5, Canada
Tel: 905-673-0699 Fax: 905-673-6509

## ASIA/PACIFIC

**Australia**
Microchip Technology Australia Pty Ltd
Suite 22, 41 Rawson Street
Epping 2121, NSW
Australia
Tel: 61-2-9868-6733 Fax: 61-2-9868-6755

**China - Beijing**
Microchip Technology Consulting (Shanghai) Co., Ltd., Beijing Liaison Office
Unit 915
Bei Hai Wan Tai Bldg.
No. 6 Chaoyangmen Beidajie
Beijing, 100027, No. China
Tel: 86-10-85282100 Fax: 86-10-85282104

**China - Chengdu**
Microchip Technology Consulting (Shanghai) Co., Ltd., Chengdu Liaison Office
Rm. 2401, 24th Floor,
Ming Xing Financial Tower
No. 88 TIDU Street
Chengdu 610016, China
Tel: 86-28-86766200 Fax: 86-28-86766599

**China - Fuzhou**
Microchip Technology Consulting (Shanghai) Co., Ltd., Fuzhou Liaison Office
Unit 28F, World Trade Plaza
No. 71 Wusi Road
Fuzhou 350001, China
Tel: 86-591-7503506 Fax: 86-591-7503521

**China - Shanghai**
Microchip Technology Consulting (Shanghai) Co., Ltd.
Room 701, Bldg. B
Far East International Plaza
No. 317 Xian Xia Road
Shanghai, 200051
Tel: 86-21-6275-5700 Fax: 86-21-6275-5060

**China - Shenzhen**
Microchip Technology Consulting (Shanghai) Co., Ltd., Shenzhen Liaison Office
Rm. 1315, 13/F, Shenzhen Kerry Centre,
Renminnan Lu
Shenzhen 518001, China
Tel: 86-755-82350361 Fax: 86-755-82366086

**China - Hong Kong SAR**
Microchip Technology Hongkong Ltd.
Unit 901-6, Tower 2, Metroplaza
223 Hing Fong Road
Kwai Fong, N.T., Hong Kong
Tel: 852-2401-1200 Fax: 852-2401-3431

**India**
Microchip Technology Inc.
India Liaison Office
Divyasree Chambers
1 Floor, Wing A (A3/A4)
No. 11, O'Shaugnessey Road
Bangalore, 560 025, India
Tel: 91-80-2290061 Fax: 91-80-2290062

**Japan**
Microchip Technology Japan K.K.
Benex S-1 6F
3-18-20, Shinyokohama
Kohoku-Ku, Yokohama-shi
Kanagawa, 222-0033, Japan
Tel: 81-45-471- 6166 Fax: 81-45-471-6122

**Korea**
Microchip Technology Korea
168-1, Youngbo Bldg. 3 Floor
Samsung-Dong, Kangnam-Ku
Seoul, Korea 135-882
Tel: 82-2-554-7200 Fax: 82-2-558-5934

**Singapore**
Microchip Technology Singapore Pte Ltd.
200 Middle Road
#07-02 Prime Centre
Singapore, 188980
Tel: 65-6334-8870 Fax: 65-6334-8850

**Taiwan**
Microchip Technology (Barbados) Inc., Taiwan Branch
11F-3, No. 207
Tung Hua North Road
Taipei, 105, Taiwan
Tel: 886-2-2717-7175 Fax: 886-2-2545-0139

## EUROPE

**Austria**
Microchip Technology Austria GmbH
Durisolstrasse 2
A-4600 Wels
Austria
Tel: 43-7242-2244-399
Fax: 43-7242-2244-393

**Denmark**
Microchip Technology Nordic ApS
Regus Business Centre
Lautrup hoj 1-3
Ballerup DK-2750 Denmark
Tel: 45 4420 9895 Fax: 45 4420 9910

**France**
Microchip Technology SARL
Parc d'Activite du Moulin de Massy
43 Rue du Saule Trapu
Batiment A - ler Etage
91300 Massy, France
Tel: 33-1-69-53-63-20 Fax: 33-1-69-30-90-79

**Germany**
Microchip Technology GmbH
Steinheilstrasse 10
D-85737 Ismaning, Germany
Tel: 49-89-627-144 0 Fax: 49-89-627-144-44

**Italy**
Microchip Technology SRL
Centro Direzionale Colleoni
Palazzo Taurus 1 V. Le Colleoni 1
20041 Agrate Brianza
Milan, Italy
Tel: 39-039-65791-1 Fax: 39-039-6899883

**United Kingdom**
Microchip Ltd.
505 Eskdale Road
Winnersh Triangle
Wokingham
Berkshire, England RG41 5TU
Tel: 44 118 921 5869 Fax: 44-118 921-5820

08/01/02