

Stepper Motor Microstepping with PIC18C452

Authors: Padmaraja Yedamale
 Sandip Chattopadhyay
 Microchip Technology Inc.

INTRODUCTION

A stepper motor, as its name suggests, moves one step at a time, unlike those conventional motors, which spin continuously. If we command a stepper motor to move some specific number of steps, it rotates incrementally that many number of steps and stops. Because of this basic nature of a stepper motor, it is widely used in low cost, open loop position control systems. Open loop control means no feedback information about the position is needed. This eliminates the need for expensive sensing and feedback devices, such as optical encoders. Motor position is known simply by keeping track of the number of input step pulses.

STEPPER MOTOR BASICS

Now let's take a closer look at a stepper motor. The first thing that we notice is that it has more than two wires leading into it. In fact, various versions have four, five, six, and sometimes more wires. Also, when we manually rotate the shaft, we get a 'notched' feeling. The simplest way to think about a stepper motor is as a bar magnet that pivots about its center with four individual, but exactly identical electromagnets, as shown in Figure 1A. If we manually rotate the magnet without energizing any coils, we get the 'notched' feeling whenever a relatively larger magnetic force is generated, because of the alignment of the permanent magnet with the core of the electromagnets, as in Figure 1A. This force is termed 'detent torque'. Let's assume that the initial position of the magnetic rotor is as shown in Figure 1A. Now turn on coil A; i.e., flow current through it to create an electromagnet, as shown in Figure 1B. The motor does not rotate, but we cannot move it freely by hand (more torque has to be applied to move it now), because of a larger 'holding torque'. This torque is generated by the attraction of the north and south poles of the rotor magnet and the electromagnet produced in the stator by the current.

FIGURE 1: NON-ENERGIZED AND CLOCKWISE CURRENT IN COIL A

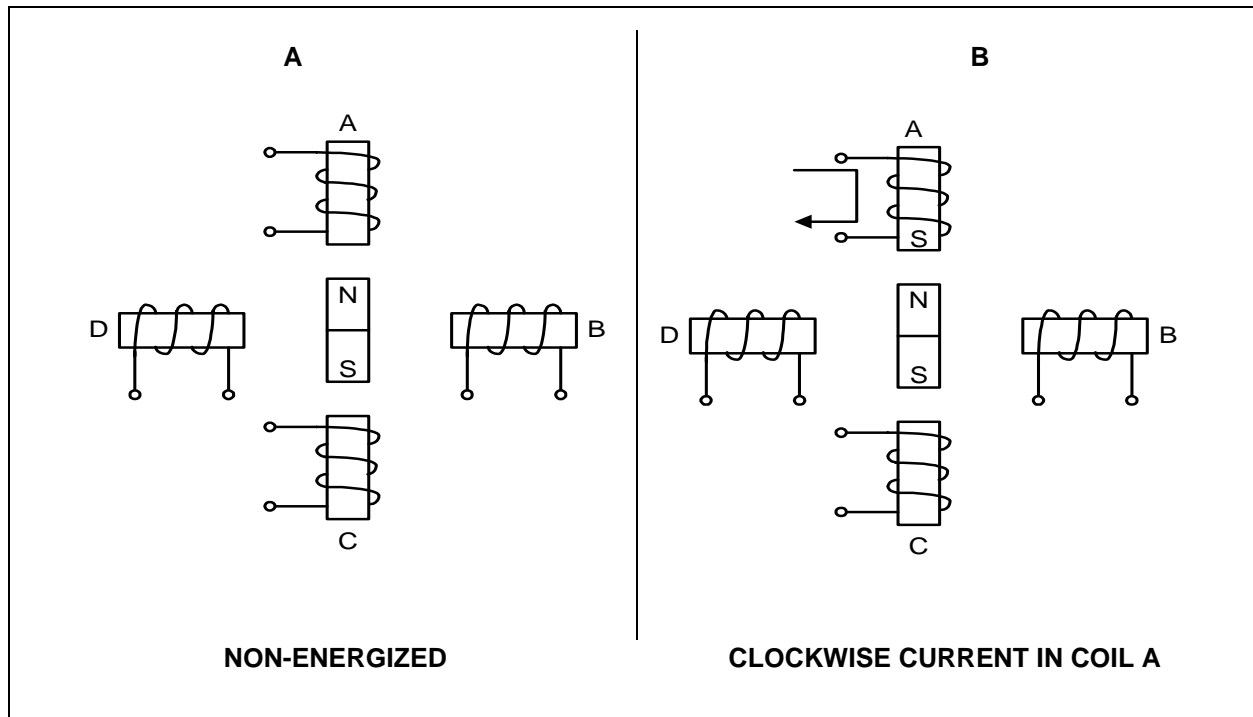
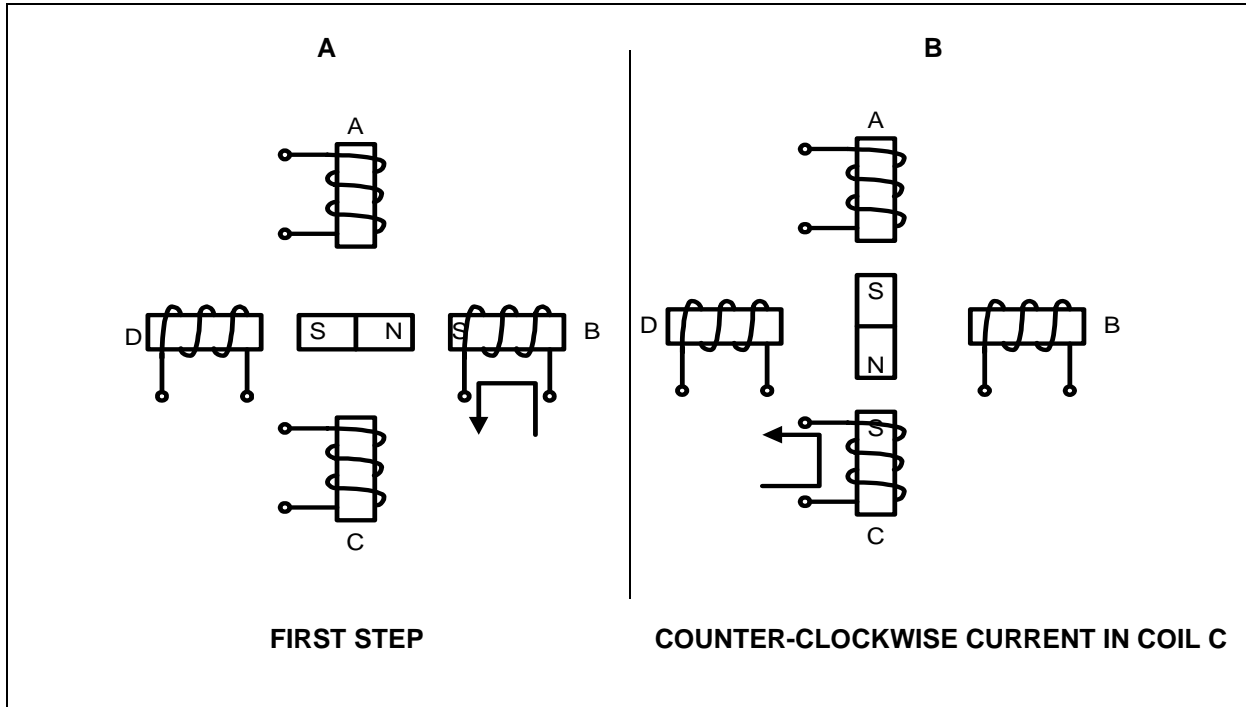


FIGURE 2: FIRST STEP MOVEMENT AND NEXT STEP



To move the motor in a clockwise direction from its initial stop position, we need to generate torque in the clockwise direction. This is done by turning off coil A, and turning on coil B. The electromagnet in coil B pulls the magnetized rotor and the rotor aligns itself with coil B, as shown in Figure 2A. Turning off coil B and turning on coil C will move the rotor one step further, as shown in Figure 2B.

Comparing Figure 1B and Figure 2B, we understand that the direction of current flow in coil C is exactly opposite to the direction of flow in coil A. This is required to generate an electromagnet of correct polarity, which will pull the rotor in the clockwise direction. By the same logic, the direction of current in coil D will be opposite to coil B when the rotor takes the next step (due to turning off coil C and turning on coil D).

A 360 degree rotation of the rotor will be completed if you turn off coil D and turn on coil A. The coil operation sequence (B, C, D, A), described is responsible for the clockwise rotation of the motor. The rotor will move counter-clockwise from its initial position at Figure 1B if we follow the opposite sequence (D, C, B, A).

UNIPOLAR AND BIPOLAR

Two leads on each of the four coils of a stepper motor can be brought out in different ways. All eight leads can be taken out of the motor separately. Alternatively, connecting A and C together, and B and D together, as shown in Figure 3, can form two coils. Leads of these two windings can be brought out of the motor in three different ways, as shown in Figure 3, Figure 4, and Figure 5.

If the coil ends are brought out as shown in Figure 3, then the motor is called a bipolar motor, and if the wires are brought out as shown in Figure 4 or Figure 5, with one or two center tap(s), it is called a unipolar motor.

FIGURE 3: BIPOLAR (4-WIRE)

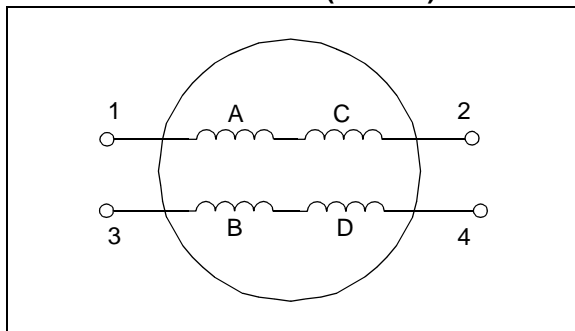


FIGURE 4: UNIPOLAR (5-WIRE)

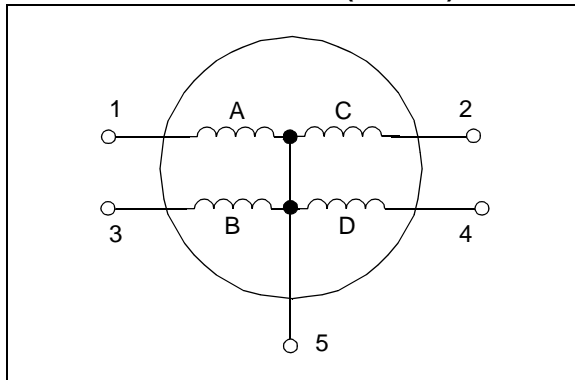
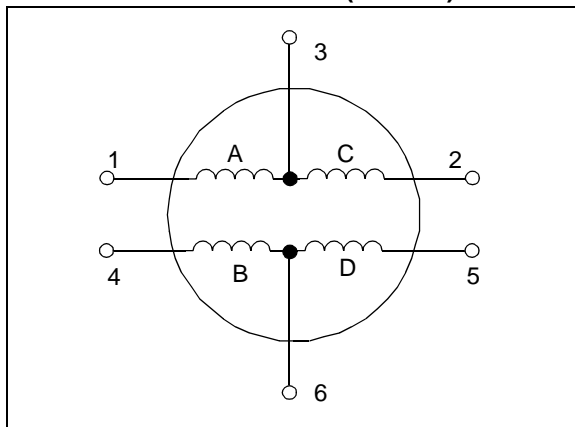


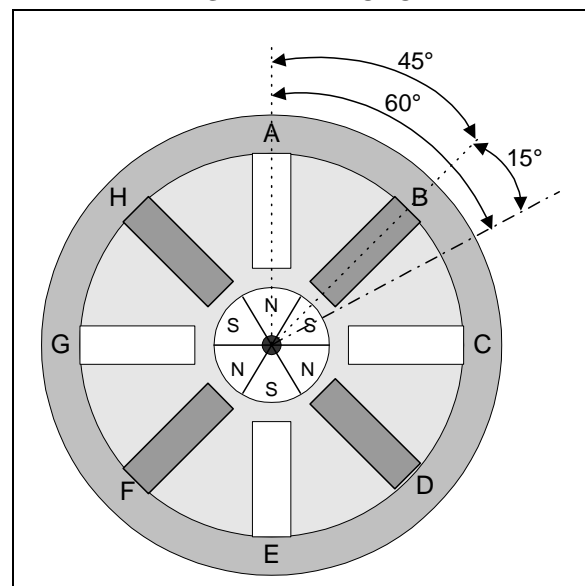
FIGURE 5: UNIPOLAR (6-WIRE)



AN ACTUAL PERMANENT MAGNET (PM) STEPPER MOTOR

The simple stepper motor described, moves in very coarse steps of 90 degrees. How do actual motors achieve movements as low as 7.5 degrees? The stator (the stationary electromagnets) of a real motor has more segments on it. A typical stator arrangement with eight stators is shown in Figure 6.

FIGURE 6: STATOR WINDING ARRANGEMENTS IN A PERMANENT MAGNET STEPPER MOTOR



The rotor is also different and a typical cylindrical rotor with 6 poles is shown in Figure 6. There are 45 degrees between each stator section and 60 degrees between each rotor pole. Using the principle of vernier mechanism, the actual movement of the rotor for each step is 60 minus 45 or 15 degrees. In this case, also, there are only two coils: one connects pole sections A, C, E and G, and the other connects B, D, F, H. Let us assume that current is flowing in a certain direction through the first coil only, and pole sections are wired in such a fashion that:

- A and C have S-polarity
- E and G have N-polarity

The rotor will be lined up accordingly, as shown in Figure 6. Let's say that we want the rotor to move 15 degrees clockwise. We would remove the current applied to the first winding and energize the second winding. The pole sections B, D, F, H are wired together with the second winding in such a way that:

- B and D have S-polarity
- F and H have N-polarity

In the next step, current through winding 2 is removed and reverse polarity current is applied in winding 1. This time A and C have N-polarity, and E and G have S-polarity; so the rotor will take a further 15 degree step in the clockwise direction. The principle of operation is the same as the basic stepper motor with a bar magnet as rotor and four individual electromagnets as stators, but in this construction, 15 degrees per step is achieved. Different 'step angles' (i.e., angular displacement in degrees per step) can be obtained by varying the design with different numbers of stators and rotor poles. In an actual motor, both rotor and stators are cylindrical, as shown in Figure 7. This type of motor is called a permanent magnet (PM) stepper because the rotor is a permanent magnet. These are low cost motors with typical step angles of 7.5 degrees to 15 degrees.

VARIABLE RELUCTANCE (VR) STEPPER MOTOR

There is a type of motor where the rotor is not cylindrical, but looks like bars with a number of teeth on it, as shown in Figure 8. The rotor teeth are made of soft iron. The electromagnet produced by activating stator coils in sequence, attracts the metal bar (rotor) towards the minimum reluctance path in the magnetic circuit. We don't get a notched feeling when we try to rotate it manually in the non-energized condition. In the non-energized condition, there is no magnetic flux in the air gap, as the stator is an electromagnet and the rotor is a piece of soft iron; hence, there is no detent torque. This type of stepper motor is called a variable reluctance stepper (VR). The motor shown in Figure 8 has four rotor teeth, 90 degrees apart and six stator poles, 60 degrees apart. So when the windings are energized in a reoccurring sequence of 2, 3, 1, and so on, the motor will rotate in a 30 degree step angle. These motors provide less holding torque at standstill compared to the PM type, but the dynamic torque characteristics are better.

Variable reluctance motors are normally constructed with three or five stator windings, as opposed to the two windings in the PM motors.

FIGURE 7: A BIPOLAR PERMANENT MAGNET STEPPER MOTOR

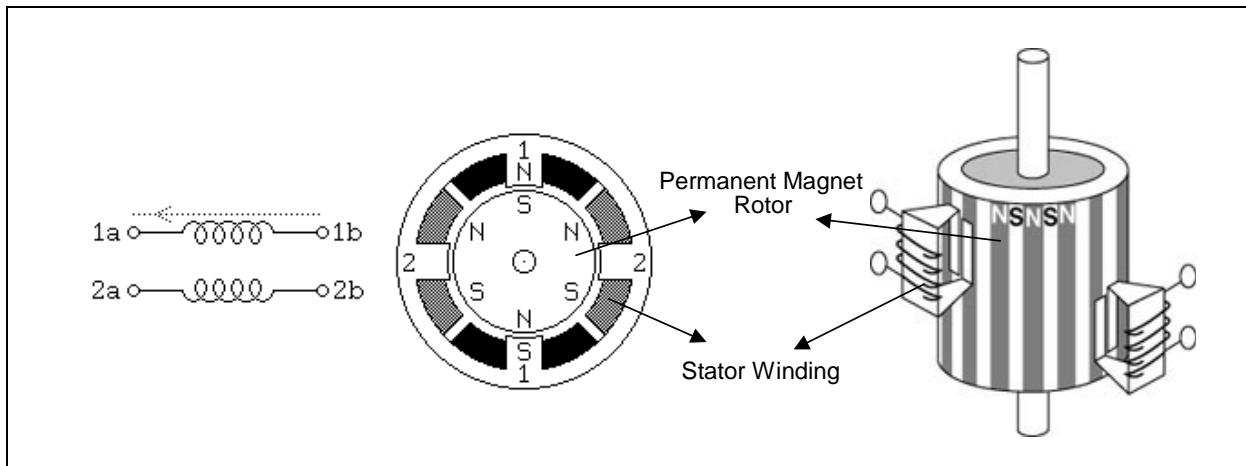
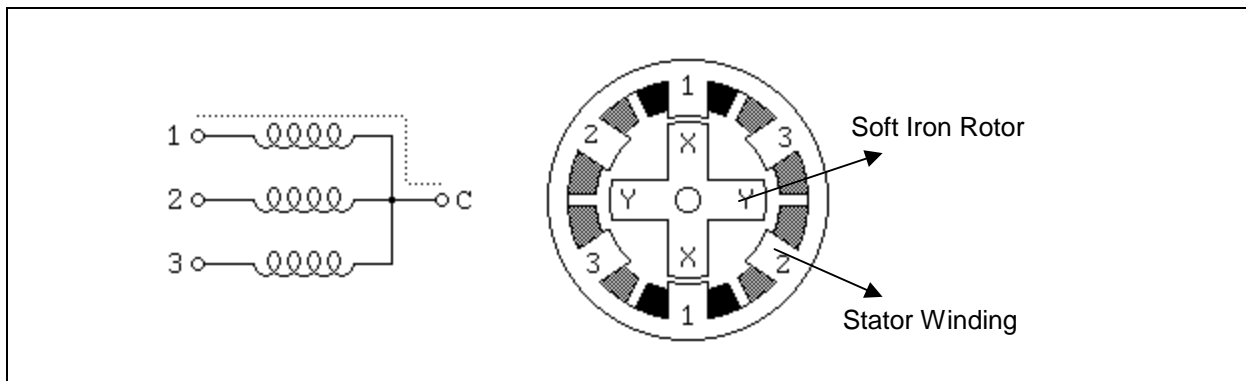


FIGURE 8: A VARIABLE RELUCTANCE MOTOR



HYBRID (HB) STEPPER MOTOR

Construction of permanent magnet motors becomes very complex below 7.5 degrees step angles. Smaller step angles can be realized by combining the variable reluctance motor and the permanent magnet motor principles. Such motors are called hybrid motors (HB), which give much smaller step angles, as small as 0.9 degrees per step.

A typical hybrid motor is shown in Figure 9. The stator construction is similar to the permanent magnet motor, and the rotor is cylindrical and magnetized like the PM motor with multiple teeth like a VR motor. The teeth on the rotor provide a better path for the flux to flow through the preferred locations in the air gap. This increases the detent, holding, and dynamic torque characteristics of the motor compared to the other two types of motors.

Hybrid motors have a smaller step angle compared to the permanent magnet motor, but they are very expensive. In low cost applications, the step angle of a permanent magnet motor is divided into smaller angles using better control techniques.

Permanent magnet motors and hybrid motors are more popular than the variable reluctance motor, and since the stator construction of these motors is very similar, a common control circuit can easily drive both types of motors.

HOW TO IDENTIFY THE PERMANENT MAGNET/HYBRID MOTOR LEADS

The color code of the wires coming out of the motor are not standard; however, using a multimeter/ohmmeter, it is easy to identify the winding ends and center tap.

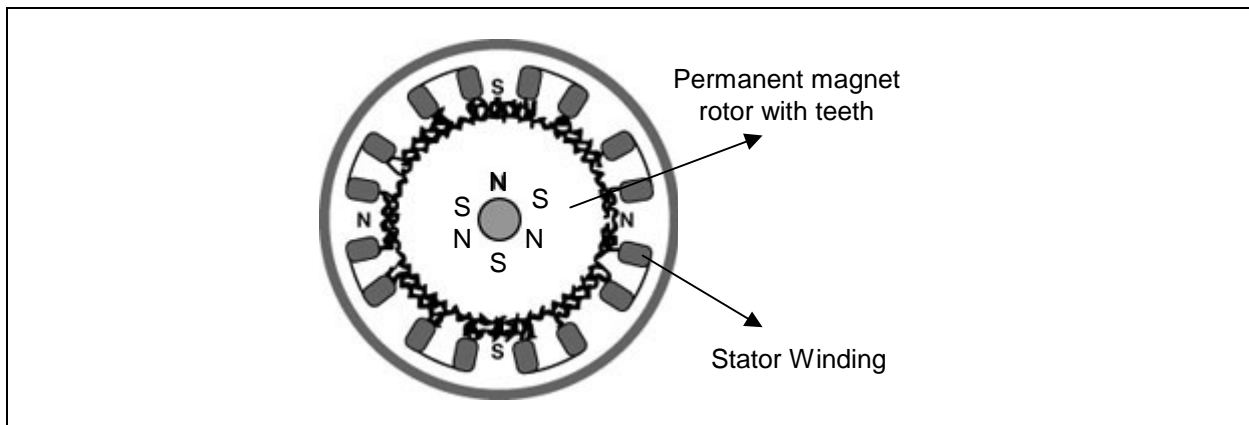
If only four leads are coming out of the motor, then the motor is a bipolar motor. If the resistance measured across two terminals, say terminals 1 and 2 in Figure 3, is finite, then those are ends of a coil. If the multimeter shows an open circuit (i.e., if you are trying to measure across the terminals 1 and 3, or 1 and 4, or 2 and 3, or 2 and 4), then the terminals are of different windings. Change your lead to another terminal and check again to find a finite resistance.

If there are five leads coming out of the motor, then the resistance across one terminal and all other terminals will be almost equal. This common terminal is the center tap and the other terminals are the ends of different windings. Figure 4 shows terminal 5 is the common terminal, while 1, 2, 3, and 4 are the ends of the windings.

In the case of a motor with six leads as in Figure 5, resistance across terminals 1 and 2 should be approximately double the resistance measured across terminals 1 and 3, and 2 and 3. The same is applicable for the other winding (the remaining 3 wires).

In all the above cases, once the terminals are identified, it is important to know the sequence in which the windings should be energized. This is done by energizing the terminals one after the other, by rated voltage. If the motor smoothly moves in a particular direction, say clockwise, when the windings are energized, then the energizing sequence is correct. If the motor hunts or moves in a jerky manner, then the sequence of winding segments has to be changed and checked again for smooth movement.

FIGURE 9: CONSTRUCTION OF A HYBRID MOTOR



TORQUE AND SPEED

The speed of a stepper motor depends on the rate at which you turn on and off the coils, and is termed the 'step-rate'. The maximum step-rate, and hence, the maximum speed, depends upon the inductance of the stator coils. Figure 10 shows the equivalent circuit of a stator winding and the relation between current rise and winding inductance. It takes a longer time to build the rated current in a winding with greater inductance compared to a winding with lesser inductance. So, when using a motor with higher winding inductance, sufficient time needs to be given for current to build up before the next step command is issued. If the time between two step commands is less than the current build-up time, it results in a 'slip', i.e., the motor misses a step. Unfortunately, the inductance of the winding is not well documented in most of the stepper motor data sheets. In general, for smaller motors, the inductance of the coil is much less than its resistance, and the time

constant is less. With a lower time constant, current rise in the coil will be faster, which enables a higher step-rate. Using a Resistance-Inductance (RL) drive can achieve a higher step rate in motors with higher inductance, which is discussed in the next section.

The best way to decide the maximum speed is by studying the torque vs. step-rate (expressed in pulse per second or pps) characteristics of a particular stepper motor (shown in Figure 11). 'Pull-in' torque is the maximum load torque that the motor can start or stop instantaneously without mis-stepping. 'Pull-out' torque is the torque available when the motor is continuously accelerated to the operating point. From the graph, we can conclude that for this particular motor, the 'maximum self-starting frequency' is 200 pps. The term 'maximum self-starting frequency' is the maximum step-rate at which the motor can start instantaneously at no-load without mis-stepping. While at no-load, this motor can be accelerated up to 275 pps.

FIGURE 10: MOTOR EQUIVALENT CIRCUIT AND CURRENT RISE RATE IN STATOR WINDING

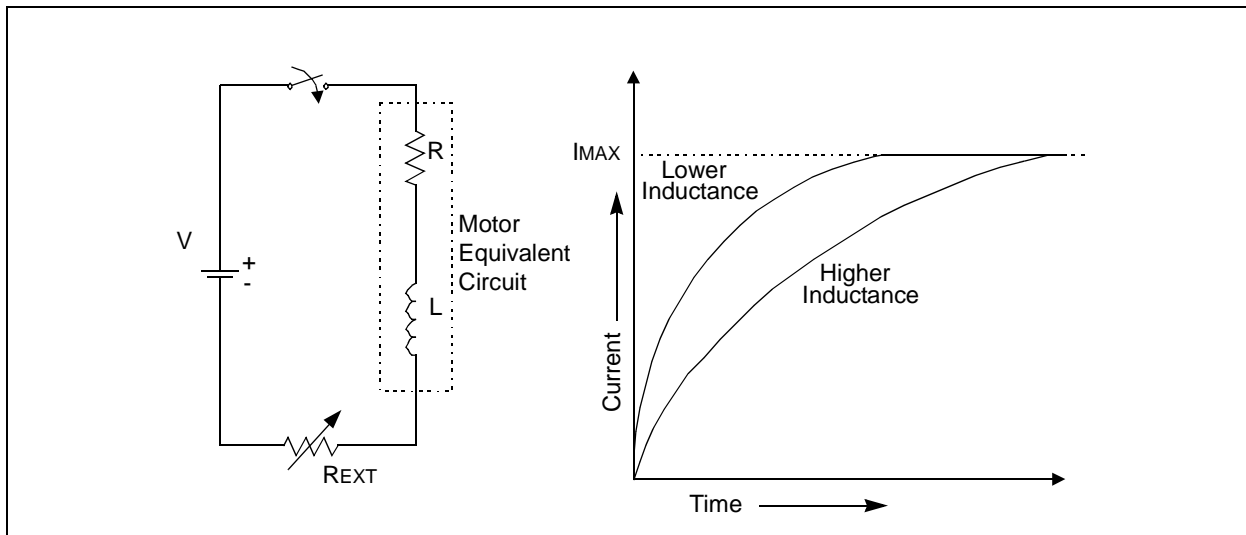
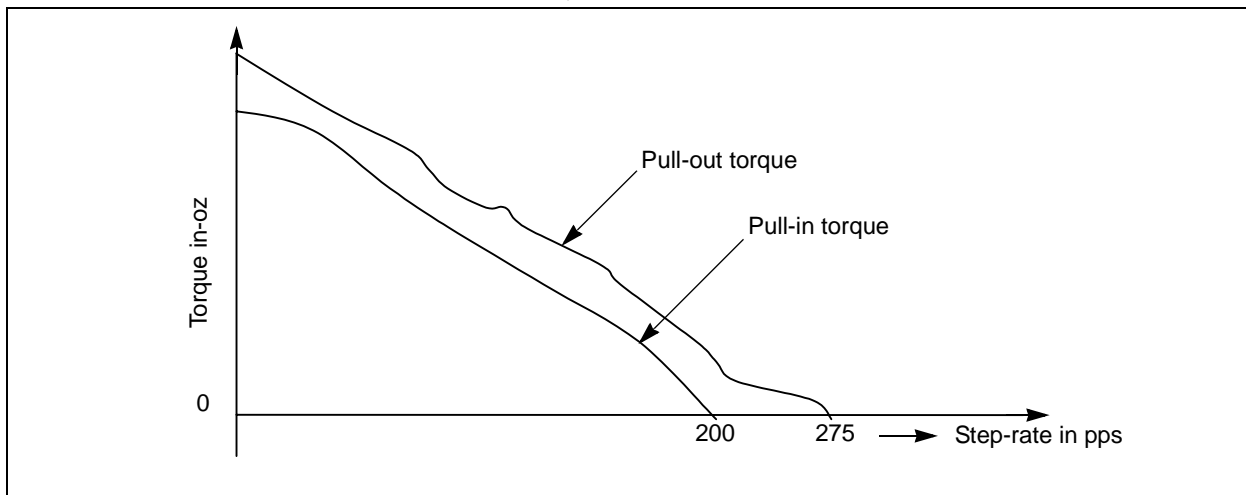


FIGURE 11: A TYPICAL SPEED VS. TORQUE CURVE



DRIVE CIRCUITS

The drive mechanism for 5-wire and 6-wire unipolar motors is fairly simple and is shown in Figure 12 (A and B). Only one coil is shown in this figure, but the other will be connected in the same way.

By comparing Figure 12A and Figure 12B, we see the direction of current flow is opposite in sections A and C of the coil, as per our explanation earlier. But the current flow in a particular section of the coil is always unidirectional, hence the name 'unipolar motor'.

Bipolar stepper motors do not have the center tap. That makes the motor construction easier, but it needs a different type of driver circuit, which reverses the current flow through the entire coil by alternating the polarity of the terminals, giving us the name 'bipolar'.

A bipolar motor is capable of higher torque since the entire coil is energized, not just half. Let's look at the mechanism for reversing the voltage across one of the coils, as shown in Figure 13.

This circuit is called an H-bridge, because it resembles a letter 'H'. The current can be reversed through the coil by closing the appropriate switches. If switches A and D are closed, then current flows in one direction, and if switches B and C are closed, then current flows in the opposite direction.

As the rating of the motor increases, the winding inductance also increases. This higher inductance results in a sluggish current rise in the windings, which limits the step-rate, as explained in the previous section. We can reduce the time constant by externally adding a suitable resistor in series with the coil and applying more than the rated voltage. The resistor should be chosen in such a way that the voltage across the coil does not exceed the rated voltage, and the additional voltage is dropped across the resistor. This method is also useful if we have a fixed power supply with an output of more than the rated coil-voltage specified. This type of drive is called a resistance-inductive (RL) drive. Electronic circuitry can be added to vary this resistor value dynamically to get the best result. The main disadvantage of this drive is that, since they are used with motors with large torque ratings, current flowing through the series resistor is large, resulting in higher heat dissipation and, hence, the size of the drive becomes bulky.

This resistor can be avoided by using PWM current control in the windings. In PWM control, current through the winding can be controlled by modulating the 'ON' time and 'OFF' time of the switches with PWM pulses, thus ensuring that only the required current flows through the coil, as shown in Figure 14.

FIGURE 12: SIMPLIFIED DRIVES FOR THE UNIPOLAR MOTOR

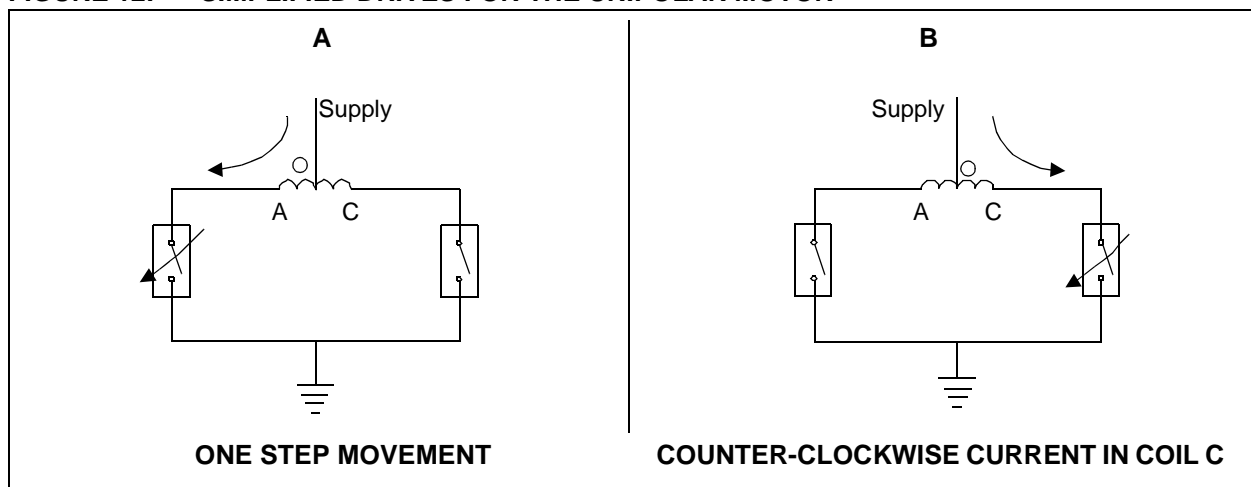


FIGURE 13: SIMPLIFIED H-BRIDGE CONFIGURATION

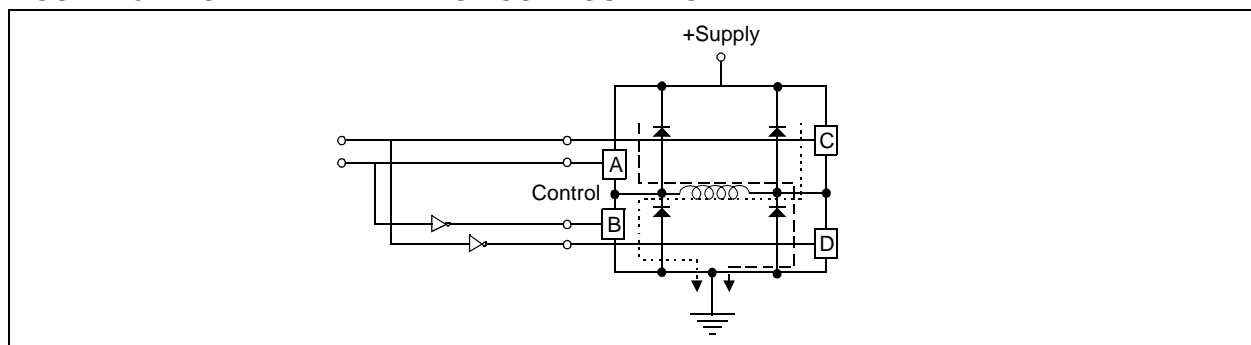
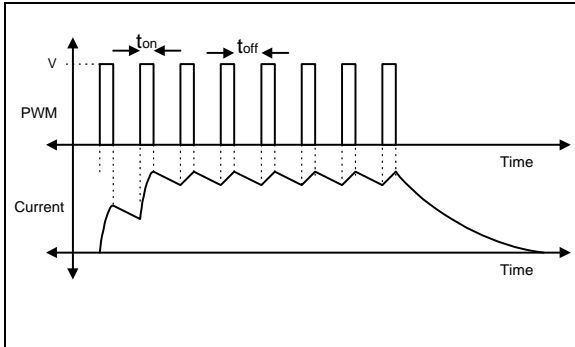


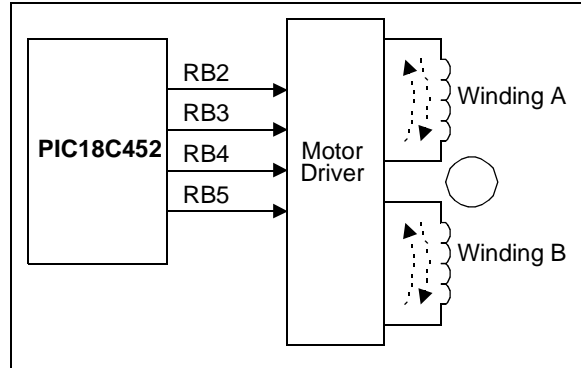
FIGURE 14: CURRENT WAVE FORM WITH PWM SWITCHING



STEPPER MOTOR CONTROL

To control a stepper motor, we need a proper driver circuit as discussed earlier. Unipolar drive can be used with unipolar motors only. In this application note, a bipolar drive is discussed, as this can be used to control both bipolar and unipolar motors. Unipolar motors can be connected to a bipolar driver by simply ignoring the center taps (by doing this, the motor becomes bipolar). Next we need a sequencer to issue proper signals in a required sequence to the H-bridges. A controller is built around the PIC18C452. Two H-bridges are used to control two windings of the stepper motors. Functional block diagram is shown in Figure 15. Example 1 shows the code required for full step control written for PIC18C452:

FIGURE 15: BLOCK DIAGRAM OF FULL STEP CONTROL



Code which configures PORTB<5:2> as output pins is not given in the example.

The code makes RB<5:2> outputs either '0' or '1' sequentially, which switches off or applies positive (+) or negative (-) polarity to Winding A and Winding B, as shown below:

Winding A	Winding B	
+	0	step 1
0	+	step 2
-	0	step 3
0	-	step 4

Legend:

- 0 = coil OFF
- + = current flows in one direction
- - = current flows in the opposite direction

Note: Step 1 follows after step 4 and the cycle continues.

EXAMPLE 1: FULL STEP WITH 'ONE PHASE ON' AT A TIME

```

#define STEP_ONE      b'00100000'    ; PortB<5:2> are used to connect the
#define STEP_TWO      b'00010000'    ; switches
#define STEP_THREE    b'00001000'
#define STEP_FOUR     b'00000100'

    clrf    STEP_NUMBER                ; Initialize start of step sequence
;*****
    Initialize here TMR0 module, enable TMR0 interrupt and load a value in TMR0
;*****
;*****
; Routine in TMR0 ISR which updates the current sequence for the next steps
;*****
    org    2000h
UPDATE_STEP
    incf    STEP_NUMBER,F              ; Increment step number
    btfsc   STEP_NUMBER,2             ; If Step number = 4h then clear the count
    clrf    STEP_NUMBER
    movf    STEP_NUMBER,W             ; Load the step number to Working register
    call    OUTPUT_STEP               ; Load the sequence from the table
    movwf   PORTB                     ; to Port B
    return
OUTPUT_STEP
    addwf   PCL,F                     ; Add Wreg content to PC and
    retlw   STEP_ONE                  ; return the corresponding sequence in Wreg
    retlw   STEP_TWO
    retlw   STEP_THREE
    retlw   STEP_FOUR

```

The step command sequence is updated in the Timer0 overflow Interrupt Service Routine. After issuing each step command in the sequence, PIC18C452 waits for the Timer 0 overflow interrupt to issue the next step sequence. This waiting time can be programmed by loading different values in the TMR0 register. Motor speed depends upon this value in the TMR0 register.

EQUATION 1: CALCULATE STEP COMMAND WAITING PERIOD

$$\begin{aligned} \text{No. Steps per Revolution} &= 360/\text{Motor Step Angle} \\ \text{pps} &= (\text{rpm}/60) * \text{No. Steps per Revolution} \\ \text{Twait} &= 1/\text{pps} \end{aligned}$$

For example, to turn a PM motor with a 7.5 degree step angle at a speed of 120 revolutions per minute (rpm), 96 pulses per second (pps) is required. This means that the waiting period should be 1/96 second to achieve this speed.

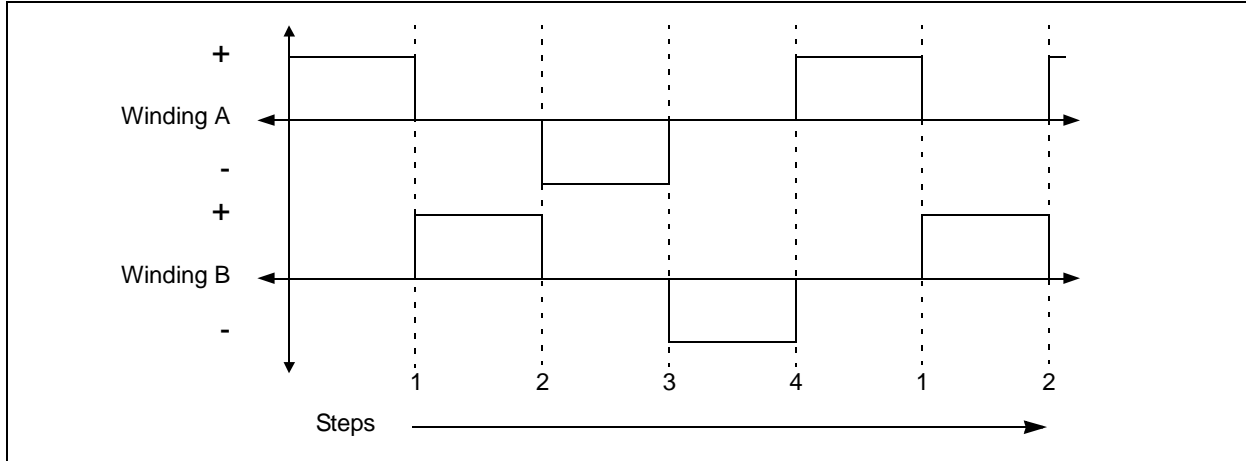
Instead of creating a software delay loop, Timer 0 module of PIC18C452 is loaded with an appropriate value to interrupt the processor every 1/96 second. Steps are updated in the Timer 0 Interrupt Service Routine. By loading different values in the Timer 0 module, the speed of the motor can be changed. The current through the two coils looks like a wave, as shown in Figure 16, so this is termed 'wave drive'.

This controller drives current through only one winding at a given time, so it is also termed 'One Phase On control'. This is the simplest kind of controller. The torque generated in this mode is less, as only one winding at a time is used. For the same stepper motor, we can improve the torque characteristics, by designing a better controller and thereby improving the drive capability.

The following are the most common drive types:

- 'Two Phase On' full step drive
- Half step drive, where the motor moves half of the full step angle (7.5/2 degrees in the case of a motor with 7.5 degrees of step angle)
- Microstepping (which requires unequal current flow in two windings), where the rotor moves a fraction of the full step angle (1/4, 1/8, 1/16 or 1/32).

FIGURE 16: FULL STEP 'ONE PHASE ON' OR WAVE CONTROL



'TWO PHASE ON' FULL STEPPING

In this method, both windings of the motor are always energized. Instead of making one winding off and another on, in sequence, only the polarity of one winding at a time is changed as shown:

Winding A: + - - + + ...
 Winding B: + + - - + ...

The code written for 'One Phase On' control is modified, as shown below in Example 2, to achieve 'Two Phase On' control.

The UPDATE_STEP function is the same as in Example 1, but in the OUTPUT_STEP function, two steps are AND'd (i.e., simultaneously two outputs of port B are '1'), which makes the two coils 'ON' simultaneously. The energizing sequence for both windings is shown in Figure 17.

EXAMPLE 2: 'TWO PHASE ON' CONTROL

```
#define STEP_ONE      b'00100000'    ; PortB<5:2> are used to connect the
#define STEP_TWO     b'00010000'    ; switches
#define STEP_THREE   b'00001000'
#define STEP_FOUR    b'00000100'

    clrf    STEP_NUMBER                ; Initialize start of step sequence
;*****
    Initialize here TMR0 module, enable TMR0 interrupt and load a value in TMR0
;*****

;*****
; Routine in ISR which updates the current sequence for the next steps
;*****

    org    2000h
UPDATE_STEP
    incf   STEP_NUMBER,F              ; Increment step number
    btfsc STEP_NUMBER,2              ; If Step number = 4h then clear the count
    clrf   STEP_NUMBER
    movf   STEP_NUMBER,W              ; Load the step number to Working register
    call  OUTPUT_STEP                 ; Load the sequence from the table
    movwf PORTB                       ; to PortB
    return

OUTPUT_STEP
    addwf  PCL,F                      ; Add Wreg content to PC and
    retlw  STEP_ONE | STEP_TWO        ; return the corresponding sequence in Wreg
    retlw  STEP_TWO | STEP_THREE
    retlw  STEP_THREE | STEP_FOUR
    retlw  STEP_FOUR | STEP_ONE
```

FIGURE 17: VOLTAGE SEQUENCE WITH 'TWO PHASE ON' AT A TIME

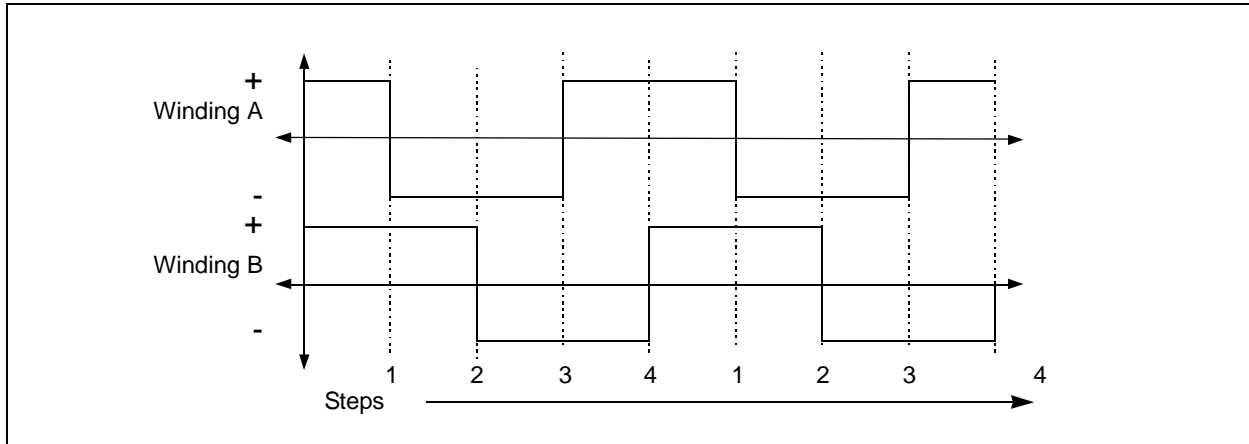
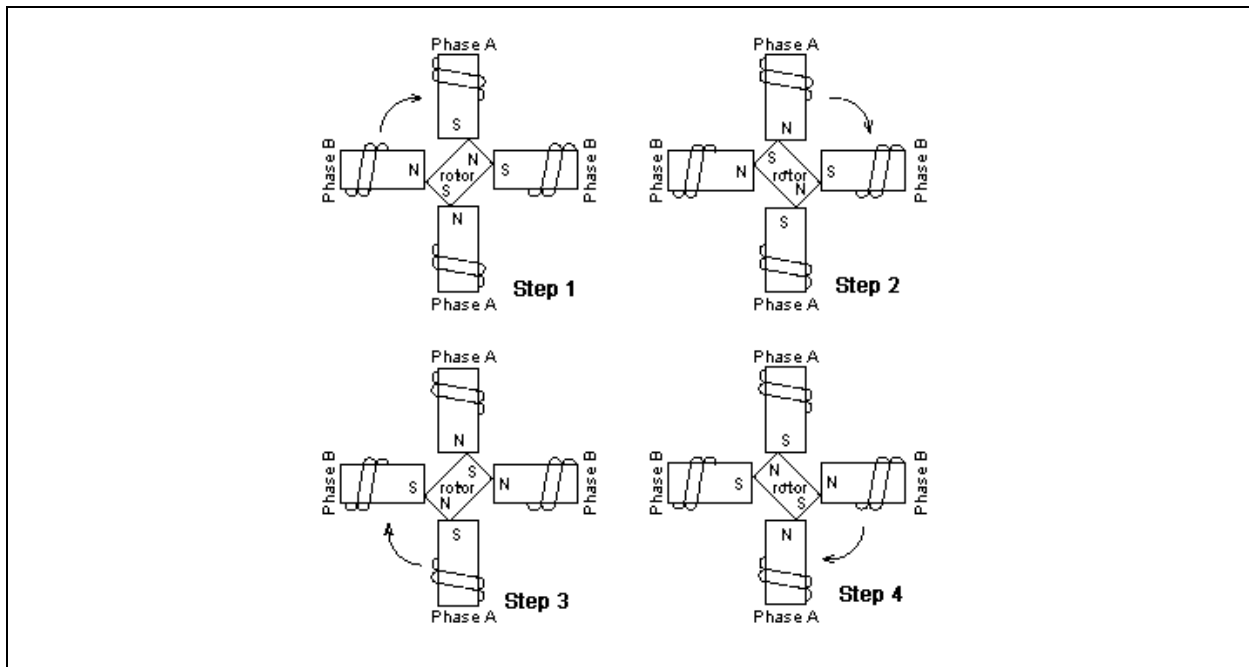


FIGURE 18: MOTOR ROTATION SEQUENCE WITH 'TWO PHASE ON' AT A TIME



With the current flowing in both windings simultaneously, the rotor aligns itself between the 'average north' and 'average south' magnetic poles, as shown in Figure 18. Since both phases are always 'ON', this method gives 41.4 percent more torque than 'One Phase On' stepping.

One drawback of a stepper motor is that it has a natural resonant frequency. When the step-rate equals this frequency, we experience an audible change in the noise made by the motor, as well as an increase in vibration. The resonance point varies with the application and load, and typically occurs at low speed. In severe cases, the motor may lose steps at the resonant frequency. The best way to reduce the problem is to drive the motor in Half Step mode or Microstep mode.

AN822

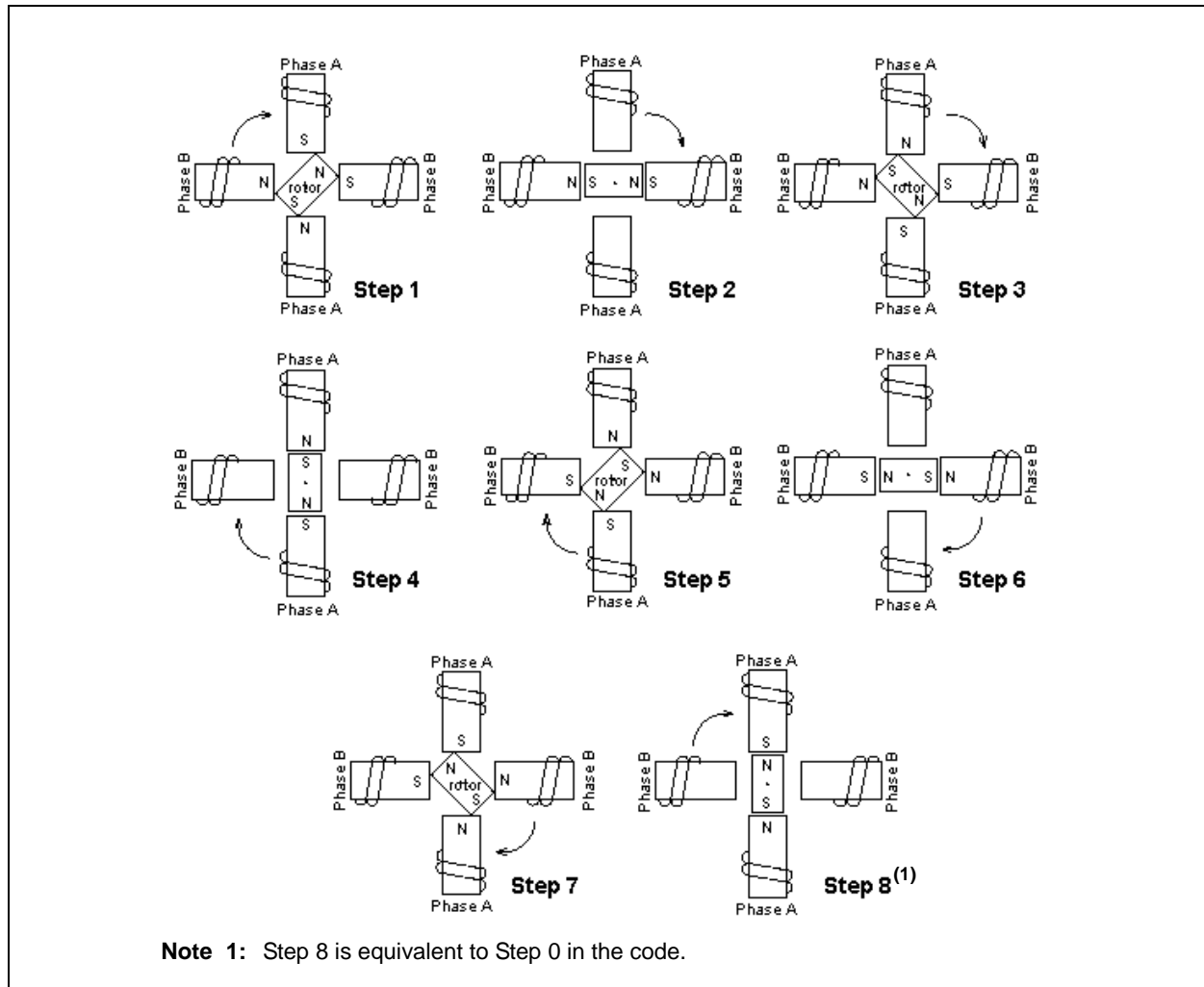
HALF STEPPING

This is actually a combination of 'One Phase On' and 'Two Phase On' full step control, as shown in Table 1.

TABLE 1: HALF STEP CONTROL

STEP_NUMBER	1	2	3	4	5	6	7	8 (0)
Rotor position	1/2	1	1 1/2	2	2 1/2	3	3 1/2	4/0
Current in Winding A	+	0	-	-	-	0	+	+
Current in Winding B	+	+	+	0	-	-	-	0

FIGURE 19: MOTOR ROTATION SEQUENCE FOR HALF STEP



When current flows in only one winding, the rotor aligns with the stator poles in positions 0, 1, 2, and 3, as shown in Figure 19. When current flows in both windings, the rotor aligns itself between two stator poles in positions $\frac{1}{2}$, $1\frac{1}{2}$, $2\frac{1}{2}$, and $3\frac{1}{2}$. So we see that, compared to a full step, the number of steps are doubled. This implies that a motor with a 7.5 degree step angle can be moved 3.75 degrees per step in Half Step mode and, hence,

will take 96 steps to complete a rotation of 360 degrees, as compared to 48 steps in Full Step mode. Now, to rotate this motor at 120 rpm, as discussed earlier, the step-rate also has to be doubled to 192 pps.

The code to achieve half stepping is given in Example 3. The energizing sequence for the stator coils is shown in Figure 20.

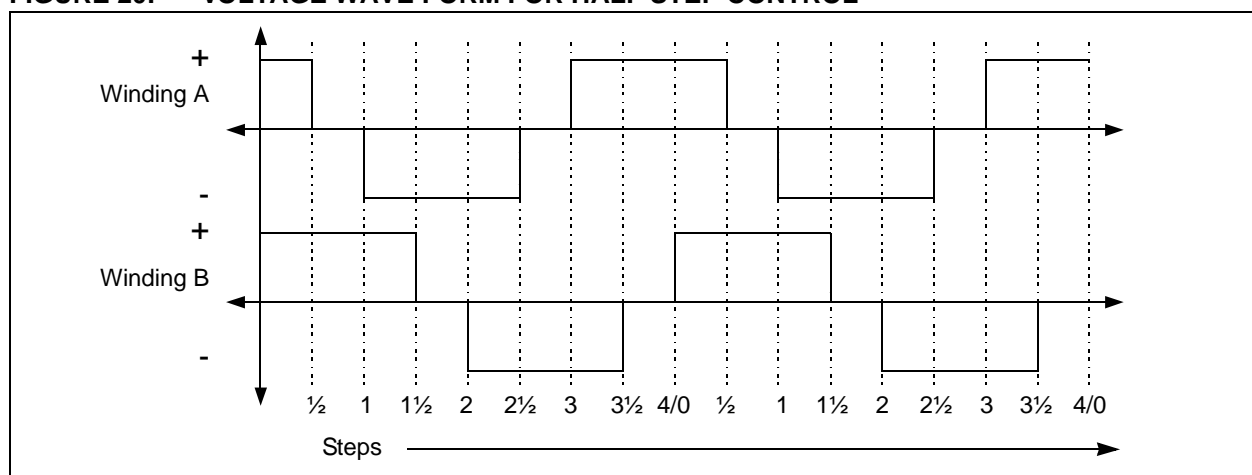
EXAMPLE 3: HALF STEPPING

```
#define STEP_ONE      b'00100000'  ; PortB<5:2> are used to connect the
#define STEP_TWO     b'00010000'  ; switches
#define STEP_THREE   b'00001000'
#define STEP_FOUR    b'00000100'

      clrf      STEP_NUMBER          ; Initialize start of step sequence
;*****
      Initialize here TMR0 module, enable TMR0 interrupt and load a value in TMR0
;*****

;*****
; Routine in ISR which updates the current sequence for the next steps
;*****
      org      2000h
UPDATE_STEP
      Incf     STEP_NUMBER,F         ; Increment step number
      btfsc   STEP_NUMBER,3         ; If Step number = 8h then clear the count
      clrf    STEP_NUMBER
      movf    STEP_NUMBER,W         ; Load the step number to Working register
      call    OUTPUT_STEP           ; Load the sequence from the table
      movwf   PORTB                 ; to Port B
      return
OUTPUT_STEP
      addwf   PCL,F                 ; Add Wreg content to PC and
      retlw   STEP_ONE              ; return the corresponding sequence in Wreg
      retlw   STEP_ONE | STEP_TWO
      retlw   STEP_TWO
      retlw   STEP_TWO | STEP_THREE
      retlw   STEP_THREE
      retlw   STEP_THREE | STEP_FOUR
      retlw   STEP_FOUR
      retlw   STEP_FOUR | STEP_ONE
```

FIGURE 20: VOLTAGE WAVE FORM FOR HALF STEP CONTROL



MICROSTEPPING

During our earlier discussion, we have mentioned that halfstepping and microstepping reduces the stepper motor's resonance problem. Although the resonance frequency depends upon the load connected to the rotor, it typically occurs at a low step-rate. We have already seen that the step-rate doubles in Half Step mode compared to Full Step mode. If we move the motor in microsteps, i.e., a fraction of a full step (1/4, 1/8, 1/16 or 1/32), then the step-rate has to be increased by a corresponding factor (4, 8, 16 or 32) for the same rpm. This further improves the stepper performance at very low rpm. Moreover, microstepping offers other advantages as well:

- Smooth movement at low speeds
- Increased step positioning resolution, as a result of a smaller step angle
- Maximum torque at both low and high step-rates

But microstepping requires more processing power. If we study the flow diagrams for current (as shown for full or half steps), we conclude that the value of current in a particular coil is either 'no current' or 'a rated current'. However, in microstepping, the magnitude of current varies in the windings.

The function of a microstepping controller is to control the magnitude of current in both coils in the proper sequence.

THEORY OF MICROSTEPPING

The current flow diagrams, as well as the sequence of operations in case of full or half stepping, reveals that the electrical sequence repeats itself after every fourth full step. This phenomenon of stepper motor signifies that one full 'electrical cycle' consists of four full steps. Please note that one full 'electrical cycle' (i.e., 360 degrees of 'electrical angle') is different from one full revolution of the rotor (360 degrees of mechanical rotation). One full 'electrical cycle' always consists of four full steps. Hence, one full step of any stepper motor with any 'step angle' corresponds to 360/4 or 90 degrees of 'electrical angle'. If this 'electrical angle' is divided into smaller, equal angles, and a corresponding current is given to the stator windings, then it will look like Figure 21. So we can vary current in one winding with a sine function of an angle 'θ' and in the other winding with a cosine function of 'θ'.

In a stepper motor, the rotor stable positions are in synchronization with the stator flux. When the windings are energized, each of the windings will produce a flux in the air gap proportional to the current in that winding. So the flux in the air gap is directly proportional to the vector sum of the winding currents, in the resultant vector direction. In Full Step and Half Step modes, rated current is supplied to the windings, which rotates the resultant flux in the air gap in 90 degrees and 45 degrees electrical, respectively, with each change in sequence. In microstepping, the current is changed in the windings in fractions of rated current. Therefore, the resultant direction of flux changes in fractions of 90 degrees electrical. Usually, a full step is further divided into 4/8/16/32 steps. (A step length shorter than 1/32 of a full step normally does not make any further improvement in the motion.)

To achieve the required rotating flux, you can calculate the magnitude of the current in the windings with the following formula:

EQUATION 2: FLUX FORMULA

$$I_a = I_{PEAK} * \sin\theta$$

$$I_b = I_{PEAK} * \cos\theta$$

Where:

I_a = instantaneous current in stator winding A

I_b = instantaneous current in stator winding B

θ = angle in electrical degrees from a full step position (OR microstep angle)

I_{PEAK} = rated current of winding

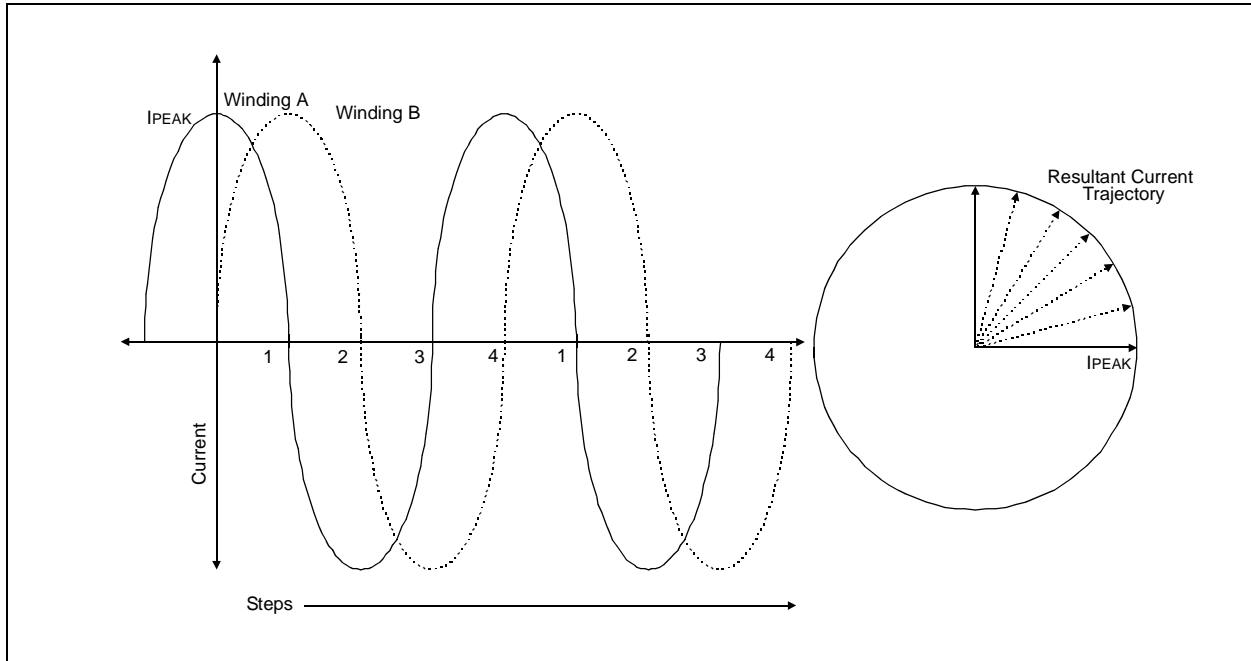
With the above equations, the resultant stator current is the vector sum of the individual winding currents.

$$= \sqrt{((I_{PEAK} * \sin \theta)^2 + (I_{PEAK} * \cos\theta)^2)}$$

$$= I_{PEAK} * \sqrt{(\sin^2\theta + \cos^2\theta)} = I_{PEAK} \angle\theta \text{ electrical degree}$$

This shows that at any angle θ, the resultant current remains same and equal to 'I_{PEAK}'.

FIGURE 21: CURRENTS IN STATOR DURING MICROSTEP AND THE RESULTANT CURRENT



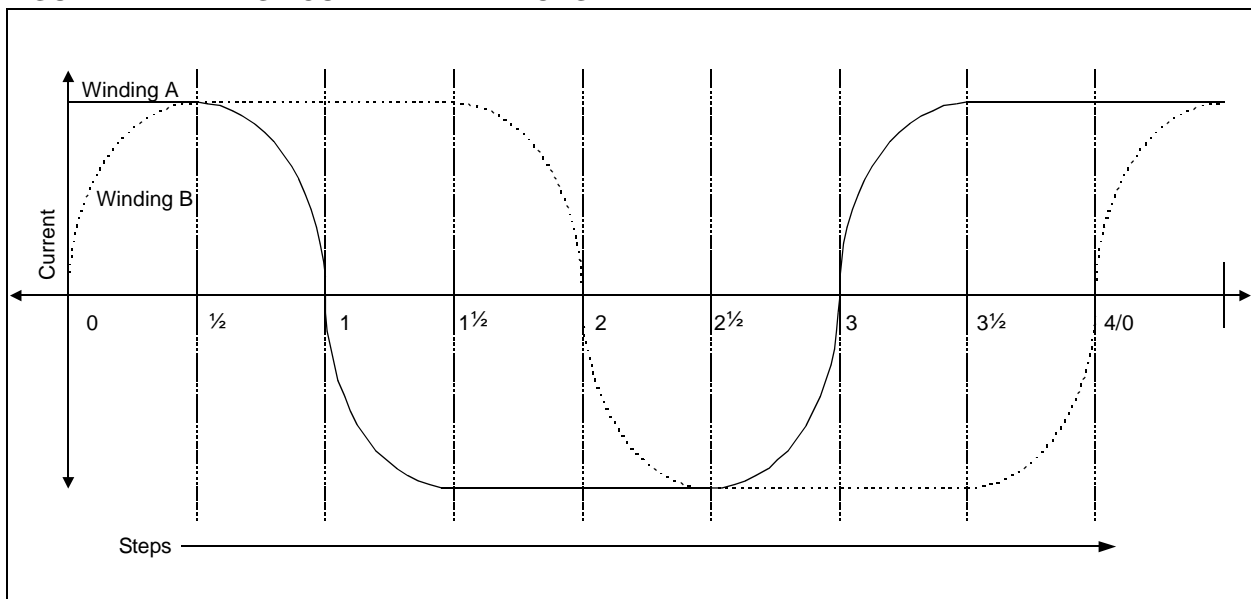
As shown in Figure 21, current in each winding will vary resulting in a rotating flux corresponding to I_{PEAK} in the air gap. So for each increment of electrical angle θ , a flux and a torque corresponding to I_{PEAK} is produced at an angle θ , thus producing a constant rotating flux/torque, which makes microstepping possible.

But in practice, the current in one winding is kept constant over half of the complete step and current in the other winding is varied as a function of $\sin\theta$ to maximize the motor torque, as shown in Figure 22.

Thus, the resultant current is:

$$\begin{aligned}
 &= \sqrt{((I_{PEAK})^2 + (I_{PEAK} * \sin\theta)^2)} \\
 &= I_{PEAK} * \sqrt{(1 + \sin^2\theta)} \geq I_{PEAK} \angle \theta \text{ electrical degrees}
 \end{aligned}$$

FIGURE 22: PHASE-CURRENT RELATIONSHIP



IMPLEMENTATION

The question is how to drive variable currents through the coil connected to a single supply source. There are different ways to achieve this, but the best way is:

1. Connect one voltage source across the H-bridge so that when one pair of opposite switches are on, rated voltage is applied to the stator coil.
2. Vary the PWM duty cycle to control current through the coil.

The controller is built around the PIC18C452 microcontroller. A block diagram is shown in Figure 23. An actual circuit schematic is given in Appendix A. Two PWM modules of PIC18C452 are used to control current through two windings of the stator, and can be used for both full or half step.

Added features in the controller are:

- Speed setting through a potentiometer connected to one of the ADC channels of the PIC18C452.
- A step switch connected to one of the inputs of PORTB. If this switch is pressed, then the motor moves only one step (full, half or microstep).
- A toggle switch connected to one of the inputs to PORTB that decides the direction: forward or reverse.
- A DIP switch, connected to PORTD, is used to select the number of microsteps.
- DIP4 is used as the "Enable" switch. This has to be closed to run the motor with microsteps selected by DIP1-3.

Details of the DIP switches are shown in Table 2.

TABLE 2: DIP SWITCHES

No. of Steps	SW4 (RD5)	SW3 (RD2)	SW2 (RD1)	SW1 (RD0)
Full Step	Close	Open	Open	Close
Half Step	Close	Open	Close	Open
4	Close	Open	Close	Close
8	Close	Close	Open	Open
16	Close	Close	Open	Close
32	Close	Close	Close	Open

Note: Invalid where switches are all open or all closed.

Theoretically, the number of microsteps can be even more than 32, but practically, that does not improve stepper performance. The motor can be driven in microsteps by changing the currents in both windings, as a function of sine and cosine, simultaneously. Alternatively, the current is kept constant in one winding, while it is varied in the other, as shown in Figure 24. In practice, the second method is followed to maximize torque. Theoretically, the variation follows a sine curve, but may vary slightly for different motors to get improved step accuracy.

Appropriate values of the PWM duty cycle (proportional to the required coil current) for each step are given in Appendix B. A table corresponding to the PWM duty cycle is stored in the program memory of PIC18C452. The Table Pointer (TBLRD instruction) of PIC18C452 is used to retrieve the value from the table and load it to the PWM registers to generate an accurate duty cycle.

The assembly code to realize the microstepping is given in Appendix C.

The serial interface with a host computer is done using an USART module on the PIC18C452.

On the Host PC side, "Hyper Terminal" is used for communication. The serial link parameters are:

```

Baud rate:      9600
Data bits:      8
Parity:         none
Stop bit:       1
Flow control:   none
    
```

The commands shown in Table 3 can be set and run from the host PC.

Memory Usage

On-chip ROM used: 3580 bytes

On-chip RAM used: 26 bytes

CONCLUSION

Microstepping a stepper motor increases stepping accuracy and reduces resonance in the motor. The two PWMs in the PIC18C452 can be used to control the voltage to the windings of a bipolar stepper motor.

A sine lookup table is entered in the program memory and accessed using the table read instructions. An on-chip USART communicates with the host PC for control parameters, and motor speed can be set using a potentiometer connected to one of the ADC channels.

TABLE 3: HOST PC COMMANDS

Command	Description	Range	Remarks/Data Value
0	Exit from PC interface	—	Control goes to the parameters set on the Reference board, like pot., FWD/REV switch, DIP switch
1	Number of microsteps	1 to 6	1. Full step 2. Half step 3. 1/4 step 4. 1/8 step 5. 1/16 step 6. 1/32 step
2	Direction of rotation	0 to 1	0 = Forward 1 = Reverse
3	Number of steps to inch	1 to 999	Inches in the selected direction and by selected step length
4	RPM	1 to 200	Rotates at set RPM, in set direction

FIGURE 23: BLOCK DIAGRAM OF CIRCUIT FOR MICROSTEPPING

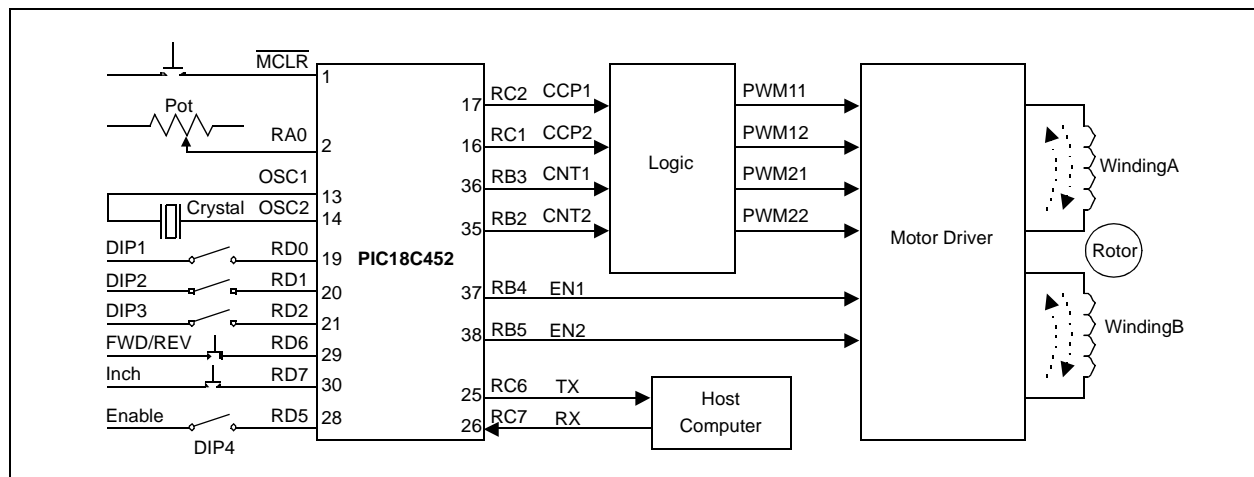
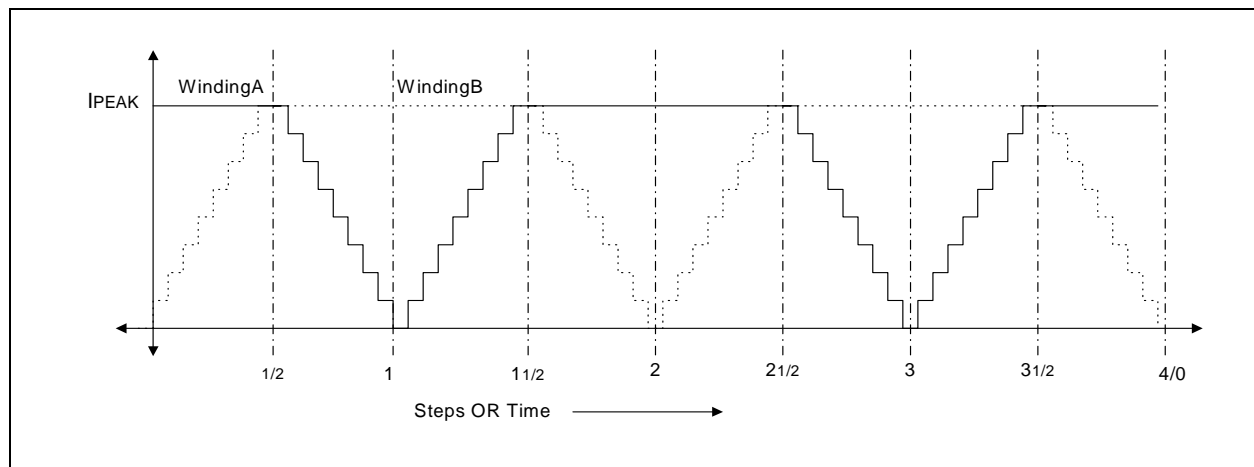


FIGURE 24: CURRENT FLOWS IN STATOR WINDINGS

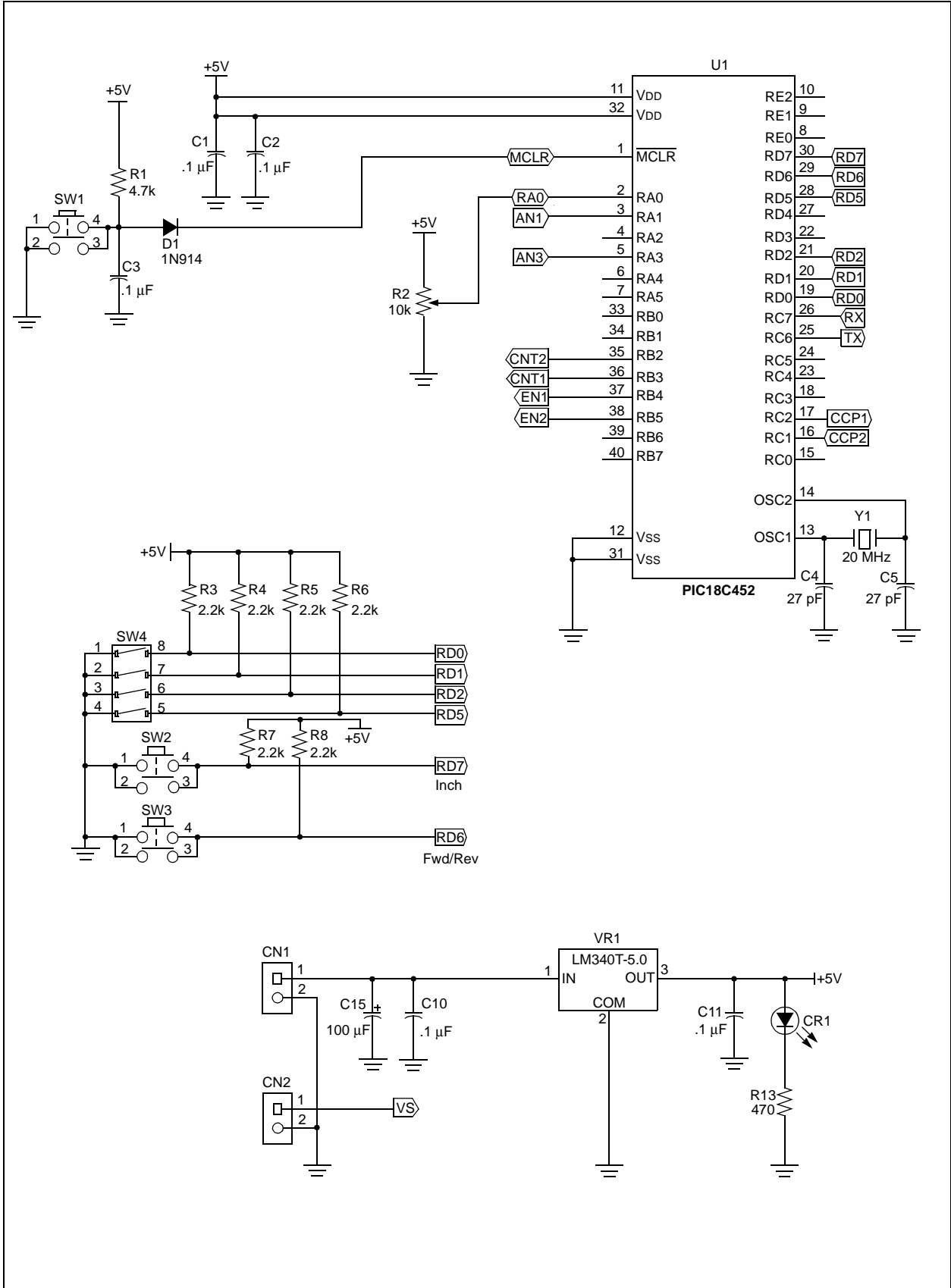


APPENDIX A: SCHEMATIC DETAILS

The control scheme uses PIC18C452 for control and a driver IC, which has two H-bridges for driving the motor.

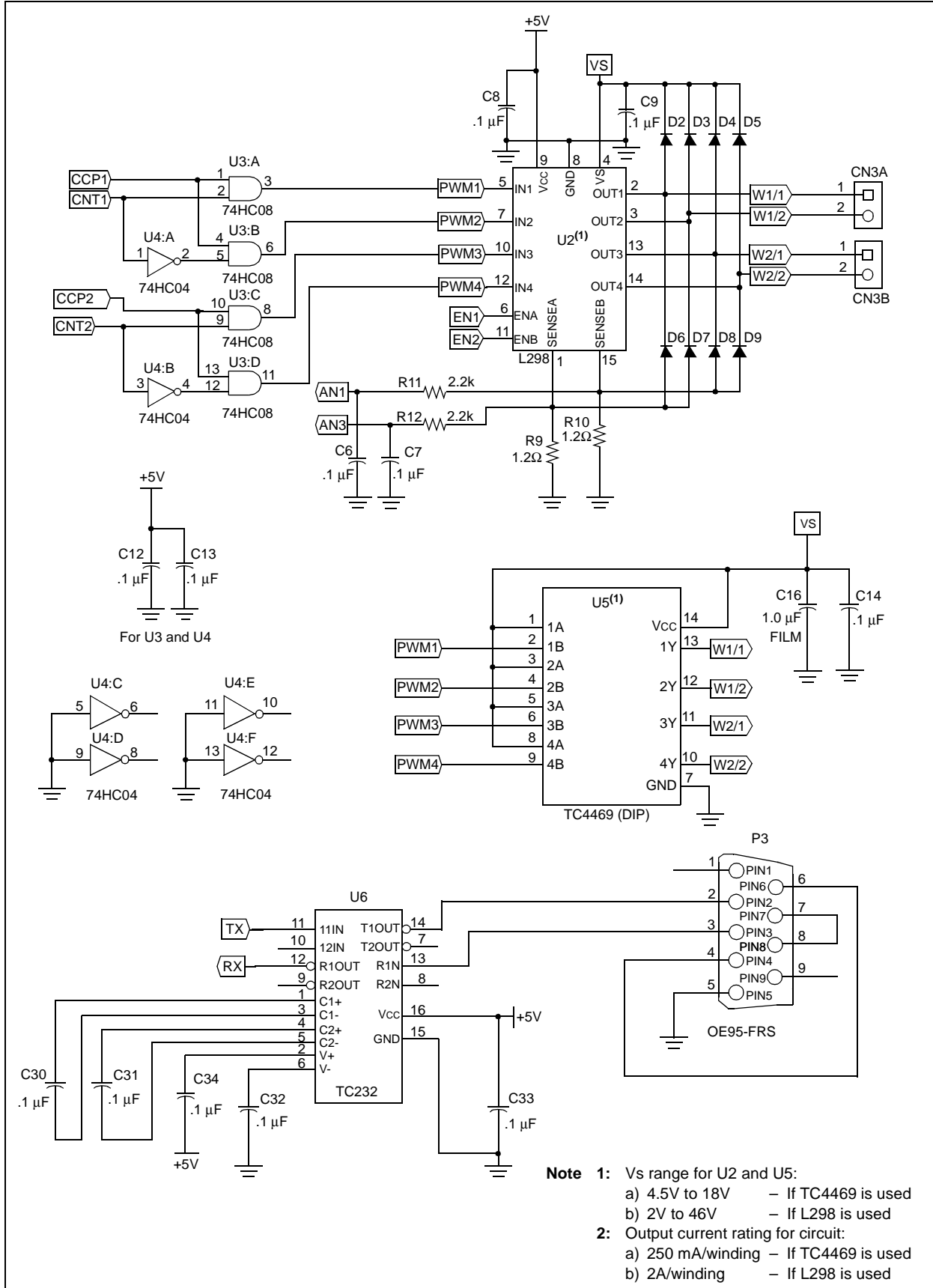
- Four PWMs required are derived from two CCPs (CCP1 and CCP2 in PWM mode). Control signals CNT1 and CNT2 switches CCP1 and CCP2 to appropriate PWM inputs of Driver IC (U2 and U5). CNT1 and CNT2 are connected to RB3 (Pin 36) and RB2 (Pin 35) of microcontroller (U1), respectively.
- EN1 and EN2 signals enable two sets of bridges in the driver IC (only for U2), connected to RB4 (Pin 37) and RB5 (Pin 38) of U1, respectively.
- Current feedbacks from the motor windings are converted to voltages by resistors R9 and R10, connected to Pin 1 and 15 of U2. These feedbacks are connected to AN1 (Pin 3) and AN3 (Pin5).
- I/O pin RD5 (Pin 28) is connected with a SPST switch for drive enable.
- I/O pin RD6 (Pin 29) is connected to a push-button switch for motor direction selection (FWD/REV). Each press of the switch will toggle the direction.
- I/O pin RD7 (Pin 30) is connected to a push-button switch for “Inch” movement of the motor. Each press of this switch will move the motor by a step, controlled by software.
- DIP switches connected to PORT<2:0> select the number of steps, as explained in the previous section.
- A 20 MHz crystal is used as the main oscillator.

FIGURE A-1: CIRCUIT DIAGRAM (SHEET 1 OF 2)



AN822

FIGURE A-2: CIRCUIT DIAGRAM (SHEET 2 OF 2)



APPENDIX B: PWM DUTY CYCLE VALUES

TABLE B-1: TRUTH TABLE FOR FULL STEP OF A STEPPER MOTOR (BIPOLAR MOTOR)

Step Number	Current in Winding 1	Current in Winding 2	PWM1 Duty Cycle CCP1	PWM2 Duty Cycle CCP2	EN1 RB4	EN2 RB5	CNT1 RB3	CNT2 RB2	PORTB Value
0	+1	0	100%	0%	H	L	H	L	0x18
1	0	+1	0%	100%	L	H	L	H	0x24
2	-1	0	100%	0%	H	L	L	L	0x10
3	0	-1	0%	100%	L	H	L	L	0x20

TABLE B-2: TRUTH TABLE FOR MICRO-STEP OF A STEPPER MOTOR (BIPOLAR MOTOR)

Step Number		Current in Winding 1	Current in Winding 2	PWM1 Duty Cycle CCP1	PWM2 Duty Cycle CCP2	EN1 RB4	EN2 RB5	CNT1 RB3		CNT2 RB2		PORTB Value	
Step Range	Micro Step							FWD	REV	FWD	REV	FWD	REV
0 to Half Section 2.1	0	+1	+ Sin 5.6°	100%	9.8%	H	H	H	H	H	L	0x3C	0x38
	1	+1	+ Sin 11.25°	100%	20%	H	H	H	H	H	L	0x3C	0x38
	2	+1	+ Sin 16.8°	100%	29%	H	H	H	H	H	L	0x3C	0x38
	3	+1	+ Sin 22.5°	100%	38%	H	H	H	H	H	L	0x3C	0x38
	4	+1	+ Sin 28°	100%	47%	H	H	H	H	H	L	0x3C	0x38
	5	+1	+ Sin 33.75°	100%	56%	H	H	H	H	H	L	0x3C	0x38
	6	+1	+ Sin 39°	100%	63%	H	H	H	H	H	L	0x3C	0x38
	7	+1	+ Sin 45°	100%	71%	H	H	H	H	H	L	0x3C	0x38
	8	+1	+ Sin 50.6°	100%	77%	H	H	H	H	H	L	0x3C	0x38
	9	+1	+ Sin 56.25°	100%	83%	H	H	H	H	H	L	0x3C	0x38
	10	+1	+ Sin 61.8°	100%	88%	H	H	H	H	H	L	0x3C	0x38
	11	+1	+ Sin 67.5°	100%	93%	H	H	H	H	H	L	0x3C	0x38
	12	+1	+ Sin 73.1°	100%	95.6%	H	H	H	H	H	L	0x3C	0x38
	13	+1	+ Sin 78.75°	100%	98%	H	H	H	H	H	L	0x3C	0x38
	14	+1	+ Sin 84.35°	100%	99.5%	H	H	H	H	H	L	0x3C	0x38
	15	+1	+ Sin 90°	100%	100%	H	H	H	H	H	L	0x3C	0x38

Note 1: Current is in one winding constant for a half of the full step and current in other winding varying sinusoidal.

2: Table is direct for 32 microsteps/step.

3: For -16, -8, -4, -2 (half step); 2, 4, and 8 microsteps are skipped, respectively, from this table.

Software License Agreement

The software supplied herewith by Microchip Technology Incorporated (the "Company") for its PICmicro® Microcontroller is intended and supplied to you, the Company's customer, for use solely and exclusively on Microchip PICmicro Microcontroller products.

The software is owned by the Company and/or its supplier, and is protected under applicable copyright laws. All rights are reserved. Any use in violation of the foregoing restrictions may subject the user to criminal sanctions under applicable laws, as well as to civil liability for the breach of the terms and conditions of this license.

THIS SOFTWARE IS PROVIDED IN AN "AS IS" CONDITION. NO WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. THE COMPANY SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.

APPENDIX C: ASSEMBLY CODE FOR MICROSTEPPING

```

;*****
;PROGRAM :          STEPPER MOTOR CONTROL
;MICROCONTROLLER :    18C452
;CRYSTAL FREQUENCY :    20MHZ
;DRIVER IC USED :      TC4469/ST's L298
;*****
;Documents to be referred with this :
; a) Diagram of control circuit
; b) Application note: Microstepping of stepper motor using 18CXXX
;*****
;AUTHOR :      Padmaraja Yedamale , IDC
;DATE :
;Version :     V1.0
;*****
;Description:-
;-----
;This module controls Stepper motor in Full steps, Half steps and
;microsteps of -4,-8,-16,-32 per full step.
;Timer0 is used for Speed control, which is rate of change of steps.
;Speed of the motor is varied by a potentiometer connected to the
;ADC channel0, which is loaded to TMR0.
;Direction of motor rotation can be changed using the Tact switch(FWD/REV)
;connected to PORTD<6>(Pin29). An internal buffer toggles and changes the
;direction with each press.
;Motor can be "Inched"(i.e. moved in steps) by using the switch(INCH)
;connected to the PORTD<7>(Pin30). Each press of this switch will move
;the motor by one step(full, half or the selected microstep), in the
;selected direction of FWD/REV.
;The DIP switches DIP1(PORTD<0>, Pin19), DIP2(PORTD<1>, Pin20), DIP3(PORTD<0>, Pin21)
;are used to select number of steps as shown in the following table
;-----
;      Sl no.      No. of Steps      DIP3 (RD2)      DIP2 (RD1)      DIP1 (RD0)
;      1           Full step(1)      Open            Open            Close
;      2           Half step(2)      Open            Close           Open
;      3           4                Open            Close           Close
;      4           8                Close           Open            Open
;      5           16               Close           Open            Close
;      6           32               Close           Close           Open
;-----
;DIP4 connected to PORT<5>, pin 28 is used as "Control enable" switch.
;If this is open, motor is inhibited from rotating.
;This module uses CCPx's in PWM mode
;
;In this module current in one of the winding is kept constant (rated)
;over half of the complete step and current in the other winding
;is varied sinusoidally, in order to maximize the rotor torque.
;Resultant rotor Torque = sqrt(1 + (Sine(angle)*Sine(angle))
;which is always > 1
;

```

;A table with PWM values is stored in the program memory. Table pointers and
 ;Table access instructions are used to read the table as required for microstepping.
 ;

;An interface with host computer is given through serial port. USART module in the
 ;PIC18Cxxx is used for the communication. Following commands are implemented.
 ;-----

Command	Explanation	Data value	Range	Remarks
0	Exit from PC interface	----	----	Control goes to the parameters set on the Reference board, like pot., FWD/REV switch, DIP switch
1	No. of microsteps	1-Full step 2-Half step 3-1/4 step 4-1/8 step 5-1/16 step 6-1/32 step	1 to 6	----
2	Direction of rotation	0-Forward 1- Reverse	0 to 1	-----
3	No. of steps to Inch	---	1 to 999	Inches in the selected direction and by selected step length
4	RPM	----	1 to 200	Rotates at set RPM in set direction

include <p18c452.inc>

;Variables definition

```

UDATA_ACS          ;Relocatable variables in access RAM
STEP_NUMBER        res    1    ;Used for tracking the microstep counts
MOTOR_DIRECTION    res    1    ;Motor direction byte
                    ;0 indicates Reverse rotation
                    ;1 indicates forward
COUNTER            res    1    ;Counter used for counting key debounce time
COUNTER1           res    1    ;Counter used for counting key debounce time
SPEED_REF_H        res    1    ;Speed referance, read from ADC0, connected
SPEED_REF_L        res    1    ;to Preset on the board
FLAG_BYTE          res    1    ;Indicates status flags
STEP_JUMP          res    1    ;Step jump count based on DIP switch setting
RECIEVED_BYTE      res    1    ;Byte recieved from host PC
COMMAND_BYTE       res    1    ;Command from host PC
INCH_VALUE         res    2    ;Inch count from host PC
RPM_VALUE          res    4    ;RPM value
MICRO_STEPS        res    1    ;No. of microsteps stored
TEMP_RPM           res    3    ;Temparary reg
TEMP_LOCATION      res    4    ;Temparary reg
TEMP               res    1    ;Temparary variable
TEMP1              res    1

```

```

;-----
#define DEBOUNCE          H'02'    ;Second bit in the FLAG_BYTE
#define TMR0_VALUE_L      H'05E'   ;Timer0 Higher byte value
#define TMR0_VALUE_H      H'0AA'   ;Timer0 Lower byte value
#define STEPS_PER_ROTATION H'30'   ;Full steps per rotation = 360/step angle

```

```

STARTUP            code 0x00
                    goto    Start    ;Reset Vector address

                    CODE    0x08
                    goto    ISR_HIGH ;Higher priority ISR at 0x0008

```

AN822

```
PRG_LOW    CODE    0x018
           goto    ISR_LOW      ;Lower priority ISR at 0x0018

;*****
PROG1      code
Start
;*****
;Used only with MPLAB2000 + PCM18XA0- For Table read/write
;This code is not required when the actual device is used
;*****
           movlw   0xb0
           movwf   0xf9c

;*****
;This routine configures the I/O ports.
;PORTB - Outputs
;PORTB<3> - CNT1 - Used for switching PWM1 logic to change the
;           direction of current in winding1
;PORTB<2> - CNT2 - Used for switching PWM2 logic to change the
;           direction of current in winding2
;PORTB<4> - EN1 - Used for Enabling the H-bridge controlling winding1
;PORTB<5> - EN2 - Used for Enabling the H-bridge controlling winding2
;PORTD - Inputs
;PORTD<5> - Enable switch connected
;PORTD<6> - Forward/Reverse Tact switch connected
;PORTD<7> - INCH Tact switch connected
;*****
IO_PORT_Init
           movlw   0x0          ;Clear PORTB
           movwf   PORTB
           movlw   0x0          ;Clear LatchB
           movwf   LATB
           movlw   0x03         ;PORTB<2:5> output,rest input
           movwf   TRISB       ;PORTB<6:7> reserved for ICD
           movlw   0x0          ;Clear PORTD
           movwf   PORTD
           movlw   0x0          ;Clear LatchD
           movwf   LATD
           movlw   0x0E7       ;PORTD<7:6> and <2:0> input,rest output
           movwf   TRISD      ;

;*****
;This routine configures Analog to Digital(ADC) module to read speed
;Referance voltage from the Preset connected to ADC Ch.0
;*****
ADC_Init
           movlw   0x81        ;ADC Clock=Fosc/32,ADCCh=0,ADON=ON
           movwf   ADCON0      ;
           movlw   0x04        ;ADC result left justified,
           movwf   ADCON1      ;ADC 1Ch.,(AD0);No ref.
           movlw   0x00        ;Clear PortA bits
           movwf   PORTA      ;
           movlw   0x0F        ;PORTA<0:3> input,rest output
           movwf   TRISA      ;
           movlw   0x0         ;Clear PORTE
           movwf   PORTE
           movlw   0x0         ;Clear ADC result higher byte
           movwf   ADRESH      ;At POR AD reult is unknown
           movlw   0x0         ;Clear ADC result lower byte
           movwf   ADRESL      ;At POR AD reult is unknown

;*****
```



```

;This routine configures CCP1 and CCP2 as PWM outputs
;PWM Frequency set to 20KHz(PR2 register)
;*****
CCP1_CCP2_Init

    movlw    0x00            ;CCP1 & CCP2 are outputs
    movwf    TRISC,ACCESS
    movlw    0x00
    movwf    TMR2,ACCESS    ;clear Timer2
    movlw    0xF9            ;PR2=PWM Period;0xF9 corresponds to 20KHz
    movwf    PR2,ACCESS     ;PWM period = [(PR2)+1]*4*Tosc*Tmr2 prescale
                                ;           = [0xF9+1]*4*20MHz*16

    movlw    0x04            ;Timer2 is ON,prescale = 1:1
    movwf    T2CON,ACCESS   ;Load to Timer2 control register
    movlw    0x00c           ;Set CCP1 to PWM mode
    movwf    CCP1CON,ACCESS ;
    movlw    0x00c           ;Set CCP2 to PWM mode
    movwf    CCP2CON,ACCESS ;

;*****
;This routine initializes USART parameters
;*****
INIT_USART

    movlw    0x81            ;Baudrate = 9600
    movwf    SPBRG

    movlw    0x24            ;8-bit transmission;Enable Transmission;
    movwf    TXSTA           ;Asynchronous mode with High speed transmission

    movlw    0x90            ;Enable the serial port
    movwf    RCSTA           ;with 8-bit continuous reception

;*****
;This routine initializes the Interrupts required
;TMR0 overflow interrupt is used to change the step sequence
;*****
INTERRUPT_init

    movlw    0x020           ;Unmask Timer0 interrupt
    movwf    INTCON          ;All other interrupts masked
    movlw    0x004           ;TMR0 overflow interrupt-High priority
    movwf    INTCON2
    movlw    0x093           ;Power ON reset status bit/Brownout reset status bit
    movwf    RCON           ;and Instruction flag bits are set
                                ;Priority level on Interrupts enabled

    movlw    0x040           ;ADC Interrupt enabled
    movwf    PIE1
    movlw    0x000           ;A/D converter interrupt-Low priority
    movwf    IPR1
    bsf     PIE1,5
    bcf     IPR1,5
    bsf     TRISC,7

;*****
;Setting of jump count and prescale value based on the DIP switch settings
    clrf    FLAG_BYTE        ;Intialising all local variables
    clrf    TEMP

    call    SET_DIP_PARAMETERS ;Parameters are set based on DIP switches
    call    STEPPER_COM       ;Displays a welcome message on the host PC screen
    call    send_command_request

;*****

```

AN822

```
;Timre0 Initialization with prescaler
;*****
movlw    TMR0_VALUE_H      ;Timer0 Initialisation
movwf    TMR0H
movwf    SPEED_REF_H
movlw    TMR0_VALUE_L      ;
movwf    TMR0L
movwf    SPEED_REF_L
;*****
;On POR, Motor is moved to a Full step positon
;*****
clrf     STEP_NUMBER      ;starting from step0
bsf     MOTOR_DIRECTION,0 ;motor in fwd direction

movlw    0x0FF             ;Set CCP1L 100% duty cycle
movwf    CCP1L             ;8MSB's of duty cycle
movlw    0x30             ;2 LSB's at CCPxCON<5:4>
iorwf    CCP1CON,1

movlw    0x000             ;set CCP2L
movwf    CCP2L             ;8MSB's of duty cycle
movlw    0x38             ;set Forward current in Winding1
movwf    PORTB

bsf     INTCON,PEIE       ;Enable all Unmasked peripheral interrupts
bsf     INTCON,GIE        ;Enable all Unmasked interrupts
;*****
;Main program starts here which does the following
; 1) Checks for Key pressed (with debounce)
;    a) Motor Forward/Reverse Key connected to RD6
;    b) Motor Inch(move by a step) Key connected to RD7
; 2) If the step is updated by Timer0 interrupt, outputs the
;    required PWM on to CCP1/CCP2
; 3) 1 and 2 are repeated continuously
;*****

MAIN_LOOP
btfsc   PORTD,5           ;Checking for DIP4(Control enable)closed
goto    STOP_MOTOR       ;If open, motor will not rotate

call    check_key         ;Routine which checks for FWD/REV and INCH keys

btfsc   FLAG_BYTE,4      ;If host PC gives command, process the command
call    PROCESS_COMMAND

btfss   PIE1,ADIE
goto    MAIN_LOOP
call    SET_ADC_GO
call    SET_DIP_PARAMETERS
goto    MAIN_LOOP        ;If not returning from TMR0 overflow interrupt
                        ;don't change the step, loop in Main routine

STOP_MOTOR
clrf    CCP1L
clrf    CCP2L
bcf     CCP1CON,4
bcf     CCP1CON,5        ;Update the PWM duty cycle from the table
bcf     CCP2CON,4
bcf     CCP2CON,5        ;Update the PWM duty cycle from the table
clrf    STEP_NUMBER
goto    MAIN_LOOP

;*****
;On TMR0 overflow program will execute the higher priority ISR
;Higher priority Interrupt Service Routine will update the Step count based
;on the Speed commanded by the Potentiometer read through ADC ch.0 in
;Low priority Interrupt
```

```

;*****
ISR_HIGH
    btfsc    INTCON,TMR0IF        ;Timer0 overflow Interrupt?
    goto     timer0_int          ;Yes
    RETFIE                        ;

timer0_int
    call     UPDATE_STEP_NUMBER  ;Update the u-Step number

    call     UPDATE_PWM_STEP
    movff   SPEED_REF_H,TMR0H    ;Load the Higher byte of SpeedCommand to TMR0H
    movff   SPEED_REF_L,TMR0L    ;Load the Lower byte of SpeedCommand to TMR0L

    btfsc   FLAG_BYTE,6
    call    DECREMENT_INCH_COUNT

    bcf     INTCON,TMR0IF        ;Clear TMR0IF
    bcf     FLAG_BYTE,0         ;Clear the flag for PWM updation
    RETFIE

;*****
;On ADC ch.0 interrupt program will execute the lower priority ISR
;Lower priority Interrupt Service Routine will read the ADC ch.0 result
;and load to the Speed command variables.
;*****
ISR_LOW
    btfsc   PIR1,ADIF           ;ADC Interrupt?
    goto    ADC_SPEED_READ      ;Yes
    btfsc   PIR1,RCIF           ;Recieve Interrupt?
    goto    RECIEVE_THE_BYTE    ;Yes
    RETFIE

ADC_SPEED_READ
    movff   ADRESH,RPM_VALUE+1  ;Load AD result
    bcf     STATUS,C
    rrcf    RPM_VALUE+1,F
    movf    RPM_VALUE+1,W
    btfsc   STATUS,Z
    incf    RPM_VALUE+1,F
    bcf     PIR1,ADIF           ;ADIF flag is cleared for next interrupt
    RETFIE

RECIEVE_THE_BYTE
    movff   RCREG,RECIEVED_BYTE  ;
    movf    RECIEVED_BYTE,W
    call    load_RX_REG_from_WREG
    bsf     FLAG_BYTE,4
    bcf     PIR1,RCIF           ;RCIF flag is cleared for next interrupt
    RETFIE

;*****
;This routine will update the PWM duty cycle on CCPx according to the count
;in STEP_NUMBER. STEP_NUMBER is updated in the Timer0 overflow interrupt
;*****
UPDATE_PWM_STEP
;-----
    movf    STEP_JUMP,W          ;Checking for full step
    btfsc   WREG,5               ;Yes, goto FULL_STEP_JUMP
    goto    FULL_STEP_JUMP      ;No,then Half step/Microstep
;-----
;Below is the routine where for microstep(including halfstep) current(PWM) values
;from the sine_table are taken and loaded to the CCPRxL and CCPxCON<5:4> as per Table-2
;-----
;Refer Table-2 Section 2.1 (microstep range from 0 to Half of a complete step)

```

AN822

```
-----  
    movlw    0x010                ;  
    cpfslt  STEP_NUMBER          ;Is the u-step>0x10?  
    goto    step_half            ;Yes, goto Step_half  
  
    movlw    0x00  
    cpfseq  STEP_NUMBER  
    goto    cont_1_15  
    movlw    UPPER_sine_table     ;Initialize Table pointer to the first  
    movwf   TBLPTRU              ;location of the table  
    movlw    HIGH_sine_table  
    movwf   TBLPTRH  
    movlw    LOW_sine_table  
    movwf   TBLPTRL  
  
    TBLRD*+  
    TBLRD*+  
    call    table_adjust_positive ;Used for skipping the table contents for  
cont_1_15  ;u-steps < 32  
    btfsc  MOTOR_DIRECTION,0     ;Is Motor Forward?  
    goto    fwd_1_15              ;  
    movlw  0x38                   ;No Reverse,Wng1 current +ve, Wng2 current -ve  
    goto    rev_1_15              ;Wng1-PORTB<3>;Wng2-PORTB<2>  
fwd_1_15  movlw    0x3C           ;Yes,Forward,Wng1 current +ve, Wng2 current +ve  
rev_1_15  movwf   PORTB  
          call    CCP2_INCREASE   ;Load the PWM2 values and increment Table pointer  
          return  
  
-----  
;Refer Table-2 Section 2.2 (microstep range from Half of a complete step to one complete step)  
-----  
step_half  
    movlw    0x020                ;Is the u-step>20?  
    cpfslt  STEP_NUMBER          ;Yes, goto step_full  
    goto    step_1full  
  
    movlw    0x10                 ;Is the microstep == 10?  
    cpfseq  STEP_NUMBER          ;No, continue loading PWM values  
    goto    cont_16_31  
    call    point_to_end_of_table ;Yes,Point the Table pointer to end of the Table  
cont_16_31  
    btfsc  MOTOR_DIRECTION,0  
    goto    fwd_16_31  
    movlw  0x38                   ;For Reverse rotation Wng1 current +ve  
    goto    rev_16_31             ;Wng2 -ve  
fwd_16_31  
    movlw  0x3C                   ;For forward rotation Wng1 current +ve,Wng2 +ve  
rev_16_31  
    movwf   PORTB  
    call    CCP1_DECREASE        ;Load the PWM1 values and decrement Table pointer  
    return  
  
-----  
;Refer Table-2 Section 2.3 (microstep range from One complete step to one and half step)  
-----  
step_1full  
    movlw    0x030                ;Is u-step>30h?  
    cpfslt  STEP_NUMBER          ;Yes, goto step_1nhalf  
    goto    step_1nhalf  
  
    movlw    0x20                 ;Is the microstep == 20?  
    cpfseq  STEP_NUMBER          ;No, continue loading PWM values  
    goto    cont_32_47  
    call    point_to_begining_of_table  
cont_32_47 ;Yes,Point the Table pointer to beginning of the Table
```

```

        btfsc    MOTOR_DIRECTION,0
        goto    fwd_32_47          ;
        movlw   0x30              ;If Motor is reverse Wng1&Wng2 current -ve
        goto    rev_32_47
fwd_32_47
        movlw   0x34              ;If Motor is forward Wng1 current -ve,Wng2 +ve
rev_32_47
        movwf   PORTB
        call    CCP1_INCREASE
        return
;-----
;Refer Table-2 Section 2.4 (microstep range from One and half step to two complete step)
;-----
step_1nhalf
        movlw   0x40
        cpfslt  STEP_NUMBER
        goto    step_two

        movlw   0x30              ;Is the microstep == 30?
        cpfseq  STEP_NUMBER
        goto    cont_48_63        ;No, continue loading PWM values
        call    point_to_end_of_table ;Yes,Point the Table pointer to end of the Table
cont_48_63
        btfsc    MOTOR_DIRECTION,0
        goto    fwd_48_63
        movlw   0x30              ;If Motor is reverse Wng1&Wng2 current -ve
        goto    rev_48_63
fwd_48_63
        movlw   0x34              ;If Motor is forward Wng1 current -ve,Wng2 +ve
rev_48_63
        movwf   PORTB
        call    CCP2_DECREASE     ;Load the PWM2 values and decrement Table pointer
        return
;-----
;Refer Table-2 Section 2.5 (microstep range from Two complete step to two and half step)
;-----
step_two
        movlw   0x50
        cpfslt  STEP_NUMBER
        goto    step_2nhalf

        movlw   0x40              ;Is the microstep == 40?
        cpfseq  STEP_NUMBER
        goto    cont_64_79        ;No, continue loading PWM values
        call    point_to_begining_of_table
cont_64_79
        btfsc    MOTOR_DIRECTION,0
        goto    fwd_64_79
        movlw   0x34              ;If Motor is reverse Wng1 current -ve,Wng2 +ve
        goto    rev_64_79
fwd_64_79
        movlw   0x30              ;If Motor is forward Wng1&Wng2 current -ve
rev_64_79
        movwf   PORTB
        call    CCP2_INCREASE     ;Load the PWM2 values and increment Table pointer
        return
;-----
;Refer Table-2 Section 2.6 (microstep range from two and half step to Three complete step)
;-----
step_2nhalf

        movlw   0x60
        cpfslt  STEP_NUMBER
        goto    step_three

```

AN822

```
    movlw    0x50                ;Is the microstep == 50?
    cpfseq  STEP_NUMBER
    goto    cont_80_95          ;No, continue loading PWM values
    call    point_to_end_of_table ;Yes,Point the Table pointer to end of the Table
cont_80_95
    btfsc  MOTOR_DIRECTION,0
    goto    fwd_80_95
    movlw  0x34                ;If Motor is reverse Wng1 current is -ve,Wng2 +ve
    goto    rev_80_95
fwd_80_95
    movlw  0x30                ;If Motor is forward Wng1&Wng2 current -ve
rev_80_95
    movwf  PORTB
    call   CCP1_DECREASE       ;Load the PWM1 values and decrement Table pointer
    return

;-----
;Refer Table-2 Section 2.7 (microstep range from 3 complete step to 3 and half step)
;-----
step_three
    movlw  0x70
    cpfslt STEP_NUMBER
    goto  step_3nhalf

    movlw  0x60                ;Is the microstep == 60?
    cpfseq STEP_NUMBER
    goto  cont_96_111          ;No, continue loading PWM values
    call  point_to_begining_of_table ;Yes,Point the Table pointer to begining of the Table
cont_96_111
    btfsc  MOTOR_DIRECTION,0
    goto    fwd_96_111
    movlw  0x3C                ;If Motor is reverse Wng1&Wng2 current +ve
    goto    rev_96_111
fwd_96_111
    movlw  0x38                ;If Motor is Forward Wng1 current +ve,Wng2 -ve
rev_96_111
    movwf  PORTB
    call   CCP1_INCREASE       ;Load the PWM1 values and increment Table pointer
    return

;-----
;Refer Table-2 Section 2.8 (microstep range from 3 and Half step to 4 complete step/0 step)
;-----
step_3nhalf
    movlw  0x80
    cpfslt STEP_NUMBER
    goto  CLEAR_STEP_NUMBER

    movlw  0x70                ;Is the microstep == 70?
    cpfseq STEP_NUMBER
    goto  cont_112_127        ;No, continue loading PWM values
    call  point_to_end_of_table ;Yes,Point the Table pointer to end of the Table
cont_112_127
    btfsc  MOTOR_DIRECTION,0
    goto    fwd_112_127
    movlw  0x3C                ;If Motor is reverse Wng1&Wng2 current +ve
    goto    rev_112_127
fwd_112_127
    movlw  0x38                ;If Motor is forward Wng1 current +ve,Wng2 -ve
rev_112_127
    movwf  PORTB
    call   CCP2_DECREASE       ;Load the PWM2 values and decrement Table pointer
    return

;*****
;If Full step control is choosen, both PWM's will be loaded with 100%
; duty cycle during initialisation and current sequence for the steps is
```

```

;controlled by Enable signals EN1(RB4) and EN2(RB5) and PWM switching signals
;CNT1(RB3) and CNT2(RB2) .
;Refer Table-1
;*****
FULL_STEP_JUMP
    movlw    0x20                ;check for 1st step
    cpfslt  STEP_NUMBER
    goto    SECOND_STEP
    btfsc   MOTOR_DIRECTION,0   ;Test motor_direction
    goto    FWD_FIRST_STEP
    movlw   0x28                ;If Motor is reverse Wng1=0,Wng2=-1
    movwf   PORTB
    call    CCP1_LOW_CCP2_HIGH
    return
FWD_FIRST_STEP
    movlw   0x18                ;If Motor is forward Wng1=+1,Wng2=0
    movwf   PORTB
    call    CCP1_HIGH_CCP2_LOW
    return
SECOND_STEP
    movlw   0x40                ;check for 2nd step
    cpfslt  STEP_NUMBER
    goto    THIRD_STEP
    btfsc   MOTOR_DIRECTION,0   ;Test motor_direction
    goto    FWD_SECOND_STEP
    movlw   0x14                ;If Motor is reverse Wng1=-1,Wng2=0
    movwf   PORTB
    call    CCP1_HIGH_CCP2_LOW
    return
FWD_SECOND_STEP
    movlw   0x24                ;If Motor is forward Wng1=0,Wng2=+1
    movwf   PORTB
    call    CCP1_LOW_CCP2_HIGH
    return
THIRD_STEP
    movlw   0x60                ;check for 3rd step
    cpfslt  STEP_NUMBER
    goto    FORTH_STEP
    btfsc   MOTOR_DIRECTION,0   ;Test motor_direction
    goto    FWD_THIRD_STEP
    movlw   0x24                ;If Motor is reverse Wng1=0,Wng2=+1
    movwf   PORTB
    call    CCP1_LOW_CCP2_HIGH
    return
FWD_THIRD_STEP
    movlw   0x14                ;If Motor is forward Wng1=-1,Wng2=0
    movwf   PORTB
    call    CCP1_HIGH_CCP2_LOW
    return
FORTH_STEP
    movlw   0x80                ;check for 4th step
    cpfslt  STEP_NUMBER
    goto    CLEAR_STEP_NUMBER
    btfsc   MOTOR_DIRECTION,0   ;Test motor_direction
    goto    FWD_FORTH_STEP
    movlw   0x18                ;If Motor is reverse Wng1=+1,Wng2=0
    movwf   PORTB
    call    CCP1_HIGH_CCP2_LOW
    return
FWD_FORTH_STEP
    movlw   0x28                ;If Motor is forward Wng1=0,Wng2=-ve
    movwf   PORTB
    call    CCP1_LOW_CCP2_HIGH
    return
CLEAR_STEP_NUMBER

```

AN822

```
    clrf      STEP_NUMBER
    return

;*****
;This routine checks for the keys pressed after waiting for the key to debounce
;
;    a) Motor Forward/Reverse Key connected to RD6
;    Toggle switch, toggles between Forward and reverse with each press
;
;    b) Motor Inch Key connected to RD
;    Moves the motor by a step with each press in the direction selected
;    by Fwd/Rev key previously
;*****
check_key
    btfsc    PORTD,7                ;Is key pressed "INCH"?
    goto    check_fwd_rev_key
    call    key_debounce            ;Yes, wait for debounce
    btfss    FLAG_BYTE,DEBOUNCE
    return
    bcf     FLAG_BYTE,DEBOUNCE      ;If key pressed < debounce time,
    bcf     INTCON,TMR0IE          ;If debounced,Disable Timer0 interrupt
    bcf     PIE1,ADIE
    call    UPDATE_STEP_NUMBER     ;Update the step
    call    SET_DIP_PARAMETERS
    call    UPDATE_PWM_STEP
    bcf     FLAG_BYTE,0            ;Clear the flag for PWM updation
    return

check_fwd_rev_key
    btfss    PORTD,6                ;Fwd/Rev key pressed?
    goto    fwd_rev_key_pressed
    clrf    COUNTER                ;No, clear debounce counter
    clrf    COUNTER1               ;No, clear debounce counter
    return
fwd_rev_key_pressed
    call    key_debounce            ;Yes Fwd/Rev key pressed,wait to debounce
    btfss    FLAG_BYTE,DEBOUNCE
    return
    bcf     FLAG_BYTE,DEBOUNCE      ;If key pressed < debounce time,
    bsf     INTCON,TMR0IE          ;Enable Timer0 Interrupt
    bsf     PIE1,ADIE
    btfsc    MOTOR_DIRECTION,0
    goto    set_revdirction_bit
    bsf     MOTOR_DIRECTION,0      ;Set Motor direction bit to Forward
    return
set_revdirction_bit
    bcf     MOTOR_DIRECTION,0      ;Set Motor direction bit to Reverse
    return
;*****
;This routine Updates the step count depending upon the Number of Microsteps/step
;entered by the user
;*****
UPDATE_STEP_NUMBER
    movf    STEP_JUMP,W
    addwf   STEP_NUMBER,1          ;Add step jump count to the present step number
    btfsc   STEP_NUMBER,7          ;If Step number = 80h then clear the count
    clrf    STEP_NUMBER
    return
;*****
;This routine waits for key to debounce after it is pressed the count value is 0x3ff
;*****
key_debounce
    incf    COUNTER,1,ACCESS        ;After key press is senced, increment COUNTER
    movlw   0x12                    ;If counter == 0xFF
    cpfseq  COUNTER,ACCESS
    goto    return_from_debounce
```



```

    incf     COUNTER1,1,ACCESS      ;Increment Counter1
    movlw   0x1                    ;If counter1 == 0x3
    cpfseq  COUNTER1,ACCESS
    goto    return_from_debounce
    bsf     FLAG_BYTE,DEBOUNCE,ACCESS ;Set debounce flag(key press success)
    return

return_from_debounce
    bcf     FLAG_BYTE,DEBOUNCE,ACCESS ;If key pressed < debounce time,
    return ;Key press is not sucessful

;*****
;This routine sets the number of microsteps based on the DIP switch settings.
;Also this sets the TMR0 prescale value based on the number of microsteps.
;*****
SET_DIP_PARAMETERS
    movlw   0x07                    ;DIP switches connected to PORTD<2:0>
    andwf   PORTD,W                 ;Other bits removed
    movwf   TEMP
    bsf     STATUS,C
    movlw   0x7
    subfwb  TEMP,F
    movff   TEMP,MICRO_STEPS       ;

SET_MICROSTEPS
    movlw   0x01                    ;Is microsteps/step setting==1?
    cpfseq  TEMP
    goto    CHECK_FOR_2            ;No, check for next
    movlw   0x20                    ;Yes, then
    movwf   STEP_JUMP              ;STEP_JUMP = 20h
    movlw   0x86                    ;Load the TOCON with value
    movwf   TOCON                  ;TMR0 ON and prescalar is 1:64
    return

CHECK_FOR_2
    movlw   0x02                    ;Is microsteps/step setting==2?
    cpfseq  TEMP
    goto    CHECK_FOR_3            ;No, check for next
    movlw   0x10                    ;Yes, then
    movwf   STEP_JUMP              ;STEP_JUMP = 10h
    movlw   0x85                    ;Load the TOCON with value
    movwf   TOCON                  ;TMR0 ON and prescalar is 1:32
    return

CHECK_FOR_3
    movlw   0x03                    ;Is microsteps/step setting==4?
    cpfseq  TEMP
    goto    CHECK_FOR_4            ;No, check for next
    movlw   0x08                    ;Yes, then
    movwf   STEP_JUMP              ;STEP_JUMP = 08h
    movlw   0x84                    ;Load the TOCON with value
    movwf   TOCON                  ;TMR0 ON and prescalar is 1:16
    bsf     FLAG_BYTE,3            ;Set FLAG_BYTE<3> for table manipulation
    return

CHECK_FOR_4
    movlw   0x04                    ;Is microsteps/step setting==8?
    cpfseq  TEMP
    goto    CHECK_FOR_5            ;No, check for next
    movlw   0x04                    ;Yes, then
    movwf   STEP_JUMP              ;STEP_JUMP = 4h
    movlw   0x83                    ;Load the TOCON with value
    movwf   TOCON                  ;TMR0 ON and prescalar is 1:8
    bsf     FLAG_BYTE,3            ;Set FLAG_BYTE<3> for table manipulation
    return

CHECK_FOR_5
    movlw   0x05                    ;Is microsteps/step setting==16?
    cpfseq  TEMP
    goto    CHECK_FOR_6            ;No, check for next
    movlw   0x02                    ;Yes, then

```

AN822

```
    movwf    STEP_JUMP                ;STEP_JUMP = 2h
    movlw   0X82                      ;Load the T0CON with value
    movwf   T0CON                    ;TMR0 ON and prescalar is 1:4
    bsf     FLAG_BYTE,3              ;Set FLAG_BYTE<3> for table manipulation
    return
CHECK_FOR_6
    movlw   0x01                      ;Yes, then
    movwf   STEP_JUMP                ;STEP_JUMP = 1h
    movlw   0X81                      ;Load the T0CON with value
    movwf   T0CON                    ;TMR0 ON and prescalar is 1:2
    return
;*****
;This routine reads the PWM values from the table and loads to the CCPR1L and
;CCP1CON<5:4> and increments Table pointer to next value appropriately,
;based on the number of microsteps selected
;*****
CCP1_INCREASE
    TBLRD*+
    movff   TABLAT,CCPR1L
    TBLRD*+
    bcf     CCP1CON,4                  ;Update the PWM duty cycle from the table
    bcf     CCP1CON,5
    movf    TABLAT,0
    iorwf   CCP1CON,1
    btfsc   FLAG_BYTE,3
    call    table_adjust_positive     ;Update the table for next value
    return
;*****
;This routine reads the PWM values from the table and loads to the CCPR1L and
;CCP1CON<5:4> and decrements Table pointer to next value appropriately,
;based on the number of microsteps selected
;*****
CCP1_DECREASE
    TBLRD*-
    bcf     CCP1CON,4                  ;Update the PWM duty cycle from the table
    bcf     CCP1CON,5
    movf    TABLAT,0
    iorwf   CCP1CON,1
    TBLRD*-
    movff   TABLAT,CCPR1L
    btfsc   FLAG_BYTE,3
    call    table_adjust_negative     ;Update the table for next value
    return
;*****
;This routine reads the PWM values from the table and loads to the CCPR2L and
;CCP2CON<5:4> and increments Table pointer to next value appropriately,
;based on the number of microsteps selected
;*****
CCP2_INCREASE
    TBLRD*+
    movff   TABLAT,CCPR2L            ;Read the values from Table and update PWM duty cycle
    TBLRD*+                          ;(10 bits)of CCP2
    bcf     CCP2CON,4
    bcf     CCP2CON,5
    movf    TABLAT,0
    iorwf   CCP2CON,1
    btfsc   FLAG_BYTE,3
    call    table_adjust_positive     ;Adjust the table pointer
    return
;*****
;This routine reads the PWM values from the table and loads to the CCPR2L and
;CCP2CON<5:4> and decrements Table pointer to next value appropriately,
```

```

;based on the number of microsteps selected
;*****
CCP2_DECREASE
    TBLRD*-
    bcf     CCP2CON,4
    bcf     CCP2CON,5
    movf   TABLAT,0
    iorwf  CCP2CON,1
    TBLRD*-
    movff  TABLAT,CCPR2L
    btfsc  FLAG_BYTE,3
    call   table_adjust_negative    ;Update the table for next value
    return

;*****
;This routine adjusts the Table pointer to the begining of the Table,
; which is changed due to the previous table operations
;*****
point_to_begining_of_table
    TBLRD*+
    TBLRD*+
    TBLRD*+
    call   table_adjust_positive    ;Used for skipping the table contents for
                                        ;ustep<16
    btfsc  FLAG_BYTE,3
    call   table_adjust_positive    ;Update the table for next value
    return

;*****
;This routine adjusts the Table pointer to the end of the Table,
; which is changed due to the previous table operations
;*****
point_to_end_of_table
    TBLRD*-
    TBLRD*-
    TBLRD*-
    call   table_adjust_negative    ;Used for skipping the table contents for
                                        ;ustep<16
    btfsc  FLAG_BYTE,3
    call   table_adjust_negative    ;Upadte the table for next value
    return

;*****
;This routine advances the Table pointer by (STEP_JUMP-1)*2 times,
;used for Table pointer updation for -16,-8,-4,-2 microsteps/step
;*****
table_adjust_positive
    movf   STEP_JUMP,W
    dcfsnz WREG,F                                ;W= STEP_JUMP-1
    return
    rlncf  WREG,W                                ;W=(STEP_JUMP-1)*2
    bcf    STATUS,C
    addwfc TBLPTRL,F                            ;TablePointer= Table_pointer+W
    clrf   WREG
    addwfc TBLPTRH,F
    addwfc TBLPTRU,F
    return

;*****
;This routine updates(subtracts) the Table pointer by (STEP_JUMP-1)*2 times,
;used for Table pointer updation for -16,-8,-4,-2 microsteps/step
;*****
table_adjust_negative
    movf   STEP_JUMP,W
    dcfsnz WREG,F                                ;W= STEP_JUMP-1

```

AN822

```
    return
    rlnsf      WREG,W                ;W=(STEP_JUMP-1)*2
    bsf       STATUS,C
    subwfb    TBLPTRL,F             ;TablePointer= Table_pointer-W
    clrf      WREG
    subwfb    TBLPTRH,F
    subwfb    TBLPTRU,F
    return

;*****
;Make PWM1 high and PWM2 Low
;*****
CCP1_HIGH_CCP2_LOW
    movlw    0x0FF                ;Set CCP1L 100% duty cycle
    movwf    CCP1L                ;8MSB's of duty cycle
    movlw    0x30                 ;2 LSB's at CCPxCON<5:4>
    iorwf    CCP1CON,F
    movlw    0X0                 ;Set CCP2 duty cycle to 0%
    movwf    CCP2L
    movlw    0X0CF
    andwf    CCP2CON,F
    return

;*****
;Make PWM1 Low and PWM2 High
;*****
CCP1_LOW_CCP2_HIGH
    movlw    0x0                 ;Set CCP1L 0% duty cycle
    movwf    CCP1L                ;8MSB's of duty cycle
    movlw    0x0CF               ;2 LSB's at CCPxCON<5:4>
    andwf    CCP1CON,F
    movlw    0X0FF              ;Set CCP2 duty cycle to 100%
    movwf    CCP2L
    movlw    0X030
    iorwf    CCP2CON,F
    return

;*****
;This routine sets the ADC GO bit high after an aquisition time of 20uS approx.
;*****
SET_ADC_GO
    call     CALCULATE_RPM1
    btfss   ADCON0,GO
    bsf     ADCON0,GO             ;Set GO bit for ADC conversion start
    return

;*****
;This routine calculates the RPM with Potentiometer
;*****
CALCULATE_RPM1
    clrf    TEMP
    clrf    TEMP1
    movf    RPM_VALUE+1,W
    btfsc   STATUS,Z
    incf    RPM_VALUE+1,F
    movlw   0x2F
    movwf   TEMP_LOCATION
    movlw   0xAE
    movwf   TEMP_LOCATION+1
continue_subtraction1
    bsf     STATUS,C
    movf    RPM_VALUE+1,W
    subwfb  TEMP_LOCATION+1,F
    movlw   0x0
    subwfb  TEMP_LOCATION,F
```

```

    btfss    STATUS,C
    goto     keep_result_in_rpm1
    incf     TEMP,F
    btfsc    STATUS,C
    incf     TEMP1,F
    goto     continue_subtraction1

keep_result_in_rpm1
;Timer0 value = FFFF-Timer0
    rlc     TEMP,F
    rlc     TEMP1,F
    bcf     STATUS,C
    rlc     TEMP,F
    rlc     TEMP1,F
    bsf     STATUS,C
    movlw   0xFF
    subfwb  TEMP,F
    subfwb  TEMP1,F
    movff   TEMP1,SPEED_REF_H
    movff   TEMP,SPEED_REF_L
    return

;*****
;Table for the microsteps.
;Even numbered values are loaded to CCPxL and
;Odd numbered values are ORed with CCPxCON(CCPxCON<5:4>)
;to load complete 10 bits of PWMx duty cycle
;*****
TABLE    code 0x02A0
sine_table db    0x0,0x00,0x20,0x30,0x34,0x20,0x42,0x10,0x50,0x0,0x5C,0x10,0x68,0x20
            db    0x74,0x30,0x80,0x30,0x8C,0x30,0x98,0x30,0xA4,0x20,0xB0,0x10,0xC3,0x10
            db    0xD6,0x20,0xEA,0x20,0xFF,0x30

;*****
;This routine loads the data in Wreg to Transmission register(TXREG) after checking
;of completion of previously loaded byte transmission
;*****
load_RX_REG_from_WREG

    btfss    PIR1,TXIF
    goto     load_RX_REG_from_WREG
    movwf   TXREG
    return

;*****
;This routine processes the command from host PC. If is is command, then
;FLAG_BYTE<5> is set and data is awaited.
;*****
PROCESS_COMMAND
    bcf     FLAG_BYTE,4                ;Flag for byte recieved interrupt

    btfsc   FLAG_BYTE,5                ;Flag for differanciating Command and data
                                        ;If set, recieved_byte is data

    goto    CHECK_DATA
    movff   RECIEVED_BYTE,COMMAND_BYTE
    bcf     PIE1,ADIE                  ;Enable Speed ref from Pot
    bcf     INTCON,TMR0IE              ;Enable Timer0 interrupt

CHECK_DATA
    movlw   0x030                      ;Is Received command=exit from steup?
    cpfseq  COMMAND_BYTE
    goto    CHECK_FOR_STEP_VALUE       ;No, check for change of usteps?
    call    DISPLAY_EXIT_SETUP         ;Display exit from setup
    call    SET_DIP_PARAMETERS        ;set the microsteps according to the DIP switches
    bcf     FLAG_BYTE,5

```

AN822

```
    bcf     FLAG_BYTE, 6
    bcf     FLAG_BYTE, 0
    bsf     PIE1, ADIE           ;Enable Speed ref from Pot
    bsf     INTCON, TMR0IE      ;Enable Timer0 interrupt
    return
;-----
CHECK_FOR_STEP_VALUE
    movlw   0x031               ;Is Received command=change no. of usteps?
    cpfseq  COMMAND_BYTE
    goto    CHECK_FOR_DIRECTION ;No, check for direction change
    btfsc   FLAG_BYTE, 5
    goto    SET_MICROSTEPS_DATA
    call    DISPLAY_STEPS_VALUE ;Display allowed microstep values
    bsf     FLAG_BYTE, 5
    return

SET_MICROSTEPS_DATA
    movlw   0x30               ;Data received for microstep
    subwf   RECIEVED_BYTE, W
    movwf   MICRO_STEPS
    btfsc   STATUS, Z
    goto    NOT_VALID_ENTRY
    movlw   0x7
    cpfslt  MICRO_STEPS
    goto    NOT_VALID_ENTRY
    movff   MICRO_STEPS, TEMP
    call    SET_MICROSTEPS     ;Set the microsteps according to the data entry
    movlw   0xA
    call    load_RX_REG_from_WREG
    bcf     FLAG_BYTE, 5
    return
;-----
CHECK_FOR_DIRECTION
    movlw   0x032               ;Is Received command=change of direction?
    cpfseq  COMMAND_BYTE
    goto    CHECK_FOR_INCH_STEP ;No, check for No. of steps to inch
    btfsc   FLAG_BYTE, 5
    goto    SET_DIRECTION_DATA
    call    DISPLAY_STEPS_DIRECTION ;Display allowed direction values
    bsf     FLAG_BYTE, 5
    return

SET_DIRECTION_DATA
    movlw   0x30               ;Test the recieved data for Forward
    cpfseq  RECIEVED_BYTE
    goto    test_0x31
    clrf    MOTOR_DIRECTION    ;Direction commanded is forward
    call    MOTOR_RUN_FORWARD
    bsf     INTCON, TMR0IE      ;Enable Timer0 interrupt
    bcf     FLAG_BYTE, 5
    return
test_0x31
    movlw   0x31               ;Check for reverse rotation command
    cpfseq  RECIEVED_BYTE
    goto    NOT_VALID_ENTRY
    movlw   0x1
    movwf   MOTOR_DIRECTION    ;Yes, set the motor direction to reverse
    call    MOTOR_RUN_REVERSE
    bsf     INTCON, TMR0IE      ;Enable Timer0 interrupt
    bcf     FLAG_BYTE, 5
    return
;-----
CHECK_FOR_INCH_STEP
    movlw   0x033               ;Is Received command=Inch step?
    cpfseq  COMMAND_BYTE
```

```

goto    CHECK_FOR_RPM                ;No,check for RPM of motor
btfsc   FLAG_BYTE,5
goto    SET_INCH_STEPS_DATA
call    DISPLAY_STEPS_INCH           ;Display allowed INCH step values
movlw   HIGH(RPM_VALUE+3)
movwf   FSR2H
movlw   LOW(RPM_VALUE+3)
movwf   FSR2L
movlw   0xFF                          ;Allow 3 digits of hata to be entered
movwf   POSTDEC2
movwf   POSTDEC2
movwf   POSTDEC2
movwf   INDF2
bsf     FLAG_BYTE,5
return

SET_INCH_STEPS_DATA
movlw   0x2F
cpfsgt  RECIEVED_BYTE
goto    INCH_COMMAND_READ           ;Check the data entered >0<9
movlw   0x3A
cpfslt  RECIEVED_BYTE               ;If the data is other than 0-9,
goto    INCH_COMMAND_READ           ;consider that data entry is over
movlw   0x30
subwf   RECIEVED_BYTE,W
movwf   POSTINC2                     ;Store the data in the RPM_VALUE
return

INCH_COMMAND_READ
call    MERGE_NUMBERS                ;Concatinate the entered numbers
call    CONVERT_TO_HEX               ;Convert the concatinated nubers to Hex
movff   RPM_VALUE,INCH_VALUE         ;Load the hex value to the registers INCH_VALUE
movff   RPM_VALUE+1,INCH_VALUE+1
bsf     INTCON,TMR0IE                ;Enable Timer0 overflow interrupt
bsf     FLAG_BYTE,6
bcf     FLAG_BYTE,5
movlw   0xA
call    load_RX_REG_from_WREG
return

;-----
CHECK_FOR_RPM
movlw   0x034                          ;Is Received command=set RPM?
cpfseq  COMMAND_BYTE
goto    NOT_VALID_ENTRY              ;No,Not a correct command
btfsc   FLAG_BYTE,5
goto    SET_RPM_DATA
call    DISPLAY_STEPS_RPM            ;Display allowed RPM range
movlw   HIGH(RPM_VALUE+3)
movwf   FSR2H
movlw   LOW(RPM_VALUE+3)
movwf   FSR2L
movlw   0xFF
movwf   POSTDEC2
movwf   POSTDEC2
movwf   POSTDEC2
movwf   INDF2
bsf     FLAG_BYTE,5
return

SET_RPM_DATA
movlw   0x2F                          ;Check the data entry range 0 to 9
cpfsgt  RECIEVED_BYTE
goto    RPM_COMMAND_READ
movlw   0x3A
cpfslt  RECIEVED_BYTE
goto    RPM_COMMAND_READ
movlw   0x30                          ;Store th ereceived data in the RPM_VALUES
subwf   RECIEVED_BYTE,W

```

AN822

```
    movwf    POSTINC2
    return
RPM_COMMAND_READ
    bcf      PIE1,ADIE          ;Disable Speed ref from Pot
    call     convert_RPM_to_HEX ;Convert the data to Hex
    call     CALCULATE_RPM     ;Calculate the Timer0 value from the RPM data entered
    call     display_motor_direction
    bsf      INTCON,TMR0IE     ;Enable Timer0 ineterrupt
    bcf      FLAG_BYTE,5
    return
;-----
NOT_VALID_ENTRY
    call     DATA_NOT_VALID   ;If the entered command is other than the valid commands,
    call     send_command_request ; then display the DATA_INVALID
    bcf      FLAG_BYTE,5
    return

;*****
;This routine decrements the INCH count given from host PC to move in INCH mode
;*****
DECREMENT_INCH_COUNT
    decf    INCH_VALUE+1,F     ;Decrement the inching counter
    btfss   STATUS,C          ;after each step movement
    decf    INCH_VALUE,F
    btfsc   STATUS,C
    return
    bcf     FLAG_BYTE,5        ;If all steps are over, clear the flag_bytes
    bcf     FLAG_BYTE,6
    bcf     INTCON,TMR0IE     ;If Inch_count==0,Disable Timer0 interrupt
    return

;*****
;RPM value from the Host PC is entered in discrete decimal digits.
;This routine merges the digits to get the RPM value and then
;converts the RPM in decimal value to Hex value
;*****
convert_RPM_to_HEX
    clrf    TEMP              ;
    clrf    TEMP1

    call    MERGE_NUMBERS     ;Merge the individual digits
    call    CONVERT_TO_HEX    ;Convert the entered number to Hex

    movlw   0xC8              ;RPM limit set to 200 RPM
    cpfslt  RPM_VALUE+1      ;
    movwf   RPM_VALUE+1
    return

;*****
;RPM value from the Host PC is entered in discrete decimal digits.
;This routine merges the numbers
;*****
MERGE_NUMBERS
    movlw   0xA              ;Third digit is checked for number
    cpfslt  RPM_VALUE+2
    goto    check_second     ;If it has a valid number,concatinate
    swapf   RPM_VALUE+1,W    ;RPM_VALUE2 and 3 and keep in RPM_VALUE2
    iorwf   RPM_VALUE+2,W
    movwf   RPM_VALUE+1
    return
check_second
    movlw   0xA              ;If second digit is a valid number,
    cpfslt  RPM_VALUE+1      ;concatinate RPM_VALUE2 and 1 and place in
    goto    place_number     ;RPM_VALUE2
    swapf   RPM_VALUE,W
    iorwf   RPM_VALUE+1,F
    clrf    RPM_VALUE
```



```

    return
place_number
    movff    RPM_VALUE,RPM_VALUE+1    ;If only one digit is entered,keep it in
    clrf    RPM_VALUE                ;RPM_VALUE2 and clear RPM_VALUE1
    return

;-----
    movlw   HIGH(RPM_VALUE+2)
    movwf   FSR2H
    movlw   LOW(RPM_VALUE+2)
    movwf   FSR2L
    movlw   0xA                        ;Third digit is checked for number
    cpfslt  POSTDEC2
    goto    check_second                ;If it has a valid number,concatinate
    swapf   POSTINC2,W                 ;RPM_VALUE2 and 3 and keep in RPM_VALUE2
    iorwf   POSTDEC2,W
    movwf   INDF2
    return
;check_second
    movlw   0xA                        ;If second digit is a valid number,
    cpfslt  POSTDEC2                   ;concatinate RPM_VALUE2 and 1 and place in
    goto    place_number                ;RPM_VALUE2
    swapf   INDF2,F
    movf    POSTINC2,W
    iorwf   INDF2,F
    clrf    RPM_VALUE
    return
;place_number
    movff    RPM_VALUE,RPM_VALUE+1    ;If only one digit is entered,keep it in
    clrf    RPM_VALUE                ;RPM_VALUE2 and clear RPM_VALUE1
    return
;*****
;This routine converts the decimal RPM value to Hex
;*****
CONVERT_TO_HEX
    clrf    TEMP
continue_conversion_hex
    bsf     STATUS,C                    ;Divide the number by 0x16 by
    movlw   0x16                        ;Subtracting continuously 0x16 from the
    subwfb  RPM_VALUE+1,F                ;value
    movlw   0x0                          ;The result is stored in TEMP
    subwfb  RPM_VALUE,F
    btfss   STATUS,C
    goto    set_LS_nibble                ;
    movlw   0x9F
    cpfsgt  RPM_VALUE+1
    goto    check_ls_nibble
    movlw   0x60
    subwf   RPM_VALUE+1,F
check_ls_nibble
    movf    RPM_VALUE+1,W
    andlw   0x0F
    movwf   TEMP1
    movlw   0x9
    cpfsgt  TEMP1
    goto    increment_count
    movlw   0x6
    subwf   RPM_VALUE+1,F

increment_count
    incf    TEMP,F
    goto    continue_conversion_hex

set_LS_nibble
    movlw   0x16

```

AN822

```
    addwf    RPM_VALUE+1,F
    movlw   0x9
    cpfsgt  RPM_VALUE+1
    goto    set_hex_value
    movlw   0x6
    subwf   RPM_VALUE+1,F
set_hex_value
    swapf   TEMP,W           ;Set the values in RPM_VALUE(H) and
    andlw   0xF0             ;RPM_VALUE+1(L)
    iorwf   RPM_VALUE+1,F
    swapf   TEMP,W
    andlw   0x0F
    movwf   RPM_VALUE
    return
;*****
;Timer0 reload value is calculated from the RPM_VALUE and the microsteps setting
;Calculation of Timer0 reload value:
;Step1: No. of steps/sec      (A) = (RPM/60) * microsteps
;Step2: Time/1 step           (B) = 1/A
;Step3: Timer0 counts for B sec (C) = B * 20 MHz
;Step4: Divide by Timer0 prescaler (D) = C/Timer0 prescaler
;Step5: Subtract count from FFFFh, E = FFFF - D
;So E is the Timer0 reload value
;*****
CALCULATE_RPM
    movf    RPM_VALUE+1,W
    btfsc   STATUS,Z
    incf    WREG,W
    mullw   STEPS_PER_ROTATION ;RPM*No of steps/revolution
    movff   PRODL,TEMP_RPM+2
    movff   PRODH,TEMP_RPM+1
    clrf    TEMP_RPM
    movf    TOCON,W           ;
    andlw   0x07
    addwf   MICRO_STEPS,W
    movwf   TEMP
    bcf     STATUS,C
repeat_shift_left
    rlcfsz  TEMP_RPM+2,F      ;(RPM*No of steps/revolution)*
    rlcfsz  TEMP_RPM+1,F      ;(TMR0 prescaler*No. of steps/step)
    rlcfsz  TEMP_RPM,F
    decfsz  TEMP,F
    goto    repeat_shift_left
    call    divide_3_8zero_by_rpm
    return
;*****
;(60/0.2*10-6)=300000000d = 11E1A300h
;Timer0 value for RPM = 11E1A300h/((steps/revolution*Timer0 prescaler value)*(RPM))
;*****
divide_3_8zero_by_rpm
    clrf    TEMP
    clrf    TEMP1
    movlw   0x11             ;Temp_loation =0x11e1a300
    movwf   TEMP_LOCATION   ;          == 60/0.2X(10e-6)
    movlw   0xE1
    movwf   TEMP_LOCATION+1
    movlw   0xA3
    movwf   TEMP_LOCATION+2
    clrf    TEMP_LOCATION+3
continue_subtraction
    movlw   HIGH(TEMP_LOCATION+3)
    movwf   FSR0H
    movlw   LOW(TEMP_LOCATION+3)
    movwf   FSR0L
    movlw   HIGH(TEMP_RPM+2)
```

```

    movwf    FSR1H
    movlw   LOW(TEMP_RPM+2)
    movwf   FSR1L
    bsf     STATUS,C
    movf    POSTDEC1,W
    subwfb  POSTDEC0,F
    movf    POSTDEC1,W
    subwfb  POSTDEC0,F
    movf    POSTDEC1,W
    subwfb  POSTDEC0,F
    movlw   0x0
    subwfb  POSTDEC0,F
    btfss   STATUS,C
    goto    keep_result_in_rpm
    incf    TEMP,F
    btfsc   STATUS,C
    incf    TEMP1,F
    goto    continue_subtraction
keep_result_in_rpm
;Timer0 value = FFFF-Timer0
    bsf     STATUS,C
    movlw   0xFF
    subwfb  TEMP,F
    subwfb  TEMP1,F
    movff   TEMP1,SPEED_REF_H
    movff   TEMP,SPEED_REF_L
    return
;*****
;Displays the direction of motor on the host PC screen
;*****
display_motor_direction
    movlw   0x0
    cpfseq  MOTOR_DIRECTION
    goto    display_reverse_dir
    call    MOTOR_RUN_FORWARD
    return
display_reverse_dir
    call    MOTOR_RUN_REVERSE
    return
;*****
;This routine intializes the USART module to communicate with host PC and displays
;a welcome message on the screen
;*****
STEPPER_COM
    movlw   0xA
    movwf   TEMP
repeat_send
    incf    TEMP,F
    incf    TEMP,F
    movlw   0x62
    cpfseq  TEMP
    goto    send
    return
send
    movf    TEMP,W
    call    send_welcome
    call    load_RX_REG_from_WREG
    goto    repeat_send

send_welcome
    bcf     STATUS,C
    addwf   PCL,W
    movwf   TEMP1
    movlw   0x0
    addwfc  PCLATH,F

```

AN822

```
addwfc    PCLATU, F
movff     TEMP1, PCL
retlw    'W'
retlw    'e'
retlw    'l'
retlw    'c'
retlw    'o'
retlw    'm'
retlw    'e'
retlw    ' '
retlw    't'
retlw    'o'
retlw    ' '
retlw    'M'
retlw    'i'
retlw    'c'
retlw    'r'
retlw    'o'
retlw    's'
retlw    't'
retlw    'e'
retlw    'p'
retlw    'p'
retlw    'i'
retlw    'n'
retlw    'g'
retlw    ' '
retlw    'o'
retlw    'f'
retlw    ' '
retlw    'S'
retlw    't'
retlw    'e'
retlw    'p'
retlw    'p'
retlw    'e'
retlw    'r'
retlw    ' '
retlw    'M'
retlw    'o'
retlw    't'
retlw    'o'
retlw    'r'
retlw    0x0A
retlw    0x0D
;*****
;This routine displays the list commands with their explanation on the host PC screen
;*****
send_command_request
    movlw    0xA
    movwf    TEMP
repeat_send_com_req
    incf     TEMP, F
    incf     TEMP, F
    movlw    0x4A
    cpfseq   TEMP
    goto     send_com_req
    call     show_commands
    return
send_com_req
    movf     TEMP, W
    call     send_command_request
    call     load_RX_REG_from_WREG
    goto     repeat_send_com_req
send_command_request
```

```

bcf      STATUS,C
addwf   PCL,W
movwf   TEMP1
movlw   0x0
addwfc  PCLATH,F
addwfc  PCLATU,F
movff   TEMP1,PCL
retlw   0x0A
retlw   0x0A
retlw   0x0D
retlw   'E'
retlw   'n'
retlw   't'
retlw   'e'
retlw   'r'
retlw   ' '
retlw   't'
retlw   'h'
retlw   'e'
retlw   ' '
retlw   'r'
retlw   'e'
retlw   'q'
retlw   'u'
retlw   'i'
retlw   'r'
retlw   'e'
retlw   'd'
retlw   ' '
retlw   'c'
retlw   'o'
retlw   'm'
retlw   'm'
retlw   'a'
retlw   'n'
retlw   'd'
retlw   0x0A
retlw   0x0D
;-----
show_commands
    movlw   0xA
    movwf   TEMP
repeat_show_com
    incf    TEMP,F
    incf    TEMP,F
    movlw   0xF6
    cpfseq  TEMP
    goto    show_com
    return
show_com
    movf    TEMP,W
    call    show_command
    call    load_RX_REG_from_WREG
    goto    repeat_show_com
show_command
    bcf     STATUS,C
    addwf   PCL,W
    movwf   TEMP1
    movlw   0x0
    addwfc  PCLATH,F
    addwfc  PCLATU,F
    movff   TEMP1,PCL
    retlw   0x0A
    retlw   0x0D
    retlw   '0'

```

AN822

```
retlw    '-'
retlw    '-'
retlw    'E'
retlw    'x'
retlw    'i'
retlw    't'
retlw    ' '
retlw    'f'
retlw    'r'
retlw    'o'
retlw    'm'
retlw    ' '
retlw    'S'
retlw    'e'
retlw    't'
retlw    'u'
retlw    'p'

retlw    0x0A
retlw    0x0D
retlw    '1'
retlw    '-'
retlw    '-'
retlw    'N'
retlw    'u'
retlw    'm'
retlw    'e'
retlw    'r'
retlw    ' '
retlw    'o'
retlw    'f'
retlw    ' '
retlw    'M'
retlw    'i'
retlw    'c'
retlw    'r'
retlw    'o'
retlw    's'
retlw    't'
retlw    'e'
retlw    'p'
retlw    's'
retlw    '/'
retlw    's'
retlw    't'
retlw    'e'
retlw    'p'

retlw    0x0A
retlw    0x0D
retlw    '2'
retlw    '-'
retlw    '-'
retlw    'D'
retlw    'i'
retlw    'r'
retlw    'e'
retlw    'c'
retlw    't'
retlw    'i'
retlw    'o'
retlw    'n'
retlw    ' '
retlw    'o'
retlw    'f'
```

```
retlw    ' '
retlw    'r'
retlw    'o'
retlw    't'
retlw    'a'
retlw    't'
retlw    'i'
retlw    'o'
retlw    'n'

retlw    0x0A
retlw    0x0D
retlw    '3'
retlw    '-'
retlw    '-'
retlw    'N'
retlw    'u'
retlw    'm'
retlw    'e'
retlw    'r'
retlw    ' '
retlw    'o'
retlw    'f'
retlw    ' '
retlw    'S'
retlw    't'
retlw    'e'
retlw    'p'
retlw    's'
retlw    ' '
retlw    't'
retlw    'o'
retlw    ' '
retlw    'I'
retlw    'n'
retlw    'c'
retlw    'h'

retlw    0x0A
retlw    0x0D
retlw    '4'
retlw    '-'
retlw    '-'
retlw    'S'
retlw    'e'
retlw    't'
retlw    ' '
retlw    'R'
retlw    'P'
retlw    'M'

retlw    0x0A
retlw    0x0A
retlw    0x0D

;*****
;This routine displays message during exit from the PC communication on the host PC screen
;*****
DISPLAY_EXIT_SETUP
    movlw    0xA
    movwf    TEMP
repeat_show_exit
    incf    TEMP,F
    incf    TEMP,F
    movlw    0x2E
```

AN822

```
    cpfseq    TEMP
    goto     show_exit
    return

show_exit
    movf     TEMP,W
    call     show_exit_setup
    call     load_RX_REG_from_WREG
    goto     repeat_show_exit

show_exit_setup
    bcf      STATUS,C
    addwf   PCL,W
    movwf   TEMP1
    movlw   0x0
    addwfc  PCLATH,F
    addwfc  PCLATU,F
    movff   TEMP1,PCL
    retlw   0x0A
    retlw   0x0D
    retlw   'B'
    retlw   'Y'
    retlw   'E'
    retlw   ' '
    retlw   'B'
    retlw   'Y'
    retlw   'E'
    retlw   ' '
    retlw   '.'
    retlw   '.'
    retlw   '.'
    retlw   '.'
    retlw   0x0A
    retlw   0x0D

;*****
;This routine displays the list of data available for microstep selection on the host PC screen
;*****
DISPLAY_STEPS_VALUE
    movlw   0xA
    movwf   TEMP
repeat_show_step_value
    incf    TEMP,F
    incf    TEMP,F
    movlw   0xD0
    cpfseq  TEMP
    goto    show_step
    return

show_step
    movf    TEMP,W
    call    show_step_command
    call    load_RX_REG_from_WREG
    goto    repeat_show_step_value

show_step_command
    bcf     STATUS,C
    addwf  PCL,W
    movwf  TEMP1
    movlw  0x0
    addwfc PCLATH,F
    addwfc PCLATU,F
    movff  TEMP1,PCL
    retlw  0x0A
    retlw  0x0D
    retlw  'E'
    retlw  'n'
    retlw  't'
```



```
retlw 'e'
retlw 'r'
retlw ' '
retlw 't'
retlw 'h'
retlw 'e'
retlw ' '
retlw 'N'
retlw 'o'
retlw '.'
retlw ' '
retlw 'o'
retlw 'f'
retlw ' '
retlw 'M'
retlw 'i'
retlw 'c'
retlw 'r'
retlw 'o'
retlw 's'
retlw 't'
retlw 'e'
retlw 'p'
retlw 's'
retlw 0x0A
retlw 0x0D
retlw 'E'
retlw 'n'
retlw 't'
retlw 'e'
retlw 'r'
retlw ' '
retlw '1'
retlw ' '
retlw 'f'
retlw 'o'
retlw 'r'
retlw ' '
retlw '1'
retlw ','
retlw '2'
retlw ' '
retlw 'f'
retlw 'o'
retlw 'r'
retlw ' '
retlw '2'
retlw ','
retlw '3'
retlw ' '
retlw 'f'
retlw 'o'
retlw 'r'
retlw ' '
retlw '4'
retlw ','
retlw '4'
retlw ' '
retlw 'f'
retlw 'o'
retlw 'r'
retlw ' '
retlw '8'
retlw ','
retlw '5'
```

AN822

```
retlw    ' '
retlw    'f'
retlw    'o'
retlw    'r'
retlw    ' '
retlw    '1'
retlw    '6'
retlw    ' '
retlw    'a'
retlw    'n'
retlw    'd'
retlw    ' '
retlw    '6'
retlw    ' '
retlw    'f'
retlw    'o'
retlw    'r'
retlw    ' '
retlw    '3'
retlw    '2'
retlw    ' '
retlw    's'
retlw    't'
retlw    'e'
retlw    'p'
retlw    's'
retlw    0x0A
retlw    0x0D
```

```
*****
;This routine displays the selction of motor direction on the host PC screen
*****
DISPLAY_STEPS_DIRECTION
    movlw    0xA
    movwf    TEMP
repeat_show_step_direction
    incf    TEMP,F
    incf    TEMP,F
    movlw    0x7C
    cpfseq  TEMP
    goto    show_step_direction
    return
show_step_direction
    movf    TEMP,W
    call    show_step_direction_values
    call    load_RX_REG_from_WREG
    goto    repeat_show_step_direction
show_step_direction_values
    bcf    STATUS,C
    addwf  PCL,W
    movwf  TEMP1
    movlw  0x0
    addwfc PCLATH,F
    addwfc PCLATU,F
    movff  TEMP1,PCL
    retlw  0x0A
    retlw  0x0D
    retlw  'E'
    retlw  'n'
    retlw  't'
    retlw  'e'
    retlw  'r'
    retlw  ' '
    retlw  'D'
    retlw  'i'
```

```
retlw    'r'
retlw    'e'
retlw    'c'
retlw    't'
retlw    'i'
retlw    'o'
retlw    'n'
retlw    ' '
retlw    'o'
retlw    'f'
retlw    ' '
retlw    'r'
retlw    'o'
retlw    't'
retlw    'a'
retlw    't'
retlw    'a'
retlw    'i'
retlw    'o'
retlw    'n'
retlw    0x0A
retlw    0x0D
retlw    '0'
retlw    '-'
retlw    '-'
retlw    'F'
retlw    'o'
retlw    'r'
retlw    'w'
retlw    'a'
retlw    'r'
retlw    'd'
retlw    0x0A
retlw    0x0D
retlw    '1'
retlw    '-'
retlw    '-'
retlw    'R'
retlw    'e'
retlw    'v'
retlw    'e'
retlw    'r'
retlw    's'
retlw    'e'
retlw    0x0A
retlw    0x0D

;*****
;This routine displays the message for INCH command on the host PC screen
;*****
DISPLAY_STEPS_INCH
    movlw    0xA
    movwf    TEMP
repeat_show_step_inch
    incf     TEMP,F
    incf     TEMP,F
    movlw    0x4E
    cpfseq   TEMP
    goto     show_step_inch
    return
show_step_inch
    movf     TEMP,W
    call     show_step_inch_values
    call     load_RX_REG_from_WREG
    goto     repeat_show_step_inch
```

AN822

```
show_step_inch_values
    bcf     STATUS,C
    addwf  PCL,W
    movwf  TEMP1
    movlw  0x0
    addwfc PCLATH,F
    addwfc PCLATU,F
    movff  TEMP1,PCL
    retlw  0x0A
    retlw  0x0D
    retlw  'E'
    retlw  'n'
    retlw  't'
    retlw  'e'
    retlw  'r'
    retlw  ' '
    retlw  'N'
    retlw  'u'
    retlw  'm'
    retlw  'b'
    retlw  'e'
    retlw  'r'
    retlw  ' '
    retlw  'o'
    retlw  'f'
    retlw  ' '
    retlw  's'
    retlw  't'
    retlw  'e'
    retlw  'p'
    retlw  's'
    retlw  ' '
    retlw  't'
    retlw  'o'
    retlw  ' '
    retlw  'I'
    retlw  'N'
    retlw  'C'
    retlw  'H'
    retlw  0x0A
    retlw  0x0D

;*****
;This routine displays the message for RPM command on the host PC screen
;*****
DISPLAY_STEPS_RPM
    movlw  0xA
    movwf  TEMP
repeat_show_step_rpm
    incf  TEMP,F
    incf  TEMP,F
    movlw  0x40
    cpfseq TEMP
    goto  show_step_rpm
    return
show_step_rpm
    movf  TEMP,W
    call  show_step_rpm_values
    call  load_RX_REG_from_WREG
    goto  repeat_show_step_rpm
show_step_rpm_values
    bcf   STATUS,C
    addwf PCL,W
    movwf TEMP1
    movlw 0x0
```

```

    addwfc    PCLATH,F
    addwfc    PCLATU,F
    movff     TEMP1,PCL
    retlw     0x0A
    retlw     0x0D
    retlw     'E'
    retlw     'n'
    retlw     't'
    retlw     'e'
    retlw     'r'
    retlw     ' '
    retlw     't'
    retlw     'h'
    retlw     'e'
    retlw     ' '
    retlw     'r'
    retlw     'e'
    retlw     'q'
    retlw     'u'
    retlw     'i'
    retlw     'r'
    retlw     'e'
    retlw     'd'
    retlw     ' '
    retlw     'R'
    retlw     'P'
    retlw     'M'
    retlw     0x0A
    retlw     0x0D
;*****
;This routine displays the message Motor running Forward on the host PC screen
;*****
MOTOR_RUN_FORWARD
    movlw     0xA
    movwf     TEMP
repeat_show_motor_fwd
    incf     TEMP,F
    incf     TEMP,F
    movlw     0x3E
    cpfseq   TEMP
    goto     show_fwd_running
    return
show_fwd_running
    movf     TEMP,W
    call     show_forward_running
    call     load_RX_REG_from_WREG
    goto     repeat_show_motor_fwd
show_forward_running
    bcf     STATUS,C
    addwf    PCL,W
    movwf   TEMP1
    movlw   0x0
    addwfc  PCLATH,F
    addwfc  PCLATU,F
    movff   TEMP1,PCL
    retlw   0x0A
    retlw   0x0D
    retlw   'M'
    retlw   'o'
    retlw   't'
    retlw   'o'
    retlw   'r'
    retlw   ' '
    retlw   'R'
    retlw   'u'

```

AN822

```
retlw    'n'
retlw    'n'
retlw    'i'
retlw    'n'
retlw    'g'
retlw    ' '
retlw    'F'
retlw    'o'
retlw    'r'
retlw    'w'
retlw    'a'
retlw    'r'
retlw    'd'
retlw    0x0A
retlw    0x0D

;*****
;This routine displays the message Motor running Reverse on the host PC screen
;*****
MOTOR_RUN_REVERSE
    movlw    0xA
    movwf    TEMP
repeat_show_motor_rev
    incf     TEMP,F
    incf     TEMP,F
    movlw    0x3E
    cpfseq   TEMP
    goto     show_rev_running
    return
show_rev_running
    movf     TEMP,W
    call     show_reverse_running
    call     load_RX_REG_from_WREG
    goto     repeat_show_motor_rev
show_reverse_running
    bcf     STATUS,C
    addwf   PCL,W
    movwf   TEMP1
    movlw   0x0
    addwfc  PCLATH,F
    addwfc  PCLATU,F
    movff   TEMP1,PCL
    retlw   0x0A
    retlw   0x0D
    retlw   'M'
    retlw   'o'
    retlw   't'
    retlw   'o'
    retlw   'r'
    retlw   ' '
    retlw   'R'
    retlw   'u'
    retlw   'n'
    retlw   'n'
    retlw   'i'
    retlw   'n'
    retlw   'g'
    retlw   ' '
    retlw   'R'
    retlw   'e'
    retlw   'v'
    retlw   'e'
    retlw   'r'
    retlw   's'
    retlw   'e'
```

```
    retlw    0x0A
    retlw    0x0D
;*****
;This routine displays the message Data not valid on the host PC screen
;*****
DATA_NOT_VALID
    movlw    0xA
    movwf    TEMP
repeat_show_data_not_valid
    incf     TEMP,F
    incf     TEMP,F
    movlw    0x3E
    cpfseq   TEMP
    goto     show_data_not_valid
    return
show_data_not_valid
    movf     TEMP,W
    call     show_not_valid_data
    call     load_RX_REG_from_WREG
    goto     repeat_show_data_not_valid
show_not_valid_data
    bcf      STATUS,C
    addwf    PCL,W
    movwf    TEMP1
    movlw    0x0
    addwfc   PCLATH,F
    addwfc   PCLATU,F
    movff    TEMP1,PCL
    retlw    0x0A
    retlw    0x0D
    retlw    'N'
    retlw    'O'
    retlw    'T'
    retlw    ' '
    retlw    'A'
    retlw    ' '
    retlw    'V'
    retlw    'A'
    retlw    'L'
    retlw    'I'
    retlw    'D'
    retlw    ' '
    retlw    'E'
    retlw    'N'
    retlw    'T'
    retlw    'R'
    retlw    'Y'
    retlw    '.'
    retlw    '.'
    retlw    '.'
    retlw    '.'
    retlw    0x0A
    retlw    0x0D
;*****
end
```

AN822

NOTES:

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.

Information contained in this publication regarding device applications and the like is intended through suggestion only and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. No representation or warranty is given and no liability is assumed by Microchip Technology Incorporated with respect to the accuracy or use of such information, or infringement of patents or other intellectual property rights arising from such use or otherwise. Use of Microchip's products as critical components in life support systems is not authorized except with express written approval by Microchip. No licenses are conveyed, implicitly or otherwise, under any intellectual property rights.

Trademarks


The Microchip name and logo, the Microchip logo, FilterLab, KEELOQ, microID, MPLAB, PIC, PICmicro, PICMASTER, PICSTART, PRO MATE, SEEVAL and The Embedded Control Solutions Company are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

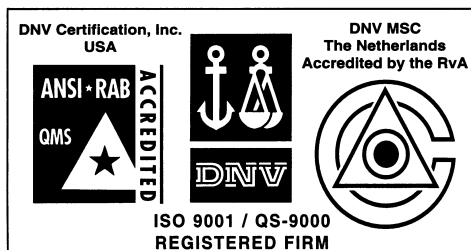
dsPIC, ECONOMONITOR, FanSense, FlexROM, fuzzyLAB, In-Circuit Serial Programming, ICSP, ICEPIC, microPort, Migratable Memory, MPASM, MPLIB, MPLINK, MPSIM, MXDEV, PICC, PICDEM, PICDEM.net, rfPIC, Select Mode and Total Endurance are trademarks of Microchip Technology Incorporated in the U.S.A.

Serialized Quick Term Programming (SQTP) is a service mark of Microchip Technology Incorporated in the U.S.A.

All other trademarks mentioned herein are property of their respective companies.

© 2002, Microchip Technology Incorporated, Printed in the U.S.A., All Rights Reserved.

 Printed on recycled paper.



Microchip received QS-9000 quality system certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona in July 1999. The Company's quality system processes and procedures are QS-9000 compliant for its PICmicro® 8-bit MCUs, KEELOQ® code hopping devices, Serial EEPROMs and microperipheral products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001 certified.



MICROCHIP

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: 480-792-7456

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: 86-28-6766200 Fax: 86-28-6766599

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: 86-755-2350361 Fax: 86-755-2366086

Hong Kong

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 Taiwan
11F-3, No. 207
Tung Hua North Road
Taipei, 105, Taiwan
Tel: 886-2-2717-7175 Fax: 886-2-2545-0139

EUROPE

Denmark

Microchip Technology Nordic ApS
Regus Business Centre
Lautrup høj 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 - 1er Etage
91300 Massy, France
Tel: 33-1-69-53-63-20 Fax: 33-1-69-30-90-79

Germany

Microchip Technology GmbH
Gustav-Heinemann Ring 125
D-81739 Munich, 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

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

03/01/02