AN1017

Sinusoidal Control of PMSM Motors with dsPIC30F DSC

Author:	Jorge Zambada
	Microchip Technology Inc.

INTRODUCTION

This application note describes a method of driving a sensored Permanent Magnet Synchronous Motor (PMSM) with sinusoidal currents controlled by a dsPIC30F Digital Signal Controller (DSC). The motor control firmware uses the dsPIC30F peripherals while the mathematical computations are performed by the DSP engine. The firmware is written in 'C' language, with some subroutines in assembly to take advantage of the special DSP operations of the dsPIC30F.

APPLICATION FEATURES

- Sinusoidal current generation for controlling PMSM motor phases using Space Vector Modulation (SVM)
- Synchronization of sinusoidal voltages to PMSM motor position
- Four-quadrant operation allowing forward, reverse and braking operation
- Closed-loop speed regulation using digital Proportional Integral Derivative (PID) control
- Phase advance operation for increased speed range
- Fractional math operations performed by the DSP engine of the dsPIC® DSC

MOTOR CONTROL WITH DIGITAL SIGNAL CONTROLLERS

The dsPIC30F Motor Control family is specifically designed to control the most popular types of motors, including AC Induction Motors (ACIM), Brushed DC Motors (BDC), Brushless DC Motors (BLDC) and Permanent Magnet Synchronous Motors (PMSM), to list a few. Several application notes have been published for ACIM operation (AN984, AN908 and GS004) and Brushless DC Motor Control operation (AN901, AN957 and AN992) based on the dsPIC30F motor control family. These application notes are available on the the Microchip web site (*www.microchip.com*).

This application note demonstrates how the dsPIC30F2010 is used to control a sensored PMSM motor with sinusoidal voltages. The design takes advantage of dsPIC30F peripherals specifically suited for motor control: Motor Control Pulse Width Modulation (MCPWM) and high-speed A/D Converter. The DSP engine of the dsPIC30F2010 supports the necessary fast mathematical operations.

The dsPIC30F2010 family member is a 28-pin 16-bit DSC specifically designed for low-cost/high efficiency motor control applications. The dsPIC30F2010 provides these key features:

- 30 MIPS processing performance
- Six independent or three complementary pairs of dedicated Motor Control PWM outputs
- Six-input, 1 Msps ADC with simultaneous sampling capability from up to four inputs
- Multiple serial communications: UART, I2C™ and SPI
- Small package (6 mm x 6 mm QFN) for embedded control applications
- DSP engine for fast response in control loops

HARDWARE REQUIRED

You will need the following hardware to implement the described motor control application:

- PICDEM™ MCLV Development Board (Figure 1)
- Hurst DMB0224C10002 BLDC Motor
- 24 VDC Power Supply

You can purchase these items from Microchip as a complete kit or as individual components. Check the Development Tools section of the Microchip web site for ordering information.

**FIGURE 1: PICDEM™ MCLV DEVELOPMENT BOARD**
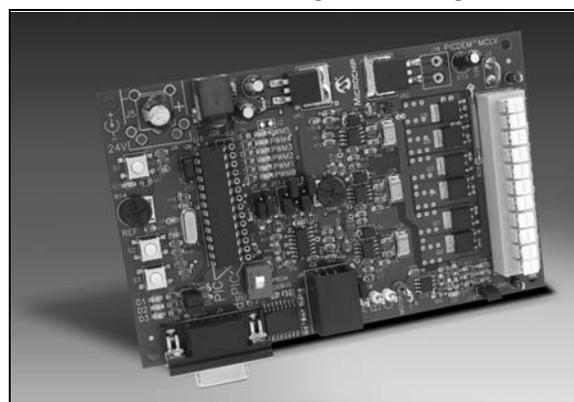
SINUSOIDAL CONTROL OF PMSM MOTORS WITH DSPIC30F DSC

It is strongly recommended that you read the *"PICDEM™ MCLV Development Board User's Guide"* (DS51554) to fully understand the hardware topology being used in this application note. This User's Guide can be downloaded from the Microchip web site.

Figure 2 is a simplified system block diagram for a Sinusoidal PMSM motor control application. This diagram will help you develop your own hardware.

FIGURE 2:   **SYSTEM BLOCK DIAGRAM**
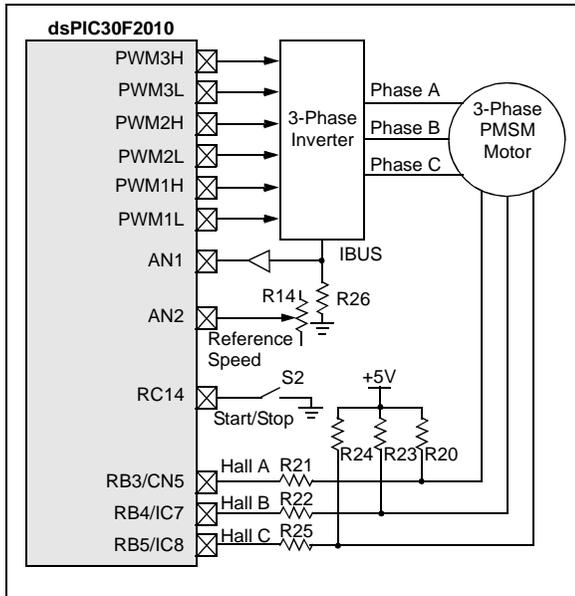
Salient aspects of this topology are:

• Potentiometer R14 selects the desired speed (Reference Speed)
• Rotor position is detected using Hall effect sensors connected to pins RB3, RB4 and RB5
• Current feedback is provided through a simple operational amplifier circuit
• Fault input is received through a comparator circuit connected with the current feedback circuit. The current is sensed using a 0.1 ohm resistor (R26)

You can easily adjust the values of the resistors to accommodate the current capabilities of the motor being used for your application. The motor drive circuit, on the other hand, is designed to drive a 24V PMSM motor. You can change the hardware to meet the drive requirement of a specific motor.

> **Note:** Refer to the "*PICDEM™ MCLV Development Board User's Guide*" (DS51554) for details on how to change the hardware for use with motors greater or less than 24V.

On the low side, the voltage limit is 10V. On the high side, the voltage limit is 48V. It is important to note that the heat sink on the IGBTs have very limited heat dissipation, so high power requirements may not be easily met with the PICDEM™ MCLV development board.

To use the PICDEM™ MCLV development board for this application, use the jumper settings shown in Table 1 and the motor connections shown in Table 2 and Table 3.

TABLE 1:   **PICDEM™ MCLV DEVELOPMENT BOARD JUMPER SETTINGS**

Jumpers	Position for Sinusoidal Control (dsPIC® DSC Sensored)
J7, J8, J11	Open
J12, J13, J14	Open
J15, J16, J17, J10	Open
J19	Short

TABLE 2:   **CONNECTIONS FOR MOTOR WINDINGS***

Connector J9	Position for Sinusoidal Control (dsPIC® DSC Sensored)
M3	Phase A (White)
M2	Phase B (Black)
M1	Phase C (Red)
G	Ground (Green) if available

TABLE 3:   **MOTOR CONNECTIONS FOR HALL SENSORS***

Connector J9	Position for Sinusoidal Control (dsPIC® DSC Sensored)
+5V	Red
GND	Black
HA	White
HB	Brown
HC	Green

*   The colors referenced in Tables 2 and 3 for the motor windings and hall sensors, respectively, pertain to the Hurst 24V motor available from Microchip. The ground wire is sometimes not available on some motors.

After your code is developed and you have downloaded it to the dsPIC30F, you will need to press switch S2 to start and stop the motor. The potentiomer marked REF (R14) sets the required speed and direction of rotation of the motor. The motor does not need to stop to change direction of rotation.

PROGRAMMING THE dsPIC30F2010 WITH THE dsPICDEM™ MCLV DEVELOPMENT BOARD

The dsPICDEM MCLV development board allows you to program the dsPIC30F2010 in-circuit. To program the part, you must set DIP switch S4 to the PRGM position. When programming is complete, you must set the DIP switch to the DEBUG position to execute the code. If the IDC2 is connected to the PICDEM™ MCLV development board as a debugger, the connector at J6 should be attached. If you use MPLAB® ICD 2 as a debugger, the RJ11 cable should be connected to the board (J6). If you use MPLAB ICD 2 as a programmer only, the RJ11 cable should be connected for programming the part and unplugged for normal program execution.

The following configuration allows the application to work on a PICDEM™ MCLV development board:

Oscillator Source:	Primary Oscillator
Primary Oscillator Mode:	XT w/PLL 16x
Comm Channel Select:	EMUC2 and EMUD2

Other settings can be enabled or disabled as needed, or modified in the application.

BACKGROUND

Many consumer and industrial applications use the BLDC motor because of its compact size, controllability and high efficiency. Increasingly, it is used in automotive applications as part of a strategy to eliminate belts and hydraulic systems, to increase functionality and to improve fuel economy. In high-performance applications, such as machine tools and low noise fan applications, the production of smooth torque is crucial.
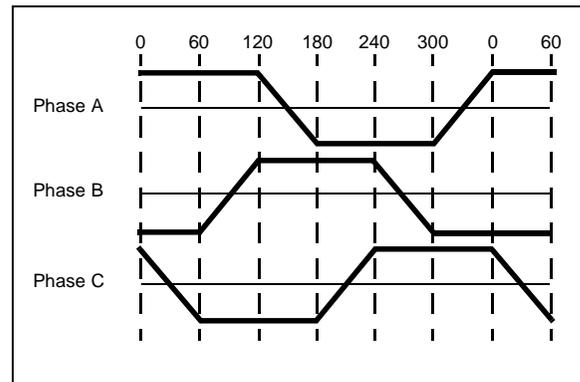
The main disadvantage of BLDC motors, when low torque ripple and quieter operation are required, is the non-sinusoidal distribution of the stator windings. BLDC motors with non-sinusoidal winding distribution generate trapezoidal back-EMF, as shown in Figure 3. Trapezoidal Back-EMF BLDC motors are specifically designed to be driven with square voltages synchronized with the motor's angular position. This control method is commonly called six-step commutation.

It is assumed that you are familiar with the six-step commutation technique, so no further elaboration is offered in this application note. However, for more detailed information on how to operate a BLDC motor with six-step commutation, you can refer to these additional Microchip application notes:

- *AN857 "Brushless DC Motor Control Made Easy"* (DS00857)
- *AN957 "Sensored BLDC Motor Control Using dsPIC30F2010"* (DS00957)

For a good introduction to BLDC motors and their basic operating principles, see also *AN885 "Brushless DC (BLDC) Motor Fundamentals"* (DS00885) .

FIGURE 3:   TRAPESOIDAL BACK-EMF

Trapezoidal distribution of the motor windings of a BLDC motor leads to torque ripple during motor operation since the current generation is also trapezoidal. This torque ripple produces a small speed oscillation, which generates audible noise. On the other hand, sinusoidal Back-EMF BLDC motors, also known as Permanent Magnet Synchronous Motors (PMSM) produce sinusoidal currents, which reduce the torque ripple, thus minimizing the audible noise. Figure 4 shows the sinusoidal back-EMF voltages generated by a motor with sinusoidal winding distribution.

FIGURE 4:   SINUSOIDAL BACK-EMF
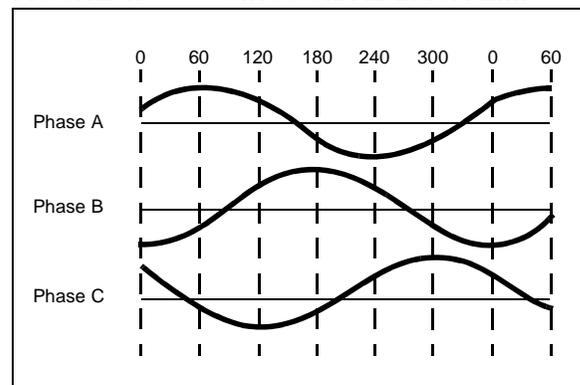
This application note assumes a 3-Phase PMSM motor with sinusoidal back-EMF and three Hall effect sensors.

SENSORED OPERATION OF BLDC MOTORS

To allow correct commutation of the motor, the absolute position within an electrical cycle must be measured. Three Hall effect sensors provide rotor position information. These sensors are distributed along the stator in such a way that they generate six different logic states per electrical cycle. The ratio between the electrical cycles and mechanical revolutions depends on the number of motor pole pairs. For instance, the motor used in this application note has five pole pairs, so every mechanical revolution requires five electrical cycles. For conventional energization (six-step commutation), six equally spaced commutations are required per electrical cycle. This is usually implemented using three Hall effect or optical switches with a suitable disk on the rotor. Continuous position information is not required. Only detection of the required commutation instances is required. Figure 5 shows the three sensor outputs along with the corresponding voltage driving each motor winding.

FIGURE 5: **SIX-STEP COMMUTATION FOR TRAPEZOIDAL BLDC MOTORS**
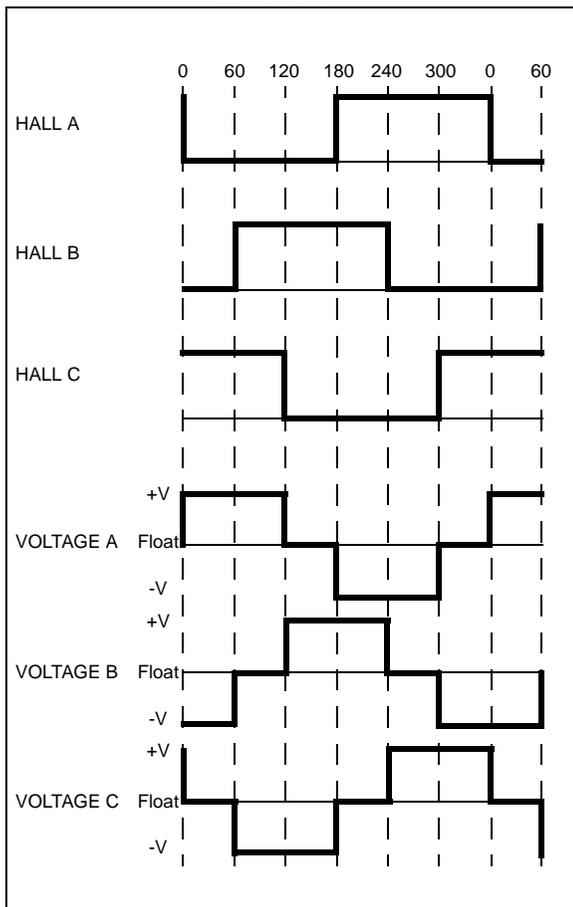
You can see that the voltage does not vary for each particular sector until a new motor position or combination of Hall effect sensor is detected. For the technique described in this application note, the three Hall effect sensors detect the rotor position, as in the six-step technique. However, instead of generating square waves, a continuous changing voltage is generated with a sine-wave shape. Figure 6 shows the resulting sinusoidal voltage generation. The relation is shown between the phase voltages and the three Hall effect sensors. The amplitude of the sinusoidal voltages determines the speed for a specific mechanical load in the motor.

FIGURE 6: **VOLTAGE GENERATION FOR SINUSOIDAL BLDC MOTORS**
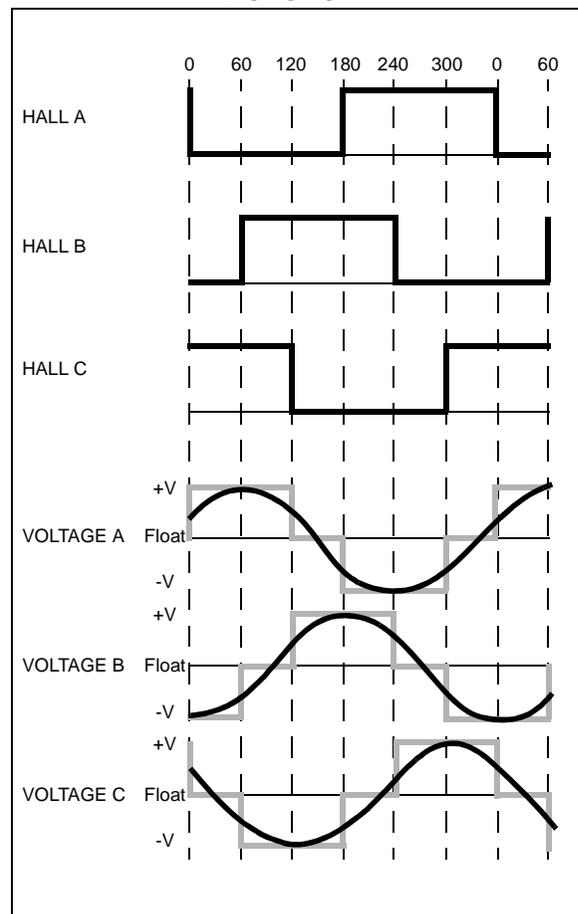
IMPLEMENTATION OF SINUSOIDAL VOLTAGE CONTROL

Figure 7 is a block diagram representation of the application software. When the motor is running, Measured Speed is subtracted from Reference Speed (desired speed) and the resulting error is processed by the PID controller to generate the amplitude of the sine wave. The Reference Speed is set by an external potentiometer, while the measured speed is derived from a Hall effect sensor.

Once Amplitude is known, two additional parameters are needed for sine generation. One parameter is the Period of the sine wave, which is taken from one of the Hall effect sensors. The other parameter is the Phase, which is calculated using Phase Advance, depending on speed range requirements and the rotor position from the Hall effect sensors.

The Amplitude variable sets the amount of motor current and the resulting torque. An increase in torque corresponds to an increase in speed. The speed control loop only controls Amplitude. The value of Phase from the Phase Advance block is derived from Hall effect sensor information to maintain the sinusoidal voltage alignment to the rotor.

Using the software block diagram as a point of reference, the following sections of this application note describe the software functionality in detail. The description starts with the Main State machine, which interacts with all the other software blocks using global variables. Then the description focuses on speed measurements (Reference Speed and Measured Speed), leading to an explanation of the software implementation of the PID controller. This discussion includes some background information on PID control.

Next, calculation of the parameters for generating three-phase sine waves is discussed, starting with the rotor sector and phase advance calculations. And finally, sinusoidal voltage generation using space vector modulation, with Amplitude, Phase and Period as parameters, concludes the discussion.

FIGURE 7:   SOFTWARE BLOCK DIAGRAM
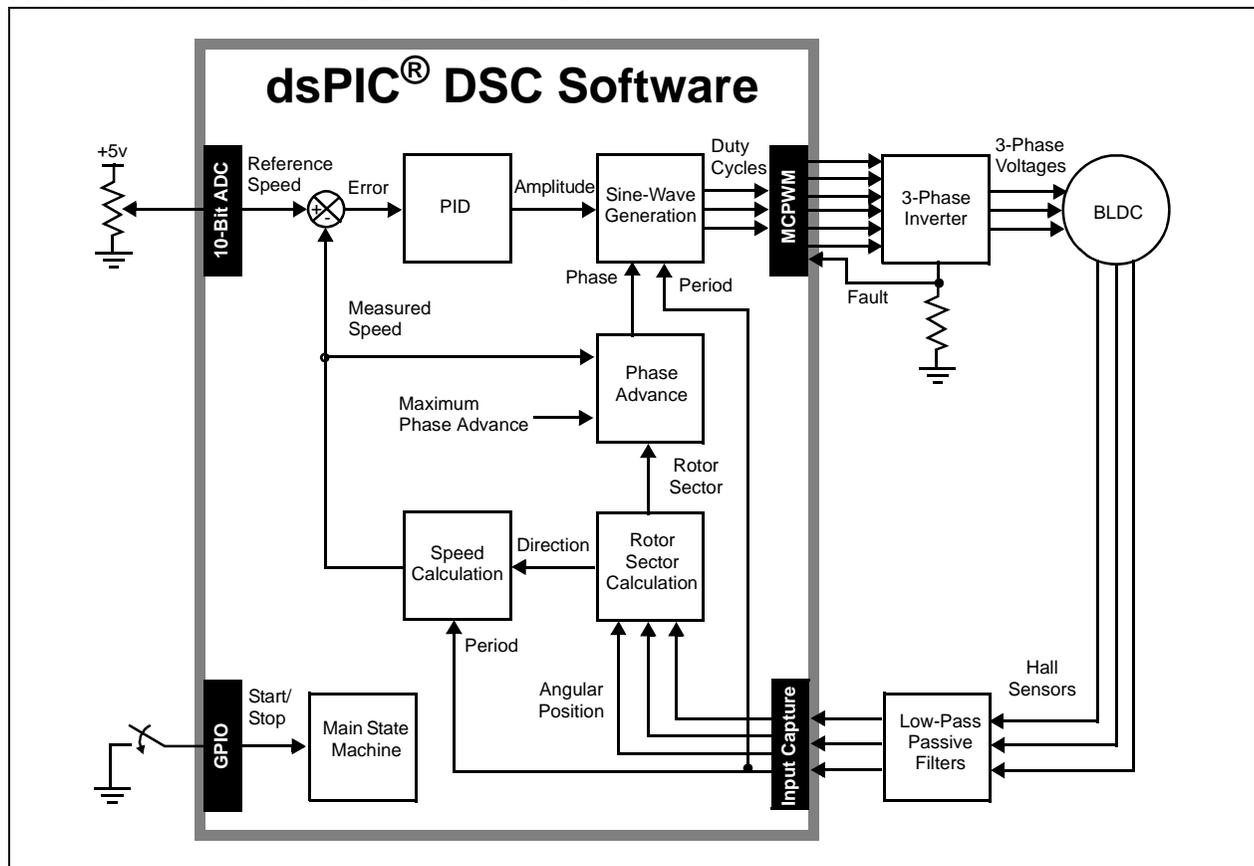
MAIN STATE MACHINE

The state diagram shown in Figure 8 illustrates how each interrupt (shown with heavy, dark lines) interacts with the motor control software. At Power-on Reset, the software initializes all the software variables and enables all the peripherals to be used by the application. After the variables and peripherals are initialized, the software enters the Motor Stopped state and remains there until a Start command is executed from the external push button (S2 pressed).

When S2 is pressed, the `RunMotor()` subroutine is called, and the first process within this subroutine (`ChargeBootstraps()`) is executed. The inverter circuit uses N-channel MOSFETs for the upper and lower devices. The bootstrap circuit generates a floating voltage source for the gate drive of the upper devices. The bootstrap supply voltage can be 15V greater than the inverter voltage rail to ensure that the upper devices turn on. The `ChargeBootstraps()` subroutine is needed to charge the bootstrap capacitors each time the motor is energized for the first time before running the motor. The `ChargeBootstraps()`

subroutine turns on the lower transistors for 10 ms, to ensure voltage on these capacitors, and then transfers control of the outputs to the PWM module.

Next, the variables used for controlling the motor are initialized. Then the PID coefficients, error accumulation and controller output variables used in the PID Speed Controller are initialized. In the `RunMotor()` subroutine, the timer counters are also initialized to zero, and the variables to capture the Hall effect sensor period are also initialized. At the end of the subroutine, the interrupt flags are cleared and the interrupts are enabled.

Once the variables have been initialized in the `RunMotor()` subroutine, all other activities within the state machine are performed by Interrupt Service Routines (ISRs). Table 4 summarizes the Interrupt Service Routines, indicates when they are called and provides a brief description of the operations executed in each particular ISR.

If S2 is pressed while the motor is running, the `StopMotor()` subroutine disables all the interrupts and stops the motor.

TABLE 4: SUMMARY OF INTERRUPT SERVICE ROUTINES

Interrupt	When Called	Operations Performed
A/D	20 kHz	• Reads new Reference Speed value from AN2
PWM	20 kHz	• Generates sine wave using SVM
T1	1 kHz	• Measures Speed Calculation using IC7 data • Executes PID control • Calculates Phase Advance • Provides Motor Stalled protection
CN5	Every Hall A transition	• Calculates rotor position • Determines rotor mechanical direction • Synchronizes sine-wave pointer to rotor position
IC7	Every Hall B transition	• Calculates rotor position • Determines rotor mechanical direction • Captures Hall transition timing using IC • Synchronizes sine-wave pointer to rotor position
IC8	Every Hall C transition	• Calculates rotor position • Determines rotor mechanical direction • Synchronizes sine-wave pointer to rotor position

FIGURE 8: MAIN STATE MACHINE DIAGRAM

SINUSOIDAL CONTROL OF PMSM MOTORS WITH DSPIC30F DSC

SPEED CALCULATION

Reference Speed Calculation

The Reference Speed variable (see Figure 7) is read by the A/D interrupt using the A/D converter (AN2 input pin). The A/D converter is configured to generate interrupts at the PWM rate (20 kHz in this application). Within the A/D ISR, the A/D conversion buffer is copied into a variable (RefSpeed) in signed fractional format. This variable represents the desired speed from -1.0 to 0.99997. Table 5 summarizes the minimum and maximum value of this variable and their interpretation for the motor used in this application note.

Measured Speed Calculation

The actual speed of the motor is calculated from one Hall effect sensor (Hall B). The calculation uses an Input Capture channel (IC7 input pin) to measure the time between two consecutive transitions in Hall effect sensor B. The timing diagram in Figure 9 shows the capture events and the variables used to store two consecutive captures to measure the time between Hall effect sensor B transitions. In the same figure, the mechanical rotation angle is plotted for the five pole-pair motor used to see the relationship between Hall effect sensor B signal and the actual mechanical movement.

TABLE 5: EXAMPLES OF REFERENCE SPEED CALCULATIONS

	RefSpeed Variable		
	Hex Value	Fractional Value	Desired Speed in RPM
Min Value	0x8000	-1.0	6000 RPM Reverse
Max Value	0x7FFF	0.99997	6000 RPM Forward

FIGURE 9: TIME RELATIONSHIP BETWEEN HALL B AND MOTOR MECHANICAL POSITION
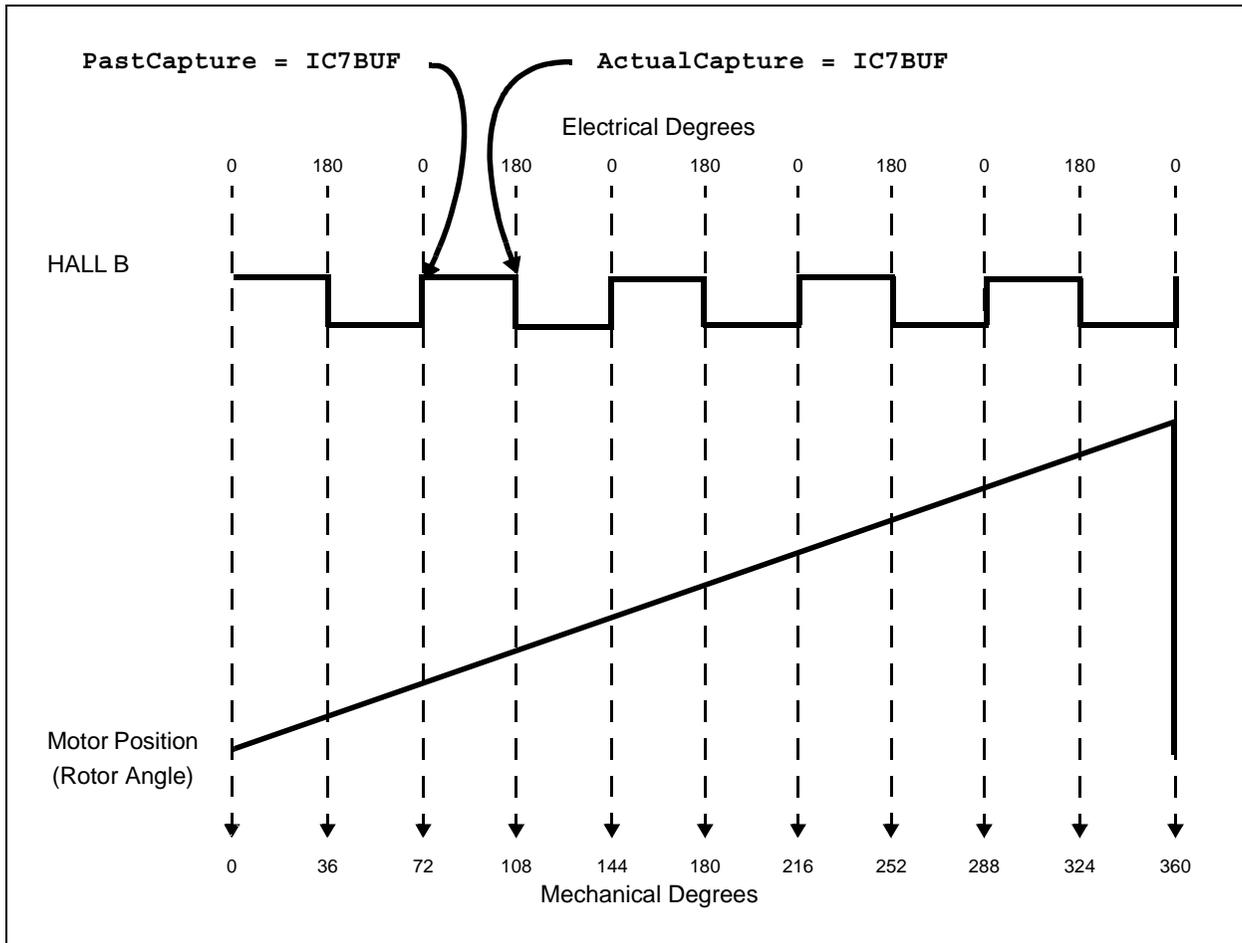
The variable used to store the time between transitions is called `Period` and is calculated in the Timer 1 ISR. This ISR is generated every 1 millisecond and executes the speed controller. The formula used to calculate the actual speed of the motor is:

$$Period = ActualCapture - PastCapture$$

where `ActualCapture` and `PastCapture` contain timing information of two consecutive Hall effect sensor B transitions as shown in Figure 9. `Period` is used for calculating the step size of the pointer used to generate the sine wave, since the generated sinusoidal voltages need to be of the same frequency compared to the Hall effect sensor information.

For Speed calculation, the following formula is used to convert Hall effect sensor time between transitions (`Period`) to the measured speed (`MeasuredSpeed`):

$$MeasuredSpeed = FractionalDivide\left(\frac{MINPERIOD}{Period}\right)$$

MINPERIOD is a constant value calculated to give a fractional value of 0.99997, if a maximum speed of 6000 RPM is detected from the Hall effect sensor. The MINPERIOD formula is:

$$MINPERIOD = \frac{(Timer\ 1\ Input\ Clock)\ x\ 60}{(Max\ Speed\ in\ RPM)\ x\ (Motor\ Poles)}$$

$$= \frac{(20\ MHz/64)\ x\ 60}{6000\ RPM\ x\ 10}$$

$$= 312$$

Table 6 provides examples of speed calculation (`MeasuredSpeed`) from the input capture values stored in `PastCapture` and `ActualCapture`.

The sign is added to the `MeasuredSpeed` depending on variable `CurrentDirection`, which is calculated in the Hall effect sensor interrupts. `CurrentDirection` is calculated based on two consecutive combinations from the Hall effect sensors (or two consecutive sectors). If the motor is moving forwards (CW), the `MeasuredSpeed` is kept positive. If the motor is moving backward (CCW), `MeasuredSpeed` is converted to a negative value.

TABLE 6:   EXAMPLES OF MEASURED SPEED CALCULATIONS

ActualCapture	PastCapture	Period	MeasuredSpeed		
			Hexadecimal	**Fractional**	**RPM**
0x0000	0xFEC7	0x0139	0x7F97	0.99679	5990.4
0x2000	0x1D8E	0x0272	0xFCB	0.49838	2995.2
0x4000	0xC5EE	0x7A12	0x0147	0.00998	60

PID SPEED CONTROLLER

The PID speed controller is executed in the T1 interrupt (Timer 1 ISR) every 1 millisecond. This subroutine is called `SpeedControl()` and uses both Reference Speed (`RefSpeed`) and the actual speed of the motor (`MeasuredSpeed`) in signed fractional format. `MeasuredSpeed` is subtracted from `RefSpeed` to determine the speed error (`Error`), which determines if the motor must speed up or slow down. To ensure smooth operation of the motor, the error value is parsed into proportional, integral and derivative components to produce a composite output that is used to compensate for the speed error.

The PID controller implementation takes advantage of the MAC instruction of the dsPIC DSC for fast execution. The formula used to generate the controller output (`ControlOutput`) depends on three error values saved in the last three T1 interrupts (`ControlDifference[0]`, `ControlDifference[1]` and `ControlDifference[2]`) and the PID coefficients (`PIDCoefficients[0]`, `PIDCoefficients[1]` and `PIDCoefficients[2]`). The formula is:

$$
\begin{aligned}
ControlOutput = ControlOutput \\
+ ControlDifference[0] \times PIDCoefficients[0] \\
+ ControlDifference[1] \times PIDCoefficients[1] \\
+ ControlDifference[2] \times PIDCoefficients[2]
\end{aligned}
$$

Table 7 provides a brief description of the PID control variables

This implementation of the PID uses the MAC instruction along with the automatic saturation feature of the dsPIC DSC when performing MAC operation. When adjusting PID gains, the user is responsible for avoiding maximum values of the PID coefficients. These values represent fractional values and have to be within the following values:

$Kp + Ki + Kd$ $<$ $0.9999\ (0x7FFF)$

$-Kp - 2*Kd$ $>$ $-1.0(0x8000)$

Kd $<$ $0.9999(0x7FFF)$

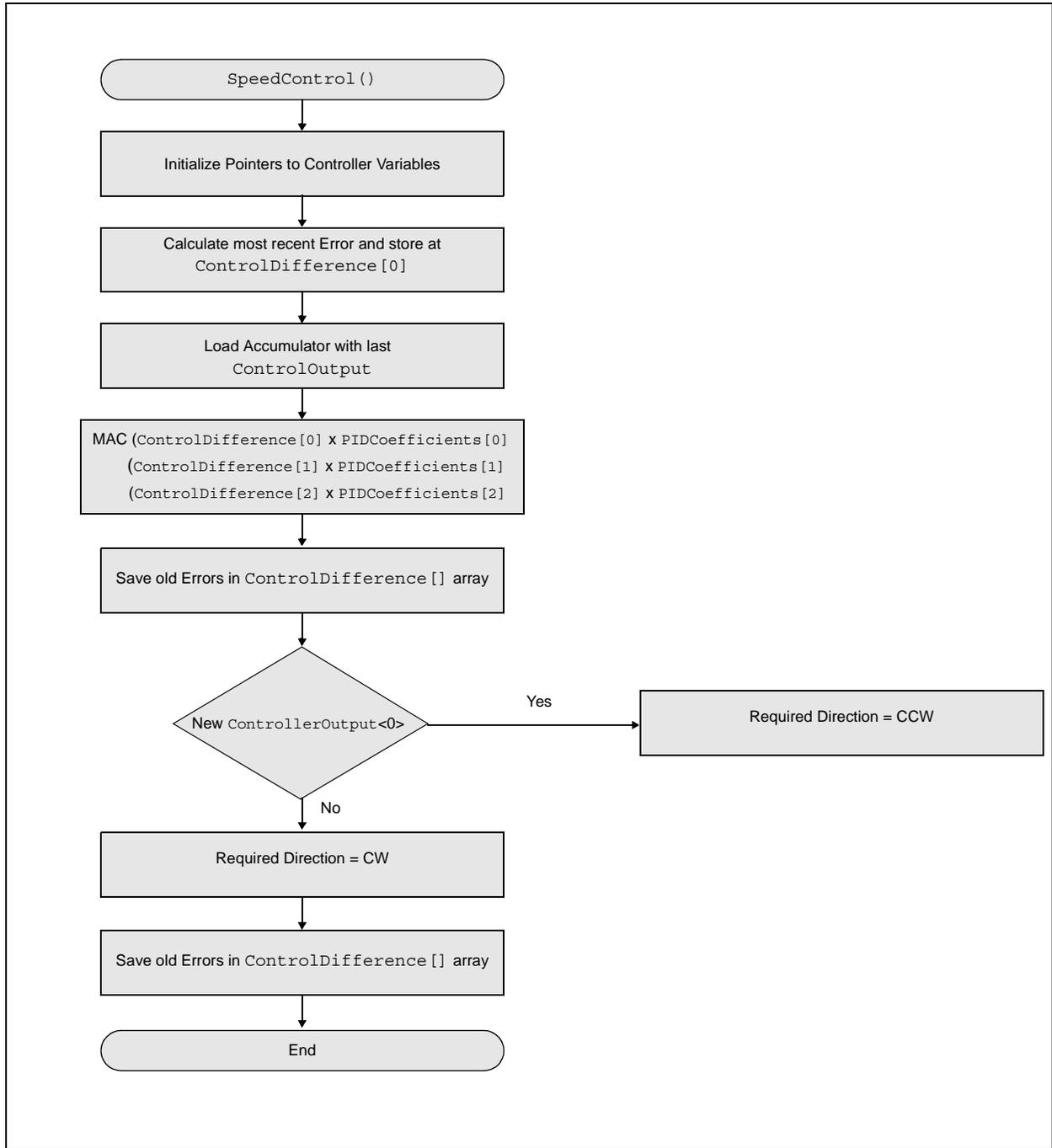Figure 10 is a flow chart that shows how the `SpeedControl()` function is implemented.

`Required_Direction` is taken from the sign of the PID controller output (`ControlOutput`) and tells the software to run the motor CW or CCW. The following code sample shows how the required direction is calculated in the software:

```
// ControlOutput determines the motor
// required direction
    if (ControlOutput < 0)
        Required_Direction = CCW;
    else
        Required_Direction = CW;
```

TABLE 7: PID CONTROL VARIABLES

Variable	Description
`ControlOutput`	The output of the controller in 16-bit signed fractional format
`ControlDifference[0]`	Most recent calculated speed error (`RefSpeed - MeasuredSpeed`)
`ControlDifference[1]`	Speed error in the previous T1 ISR (1 ms old)
`ControlDifference[2]`	Speed error before `ControlDifference[1]` (2 ms old)
`PIDCoefficients[0]`, `PIDCoefficients[1]` and `PIDCoeffients[2}`	Modified PID coefficients from regular PID form to filter-like PID implementation: $PIDCoefficients[0] = Kp + Ki = Kd$ $PIDCoefficients[1] = -Kp - 2Kd$ $PIDCoefficients[2] = Kd$

FIGURE 10:   SPEED CONTROL FLOW CHART

```
                    ┌─────────────────────────┐
                    │      SpeedControl()      │
                    └─────────────────────────┘
                                 │
                                 ▼
                    ┌───────────────────────────────────┐
                    │ Initialize Pointers to Controller  │
                    │ Variables                          │
                    └───────────────────────────────────┘
                                 │
                                 ▼
                    ┌───────────────────────────────────┐
                    │ Calculate most recent Error and    │
                    │ store at ControlDifference[0]      │
                    └───────────────────────────────────┘
                                 │
                                 ▼
                    ┌───────────────────────────────────┐
                    │ Load Accumulator with last         │
                    │ ControlOutput                      │
                    └───────────────────────────────────┘
                                 │
                                 ▼
                    ┌────────────────────────────────────────────┐
                    │ MAC (ControlDifference[0] x PIDCoefficients[0] │
                    │    (ControlDifference[1] x PIDCoefficients[1]  │
                    │    (ControlDifference[2] x PIDCoefficients[2]  │
                    └────────────────────────────────────────────┘
                                 │
                                 ▼
                    ┌───────────────────────────────────┐
                    │ Save old Errors in                 │
                    │ ControlDifference[] array          │
                    └───────────────────────────────────┘
                                 │
                                 ▼
                    ◇─────────────────────◇                    ┌──────────────────────────┐
                    │ New ControllerOutput<0> │──── Yes ───────▶│ Required Direction = CCW  │
                    ◇─────────────────────◇                    └──────────────────────────┘
                                 │
                                No
                                 ▼
                    ┌───────────────────────────────────┐
                    │ Required Direction = CW            │
                    └───────────────────────────────────┘
                                 │
                                 ▼
                    ┌───────────────────────────────────┐
                    │ Save old Errors in                 │
                    │ ControlDifference[] array          │
                    └───────────────────────────────────┘
                                 │
                                 ▼
                    ┌─────────────────────────┐
                    │          End            │
                    └─────────────────────────┘
```

PID BACKGROUND

A complete discussion of Proportional Integral Derivative (PID) controllers is beyond the scope of this application note. However, some PID operation basics are relevant.

A PID controller responds to an error signal in a closed control loop and attempts to adjust the controlled quantity to achieve the desired system response. The controlled parameter can be any measurable system quantity such as speed, torque or flux. The benefit of the PID controller is that it can be adjusted empirically by adjusting one or more gain values and observing the change in system response.

A digital PID controller is executed at a periodic sampling interval. It is assumed that the controller is executed frequently enough so that the system can be properly controlled. The error signal is formed by subtracting the desired setting of the parameter to be controlled from the actual measured value of that parameter. The sign of the error indicates the direction of change required by the control input.

The Proportional (P) term of the controller is formed by multiplying the error signal by a P gain, causing the PID controller to produce a control response that is a function of (i.e., proportional to) the error magnitude. As the error signal becomes larger, the P term of the controller becomes larger to provide more correction. The effect of the P term tends to reduce the overall error as time elapses. However, the P term has less effect as the error approaches zero. In most systems, the error of the controlled parameter gets very close to zero but does not converge. The result is a small remaining steady state error.

The Integral (I) term of the controller is used to eliminate small steady state errors. The I term calculates a continuous running total of the error signal. Therefore, a small steady state error accumulates into a large error value over time. This accumulated error signal is multiplied by an I gain factor and becomes the I output term of the PID controller.

The Differential (D) term of the PID controller is used to enhance the speed of the controller and responds to the rate of change of the error signal. The D term input is calculated by subtracting the present error value from a prior value. This delta error value is multiplied by a D gain factor that becomes the D output term of the PID controller. The D term of the controller produces more control output the faster the system error is changing.

ADJUSTING PID GAINS

The P gain of a PID controller sets the overall system response. When first tuning a controller, the I and D gains should be set to zero. The P gain can then be increased until the system responds well to set-point changes without excessive overshoot or oscillations. Using lower values of P gain will 'loosely' control the system, while higher values will give 'tighter' control. At this point, the system will probably not converge to the set-point.

After a reasonable P gain is selected, the I gain can be slowly increased to force the system error to zero. Only a small amount of I gain is required in most systems. Note that the effect of the I gain, if large enough, can overcome the action of the P term, slow the overall control response and cause the system to oscillate around the set-point. If oscillation occurs, reducing the I gain and increasing the P gain usually solves the problem.

The D gain can speed up system response. However, it must be used carefully because it can produce large output swings, which potentially cause mechanical damage on the motor.

ROTOR SECTOR CALCULATION

Again referring to the software block diagram in Figure 2, the rotor sector value is needed for sinusoidal voltage generation. Rotor sector is the absolute position of the rotor, in electrical degrees, in 60-degree steps. That information is obtained by reading the digital value from the three Hall effect sensors. The Rotor Sector Calculation software block uses the three Hall effect sensor interrupts (CN5, IC8 and IC7 interrupts, as shown in Figure 8). The sector is calculated using the relationships shown in Table 8:

TABLE 8: RELATIONSHIP OF SECTOR TO ANGULAR POSITION

Hall C	Hall B	Hall A	Sector
0	0	0	Invalid
0	0	1	4
0	1	0	2
0	1	1	3
1	0	0	0
1	0	1	5
1	1	0	1
1	1	1	Invalid

For calculating the actual motor direction of rotation (`Current_Direction`) the last two consecutive sectors in the Hall effect sensor interrupts are compared. Table 9 summarizes the direction calculation based on the last two sectors.

TABLE 9: CURRENT DIRECTION CALCULATION

LastSector	Sector	Current_Direction
0	1	CCW
1	2	CCW
2	3	CCW
3	4	CCW
4	5	CCW
5	0	CCW
0	5	CW
1	0	CW
2	1	CW
3	2	CW
4	3	CW
5	4	CW

The actual direction of motor rotation is used in the Speed Calculation block. If the actual direction is CCW, the speed is negative.

The following code illustrates this process:

```
// MeasuredSpeed sign adjustment based on
// current motor direction of rotation
    if (Current_Direction == CCW)
        MeasuredSpeed = -MeasuredSpeed;
```

The calculated Rotor Sector value is modified by the measured speed and maximum phase advance to set the phase angle of the sinusoidal voltage.

PHASE ADVANCE

Under normal conditions, motor speed is limited by the voltage supplied to the motor windings by the inverter. The voltage on the motor windings is usually synchronized with the rotor position, as shown by the relationship between Voltage A and Hall A in Figure 11. This condition creates a rotating electrical field in the stator that is 90 degrees ahead of the rotor magnetic field, which produces the maximum torque per amp from the motor.

FIGURE 11: EFFECT OF PHASE ADVANCE ON DRIVE VOLTAGE
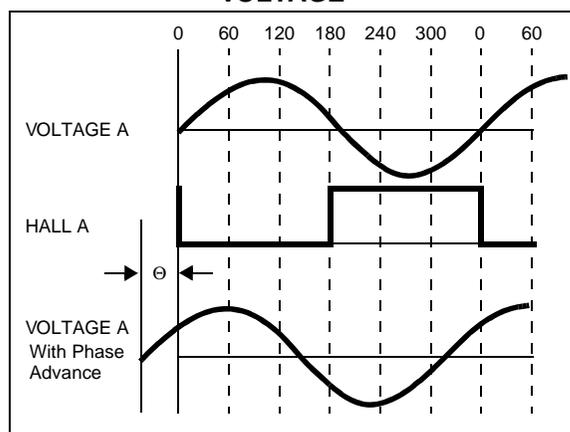
If the field generated in the stator is shifted by an angle theta, the resulting magnetic field between the stator and the rotor is increased. The addition of this phase advance creates a negative field in the rotor, which reduces the field produced by rotor permanent magnets. Some important effects of this phase shift include:

- Phase advance increases the speed range over the nominal speed limit, since the resulting field vector is increased.
- Phase advance weakens the rotor field since it produces a negative field in the rotor. Field weakening reduces the overall torque output available from the motor.
- Motor current consumption, motor operating temperature and audible noise increase when phase advance is used because the stator field is working against the rotor field.

For applications where an increase in the speed range of the motor is required, phase advance is a recommended algorithm. This algorithm can be enabled by defining PHASE_ADVANCE in the SinusoidalBLDC v1.1.c file. This feature is disabled in the context of this application note.

```
#define PHASE_ADVANCE    // for extended
                         // speed ranges this
                         // should be defined
```

SINE-WAVE GENERATION USING SPACE VECTOR MODULATION

The final step in the sinusoidal control of a PMSM motor is to generate pulse-width modulation signals for the three-phase motor voltage signals. By using Space Vector Modulation (SVM), the process of generating the pulse width for each of the three phases reduces to a few simple equations. Each of the three inverter outputs can be in one of two states: either connected to the + bus rail or the - bus rail. Thus the output can be in eight possible states ($2^3 = 8$), as shown in Table 10.

TABLE 10:   SPACE VECTOR MODULATION INVERTER STATES

C	B	A	V_{ab}	V_{bc}	V_{ca}	V_{ds}	V_{qs}	Vector
0	0	0	0	0	0	0	0	U(000)
0	0	1	V_{DC}	0	$-V_{DC}$	$2/3V_{dc}$	0	U_0
0	1	1	0	V_{DC}	$-V_{DC}$	$V_{DC}/3$	$V_{DC}/3$	U_{60}
0	1	0	$-V_{DC}$	V_{DC}	0	$-V_{DC}/3$	$V_{DC}/3$	U_{120}
1	1	0	$-V_{DC}$	0	V_{DC}	$-2V_{DC}/3$	0	U_{180}
1	0	0	0	$-V_{DC}$	V_{DC}	$-V_{DC}/3$	$-V_{DC}/3$	U_{240}
1	0	1	V_{DC}	$-V_{DC}$	0	$V_{DC}/3$	$-V_{DC}/3$	U_{300}
1	1	1	0	0	0	0	0	$U_{(111)}$

The two states where all three outputs are connected to either the + bus or the - bus are considered null states because there is no line-to-line voltage across any of the phases. These are plotted at the origin of the SVM Star. The remaining six states are represented as vectors with 60 degree rotation between each state, as shown in Figure 12.

FIGURE 12: **SPACE VECTOR MODULATION**
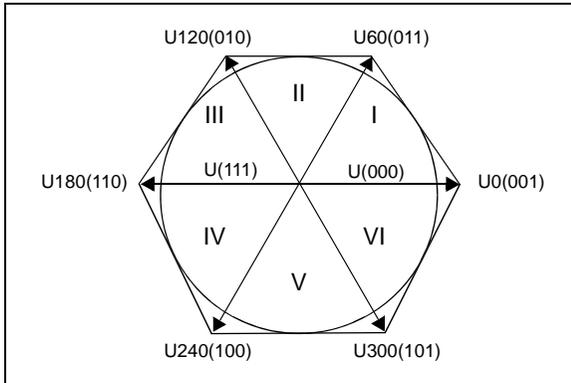
Space Vector Modulation allows any resultant vector be represented by the sum of the components of the two adjacent vectors. In Figure 13, UOUT is the desired resultant. It lies in the sector between U60 and U0. If during a given PWM period T, U0 is output for T1/T and U60 is output for T2/T, the average for the period will be UOUT.

FIGURE 13: **AVERAGE SPACE VECTOR MODULATION**

$$T = T1 + T2 + T0 = \text{PWM Period}$$

$$U_{OUT} = \frac{T1 * U0 + T2 * U60}{T}$$
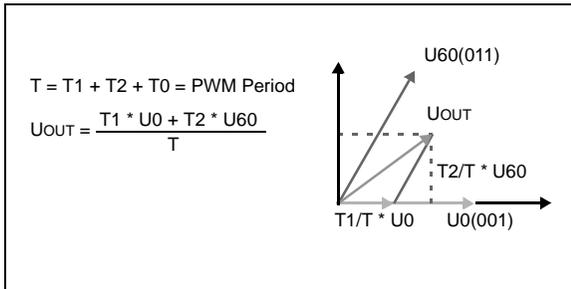
The values for T1 and T2 are taken from a look up table containing 172 fractional sinusoidal values for 0 to 60 electrical degrees. As a result, for each of the six segments one axis is exactly opposite that segment, and the other two axes symmetrically bound the segment. The values of the vector components along those two bounding axis are equal to T1 and T2. For details of the calculations, refer to the SVM.c file that accompanies this application note.

As shown in Figure 14, the PWM period T, the vector T1 is output for T1/T and the vector T2 is output for T2/T. During the remaining time the null vectors are output. The dsPIC30F device is configured for center aligned PWM, which forces symmetry about the center of the

period. This configuration produces two pulses line-to-line during each period. The effective switching frequency is doubled, reducing the ripple current while not increasing the switching losses in the power devices.

FIGURE 14: **PWM FOR PERIOD T**
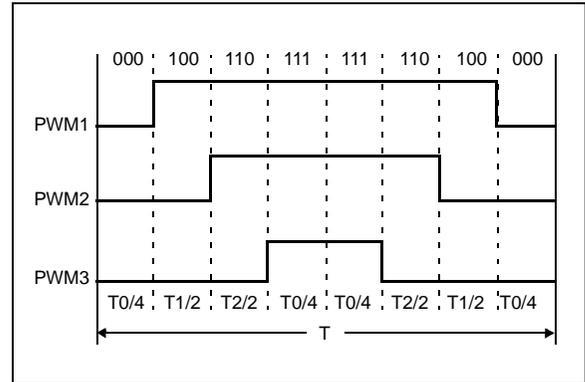
Figure 15 shows a complete electrical revolution with duty cycles using SVM. the values represent duty cycles between 0% and 100%.

FIGURE 15: **LINE-TO-GROUND VOLTAGES USING SVM**
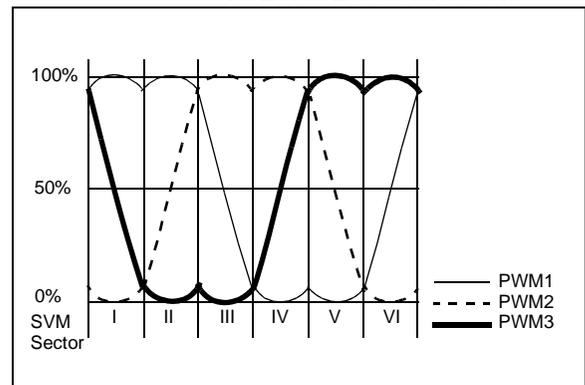
The resulting line-to-line voltages are shown in Figure 16. Sinusoidal voltages with full amplitude are fed into the motor windings.

FIGURE 16: **LINE-TO-LINE VOLTAGES USING SVM**
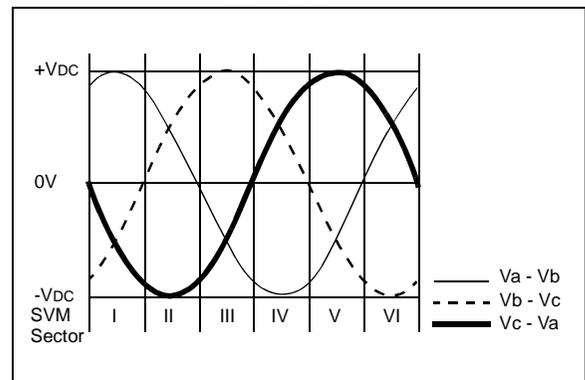
CONCLUSION

The dsPIC30F2010 provides an ideal low-cost solution for controlling a PMSM motor with sinusoidal voltages. Motor control peripherals, such as the MCPWM, in combination with processing power, makes possible several algorithms such as Space Vector Modulation for sine-wave generation, and PID loops for controlling motor speed.

The dsPICDEM™ MCLV development board has the necessary hardware to run this application while maintaining low cost to the total system.

Software files are contained in the AN1017.zip file and can be downloaded from the Microchip web site www.microchip.com

REVISION HISTORY

Revision A (December 2005)

• Initial release of this document.

Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specification contained in their particular Microchip Data Sheet.

- Microchip believes that its family of products is one of the most secure families 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 Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Most likely, the person doing so is 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 products. Attempts to break Microchip's code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

QUALITY MANAGEMENT SYSTEM
CERTIFIED BY DNV
══ ISO/TS 16949:2002 ══

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:
http://support.microchip.com
Web Address:
www.microchip.com

Atlanta
Alpharetta, GA
Tel: 770-640-0034
Fax: 770-640-0307

Boston
Westborough, MA
Tel: 774-760-0087
Fax: 774-760-0088

Chicago
Itasca, IL
Tel: 630-285-0071
Fax: 630-285-0075

Dallas
Addison, TX
Tel: 972-818-7423
Fax: 972-818-2924

Detroit
Farmington Hills, MI
Tel: 248-538-2250
Fax: 248-538-2260

Kokomo
Kokomo, IN
Tel: 765-864-8360
Fax: 765-864-8387

Los Angeles
Mission Viejo, CA
Tel: 949-462-9523
Fax: 949-462-9608

San Jose
Mountain View, CA
Tel: 650-215-1444
Fax: 650-961-0286

Toronto
Mississauga, Ontario,
Canada
Tel: 905-673-0699
Fax: 905-673-6509

ASIA/PACIFIC

Australia - Sydney
Tel: 61-2-9868-6733
Fax: 61-2-9868-6755

China - Beijing
Tel: 86-10-8528-2100
Fax: 86-10-8528-2104

China - Chengdu
Tel: 86-28-8676-6200
Fax: 86-28-8676-6599

China - Fuzhou
Tel: 86-591-8750-3506
Fax: 86-591-8750-3521

China - Hong Kong SAR
Tel: 852-2401-1200
Fax: 852-2401-3431

China - Qingdao
Tel: 86-532-8502-7355
Fax: 86-532-8502-7205

China - Shanghai
Tel: 86-21-5407-5533
Fax: 86-21-5407-5066

China - Shenyang
Tel: 86-24-2334-2829
Fax: 86-24-2334-2393

China - Shenzhen
Tel: 86-755-8203-2660
Fax: 86-755-8203-1760

China - Shunde
Tel: 86-757-2839-5507
Fax: 86-757-2839-5571

China - Wuhan
Tel: 86-27-5980-5300
Fax: 86-27-5980-5118

China - Xian
Tel: 86-29-8833-7250
Fax: 86-29-8833-7256

ASIA/PACIFIC

India - Bangalore
Tel: 91-80-2229-0061
Fax: 91-80-2229-0062

India - New Delhi
Tel: 91-11-5160-8631
Fax: 91-11-5160-8632

India - Pune
Tel: 91-20-2566-1512
Fax: 91-20-2566-1513

Japan - Yokohama
Tel: 81-45-471- 6166
Fax: 81-45-471-6122

Korea - Gumi
Tel: 82-54-473-4301
Fax: 82-54-473-4302

Korea - Seoul
Tel: 82-2-554-7200
Fax: 82-2-558-5932 or
82-2-558-5934

Malaysia - Penang
Tel: 60-4-646-8870
Fax: 60-4-646-5086

Philippines - Manila
Tel: 63-2-634-9065
Fax: 63-2-634-9069

Singapore
Tel: 65-6334-8870
Fax: 65-6334-8850

Taiwan - Hsin Chu
Tel: 886-3-572-9526
Fax: 886-3-572-6459

Taiwan - Kaohsiung
Tel: 886-7-536-4818
Fax: 886-7-536-4803

Taiwan - Taipei
Tel: 886-2-2500-6610
Fax: 886-2-2508-0102

Thailand - Bangkok
Tel: 66-2-694-1351
Fax: 66-2-694-1350

EUROPE

Austria - Wels
Tel: 43-7242-2244-399
Fax: 43-7242-2244-393

Denmark - Copenhagen
Tel: 45-4450-2828
Fax: 45-4485-2829

France - Paris
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

Germany - Munich
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

Italy - Milan
Tel: 39-0331-742611
Fax: 39-0331-466781

Netherlands - Drunen
Tel: 31-416-690399
Fax: 31-416-690340

Spain - Madrid
Tel: 34-91-708-08-90
Fax: 34-91-708-08-91

UK - Wokingham
Tel: 44-118-921-5869
Fax: 44-118-921-5820

10/31/05