

## Sensored BLDC Motor Control Using dsPIC30F2010

*Author: Stan D'Souza  
Microchip Technology*

### INTRODUCTION

The dsPIC30F2010 is a 28-pin 16-bit MCU specifically designed for embedded motor control applications. AC Induction Motors (ACIM), Brushless DC (BLDC) and DC are some typical motor types for which the dsPIC30F2010 has been specifically designed. Some of the key features on the dsPIC30F2010 are:

- 6 independent or 3 complementary pairs of dedicated Motor Control PWM outputs.
- 6 input, 500Ksps ADC with up to 4 simultaneous sampling capability.
- Multiple serial communications: UART, I<sup>2</sup>C™ and SPI
- Small package: 6 x 6 mm QFN for embedded control applications
- DSP engine for fast response in control loops.

In this application note we discuss how the dsPIC30F2010 is used to control a sensored BLDC motor. Please refer to AN901 for details on how BLDC motors operate and general information on what needs to be done to run and control BLDC motors. This application note discusses the specific implementation using the dsPIC30F2010. It touches only briefly on BLDC motor details

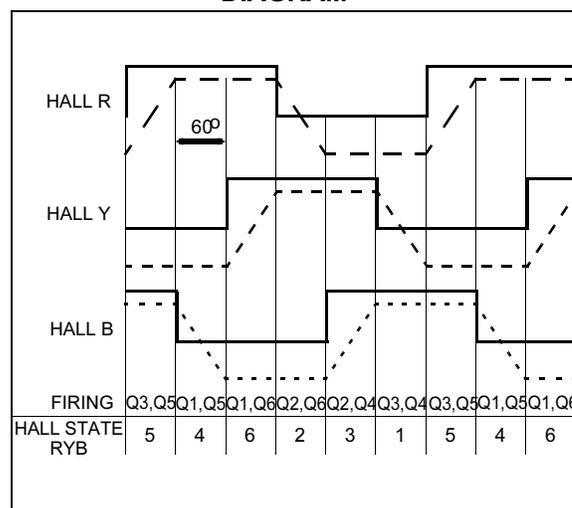
### BLDC MOTORS

BLDC motors are basically inside-out DC motors. In a DC motor the stator is a permanent magnet. The rotor has the windings, which are excited with a current. The current in the rotor is reversed to create a rotating or moving electric field by means of a split commutator and brushes. On the other hand, in a BLDC motor the windings are on the stator and the rotor is a permanent magnet. Hence the term inside-out DC motor.

To make the rotor turn, there must be a rotating electric field. Typically a three-phase BLDC motor has three stator phases that are excited two at a time to create a rotating electric field. This method is fairly easy to implement, but to prevent the permanent magnet rotor from getting locked with the stator, the excitation on the stator must be sequenced in a specific manner while knowing the exact position of the rotor magnets. Position information can be gotten by either a shaft

encoder or, more often, by Hall effect sensors that detect the rotor magnet position. For a typical three-phase, sensored BLDC motor there are six distinct regions or sectors in which two specific windings are excited. These are as shown in Figure 1.

**FIGURE 1: BLDC COMMUTATION DIAGRAM**



By reading the Hall effect sensors, a 3-bit code can be obtained with values ranging from 1 to 6. Each code value represents a sector on which the rotor is presently located. Each code value, therefore, gives us information on which windings need to be excited. Thus a simple lookup table can be used by the program to determine which two specific windings to excite and, thus, turn the rotor.

Note that state '0' and '7' are invalid states for Hall effect sensors. Software should check for these values and appropriately disable the PWM.

### Change Notification Inputs

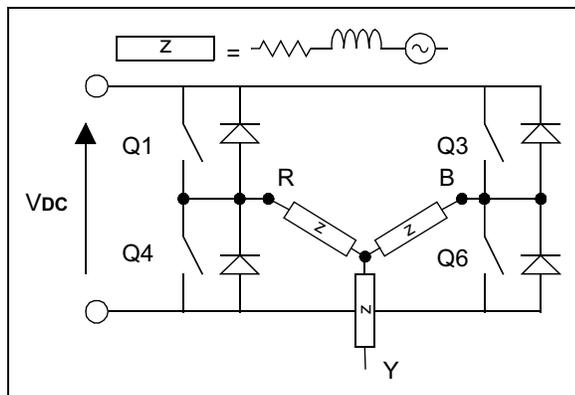
Taking this technique a step further, the Hall effect sensors can be connected to dsPIC30F2010 inputs that detect a change (Change Notification (CN) inputs). An input change on these pins generates an interrupt. In the CN Interrupt Service Routine (ISR) the user application program reads the Hall effect sensor value and uses it to generate an offset in the lookup table for driving the windings of the BLDC motor.

## MOTOR CONTROL PULSE WIDTH MODULATION (MCPWM)

Using the above method, you can get full speed rotation of the BLDC motor. However, to get variable speed of the BLDC motor, you must apply a variable voltage to the terminals of the windings. Putting this in digital terms, the variable voltage can be obtained by different duty cycles of a PWM signal going to the windings of the BLDC motor.

The dsPIC30F2010 has six PWM outputs that can be driven with the PWM signal. As shown in Figure 2, the three windings can be driven ON High, driven ON Low or not driven at all by using six switches, IGBTs or MOSFETs. When one leg of the winding is connected for example, to the high side, the variable duty cycle signal PWM can be injected on the low side driver. This has the same effect as having a PWM signal on the high side and connecting the low side to Vss or GND. When driving the PWM signal, low side drivers are preferred over high side drivers.

**FIGURE 2: BEMF SENSING HARDWARE EXAMPLE**

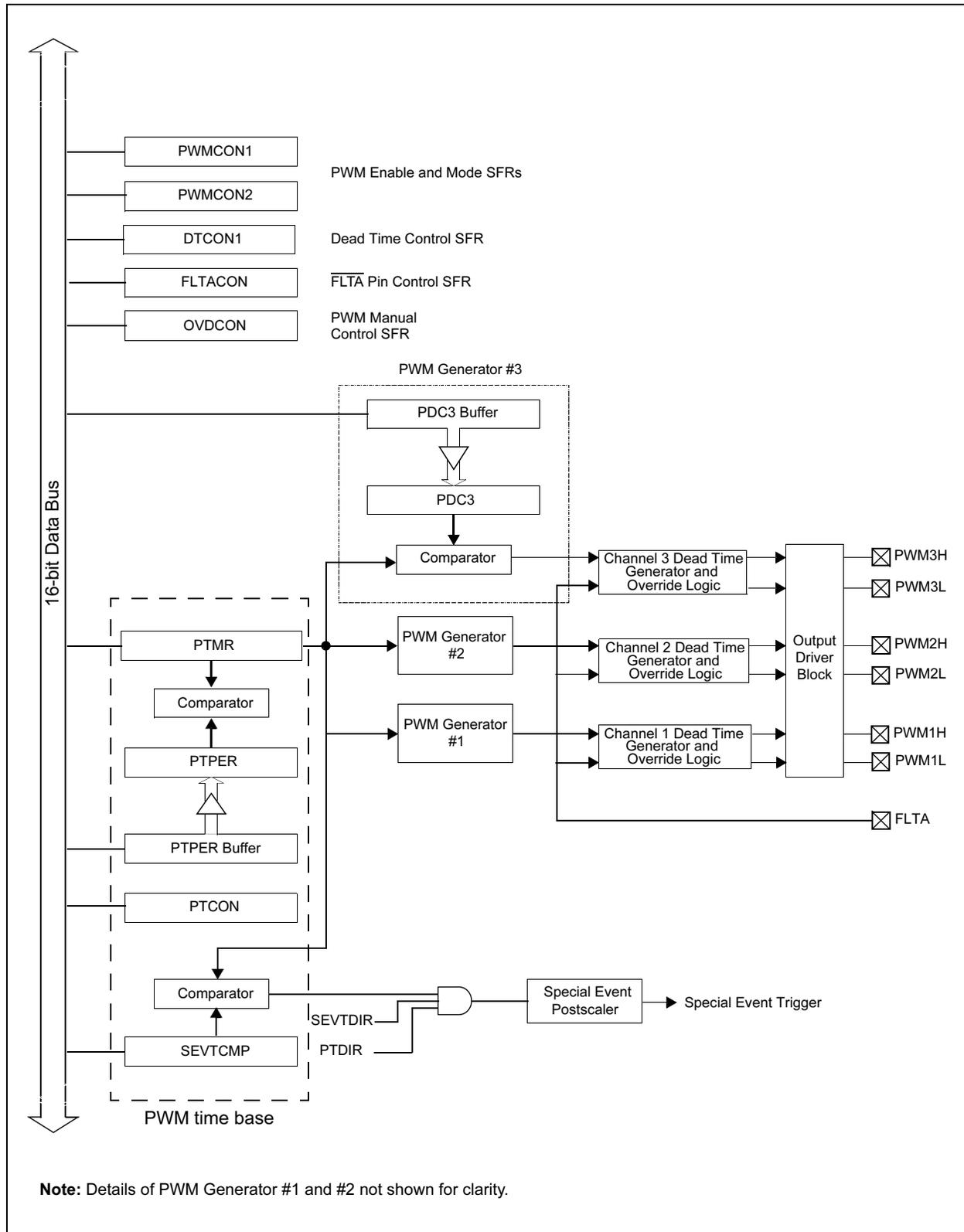


PWM is provided by the dsPIC30F2010's dedicated Motor Control (MC) PWM. The MCPWM module has been designed specifically for motor control applications. (Please refer to Figure 3 as you follow this discussion of the MCPWM module.)

The MCPWM has a dedicated 16-bit PTMR time base register. This timer is incremented by a user defined clock tick, which can be as low as  $T_{CY}$ . The user also decides the period required for the PWM by selecting a value and loading it in the PTPER registers. The PTMR is compared to the PTPER value at every  $T_{CY}$ . When there is a match, a new period is started.

The duty cycle is controlled similarly, by loading a value in the three duty cycle registers. Unlike the period compare, the value in the duty cycle register is compared at every  $T_{CY}/2$  interval (i.e., twice as fast as the period compare). If there is a match between the PTMR value and the PDCx value, then the corresponding duty cycle output is driven low or high as dictated by the PWM mode selected. The three outputs from the duty cycle compare are channeled to a complementary output pair where one output is high while the other is low, and vice versa. The two outputs can also be configured as independent outputs. When driven as complementary outputs, a dead time can be inserted between the time the high level goes low and the low level goes high. This dead time is hardware configured and has a minimum value of  $T_{CY}$ . Dead time insertion prevents inadvertent shoot-thru in output drivers.

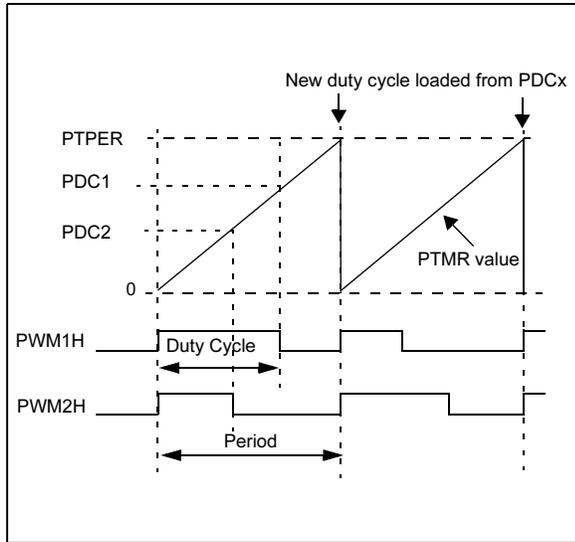
**FIGURE 3: PWM BLOCK DIAGRAM**



# AN957

There are several modes in which the MCPWM module can be configured. Edge aligned output is probably the most common mode. Figure 4 depicts the operation of an edge aligned PWM. At the start of the period, the outputs are all driven high. As the PTMR increments, a match with the duty cycle registers causes the corresponding duty cycle output to go low thereby marking the end of the duty cycle. The PTMR match with PTPER register caused a new period to start and all outputs go high to start a whole new cycle.

**FIGURE 4: EDGE-ALIGNED PWM**



The other modes that the MCPWM can be set up for are center-aligned PWM and single-shot PWM. These modes are not discussed here because they were not used for controlling the BLDC motor. For details on these modes, please refer to the *dsPIC30F Family Reference Manual (DS70046)*.

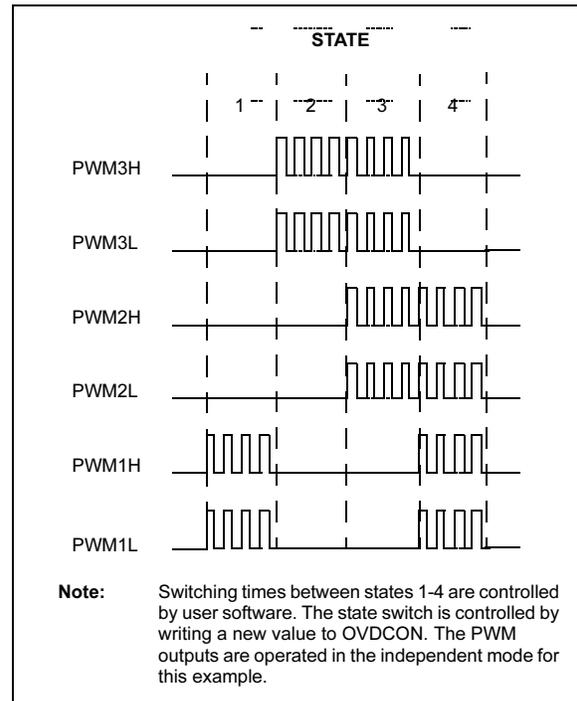
The important feature of the MCPWM used in this application is the Override Control. The Override Control is the last stage of the MCPWM module. It allows the user to directly write to the OVDCON register and control the output pins. The OVDCON register has two 6 bit fields in it. Each of the six bit fields corresponds to an output pin. The high byte portion of the OVDCON register, determines if the corresponding output pin is driven by a PWM signal (when set to 1) or (when set to 0) driven Active/Inactive by the corresponding bit field in the low byte portion of the OVDCON register. This feature allows the user to have PWM signals available, but not driving, at all output stages of the pins. For BLDC motors, the same value is written to all PDCx registers.

Depending on the value in the OVDCON register, the user can select which pin gets the PWM signal and which pin is driven active or inactive. When controlling the BLDC sensed motor it is necessary to excite two winding pairs depending on where the rotor is located and dictated by the value of the hall sensors. In the CN Interrupt Service routine the hall sensors are read and then the value of the sensors is used as an offset in a lookup table which corresponds to the value which will be loaded in the OVDCON register. Table 1 and Figure 5 show how different values are loaded in the OVDCON register depending on which sector the rotor is located in and thereby which windings need to be excited.

**TABLE 1: PWM OUTPUT OVERRIDE EXAMPLE**

State	OVDCON<15:8>	OVDCON<7:0>
1	00000011b	00000000b
2	00110000b	00000000b
3	00111100b	00000000b
4	00001111b	00000000b

**FIGURE 5: PWM OUTPUT OVERRIDE EXAMPLE**

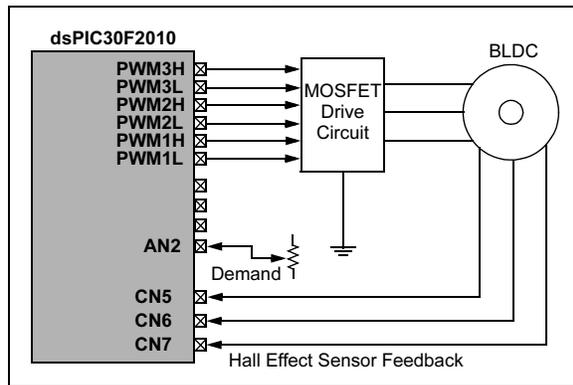


**Note:** Switching times between states 1-4 are controlled by user software. The state switch is controlled by writing a new value to OVDCON. The PWM outputs are operated in the independent mode for this example.

## HARDWARE DESCRIPTION

The block diagram in Figure 6 depicts how the BLDC motor is driven using a dsPIC30F2010. For a detailed schematic please refer to **Appendix C**.

**FIGURE 6: HARDWARE BLOCK DIAGRAM**



The six MCPWM outputs are connected to three MOSFET driver pairs (IR2101S), which in turn are connected to six MOSFETs (IRFR2407). These MOSFETs are connected in a three-phase bridge format to the three BLDC motor windings. In the current implementation, the maximum MOSFET voltage is 70 Volts, and the maximum MOSFET current is 18 Amps.

It is important to note that adequate heat dissipation must be provided if the maximum capabilities are being used. MOSFET drivers also require a higher voltage (15V) to operate, so this voltage level needs to be provided. The motor is a 24V BLDC motor so the DC+ to DC- bus voltage is 24V. A regulated 5V is provided to drive the dsPIC30F2010. The three Hall effect sensor inputs are connected to input pins that have Change Notification circuits associated with them. These inputs are enabled along with their interrupt. If a change occurs on any of these three pins, an interrupt is generated. To provide a speed demand, a potentiometer is connected to an ADC input (RB2).

To start and stop the motor, a push button switch is provided at RC14. To provide some current feedback to the motor, a low value resistor (25 milliohms) is connected between the DC- bus voltage and ground or Vss. The voltage generated by this resistor is amplified by an external op amp(MCP6002) and fed to an ADC input (RB1).

## FIRMWARE DESCRIPTION

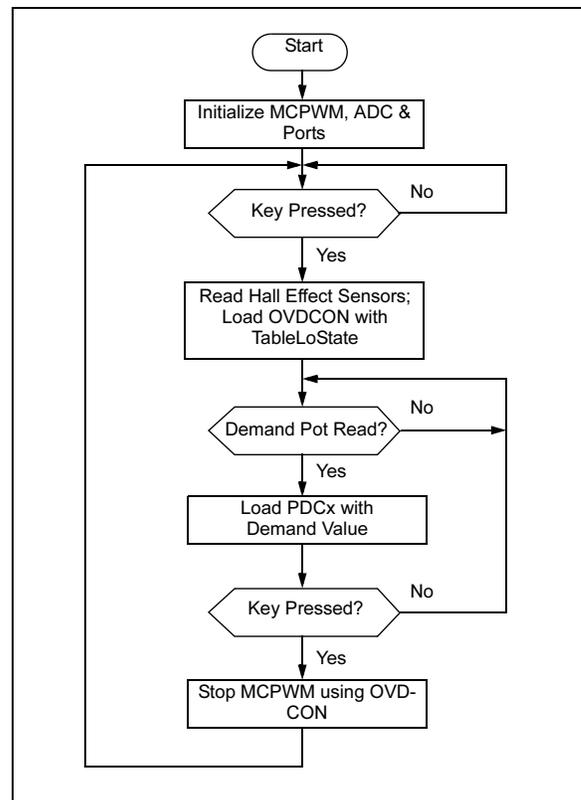
Two firmware programs are included in **Appendix A** and **Appendix B** to illustrate the methods described in the Application Note. One program uses open-loop speed control. The other uses proportional and integral feedback for closed loop speed control.

The open-loop method is generally not practical for actual applications. It is included here primarily to illustrate the BLDC motor drive methodology.

### Open-Loop Control

In open-loop control, the MCPWM directly controls motor speed based on the voltage input from the Speed Pot. After initializing the MCPWM, ADC, Ports and the Change Notification inputs, the program waits for a start activation signal (e.g., a key press) to indicate a start (see Figure 7). When the key is pressed, the Hall sensors are read. Based on their value, a corresponding value is retrieved from the table and written to the OVDCON. At this point the motor starts spinning.

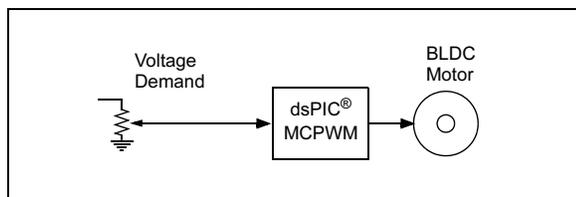
**FIGURE 7: OPEN-LOOP FLOW**



# AN957

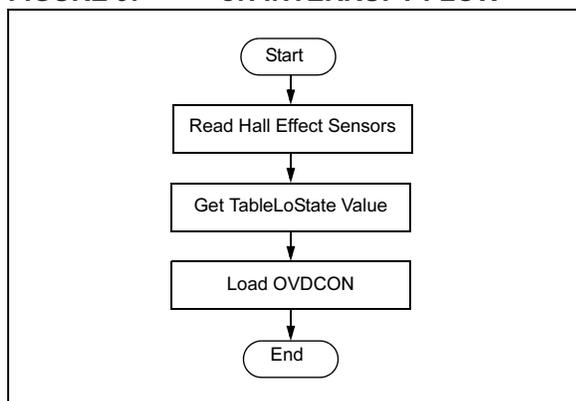
At first the duty cycle value is held at a default 50%. On the very next loop of main program, however, the potentiometer is read and its value (i.e., the correct demand value) is inserted as the duty cycle. This determines the speed of the motor. The higher the duty cycle value the faster the motor will spin. The speed is controlled by the voltage control pot, as shown in Figure 8.

**FIGURE 8: OPEN VOLTS CONTROL MODE**



The Hall effect sensors are connected to the Change Notification Pin. The CN interrupt is enabled. As the rotor spins, the position of the rotor magnet changes, and the rotor enters a different sector. Each new position is signaled by a CN Interrupt. In the CN Interrupt routine, which is shown in Figure 9, the Hall effect sensors are read and based on the value, a table lookup value is got and written to the OVDCON register. This action will insure that the correct windings are excited in the right sector and the motor will continue to spin.

**FIGURE 9: CN INTERRUPT FLOW**



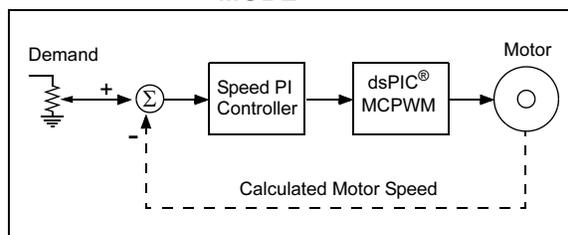
## Phase Advance

For details on Phase Advance and how to implement, please refer to AN901.

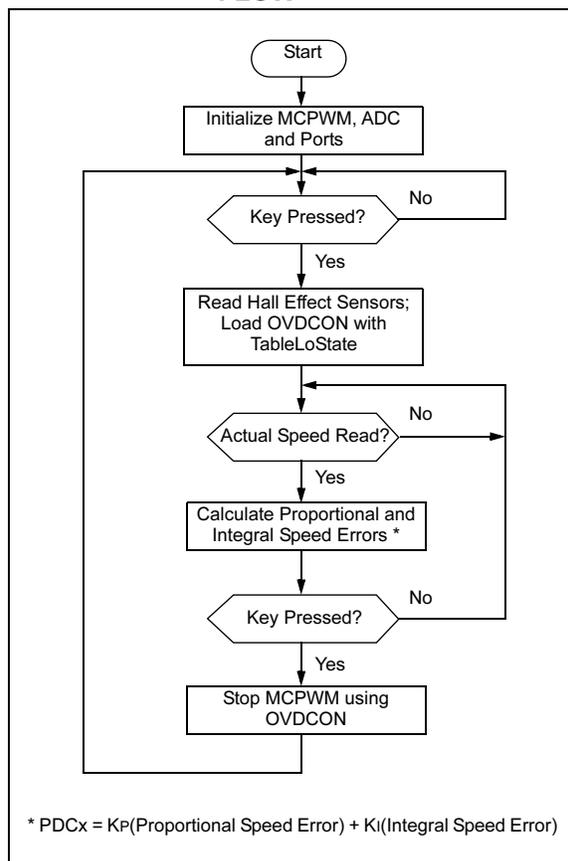
## Closed-Loop Control

In the closed-loop control version of the firmware, the main difference is that the pot is used for setting the demand. The control loop provides Proportional and Integral (PI) control of the speed. To measure the actual speed, TMR3 is used as a timer to gate a complete electrical cycle. Since we are using a 10-pole motor, five electrical cycles result in one mechanical cycle. If T seconds is the time for one electrical cycle then the speed  $S = 60/(P/2 * T)$  rpm, where P is the number of poles of the motor. The control is as shown in Figure 10. A closed-loop control flow chart is shown in Figure 11.

**FIGURE 10: CLOSED VOLTS CONTROL MODE**



**FIGURE 11: CLOSED-LOOP CONTROL FLOW**



\*  $PDCx = Kp(\text{Proportional Speed Error}) + Ki(\text{Integral Speed Error})$

## CONCLUSION

The dsPIC30F2010 is well suited for closed-loop control of a sensored BLDC motor. The peripherals and DSP engine provide an excellent bandwidth for a sensored BLDC applications with sufficient code space available for the customer's application program.

## REFERENCES

- AN885 – Brushless DC (BLDC) Motor Fundamentals
- AN901 – Using the dsPIC30F for Sensorless BLDC Control
- AN857 – Brushless DC Motor Control Made Easy
- AN889 – Brushless DC Motor Control Using PIC18FXX31 MCUs

## APPENDIX A: SOURCE CODE LISTING FOR OPEN-LOOP CONTROL

This appendix contains the source code listing for open-loop control.

### Software License Agreement

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

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.

```
//-----  
//                               Software License Agreement  
//  
// The software supplied herewith by Microchip Technology Incorporated  
// (the "Company") is intended and supplied to you, the Company's customer,  
// for use solely and exclusively with products manufacture by the Company.  
// 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.  
//-----  
// File: ClosedLoopSenBLDC.c  
//  
// Written By: Stan D'Souza, Microchip Technology  
//  
// The following files should be included in the MPLAB project:  
//  
//   ClosedLoopSenBLDC.c-- Main source code file  
//   p30f2010.gld-- Linker script file  
//  
//-----  
//  
// Revision History  
//  
// 10/01/04 -- first version  
//-----  
/*****  
    Low side driver table is as below. In this StateLoTable,  
    the Low side driver is PWM while the high side driver is  
    either on or off. This table is used in this exercise  
*****/  
  
unsigned int StateLoTable[] = {0x0000, 0x0210, 0x2004, 0x0204,  
                               0x0801, 0x0810, 0x2001, 0x0000};  
/*****  
Interrupt vector for Change Notification CN5, 6 and 7 is as below.  
When a Hall sensor changes states, an interrupt will be caused  
which will vector to the routine below.  
The user has to then read the PORTB, mask bits 3, 4 and 5,  
shift and adjust the value to read as 1, 2 ... 6. This  
value is then used as an offset in the lookup table StateLoTable  
to determine the value loaded in the OCDCON register  
*****/
```

```
void _ISR_CNInterrupt(void)
{
    IFS0bits.CNIF = 0;           // clear flag
    HallValue = PORTB & 0x0038; // mask RB3,4 & 5
    HallValue = HallValue >> 3; // shift right 3 times
    OVDCON = StateLoTable[HallValue];
}

/*****
The ADC interrupt loads the PDCx registers with the demand pot value.
This is only done when the motor is running.
*****/

void _ISR_ADCInterrupt(void)
{
    IFS0bits.ADIF = 0;
    if (Flags.RunMotor)
    {
        PDC1 = ADCBUF0;           // get value ...
        PDC2 = PDC1;             // and load all three PWMs ...
        PDC3 = PDC1;             // duty cycles
    }
}

int main(void)
{
    LATE = 0x0000;
    TRISE = 0xFFC0;              // PWMs are outputs
    CNEN1 = 0x00E0;              // CN5,6 and 7 enabled
    CNPU1 = 0x00E0;              // enable internal pullups
    IFS0bits.CNIF = 0;           // clear CNIF
    IEC0bits.CNIE = 1;           // enable CN interrupt
    InitMCPWM();
    InitADC10();
    while(1)
    {
        while (!S2);             // wait for start key hit
        while (S2)               // wait till key is released
            DelayNmSec(10);
        // read hall position sensors on PORTB
        HallValue = PORTB & 0x0038; // mask RB3,4 & 5
        HallValue = HallValue >> 3; // shift right to get value 1, 2 ... 6
        OVDCON = StateLoTable[HallValue]; // Load the override control register
        PWMCON1 = 0x0777;         // enable PWM outputs
        Flags.RunMotor = 1;       // set flag
        while (Flags.RunMotor)    // while motor is running
            if (S2)               // if S2 is pressed
            {
                PWMCON1 = 0x0700; // disable PWM outputs
                OVDCON = 0x0000;   // override PWM low.
                Flags.RunMotor = 0; // reset run flag
                while (S2)         // wait for key release
                    DelayNmSec(10);
            }
    } // end of while (1)
}
}
```

# AN957

---

```
/******  
Below is the code required to setup the ADC registers for :  
1. 1 channel conversion (in this case RB2/AN2)  
2. PWM trigger starts conversion  
3. Pot is connected to CH0 and RB2  
4. Manual Stop Sampling and start converting  
5. Manual check of Conversion complete  
*****/  
void InitADC10(void)  
{  
  
    ADPCFG = 0xFFFF8;           // all PORTB = Digital;RB0 to RB2 = analog  
    ADCON1 = 0x0064;           // PWM starts conversion  
    ADCON2 = 0x0200;           // simulataneous sample 4 channels  
    ADCHS = 0x0002;           // Connect RB2/AN2 as CH0 = pot ..  
                               // ch1 = Vbus, Ch2 = Motor, Ch3 = pot  
    ADCON3 = 0x0080;           // Tad = internal RC (4uS)  
    IFS0bits.ADIF = 0;  
    IEC0bits.ADIE = 1;  
  
    ADCON1bits.ADON = 1;       // turn ADC ON  
}  
  
/******  
InitMCPWM, intializes the PWM as follows:  
1. FPWM = 16000 hz  
2. Independant PWMs  
3. Control outputs using OVDCON  
4. Set Duty Cycle with the ADC value read from pot  
5. Set ADC to be triggered by PWM special trigger  
*****/  
void InitMCPWM(void)  
{  
    PTPER = FCY/FPWM - 1;  
  
    PWMCON1 = 0x0700;           // disable PWMs  
    OVDCON = 0x0000;           // allow control using OVD  
    PDC1 = 100;                // init PWM 1, 2 and 3 to 100  
    PDC2 = 100;  
    PDC3 = 100;  
    SEVTCMP = PTPER;  
    PWMCON2 = 0x0F00;           // 16 postscale values  
    PTCON = 0x8000;            // start PWM  
  
}  
  
//-----  
// This is a generic 1ms delay routine to give a 1mS to 65.5 Seconds delay  
// For N = 1 the delay is 1 mS, for N = 65535 the delay is 65,535 mS.  
// Note that FCY is used in the computation. Please make the necessary  
// Changes(PLLx4 or PLLx8 etc) to compute the right FCY as in the define  
// statement above.  
  
void DelayNmSec(unsigned int N)  
{  
    unsigned int j;  
    while(N--)  
        for(j=0;j < MILLISEC;j++);  
}
```

## APPENDIX B: SOURCE CODE LISTING FOR CLOSED LOOP CONTROL

This appendix contains the source code listing for closed loop control.

```
//-----
//
//          Software License Agreement
//
// The software supplied herewith by Microchip Technology Incorporated
// (the "Company") is intended and supplied to you, the Company's customer,
// for use solely and exclusively with products manufacture by the Company.
// 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 icense.
//
// 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.
//-----
// File: ClosedLoopSenBLDC.c
//
// Written By: Stan D'Souza, Microchip Technology
//
// The following files should be included in the MPLAB project:
//
//     ClosedLoopSenBLDC.c-- Main source code file
//     p30f2010.gld-- Linker script file
//
//-----
// Revision History
//
// 10/01/04 -- first version
//-----
//*****
ClosedLoopSenBLDC.c is a sensed Closed Loop Control for a BLDC motor.
The task consists of the following:
Sense changes in the Hall Sensors connected to CN5,6 & 7 (PortB)
During the CNInterrupt, read the sensors input by reading PortB
Mask and determine the state of the position 1, 2, ... 6.
Use the StateLoTable and the lookup table provided to determine the
Overload Control Register value. Set the OVDCON to this value.

The PWM is initialized to generate independant continuous PWMs.
The value of the Pot REF is used to determine the demand or desired
speed of the Motor. The desired speed value is then used with the
actual speed value to determine the Proportional Speed Error and the
Integral Speed Error. With these two values the new DutyCycle is determined
as: NewDutyCycle = Kp*(Portportional Speed Error) + Ki*(Integral Speed Error)
All 3 PWM Duty cycles are then loaded with the NewDutyCycloe 10-bit value.

The FPWM = 16000hz

The ADC is setup for a PWM trigger to start the conversion
//-----
```

# AN957

---

```
#define __dsPIC30F2010__
#include "c:\pic30_tools\support\h\p30F2010.h"

#define FCY 10000000// xtal = 5.0Mhz; PLLx8
#define MILLISEC FCY/10000// 1 mSec delay constant
#define FPWM 16000
#define Ksp1200
#define Ks10
#define RPMConstant60*(FCY/256)

#define S2!PORTCbits.RC14

void InitTMR3(void);
void InitADC10(void);
void AverageADC(void);
void DelayNmSec(unsigned int N);
void InitMCPWM(void);
void CalculateDC(void);
void GetSpeed(void);

struct {
    unsigned RunMotor : 1;
    unsigned Minus : 1;
    unsigned unused : 14;
} Flags;

unsigned int HallValue;
int Speed;
unsigned int Timer3;
unsigned char Count;
unsigned char SpeedCount;
int DesiredSpeed;
int ActualSpeed;
int SpeedError;
int DutyCycle;
int SpeedIntegral;

/*****
    Low side driver table is as below. In this StateLoTable,
    the Low side driver is PWM while the high side driver is
    either on or off. This table is used in this exercise
*****/

unsigned int StateLoTable[] = {0x0000, 0x1002, 0x0420, 0x0402,
                               0x0108, 0x1008, 0x0120, 0x0000};
/*****
Interrupt vector for Change Notification CN5, 6 and 7 is as below.
When a Hall sensor changes states, an interrupt will be caused which
will vector to the routine below.
The user has to then read the PORTB, mask bits 3, 4 and 5,
shift and adjust the value to read as 1, 2 ... 6. This
value is then used as an offset in the lookup table StateLoTable
to determine the value loaded in the OCDCON register
*****/
```

```

void _ISR_CNInterrupt(void)
{
    IFS0bits.CNIF = 0;           // clear flag
    HallValue = PORTB & 0x0038; // mask RB3,4 & 5
    HallValue = HallValue >> 3; // shift right 3 times
    OVDCON = StateLoTable[HallValue]; // Load the override control register
}

/*****
The ADC interrupt loads the DesiredSpeed variable with the demand pot
value. This is then used to determining the Speed error. When the motor
is not running, the PDC values use the direct Demand value from the pot.
*****/
void _ISR_ADCInterrupt(void)
{
    IFS0bits.ADIF = 0;
    DesiredSpeed = ADCBUF0;
    if (!Flags.RunMotor)
    {
        PDC1 = ADCBUF0;           // get value ...
        PDC2 = PDC1;             // and load all three PWMs ...
        PDC3 = PDC1;             // duty cycles
    }
}

/*****
The main routine controls the initialization, and the keypress to start
and stop the motor.
*****/

int main(void)
{
    LATE = 0x0000;
    TRISE = 0xFFC0;             // PWMs are outputs
    CNEN1 = 0x00E0;             // CN5,6 and 7 enabled
    CNPU1 = 0x00E0;             // enable internal pullups
    IFS0bits.CNIF = 0;          // clear CNIF
    IEC0bits.CNIE = 1;          // enable CN interrupt
    SpeedError = 0;
    SpeedIntegral = 0;
    InitTMR3();
    InitMCPWM();
    InitADC10();
    while(1)
    {
        while (!S2);           // wait for start key hit
        while (S2)              // wait till key is released
            DelayNmSec(10);
        // read hall position sensors on PORTB
        HallValue = PORTB & 0x0038; // mask RB3,4 & 5
        HallValue = HallValue >> 3; // shift right to get value 1, 2 ... 6
        OVDCON = StateLoTable[HallValue]; // Load the override control register
        PWMCON1 = 0x0777;        // enable PWM outputs
        Flags.RunMotor = 1;      // set flag
        T3CON = 0x8030;          // start TMR3
        while (Flags.RunMotor)   // while motor is running
            if (!S2)             // if S2 is not pressed

```

# AN957

---

```
{
    if (HallValue == 1)          //IF in sector 1
    {
        HallValue = 0xFF;       // force a new value as a sector
        if (++Count == 5)       // do this for 5 electrical revolutions or 1
                                // mechanical revolution for a 10 pole motor
        {
            Timer3 = TMR3; // read latest tmr3 value
            TMR3 = 0;
            Count = 0;
            GetSpeed(); // determine speed
        }
    }
}
else // else S2 is pressed to stop motor
{
    PWMCON1 = 0x0700; // disable PWM outputs
    OVDCON = 0x0000; // override PWM low.
    Flags.RunMotor = 0; // reset run flag
    while (S2) // wait for key release
        DelayNmSec(10);
}
} // end of while (1)
}
```

```
/******
```

Below is the code required to setup the ADC registers for :

1. 1 channel conversion (in this case RB2/AN2)
2. PWM trigger starts conversion
3. Pot is connected to CH0 and RB2
4. Manual Stop Sampling and start converting
5. Manual check of Conversion complete

```
*****/
```

```
void InitADC10(void)
```

```
{
    ADPCFG = 0xFFFF8;           // all PORTB = Digital;RB0 to RB2 = analog
    ADCON1 = 0x0064;           // PWM starts conversion
    ADCON2 = 0x0000;           // sample CH0 channel
    ADCHS = 0x0002;           // Connect RB2/AN2 as CH0 = pot.
    ADCON3 = 0x0080;           // Tad = internal RC (4uS)
    IFS0bits.ADIF = 0;         // clear flag
    IEC0bits.ADIE = 1;         // enable interrupt

    ADCON1bits.ADON = 1;       // turn ADC ON
}
```

```

/*****
InitMCPWM, initializes the PWM as follows:
1. FPWM = 16000 hz
2. Independant PWMs
3. Control outputs using OVDCON
4. Set Duty Cycle using PI algorithm and Speed Error
5. Set ADC to be triggered by PWM special trigger
*****/

void InitMCPWM(void)
{
    PTPER = FCY/FPWM - 1;

    PWMCON1 = 0x0700;          // disable PWMs
    OVDCON = 0x0000;          // allow control using OVD
    PDC1 = 100;                // init PWM 1, 2 and 3 to 100
    PDC2 = 100;
    PDC3 = 100;
    SEVTCMP = PTPER;           // special trigger is 16 period values
    PWMCON2 = 0x0F00;          // 16 postscale values
    PTCON = 0x8000;           // start PWM
}

/*****
Tmr3 is used to determine the speed so it is set to count using Tcy/256
*****/

void InitTMR3(void)
{
    T3CON = 0x0030;           // internal Tcy/256 clock
    TMR3 = 0;
    PR3 = 0x8000;
}

/*****
GetSpeed, determs the exact speed of the motor by using the value in
TMR3 for every mechanical cycle.
*****/

void GetSpeed(void)
{
    if (Timer3 > 23000)        // if TMR3 is large ignore reading
        return;
    if (Timer3 > 0)
        Speed = RPMConstant/(long)Timer3;// get speed in RPM
    ActualSpeed += Speed;
    ActualSpeed = ActualSpeed >> 1;
    if (++SpeedCount == 1)
        {SpeedCount = 0;CalculateDC();}
}

```

# AN957

---

```
/*
*****
CalculateDC, uses the PI algorithm to calculate the new DutyCycle value which
will get loaded into the PDCx registers.
*****
*/

void CalculateDC(void)
{
    DesiredSpeed = DesiredSpeed*3;
    Flags.Minus = 0;
    if (ActualSpeed > DesiredSpeed)
        SpeedError = ActualSpeed - DesiredSpeed;
    else
    {
        SpeedError = DesiredSpeed - ActualSpeed;
        Flags.Minus = 1;
    }
    SpeedIntegral += SpeedError;
    if (SpeedIntegral > 9000)
        SpeedIntegral = 0;
    DutyCycle = (((long)Ksp*(long)SpeedError + (long)Ksi*(long)SpeedIntegral) >> 12);
    DesiredSpeed = DesiredSpeed/3;
    if (Flags.Minus)
        DutyCycle = DesiredSpeed + DutyCycle;
    else DutyCycle = DesiredSpeed - DutyCycle;
    if (DutyCycle < 100)
        DutyCycle = 100;
    if (DutyCycle > 1250)
        {DutyCycle = 1250;SpeedIntegral = 0;}
    PDC1 = DutyCycle;
    PDC2 = PDC1;
    PDC3 = PDC1;
}

//-----
// This is a generic 1ms delay routine to give a 1mS to 65.5 Seconds delay
// For N = 1 the delay is 1 mS, for N = 65535 the delay is 65,535 mS.
// Note that FCY is used in the computation. Please make the necessary
// Changes(PLLx4 or PLLx8 etc) to compute the right FCY as in the define
// statement above.

void DelayNmSec(unsigned int N)
{
    unsigned int j;
    while(N--)
        for(j=0;j < MILLISEC;j++);
}

```

**APPENDIX C: SCHEMATICS**

This appendix contains schematic diagrams for using the dsPIC30F2010 to control a sensed BLDC motor.

**FIGURE C-1: MOTOR CONTROL SCHEMATIC 1**

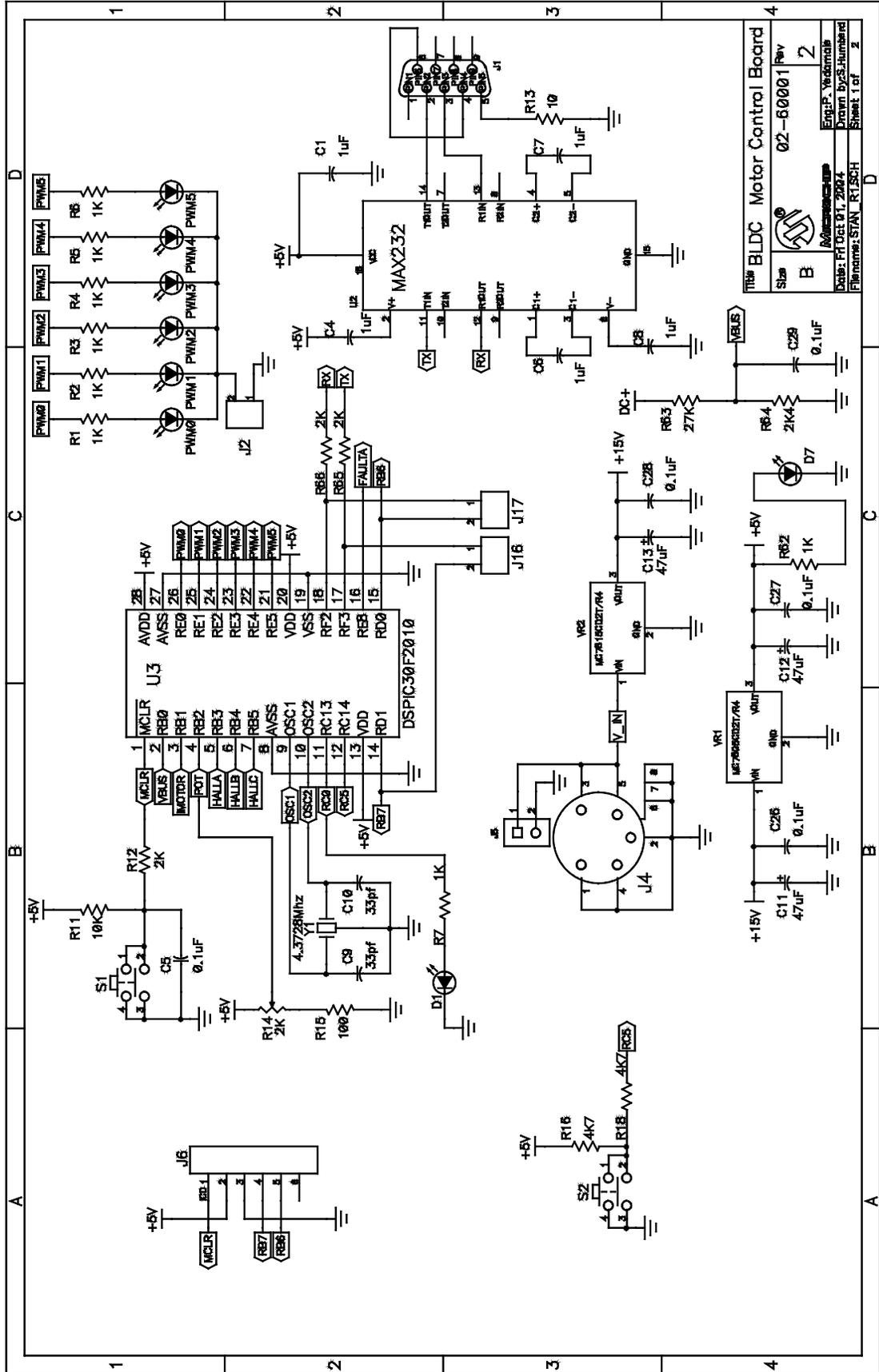
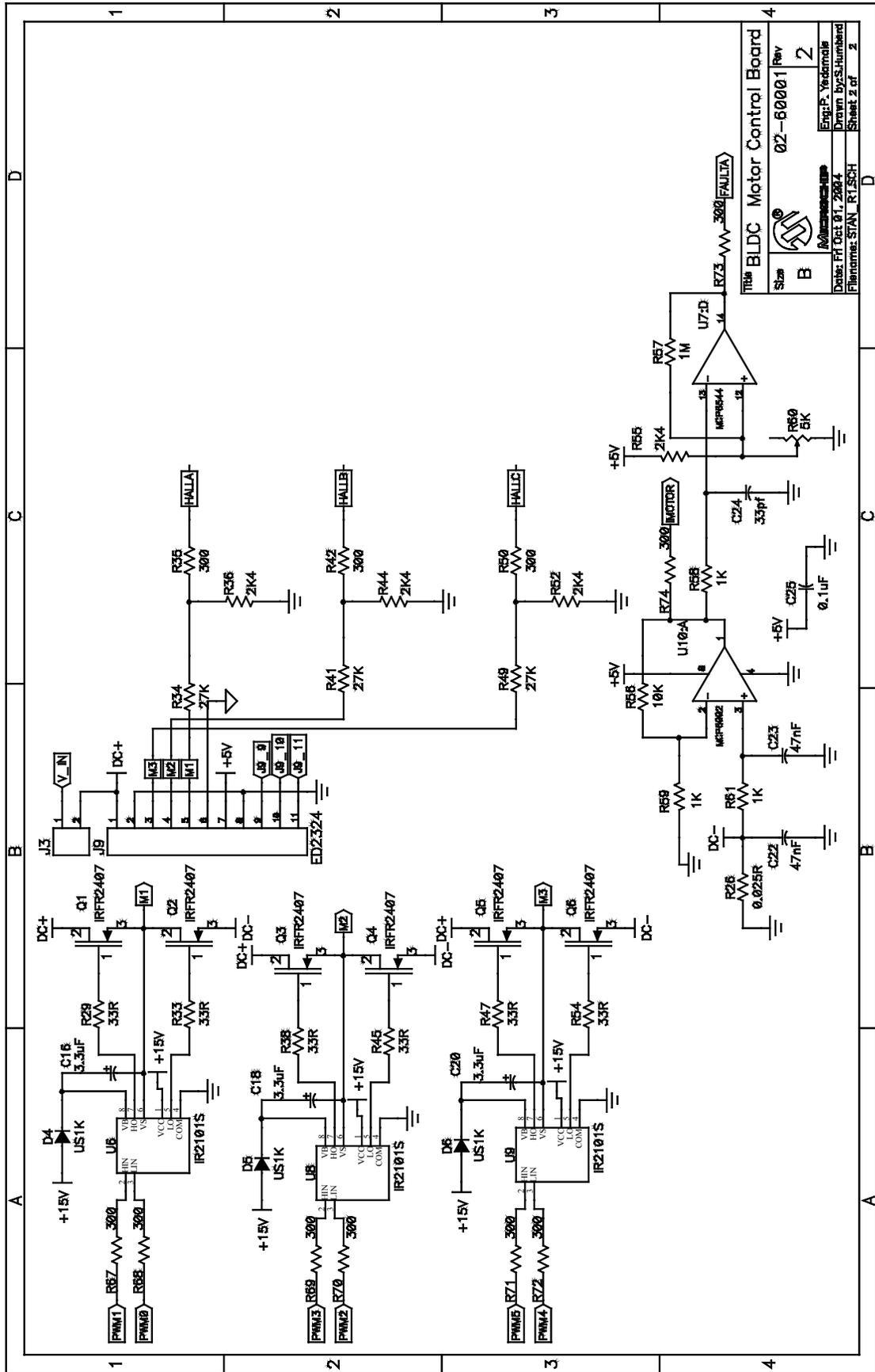


FIGURE C-2: MOTOR CONTROL SCHEMATIC 2



TIB BLDC Motor Control Board	
Size	B
Part No.	02-60001
Rev	2
Drawn By	S.Humbold
Checked By	
Date	01.2004
File Name	57AN_R1.SCH
Sheet 2 of	2

---

**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.

---

Information contained in this publication regarding device applications and the like is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. MICROCHIP MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WHETHER EXPRESS OR IMPLIED, WRITTEN OR ORAL, STATUTORY OR OTHERWISE, RELATED TO THE INFORMATION, INCLUDING BUT NOT LIMITED TO ITS CONDITION, QUALITY, PERFORMANCE, MERCHANTABILITY OR FITNESS FOR PURPOSE. Microchip disclaims all liability arising from this information and its use. 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 Microchip intellectual property rights.

**Trademarks**

The Microchip name and logo, the Microchip logo, Accuron, dsPIC, KEELOQ, microID, MPLAB, PIC, PICmicro, PICSTART, PRO MATE, PowerSmart, rPIC, and SmartShunt are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

AmpLab, FilterLab, MXDEV, MXLAB, PICMASTER, SEEVAL, SmartSensor and The Embedded Control Solutions Company are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Analog-for-the-Digital Age, Application Maestro, dsPICDEM, dsPICDEM.net, dsPICworks, ECAN, ECONOMONITOR, FanSense, FlexROM, fuzzyLAB, In-Circuit Serial Programming, ICSP, ICEPIC, Migratable Memory, MPASM, MPLIB, MPLINK, MPSIM, PICkit, PICDEM, PICDEM.net, PICLAB, PICtail, PowerCal, PowerInfo, PowerMate, PowerTool, rLAB, rPICDEM, Select Mode, Smart Serial, SmartTel and Total Endurance are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

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.

© 2004, Microchip Technology Incorporated, Printed in the U.S.A., All Rights Reserved.

 Printed on recycled paper.

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

*Microchip received ISO/TS-16949:2002 quality system certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona and Mountain View, California in October 2003. The Company's quality system processes and procedures are for its PICmicro® 8-bit MCUs, KEELOQ® code hopping devices, Serial EEPROMs, microperipherals, nonvolatile memory and analog products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001:2000 certified.*



---

## WORLDWIDE SALES AND SERVICE

---

### AMERICAS

**Corporate Office**  
2355 West Chandler Blvd.  
Chandler, AZ 85224-6199  
Tel: 480-792-7200  
Fax: 480-792-7277  
Technical Support:  
<http://support.microchip.com>  
Web Address:  
[www.microchip.com](http://www.microchip.com)

#### Atlanta

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

#### Boston

Westford, MA  
Tel: 978-692-3848  
Fax: 978-692-3821

#### 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 - 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 - Qingdao**  
Tel: 86-532-502-7355  
Fax: 86-532-502-7205

### ASIA/PACIFIC

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

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

**Japan - Kanagawa**  
Tel: 81-45-471-6166  
Fax: 81-45-471-6122

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

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

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

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

**Taiwan - Hsinchu**  
Tel: 886-3-572-9526  
Fax: 886-3-572-6459

### EUROPE

**Austria - Weis**  
Tel: 43-7242-2244-399  
Fax: 43-7242-2244-393

**Denmark - Ballerup**  
Tel: 45-4420-9895  
Fax: 45-4420-9910

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

**Germany - Ismaning**  
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

**England - Berkshire**  
Tel: 44-118-921-5869  
Fax: 44-118-921-5820