```
//------------------------------------------------------------------
//
//                          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.
//
//------------------------------------------------------------------
//  File:       SensoredBLDC.c
//
//  Written By:     Bill Anderson, Microchip Technology
//
//
// The following files should be included in the MPLAB project:
//
//      SensoredBLDC.c      -- Main source code file
//      Interrupts.c
//      Init.c
//      SensoredBLDC.h      -- Header file
//      p33FJ256MC710.gld   -- Linker script file
//
//
//------------------------------------------------------------------
//
// Revision History
//
// 06/30/07 -- first version
//------------------------------------------------------------------

#include "p33FJ32MC204.h"
#include "SensoredBLDC.h"
/*****************************************************************************/
/* Configuration bits                                                        */
/*****************************************************************************/

_FOSCSEL(FNOSC_FRC);            // Start with FRC will switch to Primary (XT, HS, EC) Oscillator
with PLL
_FOSC(FCKSM_CSECMD & POSCMD_XT);    // Clock Switching Enabled and Fail Safe Clock Monitor is
disable
                            // Primary Oscillator Mode: XT Crystal

_FBS (BSS_NO_FLASH & BWRP_WRPROTECT_OFF);
/* no Boot sector and
   write protection disabled */

_FWDT (FWDTEN_OFF);
/* Turn off Watchdog Timer */

_FGS (GSS_OFF & GCP_OFF & GWRP_OFF);
/* Set Code Protection Off for the General Segment */

_FPOR (PWMPIN_ON & HPOL_ON & LPOL_ON & FPWRT_PWR128);
/* PWM mode is Port registers
   PWM high & low active high
   alternate I2C mapped to SDA1/SCL1
   FPOR power on reset 128ms
*/

_FICD (ICS_PGD3 & JTAGEN_OFF);
/* Use PGC3/PGD3 for programming and debugging */
```

```
        void InitADC10(void);
        void DelayNmSec(unsigned int N);
        void InitMCPWM(void);
        void InitTMR3(void);
        void InitIC(void);
        void CalculateDC(void);
        void ResetPowerModule(void);
        void InitTMR1(void);
        void lockIO(void);
        void unlockIO(void);

        struct MotorFlags Flags;

        unsigned int HallValue;
        unsigned int timer3value;
        unsigned int timer3avg;
        unsigned char polecount;

        char *UartRPM,UartRPMarray[5];
        int RPM, rpmBalance;

        /************************************************************
            Low side driver table is as below.  In the StateLoTableClk
            and the StateLoTableAntiClk tables, the Low side driver is
            PWM while the high side driver is either on or off.
        ************************************************************/

        unsigned int StateTableFwd[] = {0x0000, 0x0210, 0x2004, 0x0204,
                                        0x0801, 0x0810, 0x2001, 0x0000};
        unsigned int StateTableRev[] = {0x0000, 0x2001, 0x0810, 0x0801,
                                        0x0204, 0x2004, 0x0210, 0x0000};



        int main(void)
        {
            unsigned int i;

            // Configure Oscillator to operate the device at 20Mhz
            // Fosc= Fin*M/(N1*N2), Fcy=Fosc/2
            // Fosc= 8*10/(2*2)= 20Mhz for 8M input clock

            PLLFBD =  8;                // M=10
            CLKDIVbits.PLLPOST = 0;     // N1=2
            CLKDIVbits.PLLPRE = 0;      // N2=2
            __builtin_write_OSCCONH(0x03);
            __builtin_write_OSCCONL(0x01);

            while(OSCCONbits.COSC != 0b011);
            // Wait for PLL to lock
            while(OSCCONbits.LOCK != 1);

            TRISA |= 0x0100;    // S2 as input RA8
            TRISB |= 0x0010;    // S3 as input RB4

            // Analog pin for POT already initialized in ADC init subroutine

            unlockIO();
            RPINR7bits.IC1R = 0x01;     // IC1 on RP1/RB1
            RPINR7bits.IC2R = 0x02;     // IC2 on RP2/RB2
            RPINR10bits.IC7R = 0x03;    // IC7 on RP3/RB3
            lockIO();

            InitADC10();
            InitTMR1();
            InitTMR3();
            timer3avg = 0;
            InitMCPWM();
            InitIC();
            Flags.Direction = 1;                // initialize direction CW

            for(i=0;i<1000;i++);

            while(1)
            {
                while(S2)                   // wait for start key hit
```

```
            {
                if (!S3)                // check for direction change
                {
                    while (!S3)         // wait till key is released
                        DelayNmSec(10);
                        Flags.Direction ^= 1;
                }
                Nop();
            }
            while (!S2)                 // wait till key is released
                DelayNmSec(10);

                // read hall position sensors on PORTD
            HallValue = (unsigned int)((PORTB >> 1) & 0x0007);
            if (Flags.Direction)
                OVDCON = StateTableFwd[HallValue];
            else
                OVDCON = StateTableRev[HallValue];

            PWMCON1 = 0x0777;           // enable PWM outputs
            Flags.RunMotor = 1;         // set flag
            T3CONbits.TON = 1;          // start tmr3
            polecount = 1;
            DelayNmSec(100);

            while (Flags.RunMotor)              // while motor is running
            {
                if (!S2)                        // if S2 is pressed
                    {
                    PWMCON1 = 0x0700;           // disable PWM outputs
                    OVDCON = 0x0000;            // overide PWM low.
                    Flags.RunMotor = 0;         // reset run flag
                    while (!S2)                 // wait for key release
                        DelayNmSec(10);
                    }
                Nop();
            }
        }
    }
    //------------------------------------------------------------------------
    // 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++);
    }

    /*****************************************************************************
     * Function: lockIO
     *
     * Preconditions: None.
     *
     * Overview: This executes the necessary process to set the IOLOCK bit to lock
     * I/O mapping from being modified.
     *
     * Input: None.
     *
     * Output: None.
     *
     ****************************************************************************/
    void lockIO(){

    asm volatile ("mov #OSCCON,w1 \n"
                  "mov #0x46, w2 \n"
                  "mov #0x57, w3 \n"
                  "mov.b w2,[w1] \n"
                  "mov.b w3,[w1] \n"
                  "bset OSCCON, #6");
    }
```

```
/*****************************************************************************
 * Function: unlockIO
 *
 * Preconditions: None.
 *
 * Overview: This executes the necessary process to clear the IOLOCK bit to
 * allow I/O mapping to be modified.
 *
 * Input: None.
 *
 * Output: None.
 *
 *****************************************************************************/
void unlockIO(){

asm volatile ("mov #OSCCON,w1 \n"
                "mov #0x46, w2 \n"
                "mov #0x57, w3 \n"
                "mov.b w2,[w1] \n"
                "mov.b w3,[w1] \n"
                "bclr OSCCON, #6");
}
```