// File: Interrupts.c
//
//
//
// The following files should be included in the MPLAB project:
//
//   SensoredBLDC.c   -- Main source code file
//   Interrupts.c
//   Init.c
//   SensoredBLDC.h   -- Header file
//   p33FJ32MC204.gld   -- Linker script file
//
//
//--

```
#include "p33FJ32MC204.h"
#include "SensoredBLDC.h"
int DesiredSpeed;
int ActualSpeed;
int SpeedError;
long SpeedIntegral = 0, SpeedIntegral_n_1 = 0, SpeedProportional = 0;
long DutyCycle = 0;
unsigned int Kps = 20000;                   // Kp and Ks terms need to be adjusted
unsigned int Kis = 2000;                    // as per the motor and load

/*****************************************************************
Function:       void __attribute__((interrupt, no_auto_psv)) _ADC1Interrupt (void)

Overview:       For Open loop, the ADC interrupt loads the PDCx
                registers with thedemand pot value.  This is only
                done when the motor is running.
                For Closed loop, the ADC interrupt saves into
                DesiredSpeed the demand pot value.  This is only
                done when the motor is running.

*****************************************************************/

void __attribute__((interrupt, no_auto_psv)) _ADC1Interrupt (void)
{
    if (Flags.RunMotor)
    #ifdef CLOSEDLOOP                   // For closed loop, save
        DesiredSpeed = ADC1BUF0 * POTMULT;  // value for speed control
    #else
    {                                   // For open loop control,
        P1DC1 =  (ADC1BUF0 >> 1);           // get value,
        P1DC2 = P1DC1;                  // and load all three PWM
        P1DC3 = P1DC1;                  // duty cycles
    }
    #endif


    // reset ADC interrupt flag
    IFS0bits.AD1IF = 0;
```

```
}

/*****************************************************************
Function:       void __attribute__((interrupt, no_auto_psv)) _IC1Interrupt (void)

PreCondition:   The inputs of the hall effect sensors should have low pass
                filters. A simple RC network works.

Overview:       This interrupt represents Hall A ISR.
                In Reverse, Hall reading == 3 or 4
                In Forward, Hall reading == 2 or 5
                and generates the next commutation sector.
                Hall A is used for Speed measurement
*****************************************************************/

void __attribute__((interrupt, no_auto_psv)) _IC1Interrupt (void)
{
    int Hall_Index;

    IFS0bits.IC1IF = 0; // Clear interrupt flag
    HallValue = (unsigned int)((PORTB >> 1) & 0x0007);  // Read halls

    if (Flags.Direction)
    {
        OVDCON = StateTableFwd[HallValue];
        Hall_Index = HALL_INDEX_F;
        }
    else
    {
        OVDCON = StateTableRev[HallValue];
        Hall_Index = HALL_INDEX_R;
        }

// The code below is uses TMR3 to calculate the speed of the rotor
    if (HallValue == Hall_Index)  // has the same position been sensed?
        if (polecount++ == POLEPAIRS)  //has one mech rev elasped?
        {                               // yes then read timer 3
        timer3value = TMR3;
        TMR3 = 0;
        timer3avg = ((timer3avg + timer3value) >> 1);
        polecount = 1;
        }
}

/*****************************************************************
Function:       void __attribute__((interrupt, no_auto_psv)) _IC2Interrupt (void)

PreCondition:   The inputs of the hall effect sensors should have
                low pass filters. A simple RC network works.

Overview:       This interrupt represents Hall B ISR.
                Hall reading == 1 or 6
                and generates the next commutation sector.

*****************************************************************/

void __attribute__((interrupt, no_auto_psv)) _IC2Interrupt (void)
{
    IFS0bits.IC2IF = 0; // Clear interrupt flag
    HallValue = (unsigned int)((PORTB >> 1) & 0x0007);  // Read halls

    if (Flags.Direction)
        OVDCON = StateTableFwd[HallValue];
    else
        OVDCON = StateTableRev[HallValue];
}

/*****************************************************************
Function:       void __attribute__((interrupt, no_auto_psv)) _IC7Interrupt (void)

PreCondition:   The inputs of the hall effect sensors should have
                low pass filters. A simple RC network works.

Overview:       This interrupt represents Hall C ISR.
                and generates the next commutation sector.
```

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

void __attribute__((interrupt, no_auto_psv)) _IC7Interrupt (void)
{
    IFS1bits.IC7IF = 0; // Clear interrupt flag
    HallValue = (unsigned int)((PORTB >> 1) & 0x0007);  // Read halls

    if (Flags.Direction)
        OVDCON = StateTableFwd[HallValue];
    else
        OVDCON = StateTableRev[HallValue];
}

/*****************************************************************
Function:       void __attribute__((interrupt, no_auto_psv)) _T3Interrupt (void)

PreCondition:   None.

Overview:       This interrupt a 1ms interrupt and outputs a square
                wave toggling LED4.

*****************************************************************/

void __attribute__((interrupt, no_auto_psv)) _T1Interrupt (void)
{
#ifdef CLOSEDLOOP
    ActualSpeed = SPEEDMULT/timer3avg;
    SpeedError = DesiredSpeed - ActualSpeed;
    SpeedProportional = (int)(((long)Kps*(long)SpeedError) >> 15);
    SpeedIntegral = SpeedIntegral_n_1 + (int)(((long)Kis*(long)SpeedError) >> 15);

    if (SpeedIntegral < 0)
        SpeedIntegral = 0;
    else if (SpeedIntegral > 32767)
        SpeedIntegral = 32767;
    SpeedIntegral_n_1 = SpeedIntegral;
    DutyCycle = SpeedIntegral + SpeedProportional;
    if (DutyCycle < 0)
        DutyCycle = 0;
    else if (DutyCycle > 32767)
        DutyCycle = 32767;

    PDC1 = (int)(((long)(PTPER*2)*(long)DutyCycle) >> 15);
    PDC2 = PDC1;
    PDC3 = PDC1;
#endif                              // in closed loop algorithm

    IFS0bits.T1IF = 0;
}
```