```
// 22-03-2011 Leg-driver for HI6SIM demo moving platform.
//
// Move the leg motor with potentiometer(Pos) to follow the input Pot.(ReqPos).
// Prevent overshoot end of travel(MinPos, MaxPos).
// Prevent PWM saturation(MaxRun).
// Slow run-home(SlowRun)
// Dead-zone around requested position(DeadBand).
// If Pos 0, blink red.(not connected)
// If ReqPos 0, blink green, Use SlowRun (slower)(or not connected)
// Run home closer to end.
// Protect aganst quick reverse at high speed
// Protect against Blocked motor en retry
// Fixed lower end stop problem
// Implemented PID
// If negative duty, break bij littel reverse power
// Crawl to target bij increasing power if target not reached
// Better direction setting
// PID cleanup
// Reduced int from 1ms to 10 ms
// Driver leg adress, adress in/out daisy chain,
// usart request position input from:
//       Ians (BFF)6dof code. mode BIN Baud 38.400
//
// By  Douwe Jippes  d.jippes@hccnet.nl   www.xs4all.nl/~jippes/hi6sim

#include <p18f2431.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <adc.h>
#include <delays.h>
#include <timers.h>
#include <portb.h>
#include <pcpwm.h>
#include <adc.h>
#include <usart.h>
#include "EEP.h"

/* Set up the configuration bits */
#pragma config OSC  = HSPLL, FCMEN = OFF, IESO = OFF    //1H 40MC
#pragma config PWRTEN = OFF, BOREN = OFF, BORV = 42             //2L
#pragma config WDTEN  = OFF, WINEN = OFF, WDPS = 64             //2H
#pragma config T1OSCMX = OFF, HPOL = HIGH, LPOL= HIGH, PWMPIN = OFF     //3L
#pragma config MCLRE = ON
//3H
#pragma config STVREN = ON, LVP = OFF, DEBUG = OFF              //4L
#pragma config CP0 = OFF, CP1= OFF                                              //5L
#pragma config CPB = OFF,CPD = OFF                                              //5H
#pragma config WRT0 = OFF, WRT1 = OFF                                   //6L
#pragma config WRTB = OFF, WRTC = OFF, WRTD = OFF               //6H
#pragma config EBTR0 = OFF, EBTR1 = OFF                                 //7L
#pragma config EBTRB = OFF
//7H

// Limmits & contants
const unsigned int MaxRun = 3950;               // Max Duty (PWM)
const unsigned int SlowRun = 2500;              //
const unsigned int MustRun = 1000;              // Must/should be running
const unsigned int Fast = 2;                    // Stopping if faster then. (reversing)
const unsigned int StopShiftL = 1;              // Stop delay shift mpy
const unsigned int StopTimeMax = 1000;  // Stop Delay max
```

```
const unsigned int Minimum = 300;               // Minimum duty reverse stopping
const unsigned int Speed = 2;                   // Break if faster then. (To target)
const unsigned int BreakDist = 0;               // Break at distance from target
const unsigned int MinDuty = 300;               // Minimum duty braking to target
const unsigned int BDmpy = 1;                   // Break delay mpy
const unsigned int Notjet = 5;                  // Almost there at target
const unsigned int CrawlDelay = 2;              // Almost there delay add
const unsigned int MaxCrawl = 50;               // Almost there max error add
const unsigned int ErrSumNum = 4;               // # of Errors collected
const unsigned int Kp = 50;                             // Error * = Proportional
const unsigned int Ki = 5;                              // Error sum * = Integral
const unsigned int Kd = 80;                             // -Position delta * = Derivative
const unsigned int Small = 2;                   // Dead band
const unsigned int Wide = 3;                    // ,,
const unsigned int MaxPos = 820;                // End stop up
const unsigned int MinPos = 50;                 // End stop down
const unsigned int FFdelay = 10;                // Blink delay
const unsigned int MPdelay = 0;                 // Messure Error Position & delay Hist
10+(n*10) ms
const unsigned int Dondelay = 100;              // Duty on delay if not moving
const unsigned int Doffdelay1 = 32;     // Duty off(cool down) first delay if not moving
const unsigned int SM = 0;                              // Slope mesure end
const unsigned int LogTime = 10;                // Write log to EEPROM when long atend
ReqPos (Err & Delta Pos)

#define JAM             // Jammed motor detect (comment for debug)

//PID Constants
#define F_OSC   40000000                                // 40 Mc
#define K_TMR1   65535 - (F_OSC/4000)   // 8 ms (2 setduty between usart reads)
//#define K_TMR1   65535 - (F_OSC/2286) // 14 ms  (like read usart cycle BFF)
#define K_Tnos   65535 - (F_OSC/64)             // 500 ms
#define K_Tbit   65535 - (F_OSC/32000)  // 1 ms
#define K_THbit   65535 - (F_OSC/64000)// 0.5 ms
#define K_Tdisc   65535 - (F_OSC/320)   // 100 ms


//-----PORTA CONFIGURATION-----
//#define       d0      PORTAbits.RA0           // ad selected
//#define       d1      PORTAbits.RA1           // ad current
//#define       d2      PORTAbits.RA2           // ad position / Index
//#define       d3      PORTAbits.RA3           // ad req pos / QEA
//#define       d4      PORTAbits.RA4           // QEB

//-----PORTB CONFIGURATION-----
//#define       d0              PORTBbits.RB0   // pwm 0
//#define       d1              PORTBbits.RB1   // pwm 1
//#define       d2              PORTBbits.RB2   // red
//#define       d3              PORTBbits.RB3   // green
//#define       d4              PORTBbits.RB4   // VB on
//#define       d5              PORTBbits.RB5   // SM
//#define       d6              PORTBbits.RB6   // PGC
//#define       d7              PORTBbits.RB7   // PGD

//-----PORTC CONFIGURATION-----
#define TxEn    PORTCbits.RC0           // tx enable
//#define       d1      PORTCbits.RC1           // FLTA
#define A_Out   PORTCbits.RC2           // AdrOut
#define A_In    PORTCbits.RC3           // AdrIn
#define Sel1    PORTCbits.RC4           // ad0 select 1
#define Sel2    PORTCbits.RC5           // ad0 select 2
```

```
//#define       d6      PORTCbits.RC6           // TX
//#define       d7      PORTCbits.RC7           // RX

#define         RCIF    PIR1bits.RCIF   // Reception complete int flag(off when RCREG read)
#define         FERR    RCSTAbits.FERR  // Frame error
#define         OERR    RCSTAbits.OERR  // Overrun error
#define         CREN    RCSTAbits.CREN  // Continuous Rec Enable(off resets errors)

#define         TXIF    PIR1bits.TXIF   // TXREG empty int flag(off when TXREG written)
#define         TRMT    TXSTAbits.TRMT  // TSR empty

#define         TMR1IF  PIR1bits.TMR1IF // TMR1 int flag(sw clear)

#define         GO              ADCON0bits.GO   // Start ad

#define         off             0
#define         on              1

char prid[] = " By www.xs4all.nl/~jippes/hi6sim v11";
unsigned int Pos = 1;
unsigned int PI0 = 1;                   // Hist Pos. Input
unsigned int PI1 = 1;
unsigned int PI2 = 1;
unsigned int PI3 = 1;
unsigned int PI4 = 1;
unsigned int PI5 = 1;
unsigned int OldPos = 1;                // Older Position used to calqulate Derivative * Kd
unsigned int ReqPosD = 1;
unsigned short long ReqPosL;
unsigned int ReqPos = 1;
unsigned int RP0 = 1;
unsigned int RP1 = 1;                           // History Req. Pos. input after spike
removal
unsigned int RP2 = 1;
unsigned int RP3 = 1;
unsigned int RP4 = 1;
unsigned int RP5 = 1;
unsigned int RPI0 = 1;                  // Hist of Req. Pos. Input
unsigned int RPI1 = 1;
unsigned int RPI2 = 1;
unsigned int RPI3 = 1;
unsigned int RPI4 = 1;
unsigned int RPI5 = 1;
unsigned int Error = 0;                         // position Error for Proportional * Kp
unsigned int PrevError = 0;
unsigned int E0 = 0;                    // Error history
unsigned int E1 = 0;
unsigned int E2 = 0;
unsigned int E3 = 0;
unsigned int E4 = 0;
unsigned int P0 = 1;                            // Position history after spike removal
unsigned int P1 = 1;
unsigned int P2 = 1;
unsigned int P3 = 1;
unsigned int P4 = 1;
unsigned int P5 = 1;
int DeltaPos = 0;
int DP0 = 0;                            // Delta Position hist
int DP1 = 0;                            // History
int DP2 = 0;
int DP3 = 0;
```

```
int DP4 = 0;
int DP5 = 0;
int DP6 = 0;
int DP7 = 0;
int DPsum =0;
int DRP = 0;                                    // Delta Request Position
unsigned int ErrSum = 0;                    // Last error sum for Integral * Ki
int Stopping = 0;                 // Stop delay calculated from old error sum
unsigned int i;                                 // ?
unsigned int DeadBand;
unsigned int Duty;
unsigned int FFcount = 0;                       // Blink delay
unsigned int MPcount = 0;
unsigned int Doffcount = 0;
unsigned int Doffdelay = 0;
unsigned int Doncount = 0;
unsigned int MaxDuty;
unsigned int WP=0;                                      // Write Pposition in EEPROM
unsigned int CrawlDelCnt =0;              // Crawl add delay count
unsigned int Crawl =0;                          // Crawl to targed if stoped nearby
unsigned int OldDPsum =0;

int DeltaDeltaPos = 0;                          //Speed change
int TempDuty = 0;
int TermP = 0;
int TermI = 0;
int TermD = 0;

unsigned char Direction = 0;
unsigned char PrevDirection = 0;
unsigned char Reverse = 0;
unsigned char DeCel = 0;
unsigned char Hist = 0;           //Collect History of Error an Delta Pos
unsigned char DidHist = 0;        // for dibugging
unsigned char Rec;
unsigned char RDir = 0;
unsigned char AtLowerEnd = 0;
unsigned char Break = 0;
unsigned char PWMdirSav;
unsigned char Slow = 1;
unsigned char First = 1;

// Adress of leg driver
unsigned char Adress = 0;                 // Adress of leg driver
unsigned char AdrInit = 0;                // In get adress mode
unsigned char AdrIn = 0;                  // Got adress in
unsigned char AdrStart = 0;               // Adr in started
unsigned char AdrOut = 0;                 // adr out started
unsigned char Mask;                             // adr bit scan mask
unsigned char ConLost = 0;                // Connectinon lost

// Usart
unsigned char UdId1;
unsigned char UdId2;
unsigned char UdCnt;
unsigned char Udata;
unsigned char UdBuf[39];                  // PosReq input buffer(index with Adress)
#define trl     64
unsigned char tb[trl];
unsigned char ti;
unsigned char trid;
```

```
// A/D conversion groups
int group_a;
int group_b;
int group_c;
int group_d;

void main (void);
void trace(char trid);
void Init(void);
void Int_High(void);
void DoAdr(void);
void Ain_CHange(void);
void Report(void);
void GetPos(void);
void SetDuty(void);
void SetChanADC( unsigned char channel );
void Setdc0pcpwm(unsigned int dutycycle);
void Delay(char Del);

void trace(char trid)
{
        tb[ti] = trid;
        ti++;
        if (ti > trl)
                ti = 0;
        if (trid == 2)
        {
                tb[ti] = ReqPos >> 2;
                ti++;
        }
        else
        {
                if (trid == 4)
                {
                        tb[ti] = Udata;
                        ti++;
                }
        }
        if (ti > trl)
                ti = 0;
}

void Delay(char Del)
{
 int d;
  char n;
        for (n = Del; n > 0; n--)
        {
                if (A_In)
                        return;
                for (d = 0; d < 327; d++);
        }
        ;
}

// High priority interrupt vector

#pragma code InterruptVectorHigh = 0x08
void InterruptVectorHigh (void)
{
```

```
  _asm
    goto Int_High //jump to interrupt routine
  _endasm
}

void main (void)
{
        Init();
        A_Out = on;
        while(1)
        {
//              if (Reverse || Break)
//                      PORTB |= 0x10;                          // Orange on(test)
//              else
//                      PORTB &= 0xEF;                          //Orange off (initial
test)
        }
} // End main

void Init (void)
{
                // Setup PCPWM port C
        TRISC = 0x0A;           // rc3 adr in, rc1 !flta
        A_Out = on;

                //setup A/Dconverters port A
//      PORTB |= 0x10;  //DEBUG Orange on ************************8**********
        PORTB ^= 0x04;  //DEBUG Red Flip Flop ********************************
        TRISA  = 0x1F;          //an4-0
        ADCON0 = 0x1D;          //ss,multchan,sim2,on
        ADCON1 = 0x10;          //+-,fifo
        ADCON2 = 0xB2;          //right,12tad,fosc32
        ADCON3 = 0x10;          //pwm trig
        ANSEL0 = 0x0C;          //an3,2
        ADCHS  = 0x00;          //sel 0,2,1,3
                        // Setup PCPWM port B
        TRISB   = 0x00;         //
        PTCON0  = 0x00;         // post,pre scale,free run
        PTCON1  = 0x80;         // time base on, up
        PWMCON0 = 0x21;         // pwm0&1, indipendent
        PWMCON1 = 0x01;         // post 1:1
        FLTCONFIG       = 0x03; // flta auto
        OVDCONS = 0x00;         // active via OVDCOND
        OVDCOND = 0x00;         // stop
        PTPERH  = 0x03;         // pcpwm period 19.5 kc
        PTPERL  = 0xff;         //

        Slow = 1;
        DeadBand= Small;                                //
        Direction = PrevDirection;              //Initial direction
        OVDCOND = (1 << Direction);             //Select PCPWM direction (2 OR 1)

#ifdef JAM
        Doffdelay = Doffdelay1;
#endif

        Rec = 1;
        RCONbits.IPEN = 1;                      // Priority enable
        OpenTimer1 (TIMER_INT_ON    &
        T1_8BIT_RW        &
        T1_SOURCE_INT    &
```

```
        T1_PS_1_8       &
        T1_OSC1EN_OFF   &
        T1_SYNC_EXT_OFF    );

                                        // open the serial port for 38.400
        BAUDCTLbits.BRG16 = on;
        OpenUSART(USART_TX_INT_OFF&
//** TX int off **)
                USART_RX_INT_OFF&
                USART_ASYNCH_MODE&
                USART_EIGHT_BIT&
                USART_ADDEN_OFF&
                USART_CONT_RX&
                USART_BRGH_HIGH,
                 254);
        Adress = 0;
        AdrInit = on;                           // Adres init mode
        for (i = 0; i < 39; i++)
                UdBuf[i] = 0;
                UdId1 = 0;
        for (ti = 0; ti < trl; ti++)
                tb[ti] = 0;
        ti = 0;
        UdCnt = 0;
        if (!A_In)                                      // Adr in line already low?
                Delay(100);                             // First unit on chain delay
        if (!A_In)                                      // Adr in line low?
        {
                Delay(1);                                       // First unit on chain
extra delay                                                     //
                AdrStart = on;                          // Adr in line gone low
                Mask = 16;                                      // Adr bits from previous
unit in chain
                                                                // First = start
bit(low)(s,8,4,2,1,e)
        PORTB ^= 0x04;  //DEBUG Red Flip Flop ********************************
                WriteTimer1 (K_THbit);  // chek in the midst of a bit time
                TMR1IF = off;
        }
        else
        {
                WriteTimer1 (K_Tnos);
                INTCON2bits.INTEDG0 = on;       // Adr in low ext int
                INTCONbits.INT0IE = on;         // Alow
        }
        INTCONbits.GIEH = 1;            // High priority int enable
//      INTCONbits.GIEL = 1;            // Low priority int enable
}


void Int_High(void)
{
        if (TMR1IF)                             // Timer 1 int
        {
                TMR1IF = off;
                if (AdrInit)                    // Driver adr init
                {
                        trace (1);
                        DoAdr();
                }
                else
```

```
                {
                        trace (2);
                        SetDuty();              // PID & Duty
                }
        }
        else
        {
                if (INTCONbits.INT0IF)  // Adress line in change
                {
                        INTCONbits.INT0IF = off;
                        Ain_CHange();
                        trace (3);
                }
                else
                {
                        if (RCIF)                               // Usart data in
                        {
                    INTCONbits.GIEH = 0;        //
                                RCIF = off;
                                if (Adress)
                                        GetPos();                       // & trace Usart
                        }
                        else
                        {
                                if (TXIF)                               // Usart data out
                                {
                                        TXIF = off;
                                        Report();
                                }
                                else
                                        trace (5);
                        }
                }
        }
        INTCONbits.GIEH = 1;    //
}

void Ain_CHange()
{
        char a = 0;
        for (i = 0; i < 10; i++)
        {
                if (A_In)
                        a++;
        }
        if (a > 5)              // Adr in high and previous low
        {
                INTCON2bits.INTEDG0 = 0; // Ext int on adr-in low
                Adress = 0;                             // No adress
                A_Out = on;                             //Signal next unit
                PIE1bits.RCIE = 0;                      // Usart int disble
        }
        if (a < 5)              // Adr in low and previous high
        {
        PORTB &= 0xEF;                          //Orange off
(test)************************************
                AdrInit = on;                           // Adres init mode
                AdrStart = on;                          // Adr in line gone low
                AdrOut = off;
                Mask = 16;                                      // Adr bits from previous
unit in chain
```

```
                WriteTimer1 (K_THbit);  // chek in the midst of a bit time
                INTCONbits.INT0IE = off;        // No ext int
        }
        Slow = on;
}

void DoAdr(void)
{
        PORTB ^= 0x10;  //DEBUG Orange Flip Flop ******************************
        if (!AdrOut)                                    // not jet in adr out
        {
                if (!AdrStart)                          // Adr sequence started?
                {                                                       // No
                        AdrInit = off;                          // SW's off
                        INTCON2bits.INTEDG0 = 0;
                        INTCONbits.INT0IE = on;
                        WriteTimer1 (K_TMR1);           // PID timer
                }
                else
                {                                               // Adr sequence
                        if (A_In)                               // Adr in high
                                Adress |= Mask;         // Set bit in Adr
                        Mask >>= 1;                             // next lower bit
//      PORTB ^= 0x04;  //DEBUG Red Flip Flop *********************************
                        if (Mask)                               // More bits to go
                                WriteTimer1 (K_Tbit);
                        else
                        {
                                Adress += 1;                    // His adr +1 = my adr
                                AdrOut = on;                    // Adr out mode
                                A_Out = off;                    // Adr out start bit
(s,8,4,2,1,e)
                                Mask = 8;                               // Msb first of 4
bits
                                WriteTimer1 (K_Tbit);
                        }
                }
        }
        else
        {
                if (Adress & Mask)
                        A_Out = 1;                      // Sent my adr bits
                else
                        A_Out = 0;
                if (Mask)                                               // In adr out more
bits
                {
                        Mask >>= 1;                                     // next bit
                        WriteTimer1 (K_Tbit);
                }
                else                                                    // Mask was 0 =
stop bit(low)
                {                                                               // Adr out
low.
                        AdrInit = off;                          // SW's off
                        AdrStart = off;
                        AdrOut = off;
                        INTCONbits.INT0IF = off;
                        INTCON2bits.INTEDG0 = 1;        // Ext int on adr-in low
                        INTCONbits.INT0IE = on;
                        WriteTimer1 (K_TMR1);           // PID timer
```

```
                        UdId1 = 0;                                      // No Usart user
data
                        UdId2 = 0;                                      // No Usart user
data
                        Udata = 0;
                        PIE1bits.RCIE = 1;                      // Usart int enable
//                      PIE1bits.TXIE = 1;
                        RCIF = off;
                }
        }
}

void Report()
{

}

void    GetPos()
{
    WriteTimer1 (K_TMR1);
        Udata = ReadUSART();
        trace (4);
        if (USART_Status.val & 0x0c)
        {
                UdId1 = 0;
                if (USART_Status.OVERRUN_ERROR)
                {
                        trace(254);
                        CREN = 0;
                        CREN = 1;
                }
                if (USART_Status.FRAME_ERROR)
                        trace(255);
                return;
        }
        if (UdId1)
        {
                if (UdId2)
                {
                        UdBuf[UdCnt] = Udata;
                        UdCnt ++;
                        if(UdId1 == 'A' && UdCnt == 8)
                        {
                                UdId1 = 0;
                                if (Udata = 0x0d)
                                        {
                                                ReqPosD = UdBuf[Adress];
                                                SetDuty();
                                        }
                        }
                        else
                        {
                                if (UdId1 == 'J' && UdCnt == 10)
                                {
                                        UdId1 = 0;
                                        if (Udata = 0x0d)
                                        {
                                                ReqPosD = UdBuf[Adress];
                                                SetDuty();
                                        }
                                }
```

```
                        }
                }
                else
                {
                        if (UdId1 == 'A' && Udata == 'B')
                                UdId2 = Udata;
                        if (UdId1 == 'J' && Udata == 'K')
                                UdId2 = Udata;
                        if (UdId2)
                        {
                                for (i = 0; i < 39; i++)
                                        UdBuf[i] = 0;
                                UdCnt = 0;
                        }
                }
        }
        else
        {
                switch(Udata)
            {
              case 'A':
                                UdId1 = Udata;
                                UdId2 = 0;
                                break;
              case 'J':
                                UdId1 = Udata;
                                UdId2 = 0;
                                break;
                }
        }
}

void SetDuty(void)
{
        PORTB ^= 0x10;  //DEBUG Orange Flip Flop ********************************
//      PORTB |= 0x10;                          //Orange on (initial test)
        TMR1IF=0;           //Reset IRQ Flag
    WriteTimer1 (K_TMR1);
        ADCON0 |= 2;                                    // A/D conversion go
        while( BusyADC() );                         // Wait for completion
        group_a = (((unsigned int)ADRESH)<<8)|(ADRESL); // Read result
        group_b = (((unsigned int)ADRESH)<<8)|(ADRESL); // Read result
        Pos     = (((unsigned int)ADRESH)<<8)|(ADRESL); // Read result
        PI5 = PI4;              // Hist. of Position Input
        PI4 = PI3;
        PI3 = PI2;
        PI2 = PI1;
        PI1 = PI0;
        PI0 = Pos;
        if (Adress)                             // Digital input
        {
                ReqPosL = ReqPosD;
        ReqPosL = ReqPosL * 820;
                ReqPos = ReqPosL >> 8;
        }
        else
        {
                ReqPos  = (((unsigned int)ADRESH)<<8)|(ADRESL); // Read result
        }
        if (First)
        {
```

```
                P5 = P4 = P3 = P2 = P1 = P0 = Pos;
                First = 0;
        }
        RPI5 = RPI4;            // Hist. of Requested Position Input
        RPI4 = RPI3;
        RPI3 = RPI2;
        RPI2 = RPI1;
        RPI1 = RPI0;
        RPI0 = ReqPos;
        ReqPos = (RPI0 + RPI1 + RPI2) / 3;
    if ((Pos > 3) && (ReqPos > 5))              //  Connected and not at home?
                AtLowerEnd = 0;
        else
                AtLowerEnd = 1;
        if (ReqPos > MaxPos)
                ReqPos = MaxPos;                //Stop here
        if (ReqPos < MinPos)
                ReqPos = MinPos;
    if (ReqPos > Pos + DeadBand)                        // Direction?
        {
                Direction = 0;                          // Move up
                Error = ReqPos - Pos;
        }
    else
        {
                if (ReqPos < Pos - DeadBand)
                {
                        Direction = 1;                          // Move down
                        Error = Pos - ReqPos;
                }
                        else
                                Error = 0;
        }
        if (Error > 200)
                Slow = 1;
        P5 = P4;                                        // Hist. of Positions
        P4 = P3;
        P3 = P2;
        P2 = P1;
        P1 = P0;
        P0 = Pos;
        RP1 = RP0;                                              // Hist of Requested Pos.
        RP0 = ReqPos;
        E4 = E3;                                                // Error history entry's
        E3 = E2;
        E2 = E1;
        E1 = E0;
        E0 = Error;
        ErrSum = (E1+E2+E3+E4);
        OldPos = P2;                                    // Set OldPos from a previous Pos.
in Hist.
        if (Direction)
                DeltaPos = OldPos - Pos;        // Derivative (Speed)
        else
                DeltaPos = Pos - OldPos;        // Derivative (Speed)
        DP7 = DP6;                                      // Dela position history
        DP6 = DP5;
        DP5 = DP4;
        DP4 = DP3;
        DP3 = DP2;
        DP2 = DP1;
```

```
        DP1 = DP0;
        DP0 = DeltaPos;
//      DPsum = DP0 + DP1 + DP2 + DP3 + DP4 + DP5;
        DPsum = DP0 + DP1 + DP2;
        if (DPsum < 0)
                DPsum = 0;
//      OldDPsum = DP2 + DP3 + DP4 + DP5 + DP6 + DP7;
        OldDPsum = DP2 + DP3 + DP4;
        if (OldDPsum < 0)
                OldDPsum = 0;
        if (OldDPsum > DPsum && Error > BreakDist)
                DeCel = 1;                              // Deceleration
        if (DPsum > OldDPsum + 2)
                DeCel = 0;                              // Acceleration
        if (RP1 > ReqPos)
        {
                DRP = RP1 - ReqPos;// Delta RequestPosition
                RDir = 1;                               // Request direction
        }
        else
        {
                DRP = ReqPos - RP1;// Delta Request Position
                RDir = 0;                               // Request direction
        }
        if (Reverse || Break)
        {
//              PORTB &= 0xEF;                          //Orange off (test)
                if (Stopping > 0)
                        Stopping -= 10;                 //Stopping delay
                else
                {
                        if (Break)
                        {
                                Break = 0;
                                OVDCOND = (1 << Direction);     // Restore PCPWM direction
(2 | 1)
                        }
                        else if (DP1 < 1 && DeltaPos < 1) // make sure
                        {
                        Reverse = 0;
                        if (Error > 50)
                                Slow = 1;
                        }
                }
        }
        else
        {
                if (PrevDirection == Direction)         // If same direction,
                {
                        if (Error < Small && DeltaPos < 2) // At rest?
                        {
                                Duty = 0;
                                DeltaPos = 0;                                   // Prevent
jitter
                        }
                        else
                        {
                                if (Error > 128)
                                        Error = 128;
                                if (ErrSum > 256)
                                        ErrSum = 256;
```

```
                                TermP = (Error + Crawl) * Kp;           //Scale error
                                TermI = ErrSum * Ki;
                                TermD = DPsum * Kd;
                                TempDuty = TermP + TermI - TermD;
                                if (TempDuty < 0)
                                {
                                        Break = 1;                                      //
Break by reversing lightly
                                        Stopping = 15;
                                        Duty = MinDuty;                         // brake a
bit
                                        //                                    PORTB &= 0xEF;                          // Orange
off(test)
                                        PWMdirSav = OVDCOND;            // Save direction
bit
                                        OVDCOND ^= 3;                           //Other
direction
                                }
                                else
                                {
                                        OVDCOND = (1 << Direction);             // Set
PCPWM direction (2 | 1)
                                        Duty = TempDuty;                // Set normal
duty
                                }
                        }
                        if (Error < DeadBand)
                        {                                                                       //
At position
                                Crawl = 0;
                                DeadBand = Wide;                        // At rest
                                Slow = 0;
                                Duty = 0;
                                DeCel = 0;
                        }
                    else
                        {
                                DeadBand = Small;                       // to exact
position
#ifdef EEPROM
                                if (!MPcount && DeCel && DeltaPos > 1) //Decend slope?
                                {
                                        if (Error > SM)         // Slope Mesure til here
                                                Hist = 1;               //Hist log until
below SM
                                }
                                else
                                        if (DidHist)
                                                DidHist = 0;        // Use as Dubug break
point
#endif
                                if (Error < Notjet && DPsum < 30)
                                {
                                        CrawlDelCnt += 1;
                                        if (CrawlDelCnt > CrawlDelay)
                                        {
                                                CrawlDelCnt = 0;
                                                Crawl += 1;
                                                if (Crawl > MaxCrawl)
                                                        Crawl = MaxCrawl;
                                        }
```

```
                                }
                                else
                                        Crawl = 0;
                        }
                        if ((Error < BreakDist) && DeCel)
                        {
                                if ((DPsum > Speed) && (Error > Wide))          // Moving
speed?
                                {
                                        if (!Break)
                                        {
//                                              PORTB &= 0xEF;                          //
Orange off(test)
                                                Break = 1;                              //
Break
                                                PWMdirSav = OVDCOND;            // Save
direction bit
                                                OVDCOND ^= 3;
//Other direction
                                                Duty = MinDuty;                         //
brake to target
                                        }
                                        DeCel = 0;
                                        Stopping = DP0 * BDmpy;  //Scale stopping delay
                                        if (Stopping > 15)
                                                Stopping = 15;
                                }
                        }
                }
                else                                                            // Now
reverse
                {
                        PrevDirection = Direction;
                        OVDCOND = (1 << Direction);             // Set PCPWM direction (2
| 1)
                        Break = 0;
                        if (DPsum > Fast)                       // Moving fast?
                        {
//                              PORTB &= 0xEF;                          // Orange off(test)
                                Reverse = 1;                            // Break
                                Duty = Minimum;                         //
                                Stopping = DPsum << StopShiftL; //Scale stopping delay
                                if (Stopping > StopTimeMax)
                                        Stopping = StopTimeMax;
                        }
                        else                                                    // Slow
                        {
                                if (Error < DeadBand && !DeltaPos)                      //
At position
                                {
                                        DeadBand = Wide;                        // At rest
                                        Slow = 0;
                                        Duty = 0;
                                }
                                else
                                {
                                        DeadBand = Small;                       // to
exact position
                                }
                        }
                }
```

```
        }
    if (AtLowerEnd)
        {
                if(!RPI0)                               // Disconnected or go home?
                {
                        Slow = 1;
                }
                if (!Pos)                                       // Disconnected?
                                        Duty = 0;                       // Stop
                FFcount = FFcount + 1;                  // Blink delay
                if (FFcount > FFdelay)
                {
                        FFcount = 0;
                        if(!RPI0)                       // Disconnected or go home?
                        {
                                PORTB ^= 0x08;                  //Green FlipFlop
                        }
                        if (!Pos)                               // Disconnected?
                                PORTB ^= 0x04;                  //Red FlipFlop
                        else
                        {
                                if (Duty)                               // Running?
                                        PORTB |= 0x04;          //Red on
                                else
                                        PORTB &= 0xFB;          //Red off
                        }
                }
        }
        else    // Not at end
        {
                if (Duty)                                               // Running?
                {
                        PORTB &= 0xF7;                          //Green off
                        PORTB |= 0x04;                          //Red on
                }
                else                                                    // Stopped
                {
                        PORTB |= 0x08;                          //Green on,
                        PORTB &= 0xFB;                          //Red off
                }
        }
        if (Slow)
                MaxDuty = SlowRun;
        else
                MaxDuty = MaxRun;                       //Run
    if (Duty > MaxDuty)                                 //Limit speed
                        Duty = MaxDuty;
#ifdef JAM
        if (DP1 > 1 && DP2 > 1)                                                 // Moving?
                {
                        Doncount = 0;
                        Doffcount = 0;
                        Doffdelay = Doffdelay1;
                        if (Slow)
                        {
                                Slow = 0;
                                MaxDuty = MaxRun;
                        }
                }
                else                                                            // Not moving
                {
```

```
                        if (Duty > MustRun)                     // Shoud it?
                        {
                                if (Doffcount)                          // Cooling down
period?
                                {
                                        PORTB &= 0xFB;                  //Red off
                                        Duty = 0;                               // Stop
                                        Reverse = 0;
                                        Break = 0;
                                        Doffcount += 1;
                                        if (Doffcount > Doffdelay)  // Retry?
                                        {
                                                if (Doffdelay < 32768)
                                                {
                                                        Doffdelay <<= 2;        //
Increase delay next time
                                                        Doffcount = 0;                  //
retry to get moving
                                                }
                                        }
                                }
                                else
                                {                                                               //
after cool and wait
                                        if (Doncount)                           // waiting
to move?
                                        {
                                                Doncount += 1;                  // Yes
                                                if (Doncount > Dondelay) // Tired waiting?
                                                {
                                                        Doncount = 0;
                                                        Doffcount = 1;                  //
cool down
                                                }
                                        }
                                        else
                                        {
                                                Doncount = 1;                   // now
wait to move
                                                Doffcount = 0;
                                                if (Error > 50)
                                                        Slow = 1;
                                        }
                                }
                        }
                        else
                        {
                                Doncount = 0;
                                Doffcount = 0;
                        Doffdelay = Doffdelay1;
                        }
                }
#endif
        PORTB |= 0x10;                          //Orange on (initial test)
        Setdc0pcpwm (Duty);                                     //Set new PCPWM duty
}  // End SetDuty.....................
```