Servo Control of a DC-Brush Motor

Author: Tim Bucella, Teknic Inc.

INTRODUCTION

The PIC17C42 microcontroller is an excellent choice for cost-effective servo control in embedded applications. Due to its Harvard architecture and RISC-like features, the PIC17C42 offers excellent computation speed needed for real time closed loop servo control. This application note examines the use of the PIC17C42 as a DC brush motor servo controller. It is shown that a PID (Proportional, Integral, Differential) control calculation can be performed in less than 200 µS (@16 MHz) allowing control loop sample times in the 2 KHz range. Encoder rates up to 3 MHz are easily handled by the PIC17C42's high speed peripherals. Further, the on-chip peripherals of the PIC17C42 allow an absolute minimum cost system to be constructed.

Closed-loop servo motor control is usually handled by 16-bit, high-end microcontrollers and external logic. In an attempt to increase performance many applications are upgrading to DSPs. However, the very high performance of the PIC17C42 makes it possible to implement these servo control applications at a significant reduction in overall system cost.

The servo system uses a PIC17C42 microcontroller, a programmable logic device (PLD), and a single-chip H-bridge driver. Such a system might be used as a positioning controller in a printer, plotter, or scanner. The low cost of implementing a servo control system using the PIC17C42 allows this system to compete favorably with stepper motor systems offering a number of advantages:

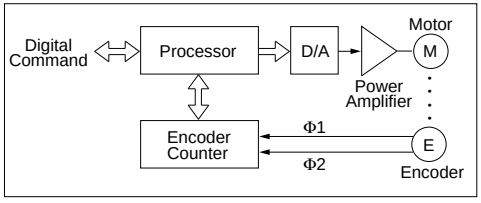
- Increased Acceleration, Velocity
- Improved Efficiency
- Reduced Audible Noise
- True Disturbance Rejection

SYSTEM OVERVIEW

DC Servo Control

Modern digital servo systems are formed as shown in Figure 1. These systems control a motor with an incremental feedback device known as a sequential encoder. They consist of an encoder counter, a processor, some form of digital-to-analog converter, and a power amplifier, which delivers current or voltage to the motor.

FIGURE 1 - A TYPICAL SERVO SYSTEM



The PIC17C42 implements both the servo compensator algorithm and the trajectory profile (trapezoidal) generation. A trajectory generation algorithm is necessary for optimum motion and its implementation is as important as the servo compensator itself. The servo compensator can be implemented as a traditional digital filter, a fuzzy logic algorithm, or the simple PID algorithm (implemented in this application note). The combination of servo compensator and trajectory calculations can place significant demands on the processor.

The digital-to-analog conversion can be handled by a conventional DAC or by using pulse-width modulation (PWM). In either case the output signal is fed to a power stage which translates the analog signal(s) into usable voltages and currents to drive the motor.

PWM output can be a duty-cycle signal in combination with a direction signal or a single signal which carries both pieces of information. In the latter case a 50% duty cycle commands a null output, a 0% duty cycle commands maximum negative output, and 100% maximum positive output.

The amplifier can be configured to supply a controlled voltage or current to the motor. Most embedded systems use voltage output because of its simplicity and reduced cost.

Sequential encoders produce quadrature pulse trains, from which position, speed, and direction of the motor rotation can be derived. The frequency is proportional to speed and each transition of Φ1 and Φ2 represents an increment of position. The phase of the signals is used to determine direction of rotation.

FIGURE 2 - THE PIC17C42 SERVO SYSTEM

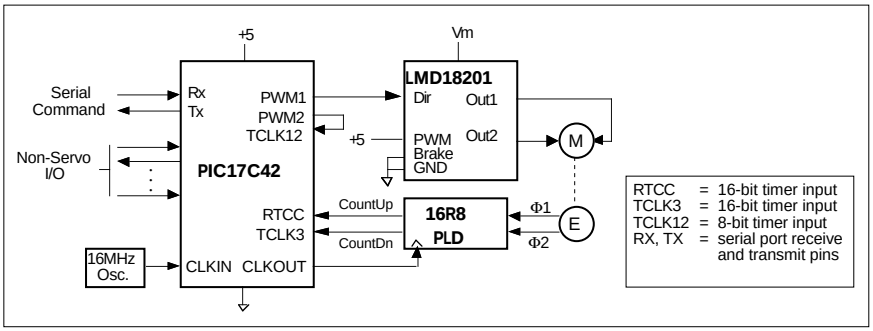
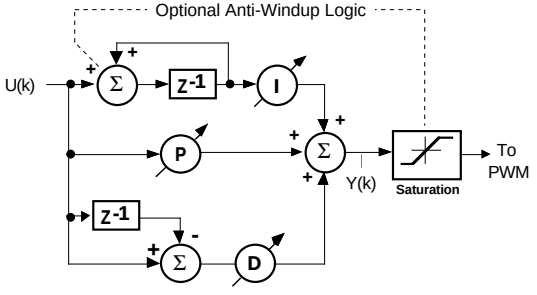


FIGURE 3 - THE PIC17C42 BASED SERVO CONTROL BOARD

Place Servo Control Board photo here.

FIGURE 4 - DIGITAL PID IMPLEMENTATION



Note: The Z-1 operator indicates a one sample time delay

These encoder signals are usually decoded using a small state machine into Count Up and Count Down pulses. These pulses are then routed to an N-bit, up-down counter whose value corresponds to the position of the motor shaft. The decoder/counter may be implemented in hardware, software, or a combination of the two.

The PIC17C42 Based Motor Control Board

The PIC17C42 based servo system described here has a full RS-232 ASCII interface, on-board switching power supply, H-bridge motor drive, over-current protection, limit switch inputs and digital I/O. The entire system measures 5" x 3.5" and is shown in Figure 3. The system can be used to evaluate the PIC17C42 in servo applications. All unused PIC17C42 pins are available at an I/O connector for prototyping.

A PID algorithm is used as a servo compensator and position trajectories are derived from linear velocity ramp segments. This system uses 50%-null PWM as the digital-to-analog conversion technique. The power stage is a high current output switching stage which steps-up the level of the PWM signal. Encoder signal decoding is accomplished using an external PLD. The up/down counter is implemented internally in the PIC17C42 as combination of hardware and software (Figures 5 and 6).

THE COMPENSATOR

PID is the most widely used algorithm for servo motor control. Although it may not be the most optimum controller for all applications, however it is easy to understand and tune.

The standard digital PID algorithm's form is shown in Figure 4. U(k) is the position or velocity error and Y(k) is the output.

This algorithm has been implemented using the PIC17C42 math library. Only 800 instruction cycles are required resulting in a 0.2mS PID execution time at 16 MHz.

Integrator wind-up is a condition which occurs in PID controllers when a large following error is present in the system, for instance when a large step disturbance is encountered. The integrator continually builds up during this following error condition even though the output is saturated. The integrator then "unwinds" when the servo system reaches its final destination causing excessive oscillation. The PID implementation shown above avoids this problem by stopping the action of the integrator during output saturation.

MOTOR ACTUATION

The PIC17C42 contains a high-resolution pulse width modulation (PWM) subsystem. This forms a very efficient power D/A converter when coupled to a simple switching power stage. The resolution of the PIC17C42 PWM subsystem is 62.5nS (at 16 MHz). This translates into 10-bit resolution at a 15.6KHz rate or 1 part in 800 (9 1/2-bit) resolution at 20KHz. This allows effective voltage control while still maintaining the modulation frequency at or above the limit of human hearing. This is especially relevant in office automation equipment where minimizing noise is a design goal.

The motor responds to a PWM output stage by time averaging the duty cycle of the output. Most motors react slowly, having an electrical time constant of 0.5mS or more and a mechanical time constant of 20.0mS or more. A 15KHz PWM output is effectively equivalent to that of a linear amplifier.

In the system shown in Figure 2, the H-bridge's direction input is wired directly to the PIC17C42's PWM output. The H-bridge is powered by a DC supply voltage, V_m. In this configuration 0 volts is presented to the motor when the PWM signal is at a 50% duty cycle, $-V_m$ volts at 0% duty cycle and $+V_m$ volts at 100% duty cycle.

ENCODER FEEDBACK

Position feedback for the example system is derived from a quadrature encoder mounted on the motor shaft. Both incremental position and direction can be derived from this inexpensive device. The quadrature encoder signals are processed by a 16R8-type PLD device as shown in Figure 2. The PLD converts the quadrature pulses into two pulse streams: Count Up and Count Down (Figure 5). These signals are then fed to two 16-bit timers of the PIC17C42 (TMR3 and RTCC). A logic description for the PLD decoder is shown in Appendix B.

The PIC17C42 keeps track of the motor shaft's incremental position by differencing these two 16-bit timers. This operation is performed each servo sample time and the current position is calculated by adding the incremental position to the previous position. Since both timers are 16-bits deep, keeping track of the overflow is unnecessary, unless the encoder signals frequency is greater than 32767 times the sample frequency. **For example, at a servo sample time of 1mS, the maximum encoder rate would be 3.2767 MHz.**

Counter wrap-around is not a concern because only the *difference* between the two counters is used. Two's-complement subtraction takes care of this automatically. Position is maintained as a three-byte, 24-bit quantity in the example program shown in Appendix F. However, there is no limit to the size of the internal position register. By adding the 16-bit incremental position each sample time to an N-byte software register, an N-byte position may be maintained.

FIGURE 5 - SEQUENTIAL ENCODER SIGNALS

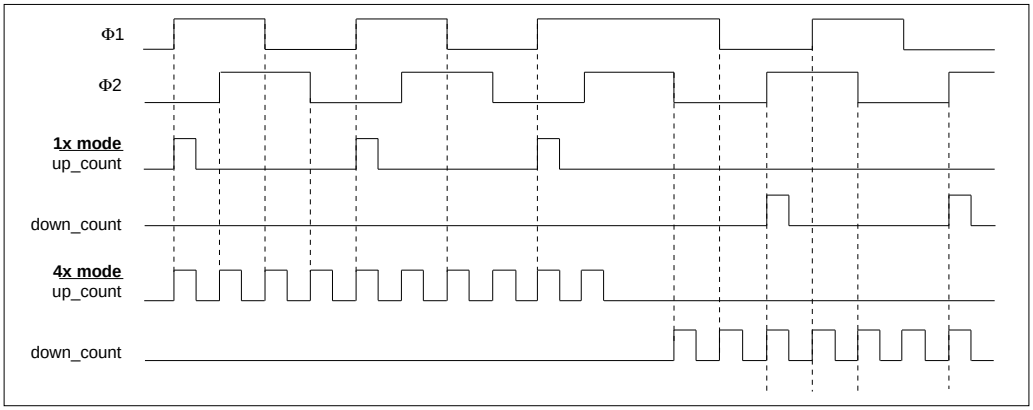
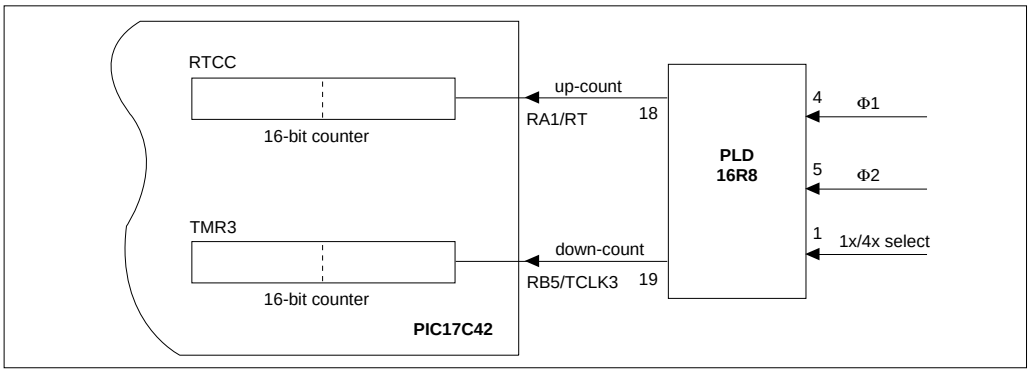


FIGURE 6 - ENCODER INTERFACE SCHEME



TRAJECTORY GENERATION

A trajectory generation algorithm is essential for optimum motion control. A linear piecewise velocity trajectory is implemented in this application. For a position move, the velocity is incremented by a constant acceleration value until a specified maximum velocity is reached. The maximum velocity is maintained for a required amount of time and then decremented by the same acceleration (deceleration) value until zero velocity is attained. The velocity trajectory is therefore trapezoidal for a long move and triangular short move where maximum velocity was not reached (Figure 8).

The doPreMove subroutine is invoked once at the beginning of a move to calculate the trajectory limits. The doMove routine is then invoked at every sample time to calculate new "desired" velocity and position values as follows:

$V_K = V_{K-1} + A$ (A = Acceleration)

$P_K = P_{K-1} + V_{K-1} + A/2$

For more details on trajectory generation, see Appendix E.

IMPLEMENTATION DETAILS

The program structure is straightforward: An interrupt service routine (ISR) processes the servo control and trajectory generation calculations, and a foreground loop is used to implement the user interface, serial communication and any exception processing (i.e. limit switches, watchdog timer, etc.).

The ISR has a simple structure. In order to effect servo control we need to read the encoder, calculate the new trajectory point and PID values, and set the output of the PWM, all at a constant, predefined rate. The ISR is initiated by a hardware timer (TMR2) on the PIC17C42. To make sure that the servo calculation always occurs synchronously with the PWM subsystem, the PWM2 output is wired to the input pin of TMR12 (TMR1 in internally-clocked, 8-bit timer mode; TMR2 in externally-clocked, 8-bit counter mode). N is loaded into the PR2 register. The sample rate then becomes the PWM rate divided by N. In this implementation N=16 (Figure 7).

FIGURE 7 - SAMPLING SCHEME

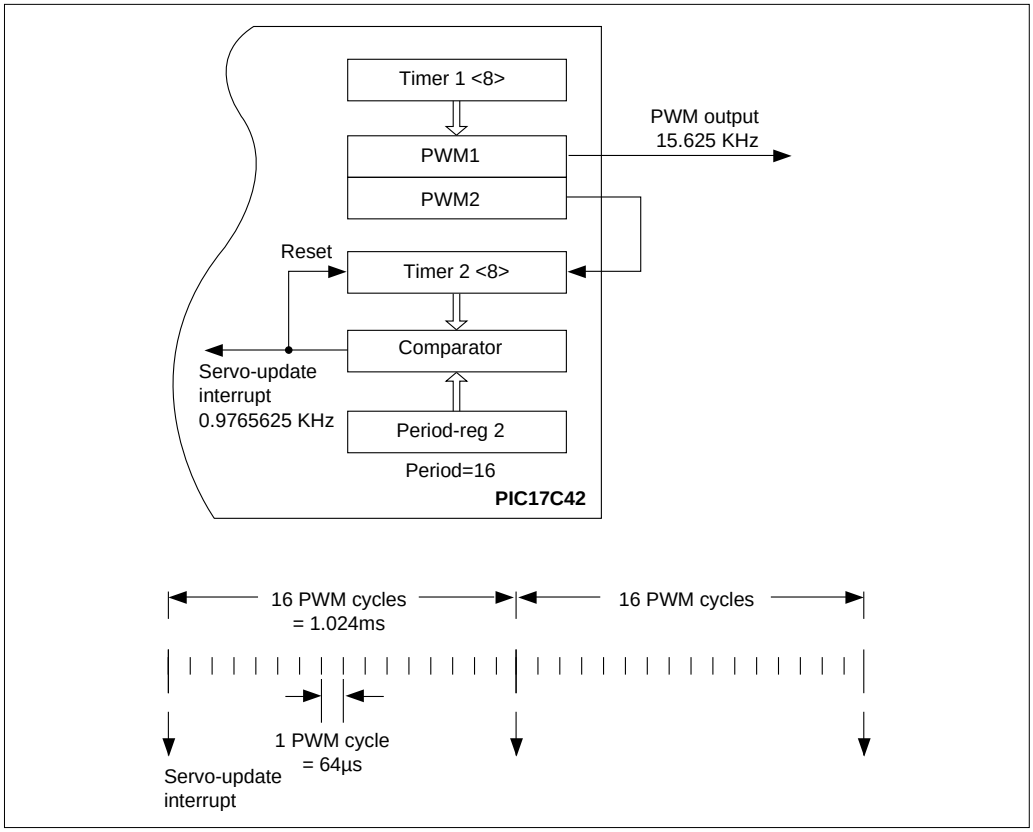


FIGURE 8 - VELOCITY RAMP SEGMENTS FOR POSITION MOVES

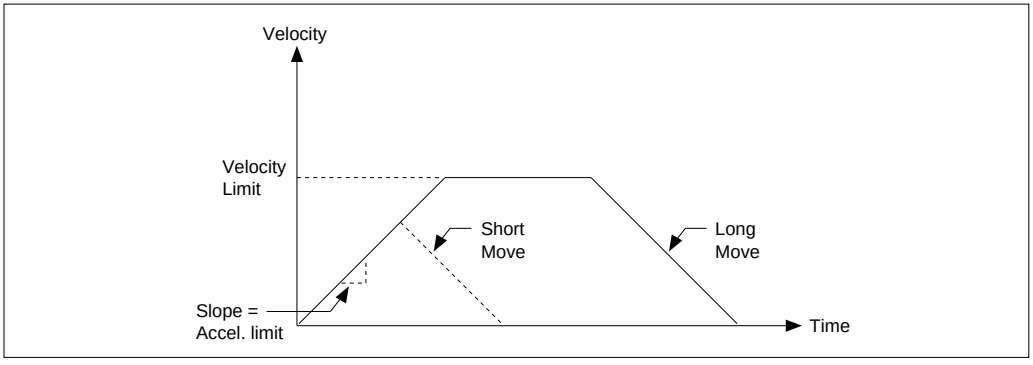


FIGURE 9 - FLOWCHART FOR FORE-GROUND PROCESSING

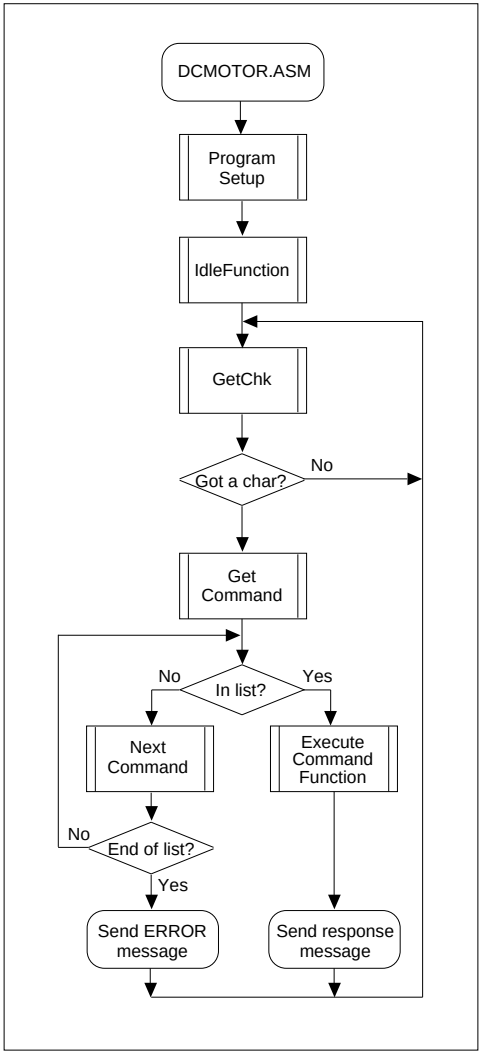
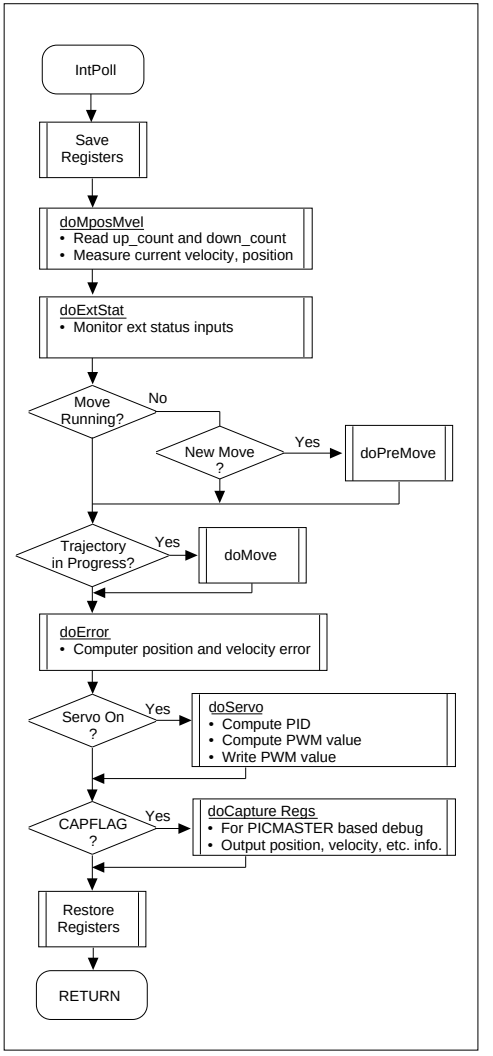


FIGURE 10 - FLOWCHART FOR INTERRUPT SERVICE ROUTINE



The following events must occur in the interrupt service routine:

- Read Timers (RTCC & TMR3)
- Calculate the new Reference Position using the Trajectory Generation Routine.
- Calculate Error: U(k) = Reference Position - Current Position
- Calculate Y(k) using PID
- Set PWM output
- Manage other housekeeping tasks (i.e. service serial characters)

The entire ISR requires only 0.250mS to execute with 16MHz processor clock frequency.

COMMAND INTERFACE

The following commands are implemented and recognized by the user interface in the foreground loop.

Move (Value): M, [$-8,388,608_{10}$ to $8,388,607_{10}$]

Commands the axis to move to a new position or velocity. Position data is relative, velocity data is absolute. Position data is in encoder counts. Velocity data is given in encoder counts per sample time multiplied by 256. All moves are performed by the controller such that velocity and acceleration limits set into parameter memory will not be violated.

All move commands are kept in a one deep FIFO buffer. The command in the buffer is executed as soon as the executing command is complete. If no move is currently executing the commanded move will start immediately.

Mode: O, (Type), [P,V, T]

An argument of “P” will cause all subsequent move commands to be incremental position moves. A “V” argument will cause all subsequent moves to be absolute velocity moves. A "T" argument sets a "Torque mode" where all subsequent M commands directly write to the PWM. This is useful for debug purposes.

Set Parameter: S, (#,Value) [00_h to FF_h, $-8,388,608_{10}$ to $8,388,607_{10}$]

Sets controller parameters to the value given. Parameters are shown in Table 1.

TABLE 1 - PARAMETERS

Parameter	$\#_h$	Range
Velocity Limit	00	0 to $8,388,607_{10}$ *
Acceleration Limit	01	0 to $8,388,607_{10}$ **
Kp: Proportional Gain	02	-32768_{10} to 32767_{10}
Kd: Differential Gain	03	-32768_{10} to 32767_{10}
Ki: Integral Gain	04	-32768_{10} to 32767_{10}

* (counts per sample time multiplied by 256)

** (counts per sample time per sample time multiplied by 256)

Read Parameter: R, (#) [00_h to FF_h]

Returns the present value of a parameter.

Shutter: C

Returns the time (in sample time counts 0 to $65,536_{10}$) since the start of the present move and captures the commanded and actual values of position and velocity at the time of the command.

Read commanded position: P

Returns the commanded position count which was captured during the last Shutter command.
Range: $-8,388,608_{10}$ to $8,388,607_{10}$.

Read commanded velocity: V

Returns the commanded velocity multiplied by 256 which was captured during the last Shutter command.
Range: $-8,388,608_{10}$ to $8,388,607_{10}$

Read actual position: p

Returns the actual position count which was captured during the last Shutter command.
Range: $-8,388,608_{10}$ to $8,388,607_{10}$.

Read actual velocity: v

Returns the actual velocity multiplied by 256 which was captured during the last Shutter command.
Range: $-8,388,608_{10}$ to $8,388,607_{10}$.

External Status: X

Returns a two digit hex number which defines the state of the bits in the external status register. Issuing this command will clear all the bits in the external status register unless the event which set the bit is still true. The bits are defined in Table 2.

TABLE 2 - EXTERNAL STATUS REGISTER BITS

Bit 7	index marker detected
Bit 6	+limit reached
Bit 5	-limit reached
Bit 4	input true
Bit 3-0	n/a

Move Status: Y

Returns a two-digit hex number which defines the state of the bits in the move status register. Issuing this command will clear all the bits in the move status register unless the event which set the bit is still true. The bits are defined in Table 3.

TABLE 3 - MOVE STATUS REGISTER BITS

Bit 7	move buffer empty
Bit 6	move complete
Bit 5-0	n/a

Read Index position: I

Returns the last index position captured in position counts.

Set Position (Value): H, [-8,388,608$_{10}$ to 8,388,607$_{10}$]

Sets the actual and commanded positions to the value given. Should not be sent unless the move FIFO buffer is empty.

Reset: Z

Performs a software reset.

Capture Servo-Response: c (#Count)

The c command will set a flag inside indicating that starting with the next M (servo move) command, velocity and position information will be sent out (by invoking the doCaptureRegs procedure) during every servo-loop for #count times. At the end of the #count, the processor will halt (see doCaptureRegs procedure). This is useful for debug purposes.

Disable Servo: s

This command disables servo actuation. The servo will activate again with the execution of the next M (move) command. This is useful for debug purposes.

Examples:

```
Z           ;Reset software (No <CR> required)
OV          ;Set velocity servo mode (No <CR>
                    ;required)
M 1000<CR>  ;Set velocity to 1000
M-1000<CR>  ;Set velocity to 1000 in reverse
                    ;direction
```

OPTIMIZING THE SYSTEM

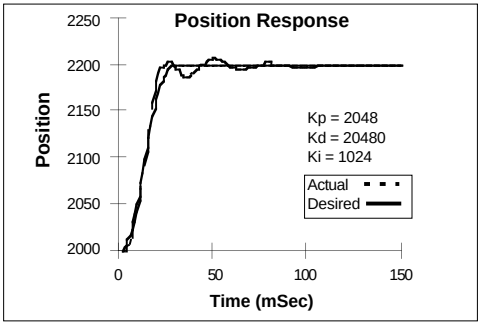
Once the PID loop is successfully implemented, the next challenge is to tune it. This was made simple through extensive use of the PICMASTER™ In-Circuit Emulator for the PIC17C42.

The PICMASTER is a highly sophisticated real-time in-circuit emulator with unlimited break-point capability, 8K deep trace buffer and external logic probes. It's user interface software runs under Windows™ 3.1 with pull-down menus and on-line help. The PICMASTER software also support dynamic data exchange (DDE) through which it is possible to send its trace buffer information to a spreadsheet, such as EXCEL™, also running under windows.

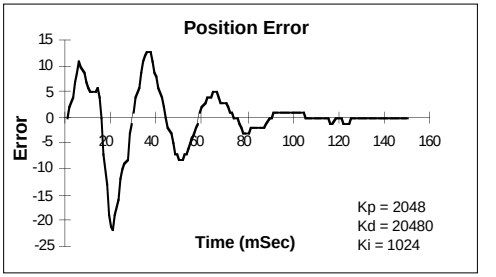
To tune the PID, first a small amount of diagnostics code is added in the servo routine (doCaptureRegs). This code simply outputs, at every sample point, the actual and desired position values, actual and desired velocity values, position error and velocity error by using TABLWT instruction. These are captured in the trace buffer of the emulator. The 'trace' condition is set up to only trace the data cycles of the 2-cycle TABLWT instructions. Next, the trace buffer is transferred to EXCEL and the various

FIGURE 11 - TYPICAL SERVO RESPONSE:

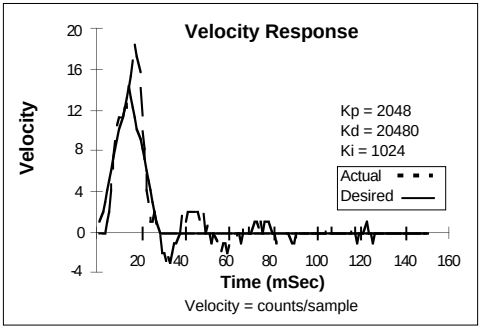
DESIRED/ACTUAL POSITION



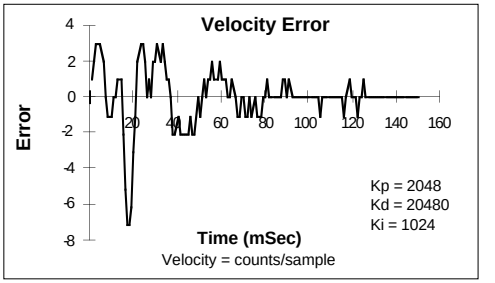
POSITION ERROR



DESIRED/ACTUAL VELOCITY



VELOCITY ERROR



parameters are plotted. The plots graphically show the amounts of overshoot, ripple and response time. By altering Kp, Ki and Kd, and plotting the results, the system can be fine tuned.

Under windows multitasking environment, using PICMASTER emulator this can be done in real time as described below.

Three sessions are set up under windows:

1. A terminal emulator session to send commands to the motor control board. The "terminal" program provided with windows is used, although any communications software such as PROCOMM will work.
2. Second, a PICMASTER emulation session is invoked. The actual PIC17C42 is replaced in-circuit by the emulator probe. Within the emulator, trace points are setup to capture the actual and desired position and velocity values on appropriate bus cycles.
3. Third, a session of EXCEL is started and dynamically linked to the PICMASTER sessions such that whenever the trace buffer is full, the data is sent over to EXCEL. A few simple filtering commands in EXCEL are used to separate the various data types, i.e. actual position data from desired position from actual velocity etc. Next, various plot windows are set up within EXCEL to plot these information.

Once these setup have been done, for every servo move, the responses are automatically plotted. It is then a simple matter of varying the PID coefficients and observing the responses to achieve the desired system response. At any point, the responses can be stored in files and/or printed out.

Except for very long "move" commands, most position and velocity commands are executed (i.e. system settled) in less than 500 samples, making it possible to capture all variables (actual and desired position and velocity, and position errors and servo output) in PICMASTER's 8K trace buffer.

CONCLUSIONS

Using a high-performance 8-bit microcontroller as the heart of a servo control system is a cost-effective solution which requires very few external components. A comparison with a popular dedicated servo-control chip, is presented in Table 4.

TABLE 4 - SERVO CONTROL CHIP COMPARISON

	LM629 @8MHz	PIC17C42 @16MHz	PIC17C42 @25MHz
Max Encoder Rate	1MHz	3.3 MHz	4.5 MHz
Servo Update Time	-	0.25 ms	0.16 ms
Max Sampling Frequency	4 KHz	2-3 KHz	4-5 KHz

Also apparent in the comparison table is the additional processing power available when using the microcontroller. This processing can be used to provide a user interface, handle other I/O, etc. Alternatively, the additional processing time might be used to improve the performance of compensator and trajectory generation algorithms. A further advantage is that for many embedded applications using motor control the microcontroller proves to be a complete, minimum cost solution.

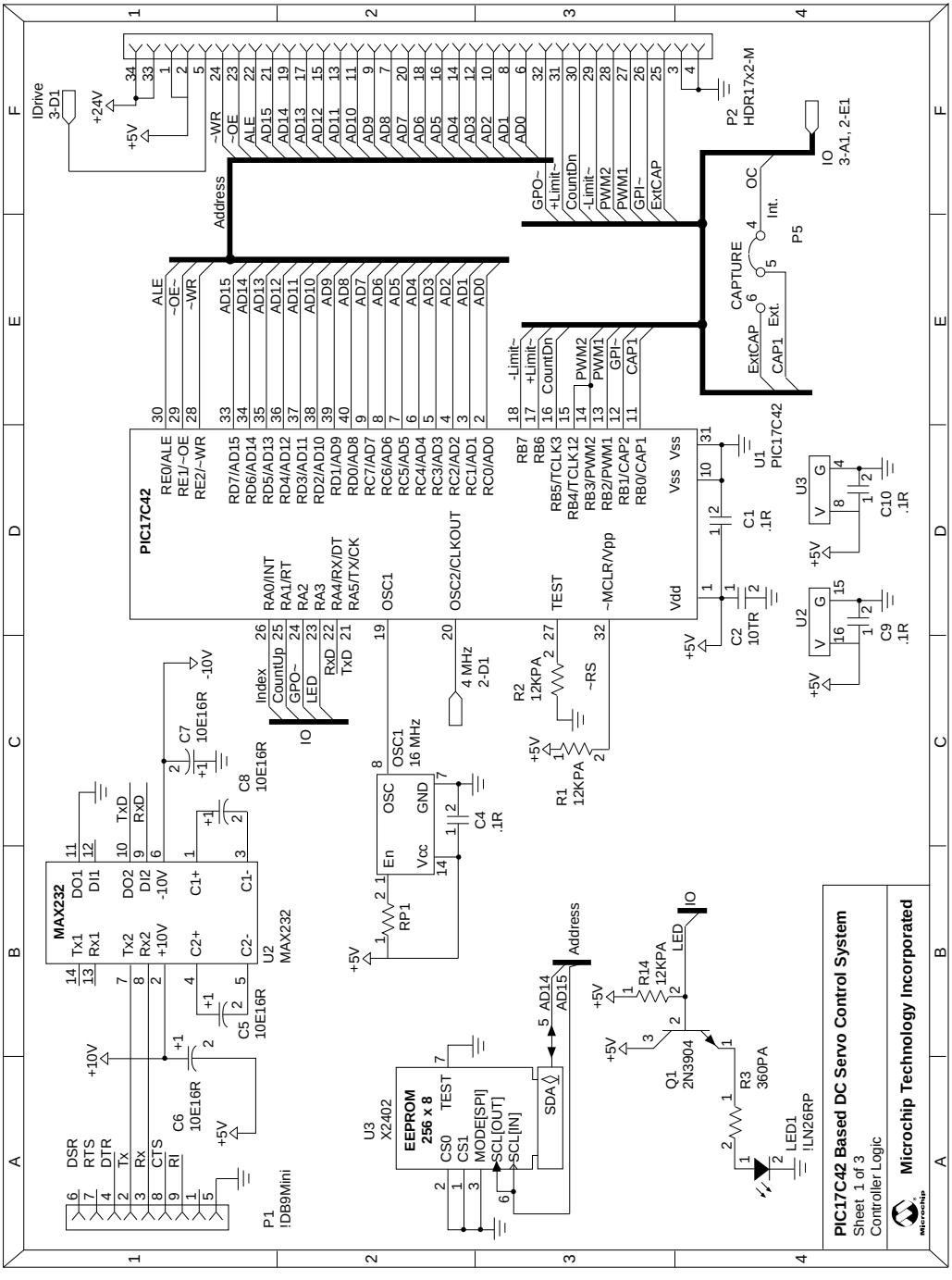
Credit

This application note and a working demo board has been developed by Teknic Inc. Teknic (Rochester, N.Y.) specializes in Motor Control Systems.

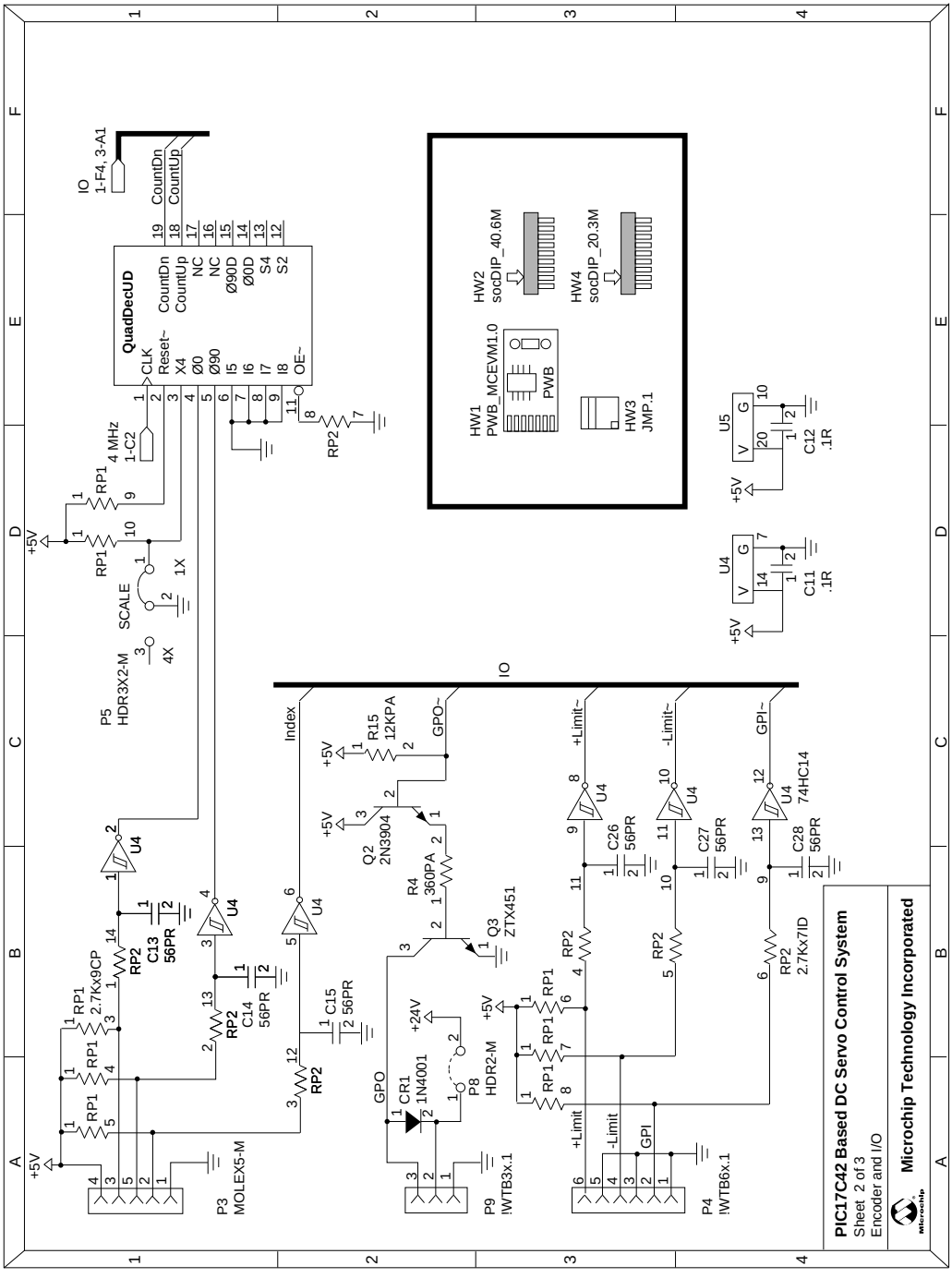
References

1. Thomas Bucella, "Comparing DSPs to Microprocessors in Motion Control Systems-Some Real World Data", PCIM conference proceedings © 1990 Intertec Communications, Inc.
2. David M. Auslander, Cheng H. Tham, "Real-Time Software for Control" © 1990 Prentice-Hall, Inc., Englewood Cliffs, NJ
3. "DC Motors, Speed Controls, Servo Systems" Fifth Edition © 1980 Electro-Craft Corporation, Hopkins, MN

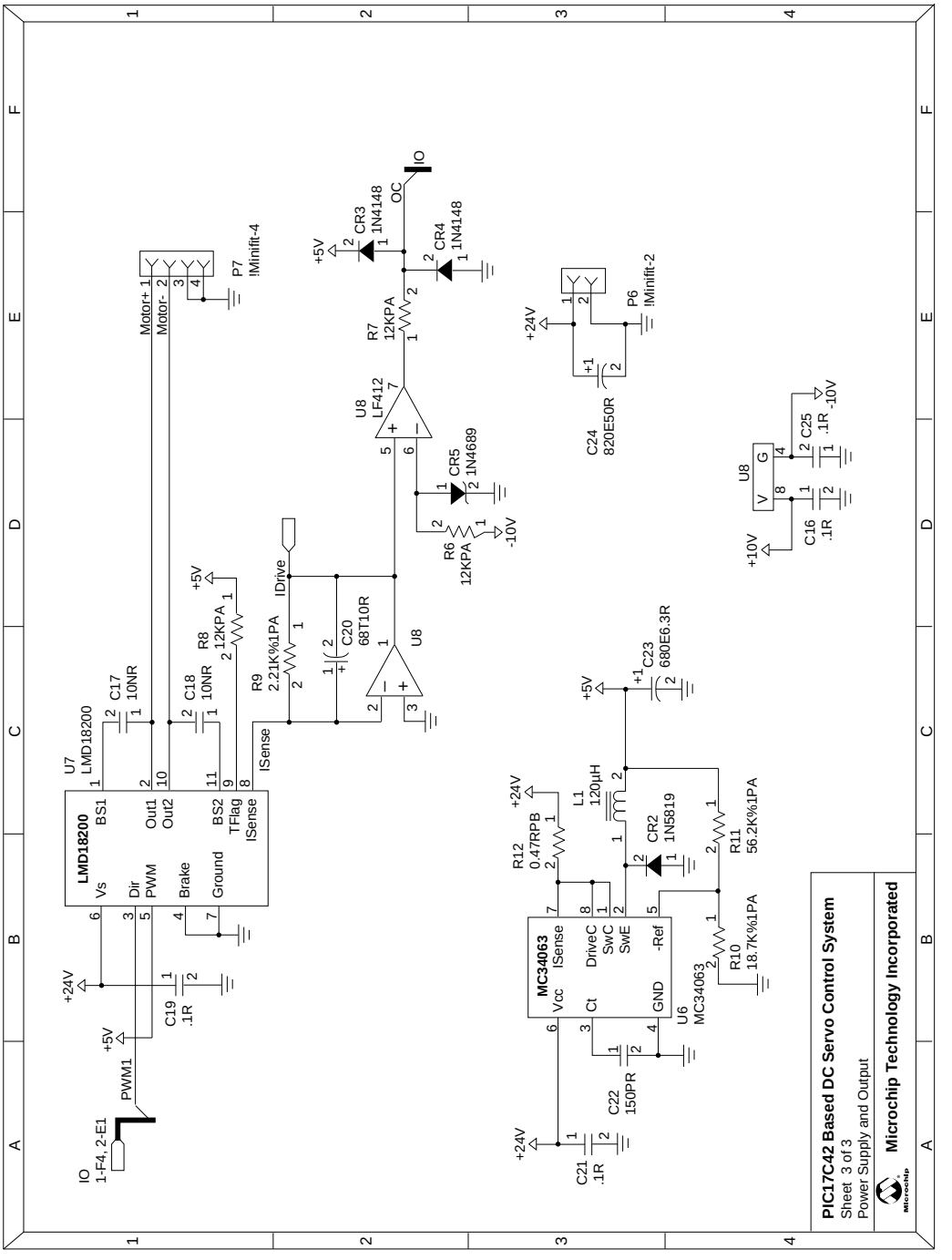
APPENDIX A: SCHEMATIC DIAGRAM



APPENDIX A (CONT.): SCHEMATIC DIAGRAM



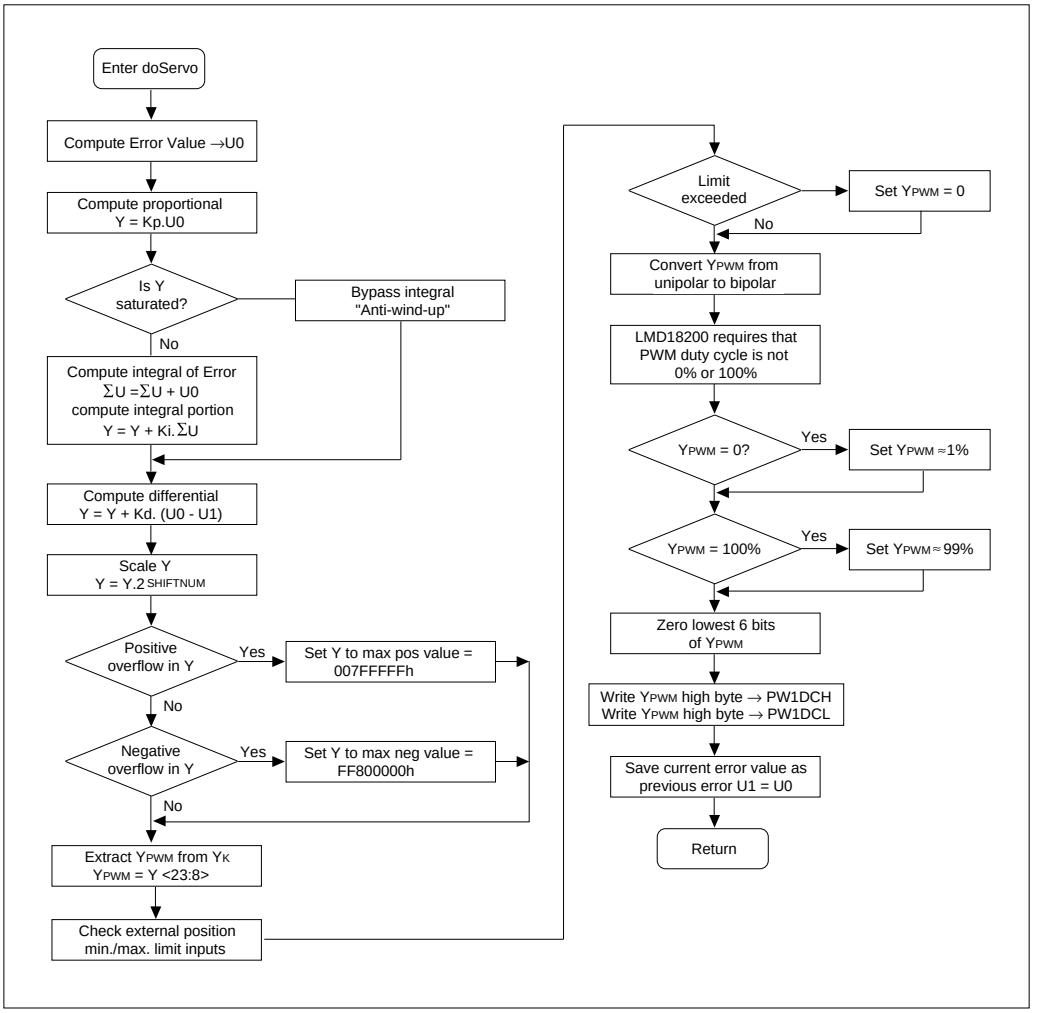
APPENDIX A (CONT.): SCHEMATIC DIAGRAM



APPENDIX B: ENCODER PLD EQUATIONS

```
Combination quadrature decoder and input synchronizer.  This design
allows 1x decoding or 4x decoding based on the X4 pin.

       * Ver 1.0 - November 8, 1991
}
MODULE QuadDivider;
TITLE QuadDivider V1.0;
COMMENT Device: 16R8;

TYPE MMI 16R8;
INPUTS;
       RESET NODE[PIN2] INVERTED;
       X4 NODE[PIN3];
       P0 NODE[PIN4];                { Phi0 }
       P90 NODE[PIN5];                       { Phi90 }
       INDX NODE[PIN6];
       { Feedback pins }
       S2 NODE[PIN12];
       S4 NODE[PIN13];
       P0D NODE[PIN14];
       P90D NODE[PIN15];
       CntUp NODE[PIN18];
       CntDn NODE[PIN19];
       UP NODE[PIN16];
       COUNT NODE[PIN17] INVERTED;
OUTPUTS;
       S2 NODE[PIN12];
       S4 NODE[PIN13];
       P0D NODE[PIN14];
       P90D NODE[PIN15];
       CntUp NODE[PIN18];
       CntDn NODE[PIN19];
       UP NODE[PIN16];
       COUNT NODE[PIN17] INVERTED;
TABLE;
       S2 := P0D & !RESET;
       S4 := P90D & !RESET;
       P0D := P0 & !RESET;
       P90D := P90 & !RESET;

       CntUp := COUNT & UP;
       CntDn := COUNT & !UP;

       COUNT :=
             (  P0D &  S2 & !P90D &  S4 & X4 { C1 }
              +!P0D & !S2 &  P90D & !S4      { C2 }
              +!P0D &  S2 & !P90D & !S4 & X4 { C3 }
              + P0D & !S2 &  P90D &  S4 & X4 { C4 }
              + P0D &  S2 &  P90D & !S4 & X4 { C5 }
              +!P0D & !S2 & !P90D &  S4      { C6 }
              +!P0D &  S2 &  P90D &  S4 & X4 { C7 }
              + P0D & !S2 & !P90D & !S4 & X4 { C8 }
              ) & !RESET;
       UP :=
              (
               !P0D &  S2 & !P90D &  S4
              +!P0D &  S2 &  P90D &  S4
              +!P0D &  S2 &  P90D & !S4
              + P0D &  S2 &  P90D & !S4
              + P0D & !S2 &  P90D & !S4
              + P0D & !S2 & !P90D & !S4
              + P0D & !S2 & !P90D &  S4
              +!P0D & !S2 & !P90D &  S4
              ) & !RESET;
END;

END QuadDivider;
```

APPENDIX C: (PART 1): PID ALGORITHM FLOWCHART



APPENDIX C: (PART 2): PID ALGORITHM CODE LISTING

```
;*************************************************************************
; NAME:        doServo
;
; DESCRIPTION: Performs the servo loop calculations.
;
doServo

        MOV16   POSERROR,U0           ; save new position error in U0

        LOADAB  U0,KP                   ; compute KP*U0                          Basic PID
        CALL    Dmult                                                            calculation
        MVPF32  DPX,Y                   ; Y=KP*U0

        CLRF    WREG                    ; if previous output saturated, do anti-
        CPFSGT  SATFLAG                 ; not accumulate integrator       wind-
        CALL    doIntegral                                                up

        LOADAB  INTEGRAL,KI             ; compute KI*INTEGRAL
        CALL    Dmult
        ADD32   DPX,Y                   ; Y=KP*U0+KI*INTEGRAL

        MVFP16  U0,AARG                 ; compute KV*(U0-U1)
        SUB16   U1,AARG
        MVFP16  KV,BARG
        CALL    Dmult
        ADD32   DPX,Y                   ; Y=KP*U0+KI*INTEGRAL+KV*(U0-U1)

        CLRF    WREG                                                             Scale Y
        CPFSGT  SHIFTNUM                    ; scale Y by SHIFTNUM
        GOTO    grabok                ; Y = Y * (2**SHIFTNUM)
        MOVFP   SHIFTNUM,TMP
grabloop
        RLC32   Y
        DECFSZ  TMP
        GOTO    grabloop

grabok
        CLRF    SATFLAG
        BTFSC   Y+B3,MSB                ; saturate to middle 16 bits,
        GOTO    negs                    ; keeping top 10 bits for PW1DCH
poss                                     ; and PW1DCL
        MOVFP   Y+B2,WREG               ; check if Y >= 2**23                    If positive
        ANDLW   0x80                                                             overflow, saturate
        IORWF   Y+B3                                                             y to maximum
        CLRF    WREG                                                             positive number
        CPFSGT  Y+B3
        GOTO    zero6bits               ; if not, zero 6 bits

        INCF    SATFLAG                 ; if so, set Y=0x007FFFFF
        CLRF    Y+B3                    ; clear for debug purposes
        MOVLW   0x7F
        MOVPF   WREG,Y+B2
        SETF    Y+B1
        SETF    Y+B0
        GOTO    zero6bits
negs
        MOVFP   Y+B2,WREG               ; check if Y <= -2**23                   If negative
        IORLW   0x7F                                                             overflow, saturate
        ANDWF   Y+B3                                                             y to maximum
        SETF    WREG                                                             negative value
        CPFSLT  Y+B3
        GOTO    zero6bits               ; if not, zero 6 bits

        SETF    SATFLAG                 ; if so, set Y = 0xFF800000
        SETF    Y+B3
        CLRF    Y+B2
        BSF     Y+B2,MSB
        CLRF    Y+B1
        CLRF    Y+B0
```

```
zero6bits
        MOV24   Y+B1,YPWM+B0            ; move Y to YPWM and zero 6 bits
doTorque                                 ; entry point for torque mode
        MOVLW   0xC0
        ANDWF   YPWM+B0

        BTFSC   YPWM+B1,MSB
        GOTO    tmlimit
tplimit                                                                  If external
        BTFSS   EXTSTAT,BIT6                                             position
        GOTO    mplimitok                                                limits have been
        CLR32   YPWM                                                     reached then zero
        GOTO    mplimitok                                                PWM output.
tmlimit
        BTFSS   EXTSTAT,BIT5
        GOTO    mplimitok
        CLR32   YPWM
mplimitok                                                                Convert PWM from
        MOVLW   PW1DCH_INIT             ; adjustment from bipolar to unipolar  unipolar to bipolar
        MOVPF   WREG,TMP+B1             ; for 50% duty cycle
        MOVLW   PW1DCL_INIT
        MOVPF   WREG,TMP+B0
        ADD16   TMP,YPWM

        CLRF    TMP+B1                  ; correct by 1 LSB
        MOVLW   0x40                    ; add one to bit5 of PW1DCL
        MOVPF   WREG,TMP+B0
        ADD16   TMP,YPWM
                                                                         PWM cycle must not
testmax                                                                  be 0% of 100%
        CLRF    TMP+B2                  ; check pwm maximum limit
        CLRF    YPWM+B2                 ; LMD18200 must have a minimum pulse
        CLRF    YPWM+B3                 ; so duty cycle must not be 0 or 100%
        MVFP16  YPWMAX,TMP
        SUB24   YPWM,TMP
        BTFSS   TMP+B2,MSB
        GOTO    testmin
        MOV16   YPWMAX,YPWM             ; saturate to max
        GOTO    limitok
testmin
        CLRF    TMP+B2                  ; check pwm minimum limit
        CLRF    YPWM+B2
        CLRF    YPWM+B3
        MVFP16  YPWMIN,TMP
        SUB24   YPWM,TMP
        BTFSC   TMP+B2,MSB
        GOTO    limitok
        MOV16   YPWMIN,YPWM             ; saturate to min
limitok
        MOVLB   BANK3                   ; set new duty cycle                 Write PWM values
        MOVFP   YPWM+B0,PW1DCL                                               to PWM registers
        MOVFP   YPWM+B1,PW1DCH

        MOV16   U0,U1                   ; push errors into U(k-1)

        RETURN

;*****************************************************************************
```

APPENDIX D: ENCODER INTERFACE ROUTINE

```
;*****************************************************************************
; NAME:                              doMPosMVel
;
; DESCRIPTION:                Calculates current position from UpCount and DownCount
;

doMPosMVel

; Do UpCounter first

                            MVFP16  UPCOUNT,TMP+B0        ; save old upcount
readUp
                            MOVPF   RTCCH,WREG
                            MOVPF   RTCCL,UPCOUNT+B0
                            CPFSEQ  RTCCH                 ; Skip next if HI hasn't changed
                            GOTO    readUp                ; HI changed, re-read LO
                            MOVPF   WREG,UPCOUNT+B1       ; OK to store HI now

                            CLRF    MVELOCITY+B0          ; clear bits below binary point

                            MOV16   UPCOUNT,MVELOCITY+B1  ; compute upcount increment
                            SUB16   TMP+B0,MVELOCITY+B1

; Now do DownCounter

                            MVFP16  DOWNCOUNT,TMP+B0      ; save old downcount
readDown
                            MOVLB   BANK2                 ; timers in Bank 2
                            MOVPF   TMR3H,WREG
                            MOVPF   TMR3L,DOWNCOUNT+B0
                            CPFSEQ  TMR3H                 ; Skip next if HI hasn't changed
                            GOTO    readDown              ; HI changed, re-read LO
                            MOVPF   WREG,DOWNCOUNT+B1     ; OK to store HI now

                            MVFP16  DOWNCOUNT+B0,TMP+B2   ; compute downcount increment
                            SUB16   TMP+B0,TMP+B2

                            SUB16   TMP+B2,MVELOCITY+B1   ; compute new measured velocity

                            CLRF    MVELOCITY+B3          ; sign extend measured velocity for
                            BTFSC   MVELOCITY+B2,MSB      ; 24 bit addition to measured posi-
tion
                            SETF    MVELOCITY+B3



                            ADD24   MVELOCITY+B1,MPOSITION; compute new measured position
                                                          ; delta position = measured velocity

                            RETURN

;*****************************************************************************
```

APPENDIX E: IMPLEMENTATION DETAILS OF TRAJECTORY GENERATION

doPreMove:

This routine is executed only once at the beginning of each move. First, various buffers and flags are initialized and a test for modetype is performed. In position mode, the minimum move is triangular and consists of two steps. Therefore, if abs (MOVVAL) > 2, an immediate move is performed. Otherwise, normal move generation is possible with the sign of the move in MOVSIGN and the appropriate signed velocity and acceleration limits in V and A, and MOVVAL/2 in HMOVVAL.

In velocity mode, the sign of the move is calculated in MOVSIGN and the appropriate signed velocity and acceleration limits are placed in V and A. Finally, at moderady, MOVVAL is sign extended for higher precision arithmetic and the servo is enabled.

In torque mode, MOVVAL is output directly to the PWM and the servo is disabled, and doMove is not executed.

doMove:

Move generation is based on a piecewise constant acceleration model. During constant acceleration, this results in the standard equations for position and velocity given by

$$x(t) = x0 + v0*t + a*(t**2)/2, v(t) = v0 + a*t$$

With the units for t in sample times, the time increment between subsequent sample times is 1, yielding the iterative equations for updating position and velocity implemented in doPosVel and given by

$$P(k) = P(k-1) + V(k-1) + A/2, \quad V(k) = V(k-1) + A,$$

where A is the signed acceleration limit calculated in doPreMove. The inverse equations of this iteration, necessary for undoing an unwanted step, are contained in undoPosVel and given by

$$P(k-1) = P(k) - V(K-1) - A/2, \quad V(K-1) = V(k) - A.$$

In position mode, the actual shape of the velocity profile depends on the values of V, A and the size of the move. Either the velocity limit is reached before half the move is completed, resulting in a trapezoidal velocity profile, or half the move is completed before the velocity limit is realized, resulting in a triangular velocity profile.

In the algorithm employed here, the velocity limit is treated as a bound on the actual velocity limit, thereby permitting exactly the same number of steps during the speedup and speed down sections of the move. Phase 1 is defined as the section of the move where the commanded position is less than half the move, and phase 2 is the remaining portion of the move. T1 is time when the actual velocity limit is reached and T2 is the time at the end of phase 1.

FIGURE A - SPEED PROFILE FOR TRAPEZOIDAL MOVES

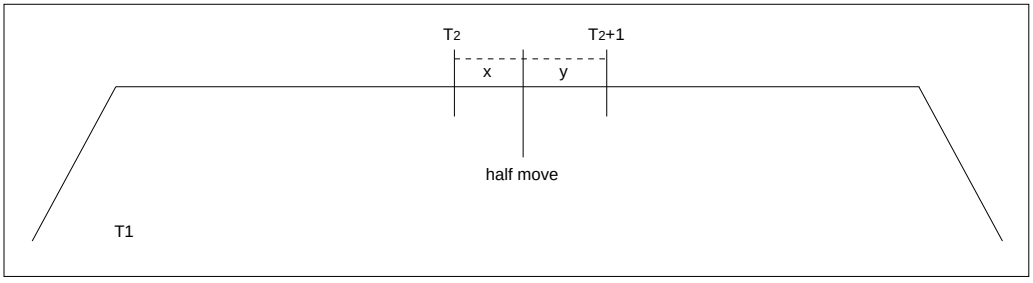


FIGURE B - SPEED PROFILE FOR TRIANGULAR MOVES

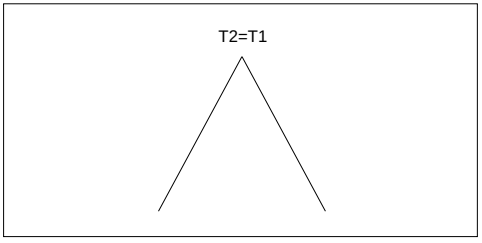
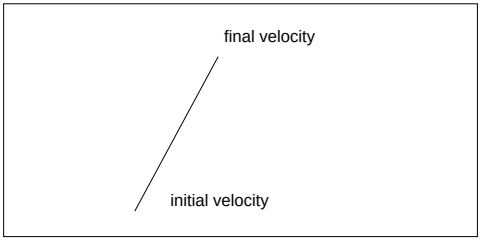


FIGURE C - SPEED PROFILE FOR VELOCITY MOVES



Furthermore, let x be the amount of undershoot and y the amount of overshoot of half the move at T2. Discretization error is minimized by using the values of x and y whether one more step will reduce the size of the final immediate move during the last step of the move. For a triangular move, the discretization error is given by min. (2x, 2y), resulting in the condition that if 2x > 2y, then take one more speedup step. In the case of a trapezoidal move, the discretization error is given by min. (2x, y-x), yielding the condition that if 3x > y, take one more step during the flat section of phase2.

At the beginning of doMove, MOVTIME is incremented and doPosVel is called to evaluate the next proposed values of commanded position and velocity under the current value of A. In position mode, phase1, the original position plus half the move minus the new proposed commanded position is calculated and placed in MOVDEL, with the previous MOVDEL saved in MOVTMP. As half the move would be passed, MOVTMP = -x and MOVDEL = y, with y>0 for the first time indicating that phase1 is about to be completed. Therefore, if y<0, we continue in phase1, where if maximum velocity has not been reached, the new proposed commanded position is executed. On the other hand, if the proposed move would exceed the maximum velocity, we undo the proposed move, set the current acceleration to zero, reevaluate the iterative equations with the new acceleration, set T1=MOVTIME-1, and execute the move.

Since T1 is cleared in doPreMove, it is used as a flag to indicate if this corner in the velocity profile has been reached. Once we find that y>0, we drop into code that is executed only one time, with phase2 beginning on the next step. If T1=0, maximum velocity has not yet been reached, so T1=T2 and the velocity profile is triangular. In this case, A is negated for speed down, and if x>y, one more step is needed to minimize the discretization error. So A is negated, the proposed step undone, A is again negated for speed down and the step recalculated and executed, with T2=T1=MOVTIME-1.

If T1 is not zero, indicating that we are in the flat section of phase1, then go to t2net1, where T2=MOVTIME-1, and if 3x>y, then one more phase2 flat step is necessary to minimize the discretization error. PH2FLAT is defined as the number of steps in the flat section of phase2, and is used as a counter during its completion. If 3x>y, then PH2FLAT=T2-T1, otherwise PH2FLAT=T2-T1-1 and phase1 is finally complete. All subsequent steps will proceed through phase2, first deciding if the flat section is finished by checking if PH2FLAT has reached zero. If not, go to flat where PH2FLAT is decremented, and tested if zero. If so, the speed down section is begun by calculating the appropriate signed acceleration limit A, and executing the last of the flat section moves. For all following steps, PH2FLAT=0, leaving only the final test for zero commanded velocity to indicate the end of the move. This will always occur since the actual maximum velocity, bounded above by the user supplied limit, is always an integer multiple of the user supplied acceleration limit, with exactly the same number of steps taken during speedup and speed down.

The velocity mode is much more straightforward, with the velocity profile in the form of a ramp. If the final velocity has not been reached, the move continues at maximum acceleration. If the final velocity has been reached, the acceleration is set to zero and the move generation of commanded position and velocity continued unless the final velocity is zero.

APPENDIX F: COMPLETE CODE LISTING (DCMOTOR.LST)

```
MPASM B0.54                                                          PAGE  1

"Revision:   2.0, 27 June 93"


                        SubTitle        "Revision:   2.0, 27 June 93"

                      ;*************************************************************************
                      ;
                      ;       Revised: 8/5/92
                      ;       CREDIT:  Developed by Teknic Inc. 1992
                      ;
                      ;  Assembled using MPASM. User's with ASM17 are suggested to get Microchip's
                      ;  new universal assembler (MPASM). To assemble with ASM17, all "if", "else"
                      ;  "endif" directives must be replaced by "#if", "#else" and "#endif"
                      ;  respectively.
                      ;*************************************************************************

                              PROCESSOR       PIC17C42
                              LIST            COLUMNS=120, XREF=YES, NOWRAP,LINES=255, R=DEC

                      ;*************************************************************************
                      ;

00F4 2400             MASTER_CLOCK    set    16000000            ; Input Clock Freq in Hz

03E8                  _SAMPLE_RATE    set     1000               ; Sample rate in Hz
07D0                  _ENCODER_RATE   set     2000               ; 2000 Pulses/rev
1770                  _RATED_SPEED    set     6000               ; in RPM

2580                  _BAUDRATE_      set     9600

                      ;*************************************************************************


                               include "17c42.h"


                               include "17c42.mac"    ; General Purpose Macros


0058                   #define _PICMASTER_DEBUG        TRUE      ; Enable PIC-MASTER TRACE Capture
0059                   #define _SERVO_PID              TRUE      ; PID computation based on error
005A                   #define DECIO                   TRUE      ; true for decimal, false for hex
005B                   #define _SERIAL_IO              TRUE

                               include "dcmotor.h17"             ; Initialization, Global Defs,
                      ;*************************************************************************
                      ;
                      ; Header file for dcmotor.asm:
                      ; Revised: 8/5/92
                      ;*************************************************************************
                      ;
                      ; hardware constants
                      ;
                      ;

005C                   #define _SET_BAUD_RATE(bps)     ((10*MASTER_CLOCK/(64*bps))+5)/10 - 1
003D 0900             CLKOUT          set    MASTER_CLOCK >> 2   ; Clock Out = CLKIN/4


0006                  TCON1_INIT      set    0x06
003F                  TCON2_INIT      set    0x3F
00FF                  PR1_INIT        set    0xFF                ; set pwm frequency to CLKOUT/256
```

```
000F                PR2_INIT        set     ((10*MASTER_CLOCK/(4*(PR1_INIT+1)*_SAMPLE_RATE))+5)/

                    ;pw1dcH_INIT     set    (((PR1_INIT+1) << 8)) >> 9
                    ;pw1dcL_INIT     set    (((((PR1_INIT+1) << 8)) >> 1) & 0xff)

007F                PW1DCH_INIT     set    0x7F
00C0                PW1DCL_INIT     set    0xC0

0080                RTCSTA_INIT     set    0x80
0090                RCSTA_INIT      set    0x90
0020                TXSTA_INIT      set    0x20
0019                SPBRG_INIT      set    _SET_BAUD_RATE(_BAUDRATE_)

00F3                DDRB_INIT       set    0xF3
0000                DDRD_INIT       set    0x00
                    ;
                    ;
                    ;       max and min pwm values
                    ;
0040                PWMINL          set    0x40
0001                PWMINH          set    0x01                    ; 0x0000 + 0x0140 (min 10 bit
0080                PWMAXL          set    0x80
00FE                PWMAXH          set    0xFE                    ; 0xFFC0 - 0x0140 ( max 10 bit
                    ;
                    ;
                    ;*****************************************************************************
                    ;       global  variables
                    ;
                            CBLOCK  0x18
0018 0004                           DPX,DPX1,DPX2,DPX3                  ; arithmetic accumula
001C 0004                           AARG,AARG1,BARG,BARG1               ; multiply arguments
                            ENDC

                            CBLOCK  0x18
0018 0004                           TMP,TMP1,TMP2,TMP3                  ; temporary variables
001C 0004                           MOVTMP,MOVTMP1,MOVTMP2,MOVTMP3      ; move temporary
                            ENDC

                            CBLOCK  0x20
0020 0003                           VL,VL1,VL2                          ; velocity limit
0023 0003                           AL,AL1,AL2                          ; acceleration limit
0026 0000
0026 0002                           KP,KP1                              ; proportional gain
0028 0002                           KV,KV1                              ; velocity gain
002A 0002                           KI,KI1                              ; integral gain
002C 0000
002C 0001                           IM                                  ; integrator mode
002D 0002                           FV,FV1                              ; velocity feedforward
002F 0002                           FA,FA1                              ; acceleration
0031 0000
0031 0003                           VALBUF,VALBUF1,VALBUF2              ; iovalue buffer
0034 0000
0034 0003                           DVALBUF,DVALBUF1,DVALBUF2           ; iovalue buffer
0037 0002                           ISRBSR,ISRWREG                      ; isr save storage
0039 0004                           CMDCHAR,CMDTEMP,CMDPTRH,CMDPTRL     ; command interface
003D 0003                           PARTEMP,PARLEN,PARPTR               ; parameter variables
0040 0000
0040 0003                           CPOSITION,CPOSITION1,CPOSITION2     ; shutter commanded
0043 0003                           CVELOCITY,CVELOCITY1,CVELOCITY2     ; shutter commanded
0046 0003                           CMPOSITION,CMPOSITION1,CMPOSITION2  ; shutter measured
0049 0003                           CMVELOCITY,CMVELOCITY1,CMVELOCITY2  ; shutter measured
004C 0000
004C 0002                           STRVALH,STRVALL                     ; string io variables
004E 0003                           HEXVAL,HEXTMP,HEXSTAT               ; hex io variables
0051 0000
0051 0002                           OPOSITION,OPOSITION1
0053 0002                           OPOSITION2,OPOSITION3               ; original commanded
0055 0003                           POSITION,POSITION1,POSITION2        ; commanded position
0058 0003                           VELOCITY,VELOCITY1,VELOCITY2        ; commanded velocity
005B 0000
```

```
005B 0004                       NMOVVAL,NMOVVAL1,NMOVVAL2,NMOVVAL3    ; move value
005F 0004                       MOVVAL,MOVVAL1,MOVVAL2,MOVVAL3        ; move value
0063 0004                       HMOVVAL,HMOVVAL1,HMOVVAL2,HMOVVAL3    ; half move value
0067 0002                       MOVTIME,MOVTIME1                      ; move time in sample
0069 0000
0069 0001                       MOVSIGN                               ; 0x00 for positive,
0x80 for
006A 0002                       T1,T11                                ; time to maximum
006C 0002                       T2,T21                                ; time for half the
006E 0002                       TAU,TAU1                              ; total move time
0070 0001                       NMODE                                 ; next move modetype
0071 0001                       MODE                                  ; move modetype
0072 0000
0072 0003                       MPOSITION,MPOSITION1,MPOSITION2       ; measured position
0075 0002                       MVELOCITY,MVELOCITY1
0077 0002                       MVELOCITY2,MVELOCITY3                 ; measured velocity
0079 0003                       POSERROR,POSERROR1,POSERROR2          ; position error
007C 0003                       VELERROR,VELERROR1,VELERROR2          ; velocity error
007F 0000
007F 0001                       SIGN                                  ; multiply sign
0080 0000
0080 0004                       Y,Y1,Y2,Y3                            ; Y(k) before pwm
0084 0004                       U0,U01,U1,U11                         ; saturated error at
0088 0000
0088 0004                       YPWM,YPWM1,YPWM2,YPWM3                ; pwm input
008C 0004                       YPWMIN,YPWMIN1,YPWMAX,YPWMAX1         ; pwm input limits
0090 0000
0090 0001                       SERVOFLAG                             ; servoflag = 0 => no
0091 0001                       MODETYPE                              ; mode
0092 0001                       EXTSTAT                               ; external status
0093 0001                       MOVSTAT                               ; move status register
0094 0001                       MOVFLAG                               ; move flag
0095 0001                       SATFLAG                               ; saturation flag
0096 0002                       INTEGRAL,INTEGRAL1                    ; integrator
0098 0000
0098 0004                       DECVAL,DECSTAT,DECTMP,DECSIGN         ; decimal io variables
009C 0000
009C 0004                       A,A1,A2,A3                            ; commanded accelera
00A0 0004                       V,V1,V2,V3                            ; commanded velocity =
00A4 0004                       MOVPBUF,MOVPBUF1,MOVPBUF2,MOVPBUF3    ; commanded position
00A8 0004                       MOVVBUF,MOVVBUF1,MOVVBUF2,MOVVBUF3    ; commanded velocity
00AC 0000
00AC 0002                       UPCOUNT,UPCOUNT1                      ; running up counter
00AE 0002                       DOWNCOUNT,DOWNCOUNT1                  ; running down counter
00B0 0000
00B0 0004                       MOVDEL,MOVDEL1,MOVDEL2,MOVDEL3        ; move discretization
00B4 0002                       PH2FLAT,PH2FLAT1                      ; phase 2 flat itera
00B6 0003                       INDEXPOS,INDEXPOS1,INDEXPOS2          ; position at last
00B9 0000
00B9 0001                       SHIFTNUM                              ; # of bit shifts from
00BA 0000
00BA 0001            if     _PICMASTER_DEBUG
00BB 0000           ;*************************************************************************
00BB 0000           ;       For PICMASTER Debug/servo tuning Purposes Only
00BB 0000
00BB 0001                       CAPFLAG                               ; trace capture flag
00BC 0002                       CAPCOUNT,CAPCOUNT1                    ; PICMASTER trace
00BE 0002                       CAPTMP,CAPTMP1                        ; trace capture
00C0 0000
00C0 0000           ;*************************************************************************
```

```
00C0 0001             endif
00C1 0000
00C1 0002                       ZERO,ONE                        ; constants
00C3 0002                       tblptrlTemp,tblptrhTemp         ; temp TABLE Pointers for
00C5 0002                       TblLatLo, TblLatHi              ; Table Latch for ISR save
00C7 0001                       alustaTemp                      ; temp alusta
00C8 0000
                        ENDC

                ;*************************************************************************
                ; NAME:         AUTONO
                ;
                ; DESCRIPTION:  Sets no auto increment or decrement
                ;
                ; TIMING (cycles): 4

                AUTONO MACRO

                                BSF     _fs0
                                BSF     _fs1
                                BSF     _fs2
                                BSF     _fs3

                ENDM

                ;*************************************************************************

                ;*************************************************************************
                ; NAME:         AUTOINC
                ;
                ; DESCRIPTION:  Set auto increment
                ;
                ; TIMING (cycles): 4
                ;

                AUTOINC MACRO

                                BSF     _fs0
                                BCF     _fs1
                                BSF     _fs2
                                BCF     _fs3

                ENDM

                ;*************************************************************************

                ;*************************************************************************
                ; NAME:         AUTODEC
                ;
                ; DESCRIPTION:  Sets auto decrement
                ;
                ; TIMING (cycles): 4
                ;

                AUTODEC MACRO

                                BCF     _fs0
                                BCF     _fs1
                                BCF     _fs2
                                BCF     _fs3

                ENDM

                ;*************************************************************************


                ;*************************************************************************
```

```
                        ; NAME:         LOADAB
                        ;
                        ; DESCRIPTION:  Loads extended math library AARG and BARG
                        ;
                        ; ARGUMENTS:    A => AARG
                        ;               B => BARG
                        ;
                        ; TIMING (cycles): 4

                        LOADAB MACRO    A,B

                                        MOVFP   A+B0,AARG+B0  ; load lo byte of A to AARG
                                        MOVFP   A+B1,AARG+B1  ; load hi byte of A to AARG
                                        MOVFP   B+B0,BARG+B0  ; load lo byte of B to BARG
                                        MOVFP   B+B1,BARG+B1  ; load hi byte of B to BARG


                                        ENDM

                        ;*****************************************************************************

                        ;*****************************************************************************
                        ;     ascii constants
                        ;
000D                    CR      set    0x0D
0018                    CAN     set    0x18
0008                    BS      set    0x08
0020                    SP      set    0x20
000A                    LF      set    0x0A
002D                    MN      set    '-'
                        ;
                        ;*******************************************************
                        ;     cmds constants and macros
                        ;
                        ;
0001                    CHARREADY       set    0x01
                        ;
                        ;
0008                    NUMPAR          set    0x08
                        ;
                        ; Response characters
                        ;
0021                    CMD_OK          set    '!'
003F                    CMD_BAD         set    '?'
                        ;
                        ; Exit values
                        ;
                        ;
0000                    HEX_SP          set    0x00
0001                    HEX_MN          set    0x01
0002                    HEX_CR          set    0x02
0003                    HEX_CAN         set    0x03
                        ;
0000                    DEC_SP          set    0x00
0001                    DEC_MN          set    0x01
0002                    DEC_CR          set    0x02
0003                    DEC_CAN         set    0x03
                        ;
                        ;
                        ; Command characters
                        ;
000D                    DO_NULL                 set    CR
004D                    DO_MOVE                 set    'M'     ; M
004F                    DO_MODE                 set    'O'     ; O
0053                    DO_SETPARAMETER         set    'S'     ; S
0052                    DO_READPARAMETER        set    'R'     ; R
0043                    DO_SHUTTER              set    'C'     ; C
0050                    DO_READCOMPOSITION      set    'P'     ; P
0056                    DO_READCOMVELOCITY      set    'V'     ; V
```

```
0070                DO_READACTPOSITION      set    'p'     ; p
0076                DO_READACTVELOCITY      set    'v'     ; v
0058                DO_EXTERNALSTATUS       set    'X'     ; X
0059                DO_MOVESTATUS           set    'Y'     ; Y
0049                DO_READINDPOSITION      set    'I'     ; I
0048                DO_SETPOSITION          set    'H'     ; H
005A                DO_RESET                set    'Z'     ; Z
0073                DO_STOP                 set    's'     ; s
0063                DO_CAPTURE              set    'c'     ; c

                    ;*************************************************************************
                    ; NAME:         CMD_DEF
                    ;
                    ; DESCRIPTION:  Creates all the definitions for a command table data struc-
                    ;               ture. The first word is at the command character used, and
                    ;               the second word is a pointer to the function that handles
                    ;               this command function.
                    ;
                    ; ENTRY CONDITIONS:     Must be contiguous with the other entries for the
                    ;                       function to work.
                    ;
                    ; ARGUMENTS:            FUNC    command execution function
                    ;                       ROOT    NAME ROOT
                    ;

                    CMD_DEF MACRO   FUNC,ROOT

                                    DATA    ROOT
                                    DATA    FUNC
                                    ENDM

0002                CMD_ENTRY_LENGTH        set    2

                    ;*************************************************************************

                    ;*************************************************************************
                    ; NAME:         CMD_START
                    ;
                    ; DESCRIPTION:  Labels the start of the command table.
                    ;

                    CMD_START MACRO LABEL

                    LABEL
                                    ENDM                                    ;

                    ;*************************************************************************
                    ;*************************************************************************
                    ; NAME:         CMD_END
                    ;
                    ; DESCRIPTION:  Marks the end of the command table with an entry of 0x00
                    ;

                    CMD_END MACRO
                    ;                                                       ;
                                    DATA    0x00
                                    ENDM                                    ;

                    ;*************************************************************************




                    ;
                    ;
                    ;*************************************************************************
                    ;                               PID Constantnts
                    ; define PIV parameters, computation based on errors only. Does not involve
```

```
                    ;
                    ; No Load @ 2Khz :              Kp=3600, Ki=112, Kd= 28800, Shiftcount = 3
                    ; With Indicator Load @ 2Khz,   Kp=2300 Ki= 41, Kd= 32200, Shiftcount = 4
                    ;     " " @ 4Khz                Kp=1024, Ki=8, Kd=31405 , ShiftCount = 5
                    ;     " " @ 0.5 Khz             Kp=5400, Ki=310, Kd=23400 , ShiftCount = 2
                    ;
                    ; No Load @ 1Khz                Kp=3600, Ki=192, Kd=16800, ShiftNum = 3
                    ; No Load @ 1Khz                Kp=1800, Ki=52, Kd=15600, ShiftNum = 4
                    ;
                    ; Adjust Shiftcount by maximizing Kd (for a 16 bit signed num)
                    ;
                    ;*****************************************************************************

005D                 #define _KP     1800                                    ; 16 bit Kp
005E                 #define _KI     52                                      ; 16 bit Ki
005F                 #define _KV     15600                                   ; 16 bit Kv

3200                _VEL_LIMIT      set     ((_RATED_SPEED*_ENCODER_RATE)/; 1/4 Rated sp
2000                _ACCL_LIMIT     set     0x2000                          ; use smaller


                     if _SERVO_PID == TRUE
0004                        _SHIFTNUM       equ     4
                     endif

                    ;*****************************************************************************


                            ORG     0x0                                     ; reset vector
0000 C021                   goto    Startup                                 ; startup

                            ORG     0x20
0020 C07D                   goto    InterruptPoll                           ; interrupt


                    ;*****************************************************************************
                    ; NAME:         Startup
                    ;
                    ; DESCRIPTION:  This routine is called on the hardware reset or when the
                    ;               program wishes to restore initial conditions. Initiali-
                    ;               zation of run-time constants takes place here.
                    ;
                    ; RETURNS:      restart to safe and initial state
                    ;
                    ; STACK UTILIZATION: none
                    ; TIMING (in cycles): X

                    Startup

0021 8406                   bsf     _glintd                                 ; disable all
                                        AUTONO
                             ; no auto

0022 8404                   BSF     _fs0
0023 8504                   BSF     _fs1
0024 8604                   BSF     _fs2
0025 8704                   BSF     _fs3


0026 B018                   movlw   0x18                                    ; clear all
0027 4A01                   movpf   wreg,fsr0

                    memloop
0028 2900                   clrf    indf0
0029 1F01                   incfsz  fsr0
002A C028                   goto    memloop

002B 15C2                   incf    ONE
```

```
002C B803                       movlb   bank3           ; bank3 ini
002D B03F                       movlw   TCON2_INIT
002E 770A                       movfp   wreg,tcon2

002F B07F                       movlw   PW1DCH_INIT     ; set duty
0030 720A                       movfp   wreg,pw1dch
0031 730A                       movfp   wreg,pw2dch

0032 B0C0                       movlw   PW1DCL_INIT
0033 700A                       movfp   wreg,pw1dcl
0034 710A                       movfp   wreg,pw2dcl

0035 B006                       movlw   TCON1_INIT      ; set organization of timers
0036 760A                       movfp   wreg,tcon1

0037 B802                       movlb   bank2           ; bank2 initialization

0038 B0FF                       movlw   PR1_INIT
0039 740A                       movfp   wreg,pr1        ; initialize timer1 period

003A B00F                       movlw   PR2_INIT
003B 750A                       movfp   wreg,pr2        ; initialize timer2 period


003C B800                       movlb   bank0           ; bank0 initialization

003D B080                       movlw   RTCSTA_INIT
003E 650A                       movfp   wreg,rtcsta     ; sets RTC for external input

003F B0F8                       movlw   0xF8            ; RA2 connected to BREAK Input of LMD18200
0040 0110                       movwf   porta           ; On Reset, thus pulled high breaking the motor

                        if _SERIAL_IO

0041 B090                       movlw   RCSTA_INIT
0042 730A                       movfp   wreg,rcsta      ; set receive status

0043 B020                       movlw   TXSTA_INIT      ; set transmit status
0044 750A                       movfp   wreg,txsta

0045 B019                       movlw   SPBRG_INIT      ; set baud rate
0046 770A                       movfp   wreg,spbrg
                        endif

0047 B0F3                       movlw   DDRB_INIT
0048 710A                       movfp   wreg,ddrb       ; set port B for whatever

0049 B801                       movlb   bank1           ; bank1 initialization

                        if (_SERVO_PID == TRUE)
                                        MOVK16  _KP,KP

004A B008                       MOVLW   (1800) & 0xff
004B 0126                       MOVWF   KP+B0
004C B007                       MOVLW   ((1800) >> 8)
004D 0127                       MOVWF   KP+B1


                                        MOVK16  _KI,KI

004E B034                       MOVLW   (52) & 0xff
004F 012A                       MOVWF   KI+B0
0050 B000                       MOVLW   ((52) >> 8)
0051 012B                       MOVWF   KI+B1


                                        MOVK16  _KV,KV
```

```
0052 B0F0                   MOVLW   (15600) & 0xff
0053 0128                   MOVWF   KV+B0
0054 B03C                   MOVLW   ((15600) >> 8)
0055 0129                   MOVWF   KV+B1


                    endif

                            MOVK16  _ACCL_LIMIT,AL

0056 B000                   MOVLW   (_ACCL_LIMIT) & 0xff
0057 0123                   MOVWF   AL+B0
0058 B020                   MOVLW   ((_ACCL_LIMIT) >> 8)
0059 0124                   MOVWF   AL+B1


                            MOVK16  _VEL_LIMIT,VL

005A B000                   MOVLW   (_VEL_LIMIT) & 0xff
005B 0120                   MOVWF   VL+B0
005C B032                   MOVLW   ((_VEL_LIMIT) >> 8)
005D 0121                   MOVWF   VL+B1



005E B004                   movlw   _SHIFTNUM
005F 01B9                   movwf   SHIFTNUM

0060 5289                   movpf   pw1dch,YPWM+B1

0061 B080                   movlw   PWMAXL          ; initialize pwm limits
0062 4A8E                   movpf   wreg,YPWMAX+B0
0063 B0FE                   movlw   PWMAXH
0064 4A8F                   movpf   wreg,YPWMAX+B1
0065 B040                   movlw   PWMINL
0066 4A8C                   movpf   wreg,YPWMIN+B0
0067 B001                   movlw   PWMINH
0068 4A8D                   movpf   wreg,YPWMIN+B1

0069 2916                   clrf    pir             ; clear flags, set indiviual interrupts
006A 2907                   clrf    intsta
006B 8517                   bsf     _tm2ie
006C 8307                   bsf     _peie

006D 8C06                   bcf     _glintd         ; enable interrupts

006E B802                   movlb   bank2

                    zeroctrs
006F 290B                   clrf    rtccl           ; clear up counter
0070 290C                   clrf    rtcch

0071 2912                   clrf    tmr3l           ; clear down counter
0072 2913                   clrf    tmr3h

0073 B0FF                   movlw   0xFF
0074 170A           delay   decfsz  wreg
0075 C074                   goto    delay

0076 6A0B                   movfp   rtccl,wreg
0077 080C                   iorwf   rtcch,W
0078 0812                   iorwf   tmr3l,W
0079 0813                   iorwf   tmr3h,W
007A 330A                   tstfsz  wreg
007B C06F                   goto    zeroctrs        ; motor still moving

007C C124                   goto    PollingLoop

                    ;*****************************************************************
```

```
                        ;*************************************************************************
                        ; NAME:         InterruptPoll


                        InterruptPoll


007D 0138                       movwf   ISRWREG                 ; save W Reg
007E 6A04                       movfp   alusta,wreg
007F 01C7                       movwf   alustaTemp              ; save alusta
0080 4F37                       movpf   bsr,ISRBSR              ; save BSR,wreg
0081 4DC3                       movpf   tblptrl,tblptrlTemp     ; save Table Pointers
0082 4EC4                       movpf   tblptrh,tblptrhTemp
0083 A0C5                       tlrd    0,TblLatLo
0084 A2C6                       tlrd    1,TblLatHi              ; save Table Latch


0085 B801                       movlb   bank1

0086 E0E5                       call    doMPosMVel              ; calculate measured position and
                                                                ; velocity

0087 E111                       call    doExtstat               ; evaluate external status

0088 2293                       rlncf   MOVSTAT,W               ; if MOVFLAG=0 and MOVSTAT,bit7=1
0089 B501                       andlw   0x01                    ; then do premove. This is only
008A 0494                       subwf   MOVFLAG,W               ; executed once at the beginning of
008B 9F0A                       btfsc   wreg,MSB                ; each move
008C E23D                       call    doPreMove

008D 9E93                       btfsc   MOVSTAT,bit6            ; is motion continuing?
008E E30F                       call    doMove                  ; if so, do move

008F E09E                       call    doError                 ; calculate position and velocity
                                                                                        ; error
0090 3390                       tstfsz  SERVOFLAG               ; test servoflag, if 0 then no servo
0091 E4AC                       call    doServo                 ; do servo

                         if     _PICMASTER_DEBUG
0092 33BB                       tstfsz  CAPFLAG
0093 E79A                       call    doCaptureRegs           ; for PIC-MASTER Trace Capture, demo
                         endif

0094 B801                       movlb   bank1

0095 2916                       clrf    pir                     ; clear all interrupt request flags

0096 A4C5                       tlwt    0,TblLatLo
0097 A6C6                       tlwt    1,TblLatHi              ; restored table latch
0098 6DC3                       movfp   tblptrlTemp,tblptrl
0099 6EC4                       movfp   tblptrhTemp,tblptrh     ; restored table pointers
009A 6F37                       movfp   ISRBSR,bsr              ; restore BSR,wreg,alusta & Table
009B 64C7                       movfp   alustaTemp,alusta
009C 6A38                       movfp   ISRWREG,wreg

009D 0005                       retfie

                        ;*************************************************************************
                        ;*************************************************************************
                        ; NAME:         doError
                        ;
                        ; DESCRIPTION:  Calculates the position and velocity error.
                        ;

                        doError

                         MOV24   POSITION,POSERROR              ; calculate position error
```

```
009E 6A55                   MOVFP   POSITION+B0,wreg         ; get byte of POSITION into w
009F 4A79                   MOVPF   wreg,POSERROR+B0         ; move to POSERROR(B0)
00A0 6A56                   MOVFP   POSITION+B1,wreg         ; get byte of POSITION into w
00A1 4A7A                   MOVPF   wreg,POSERROR+B1         ; move to POSERROR(B1)
00A2 6A57                   MOVFP   POSITION+B2,wreg         ; get byte of POSITION into w
00A3 4A7B                   MOVPF   wreg,POSERROR+B2         ; move to POSERROR(B2)


                           SUB24   MPOSITION,POSERROR

00A4 6A72                   MOVFP   MPOSITION+B0,wreg       ; get lowest byte of MPOSITION into
00A5 0579                   SUBWF   POSERROR+B0             ; sub lowest byte of POSERROR,
00A6 6A73                   MOVFP   MPOSITION+B1,wreg       ; get 2nd byte of MPOSITION into
00A7 037A                   SUBWFB  POSERROR+B1             ; sub 2nd byte of POSERROR, save
00A8 6A74                   MOVFP   MPOSITION+B2,wreg       ; get 3rd byte of MPOSITION into
00A9 037B                   SUBWFB  POSERROR+B2             ; sub 3rd byte of POSERROR, save


00AA 9F7B                   btfsc   POSERROR+B2,MSB         ; saturate error to lowest 16 bits
00AB C0B7                   goto    pneg
                    ppos
00AC 6A7A                   movfp   POSERROR+B1,wreg
00AD B580                   andlw   0x80
00AE 097B                   iorwf   POSERROR+B2
00AF 290A                   clrf    wreg
00B0 327B                   cpfsgt  POSERROR+B2
00B1 C0C1                   goto    psatok
00B2 297B                   clrf    POSERROR+B2             ; clear high byte for debug purposes
00B3 B07F                   movlw   0x7F
00B4 4A7A                   movpf   wreg,POSERROR+B1
00B5 2B79                   setf    POSERROR
00B6 C0C1                   goto    psatok
                    pneg
00B7 6A7A                   movfp   POSERROR+B1,wreg
00B8 B37F                   iorlw   0x7F
00B9 0B7B                   andwf   POSERROR+B2
00BA 2B0A                   setf    wreg
00BB 307B                   cpfslt  POSERROR+B2
00BC C0C1                   goto    psatok
00BD 2B7B                   setf    POSERROR+B2             ; set high byte to 0xFF for debug
00BE 297A                   clrf    POSERROR+B1
00BF 877A                   bsf     POSERROR+B1,MSB
00C0 2979                   clrf    POSERROR
                    psatok

                           MOV24   VELOCITY,VELERROR      ; calculate velocity error

00C1 6A58                   MOVFP   VELOCITY+B0,wreg        ; get byte of VELOCITY into w
00C2 4A7C                   MOVPF   wreg,VELERROR+B0        ; move to VELERROR(B0)
00C3 6A59                   MOVFP   VELOCITY+B1,wreg        ; get byte of VELOCITY into w
00C4 4A7D                   MOVPF   wreg,VELERROR+B1        ; move to VELERROR(B1)
00C5 6A5A                   MOVFP   VELOCITY+B2,wreg        ; get byte of VELOCITY into w
00C6 4A7E                   MOVPF   wreg,VELERROR+B2        ; move to VELERROR(B2)


                           SUB24   MVELOCITY,VELERROR

00C7 6A75                   MOVFP   MVELOCITY+B0,wreg       ; get lowest byte of MVELOCITY into w
00C8 057C                   SUBWF   VELERROR+B0             ; sub lowest byte of VELERROR, save
00C9 6A76                   MOVFP   MVELOCITY+B1,wreg       ; get 2nd byte of MVELOCITY into w
00CA 037D                   SUBWFB  VELERROR+B1             ; sub 2nd byte of VELERROR, save in
00CB 6A77                   MOVFP   MVELOCITY+B2,wreg       ; get 3rd byte of MVELOCITY into w
00CC 037E                   SUBWFB  VELERROR+B2             ; sub 3rd byte of VELERROR, save in


00CD 9F7E                   btfsc   VELERROR+B2,MSB         ; saturate error to lowest 16 bits
00CE C0DA                   goto    vneg
                    vpos
```

```
00CF 6A7D                       movfp   VELERROR+B1,wreg
00D0 B580                       andlw   0x80
00D1 097E                       iorwf   VELERROR+B2
00D2 290A                       clrf    wreg
00D3 327E                       cpfsgt  VELERROR+B2
00D4 C0E4                       goto    vsatok
00D5 297E                       clrf    VELERROR+B2
00D6 B07F                       movlw   0x7F
00D7 4A7D                       movpf   wreg,VELERROR+B1
00D8 2B7C                       setf    VELERROR
00D9 C0E4                       goto    vsatok
                        vneg
00DA 6A7D                       movfp   VELERROR+B1,wreg
00DB B37F                       iorlw   0x7F
00DC 0B7E                       andwf   VELERROR+B2
00DD 2B0A                       setf    wreg
00DE 307E                       cpfslt  VELERROR+B2
00DF C0E4                       goto    vsatok
00E0 2B7E                       setf    VELERROR+B2
00E1 297D                       clrf    VELERROR+B1
00E2 877D                       bsf     VELERROR+B1,MSB
00E3 297C                       clrf    VELERROR
                        vsatok
00E4 0002                       return

                        ;*****************************************************************************


                        ;*****************************************************************************
                        ; NAME:         doMPosMVel
                        ;
                        ; DESCRIPTION:  Calculates current position from UpCount and DownCount
                        ;

                        doMPosMVel

                        ; Do UpCounter first

                                MOVFP16 UPCOUNT,TMP+B0              ; save old upcount

00E5 78AC                       MOVFP   UPCOUNT+B0,TMP+B0+B0        ; move UPCOUNT(B0) to TMP+B0(B0)
00E6 79AD                       MOVFP   UPCOUNT+B1,TMP+B0+B1        ; move UPCOUNT(B1) to TMP+B0(B1)


                        readUp
00E7 4C0A                       movpf   rtcch,wreg
00E8 4BAC                       movpf   rtccl,UPCOUNT+B0
00E9 310C                       cpfseq  rtcch                       ; Skip next if HI hasn't changed
00EA C0E7                       goto    readUp                      ; HI changed, re-read LO
00EB 4AAD                       movpf   wreg,UPCOUNT+B1             ; OK to store HI now

00EC 2975                       clrf    MVELOCITY+B0                ; clear bits below binary point

                                MOV16   UPCOUNT,MVELOCITY+B1        ; compute upcount increment

00ED 6AAC                       MOVFP   UPCOUNT+B0,wreg             ; get byte of UPCOUNT into w
00EE 0176                       MOVWF   MVELOCITY+B1+B0             ; move to MVELOCITY+B1(B0)
00EF 6AAD                       MOVFP   UPCOUNT+B1,wreg             ; get byte of UPCOUNT into w
00F0 0177                       MOVWF   MVELOCITY+B1+B1             ; move to MVELOCITY+B1(B1)


                                SUB16   TMP+B0,MVELOCITY+B1

00F1 6A18                       MOVFP   TMP+B0+B0,wreg              ; get lowest byte of TMP+B0 into
00F2 0576                       SUBWF   MVELOCITY+B1+B0             ; sub lowest byte of
00F3 6A19                       MOVFP   TMP+B0+B1,wreg              ; get 2nd byte of TMP+B0 into w
00F4 0377                       SUBWFB  MVELOCITY+B1+B1             ; sub 2nd byte of MVELOCITY+B1,
```

```
                    ; Now do DownCounter

                            MOVFP16  DOWNCOUNT,TMP+B0          ; save old downcount

00F5 78AE                   MOVFP   DOWNCOUNT+B0,TMP+B0+B0     ; move DOWNCOUNT(B0) to
00F6 79AF                   MOVFP   DOWNCOUNT+B1,TMP+B0+B1     ; move DOWNCOUNT(B1) to


                    readDown
00F7 B802                   movlb   bank2   ; timers in Bank 2
00F8 530A                   movpf   tmr3h,wreg
00F9 52AE                   movpf   tmr3l,DOWNCOUNT+B0
00FA 3113                   cpfseq  tmr3h                      ; Skip next if HI hasn't changed
00FB C0F7                   goto    readDown                   ; HI changed, re-read LO
00FC 4AAF                   movpf   wreg,DOWNCOUNT+B1          ; OK to store HI now

                            MOVFP16  DOWNCOUNT+B0,TMP+B2       ; compute downcount increment

00FD 7AAE                   MOVFP   DOWNCOUNT+B0+B0,TMP+B2+B0  ; move DOWNCOUNT+B0(B0) to
00FE 7BAF                   MOVFP   DOWNCOUNT+B0+B1,TMP+B2+B1  ; move DOWNCOUNT+B0(B1) to


                            SUB16   TMP+B0,TMP+B2

00FF 6A18                   MOVFP   TMP+B0+B0,wreg             ; get lowest byte of TMP+B0 into
0100 051A                   SUBWF   TMP+B2+B0                  ; sub lowest byte of TMP+B2, save
0101 6A19                   MOVFP   TMP+B0+B1,wreg             ; get 2nd byte of TMP+B0 into w
0102 031B                   SUBWFB  TMP+B2+B1                  ; sub 2nd byte of TMP+B2, save in



                            SUB16   TMP+B2,MVELOCITY+B1        ; compute new measured velocity

0103 6A1A                   MOVFP   TMP+B2+B0,wreg             ; get lowest byte of TMP+B2 into
0104 0576                   SUBWF   MVELOCITY+B1+B0            ; sub lowest byte of
0105 6A1B                   MOVFP   TMP+B2+B1,wreg             ; get 2nd byte of TMP+B2 into w
0106 0377                   SUBWFB  MVELOCITY+B1+B1            ; sub 2nd byte of MVELOCITY+B1,



0107 2978                   clrf    MVELOCITY+B3              ; sign extend measured velocity
0108 9F77                   btfsc   MVELOCITY+B2,MSB          ; 24 bit addition to measured
0109 2B78                   setf    MVELOCITY+B3



                            ADD24   MVELOCITY+B1,MPOSITION    ; compute new measured position

010A 6A76                   MOVFP   MVELOCITY+B1+B0,wreg      ; get lowest byte of MVELOCITY+B1
010B 0F72                   ADDWF   MPOSITION+B0              ; add lowest byte of MPOSITION,
010C 6A77                   MOVFP   MVELOCITY+B1+B1,wreg      ; get 2nd byte of MVELOCITY+B1
010D 1173                   ADDWFC  MPOSITION+B1              ; add 2nd byte of MPOSITION, save
010E 6A78                   MOVFP   MVELOCITY+B1+B2,wreg      ; get 3rd byte of MVELOCITY+B1
010F 1174                   ADDWFC  MPOSITION+B2              ; add 3rd byte of MPOSITION, save


                                                              ; delta position = measured

0110 0002                   return

                    ;*************************************************************************
                    ; NAME:         doExtstat
                    ;
                    ; DESCRIPTION:  Get +limit,-limit,GPI from PORTB and set in EXTSTAT
                    ;
                    doExtstat
0111 9407                   btfss   _intir
0112 C11B                   goto    otherbits
                            MOV24   MPOSITION,INDEXPOS
```

```
0113 6A72                   MOVFP   MPOSITION+B0,wreg          ; get byte of MPOSITION into w
0114 4AB6                   MOVPF   wreg,INDEXPOS+B0           ; move to INDEXPOS(B0)
0115 6A73                   MOVFP   MPOSITION+B1,wreg          ; get byte of MPOSITION into w
0116 4AB7                   MOVPF   wreg,INDEXPOS+B1           ; move to INDEXPOS(B1)
0117 6A74                   MOVFP   MPOSITION+B2,wreg          ; get byte of MPOSITION into w
0118 4AB8                   MOVPF   wreg,INDEXPOS+B2           ; move to INDEXPOS(B2)


0119 8C07                   bcf     _intir
011A 8792                   bsf     EXTSTAT,MSB

                          otherbits
011B B800                   movlb   bank0                      ; get +limit,-limit and GPI
011C 6A12                   movfp   portb,wreg
011D 190A                   rrcf    wreg                       ; arrange in correct bit posi
011E B561                   andlw   0x61
011F 4A18                   movpf   wreg,TMP
0120 1D18                   swapf   TMP
0121 0818                   iorwf   TMP,W
0122 0992                   iorwf   EXTSTAT                    ; set in EXTSTAT

0123 0002                   return

                          ;*************************************************************************
                          ;*************************************************************************
                          ; NAME:                 PollingLoop
                          ;
                          ; DESCRIPTION:          The actual polling loop called after the board's
                          ;                       initialization
                          ;
                          ; ENTRY CONDITIONS:     System globals and hardware initialized and the
                          ;                       interrupt processes started.
                          ;

                          PollingLoop

                           if _SERIAL_IO
0124 E557                         call    IdleFunction
0125 E681                         call    GetChk
0126 31C2                         cpfseq  ONE                  ; GetChk, is receive buffer full?
0127 C124                         goto    PollingLoop

0128 E676                         call    GetChar              ; if so, get character
0129 4A39                         movpf   wreg,CMDCHAR         ; put in CMDCHAR
012A C559                         goto    DoCommand
                           else
                                  clrwdt
                                  goto    PollingLoop          ; wait for Interrupt
                           endif

                          ;*************************************************************************

                                              include "mult.asm"   ; Double Precision
                          Multiplication Routine
                          ;*************************************************************************
                          ;                       Double Precision Multiplication Routine
                          ;
                          ; NAME:         Dmult
                          ;
                          ; DESCRIPTION:  Mult: AARG (16 bits) * BARG (16 bits) -> DPX (32 bits)
                          ;
                          ;      (a) Load the 1st operand in locations AARG+B0 & AARG+B1 (16 bits)
                          ;      (b) Load the 2nd operand in locations BARG+B0 & BARG+B1 (16 bits)
                          ;      (c) call Dmult
                          ;      (d) The 32 bit result is in locations (DPX+B0,DPX+B1,DPX+B2,DPX+B3)
                          ;
                          ;       In the signed case, a savings of 9 clks can be realized by choosing
```

```
                    ;       BARG as the positive factor in the product when possible.
                    ;
                    ; TIMING (worst case):  unsigned:                    173 clks
                    ;                       signed:    if BARG positive:  170 clks
                    ;                                  if BARG negative:  179 clks
                    ;

                    ;***************************************************************************

0001                SIGNED  equ     TRUE            ; Set This To 'TRUE' for signed multiply
                                                    ; and 'FALSE' for unsigned.
                    ;***************************************************************************
                    ;               Multiplication Macro
                    ;***************************************************************************
                    ;
                    ; TIMING:       unsigned:     11+7*10+8*11   = 169 clks
                    ;(worst case)   signed:       11+7*10+7*11+5 = 163 clks
                    ;
                    MULTMAC  MACRO
                             variable i
                                    i = 0
                               if   SIGNED
                                  while i < 15
                               else
                                  while i < 16
                               endif
                                  if i < 8
                                    btfsc      BARG+B0,i          ; test low byte
                                    else
                                    btfsc      BARG+B1,i-8        ; test high byte
                                endif
                                    goto    add#v(i)
                                  if i < 8
                                    rlcf    DPX+B3,W              ; rotate sign into carry bit
                                    rrcf    DPX+B3                ; for i < 8, no meaningful
                                    rrcf    DPX+B2                ; are in DPX+B0
                                    rrcf    DPX+B1
                                  else
                                    rlcf    DPX+B3,W              ; rotate sign into carry bit
                                    rrcf    DPX+B3
                                    rrcf    DPX+B2
                                    rrcf    DPX+B1
                                    rrcf    DPX+B0
                                  endif
                                    i = i+1
                                  endw
                                    clrf    DPX+B0                ; if we get here, BARG = 0
                                    return
                             add0
                                    movfp   AARG+B0,wreg
                                    addwf   DPX+B2                ;add lsb
                                    movfp   AARG+B1,wreg
                                    addwfc  DPX+B3                ;add msb
                                    rlcf    AARG+B1,W             ; rotate sign into carry bit
                                    rrcf    DPX+B3                ; for i < 8, no meaningful
                                    rrcf    DPX+B2                ; are in DPX+B0
                                    rrcf    DPX+B1
                               i = 1
                               if   SIGNED
                                      while i < 15
                               else
                                  while i < 16
                               endif
                                  if i < 8
                                    btfss      BARG+B0,i          ; test low byte
                                  else
                                    btfss      BARG+B1,i-8        ; test high byte
                                  endif
                                    goto    noadd#v(i)
```

```
                                  add#v(i)
                                  movfp   AARG+B0,wreg
                                  addwf   DPX+B2                ;add lsb
                                  movfp   AARG+B1,wreg
                                  addwfc  DPX+B3                ;add msb
                             noadd#v(i)
                                  if i < 8
                                    rlcf    AARG+B1,W           ; rotate sign into carry bit
                                    rrcf    DPX+B3              ; for i < 8, no meaningful
                                    rrcf    DPX+B2              ; are in DPX+B0
                                    rrcf    DPX+B1
                                  else
                                    rlcf    AARG+B1,W           ; rotate sign into carry bit
                                    rrcf    DPX+B3
                                    rrcf    DPX+B2
                                    rrcf    DPX+B1
                                    rrcf    DPX+B0
                                  endf
                                  i = i+1
                                  endw
                                if   SIGNED
                                    rlcf    AARG+B1,W           ; since BARG is always made
                                    rrcf    DPX+B3              ; the last bit is known to be
                                    rrcf    DPX+B2
                                    rrcf    DPX+B1
                                    rrcf    DPX+B0
                                 endif
                             ENDM
                                                                ; Double Precision Multiply (
                                                                ; ( AARG*BARG -> : 32 bit
                        ;
                        Dmult
                             if   SIGNED
012B 971F                       btfss   BARG+B1,MSB             ; test sign of BARG
012C C137                       goto    argsok                  ; if positive, ok
                                        NEG16   AARG+B0         ; if negative, then negate

012D 131C                       COMF    AARG+B0+B0
012E 131D                       COMF    AARG+B0+B1
012F 290A                       CLRF    wreg
0130 151C                       INCF    AARG+B0+B0
0131 111D                       ADDWFC  AARG+B0+B1
                                        NEG16   BARG+B0         ; AARG and BARG
0132 131E                       COMF    BARG+B0+B0
0133 131F                       COMF    BARG+B0+B1
0134 290A                       CLRF    wreg
0135 151E                       INCF    BARG+B0+B0
0136 111F                       ADDWFC  BARG+B0+B1
                        endif
                        argsok
0137 291B                       clrf    DPX+B3          ; clear initial partial product
0138 291A                       clrf    DPX+B2
                      MULTMAC                           ; use macro for multiplication
0000                            variable i
0000                                   i = 0
                                if   SIGNED
                                   while i < 15
                                                        else
                                   while i < 16
                                endif
                                    if i < 8
                                      btfsc      BARG+B0,i          ; test low byte
                                        else
                                      btfsc      BARG+B1,i-8        ; test high byte
                                    endif
                                      goto    add#v(i)
                                    if i < 8
                                      rlcf    DPX+B3,W            ; rotate sign into carry bit
                                      rrcf    DPX+B3              ; for i < 8, no meaningful bits
```

```
                                    rrcf    DPX+B2              ; are in DPX+B0
                                    rrcf    DPX+B1
                                  else
                                    rlcf    DPX+B3,W            ; rotate sign into carry bit
                                    rrcf    DPX+B3
                                    rrcf    DPX+B2
                                    rrcf    DPX+B1
                                    rrcf    DPX+B0
                                    endif
                             i = i+1
                                  endw
                                if i < 8
0139 981E                          btfsc      BARG+B0,i         ; test low byte
                                     else
                                   btfsc      BARG+B1,i-8       ; test high byte
                                endif
013A C19C                          goto    add0
                                if i < 8
013B 1A1B                          rlcf    DPX+B3,W            ; rotate sign into carry bit
013C 191B                          rrcf    DPX+B3              ; for i < 8, no meaningful
013D 191A                          rrcf    DPX+B2              ; are in DPX+B0
013E 1919                          rrcf    DPX+B1
                                else
                                   rlcf    DPX+B3,W            ; rotate sign into carry bit
                                   rrcf    DPX+B3
                                   rrcf    DPX+B2
                                   rrcf    DPX+B1
                                   rrcf    DPX+B0
                                endif
0001                               i = i+1
                                if i < 8
013F 991E                          btfsc      BARG+B0,i         ; test low byte
                                     else
                                   btfsc      BARG+B1,i-8       ; test high byte
                                endif
0140 C1A6                          goto    add1
                                if i < 8
0141 1A1B                          rlcf    DPX+B3,W            ; rotate sign into carry bit
0142 191B                          rrcf    DPX+B3              ; for i < 8, no meaningful
0143 191A                          rrcf    DPX+B2              ; are in DPX+B0
0144 1919                          rrcf    DPX+B1
                                else
                                   rlcf    DPX+B3,W            ; rotate sign into carry bit
                                   rrcf    DPX+B3
                                   rrcf    DPX+B2
                                   rrcf    DPX+B1
                                   rrcf    DPX+B0

                                endif

0002                               i = i+1
                                if i < 8
0145 9A1E                          btfsc      BARG+B0,i         ; test low byte
                                     else
                                   btfsc      BARG+B1,i-8       ; test high byte

                                endif

0146 C1B0                          goto    add2
                                if i < 8
0147 1A1B                          rlcf    DPX+B3,W            ; rotate sign into carry bit
0148 191B                          rrcf    DPX+B3              ; for i < 8, no meaningful
0149 191A                          rrcf    DPX+B2              ; are in DPX+B0
014A 1919                          rrcf    DPX+B1
                                else
                                   rlcf    DPX+B3,W            ; rotate sign into carry bit
                                   rrcf    DPX+B3
                                   rrcf    DPX+B2
                                   rrcf    DPX+B1
```

```
                              rrcf    DPX+B0
                           endif
0003                          i = i+1
                           if i < 8
014B 9B1E                     btfsc      BARG+B0,i          ; test low byte
                                else
                              btfsc      BARG+B1,i-8        ; test high byte
                           endif
014C C1BA                     goto    add3
                           if i < 8
014D 1A1B                     rlcf    DPX+B3,W              ; rotate sign into carry bit
014E 191B                     rrcf    DPX+B3                ; for i < 8, no meaningful
014F 191A                     rrcf    DPX+B2                ; are in DPX+B0
0150 1919                     rrcf    DPX+B1
                           else
                              rlcf    DPX+B3,W              ; rotate sign into carry bit
                              rrcf    DPX+B3
                              rrcf    DPX+B2
                              rrcf    DPX+B1
                              rrcf    DPX+B0
                           endif
0004                          i = i+1
                           if i < 8
0151 9C1E                     btfsc      BARG+B0,i          ; test low byte
                                else
                              btfsc      BARG+B1,i-8        ; test high byte
                           endif
0152 C1C4                     goto    add4
                           if i < 8
0153 1A1B                     rlcf    DPX+B3,W              ; rotate sign into carry bit
0154 191B                     rrcf    DPX+B3                ; for i < 8, no meaningful
0155 191A                     rrcf    DPX+B2                ; are in DPX+B0
0156 1919                     rrcf    DPX+B1
                           else
                              rlcf    DPX+B3,W              ; rotate sign into carry bit
                              rrcf    DPX+B3
                              rrcf    DPX+B2
                              rrcf    DPX+B1
                              rrcf    DPX+B0
                           endif
0005                          i = i+1
                           if i < 8
0157 9D1E                     btfsc      BARG+B0,i          ; test low byte
                                else
                              btfsc      BARG+B1,i-8        ; test high byte
                           endif
0158 C1CE                     goto    add5
                           if i < 8
0159 1A1B                     rlcf    DPX+B3,W              ; rotate sign into carry bit
015A 191B                     rrcf    DPX+B3                ; for i < 8, no meaningful
015B 191A                     rrcf    DPX+B2                ; are in DPX+B0
015C 1919                     rrcf    DPX+B1
                           else
                              rlcf    DPX+B3,W              ; rotate sign into carry bit
                              rrcf    DPX+B3
                              rrcf    DPX+B2
                              rrcf    DPX+B1
                              rrcf    DPX+B0
                           endif
0006                          i = i+1
                           if i < 8
015D 9E1E                     btfsc      BARG+B0,i          ; test low byte
                                else
                              btfsc      BARG+B1,i-8        ; test high byte
                           endif
015E C1D8                     goto    add6
                           if i < 8
015F 1A1B                     rlcf    DPX+B3,W              ; rotate sign into carry bit
0160 191B                     rrcf    DPX+B3                ; for i < 8, no meaningful
```

```
0161 191A                           rrcf    DPX+B2              ; are in DPX+B0
0162 1919                           rrcf    DPX+B1
                                  else
                                    rlcf    DPX+B3,W            ; rotate sign into carry bit
                                    rrcf    DPX+B3
                                    rrcf    DPX+B2
                                    rrcf    DPX+B1
                                    rrcf    DPX+B0
                                  endif
0007                                i = i+1
                                  if i < 8
0163 9F1E                           btfsc       BARG+B0,i           ; test low byte
                                      else
                                    btfsc       BARG+B1,i-8         ; test high byte
                                  endif
0164 C1E2                           goto    add7
                                  if i < 8
0165 1A1B                           rlcf    DPX+B3,W            ; rotate sign into carry bit
0166 191B                           rrcf    DPX+B3              ; for i < 8, no meaningful
0167 191A                           rrcf    DPX+B2              ; are in DPX+B0
0168 1919                           rrcf    DPX+B1
                                  else
                                    rlcf    DPX+B3,W            ; rotate sign into carry bit
                                    rrcf    DPX+B3
                                    rrcf    DPX+B2
                                    rrcf    DPX+B1
                                    rrcf    DPX+B0
                                  endif
0008                                i = i+1
                                  if i < 8
                                    btfsc       BARG+B0,i           ; test low byte
                                       else
0169 981F                           btfsc       BARG+B1,i-8         ; test high byte
                                  endif
016A C1EC                           goto    add8
                                  if i < 8
                                    rlcf    DPX+B3,W            ; rotate sign into carry bit
                                    rrcf    DPX+B3              ; for i < 8, no meaningful
                                    rrcf    DPX+B2              ; are in DPX+B0
                                    rrcf    DPX+B1
                                  else
016B 1A1B                           rlcf    DPX+B3,W            ; rotate sign into carry bit
016C 191B                           rrcf    DPX+B3
016D 191A                           rrcf    DPX+B2
016E 1919                           rrcf    DPX+B1
016F 1918                           rrcf    DPX+B0
                                  endif
0009                                i = i+1
                                  if i < 8
                                    btfsc       BARG+B0,i           ; test low byte
                                    else
0170 991F                           btfsc       BARG+B1,i-8         ; test high byte
                                  endif
0171 C1F7                           goto    add9
                                  if i < 8
                                    rlcf    DPX+B3,W            ; rotate sign into carry bit
                                    rrcf    DPX+B3              ; for i < 8, no meaningful
                                    rrcf    DPX+B2              ; are in DPX+B0
                                    rrcf    DPX+B1
                                  else
0172 1A1B                           rlcf    DPX+B3,W            ; rotate sign into carry bit
0173 191B                           rrcf    DPX+B3
0174 191A                           rrcf    DPX+B2
0175 1919                           rrcf    DPX+B1
0176 1918                           rrcf    DPX+B0
                                  endif
000A                                i = i+1
                                  if i < 8
                                    btfsc       BARG+B0,i           ; test low byte
```

```
                                    else
0177 9A1F                              btfsc      BARG+B1,i-8         ; test high byte
                                    endif
0178 C202                              goto    add10
                                    if i < 8
                                       rlcf    DPX+B3,W               ; rotate sign into carry bit
                                       rrcf    DPX+B3                 ; for i < 8, no meaningful
                                       rrcf    DPX+B2                 ; are in DPX+B0
                                       rrcf    DPX+B1
                                    else
0179 1A1B                              rlcf    DPX+B3,W               ; rotate sign into carry bit
017A 191B                              rrcf    DPX+B3
017B 191A                              rrcf    DPX+B2
017C 1919                              rrcf    DPX+B1
017D 1918                              rrcf    DPX+B0
                                    endif
000B                                   i = i+1
                                    if i < 8
                                       btfsc      BARG+B0,i           ; test low byte
                                         else
017E 9B1F                              btfsc      BARG+B1,i-8         ; test high byte
                                    endif
017F C20D                              goto    add11
                                    if i < 8
                                       rlcf    DPX+B3,W               ; rotate sign into carry bit
                                       rrcf    DPX+B3                 ; for i < 8, no meaningful
                                       rrcf    DPX+B2                 ; are in DPX+B0
                                       rrcf    DPX+B1
                                    else
0180 1A1B                              rlcf    DPX+B3,W               ; rotate sign into carry bit
0181 191B                              rrcf    DPX+B3
0182 191A                              rrcf    DPX+B2
0183 1919                              rrcf    DPX+B1
0184 1918                              rrcf    DPX+B0
                                    endif
000C                                   i = i+1
                                    if i < 8
                                       btfsc      BARG+B0,i           ; test low byte
                                         else
0185 9C1F                              btfsc      BARG+B1,i-8         ; test high byte
                                    endif
0186 C218                              goto    add12
                                    if i < 8
                                       rlcf    DPX+B3,W               ; rotate sign into carry bit
                                       rrcf    DPX+B3                 ; for i < 8, no meaningful
                                       rrcf    DPX+B2                 ; are in DPX+B0
                                       rrcf    DPX+B1
                                    else
0187 1A1B                              rlcf    DPX+B3,W               ; rotate sign into carry bit
0188 191B                              rrcf    DPX+B3
0189 191A                              rrcf    DPX+B2
018A 1919                              rrcf    DPX+B1
018B 1918                              rrcf    DPX+B0
                                    endif
000D                                   i = i+1
                                    if i < 8
                                       btfsc      BARG+B0,i           ; test low byte
                                         else
018C 9D1F                              btfsc      BARG+B1,i-8         ; test high byte
                                    endif
018D C223                              goto    add13
                                    if i < 8
                                       rlcf    DPX+B3,W               ; rotate sign into carry bit
                                       rrcf    DPX+B3                 ; for i < 8, no meaningful
                                       rrcf    DPX+B2                 ; are in DPX+B0
                                       rrcf    DPX+B1
                                    else
018E 1A1B                              rlcf    DPX+B3,W               ; rotate sign into carry bit
018F 191B                              rrcf    DPX+B3
```

```
0190 191A                           rrcf    DPX+B2
0191 1919                           rrcf    DPX+B1
0192 1918                           rrcf    DPX+B0
                                 endif
000E                                i = i+1
                                 if i < 8
                                    btfsc      BARG+B0,i          ; test low byte
                                      else
0193 9E1F                           btfsc      BARG+B1,i-8        ; test high byte
                                 endif
0194 C22E                           goto    add14
                                 if i < 8
                                    rlcf    DPX+B3,W              ; rotate sign into carry bit
                                    rrcf    DPX+B3                ; for i < 8, no meaningful
                                    rrcf    DPX+B2                ; are in DPX+B0
                                    rrcf    DPX+B1
                                 else
0195 1A1B                           rlcf    DPX+B3,W              ; rotate sign into carry bit
0196 191B                           rrcf    DPX+B3
0197 191A                           rrcf    DPX+B2
0198 1919                           rrcf    DPX+B1
0199 1918                           rrcf    DPX+B0
                                 endif
000F                                i = i+1
019A 2918                           clrf    DPX+B0                ; if we get here, BARG = 0
019B 0002                           return
                        add0
019C 6A1C                           movfp   AARG+B0,wreg
019D 0F1A                           addwf   DPX+B2                ;add lsb
019E 6A1D                           movfp   AARG+B1,wreg
019F 111B                           addwfc  DPX+B3  ;add msb
01A0 1A1D                           rlcf    AARG+B1,W             ; rotate sign into carry bit
01A1 191B                           rrcf    DPX+B3                ; for i < 8, no meaningful
01A2 191A                           rrcf    DPX+B2                ; are in DPX+B0
01A3 1919                           rrcf    DPX+B1
0001                             i = 1
                                 if   SIGNED
                                        while i < 15
                                  else
                                    while i < 16
                                 endif
                                     if i < 8
                                       btfss      BARG+B0,i   ; test low byte
                                     else
                                       btfss      BARG+B1,i-8 ; test high byte
                                     endif
                                       goto    noadd#v(i)
                        add#v(i)
                                       movfp   AARG+B0,wreg
                                       addwf   DPX+B2 ;add lsb
                                       movfp   AARG+B1,wreg
                                       addwfc  DPX+B3 ;add msb
                        noadd#v(i)
                                     if i < 8
                                       rlcf    AARG+B1,W       ; rotate sign into carry bit
                                       rrcf    DPX+B3          ; for i < 8, no meaningful bits
                                       rrcf    DPX+B2          ; are in DPX+B0
                                       rrcf    DPX+B1
                                     else
                                       rlcf    AARG+B1,W       ; rotate sign into carry bit
                                       rrcf    DPX+B3
                                       rrcf    DPX+B2
                                       rrcf    DPX+B1
                                       rrcf    DPX+B0
                                     endif
                                     i = i+1
                                     endw
                                  if i < 8
01A4 911E                            btfss      BARG+B0,i    ; test low byte
```

```
                                  else
                                     btfss      BARG+B1,i-8  ; test high byte
                                  endif
01A5 C1AA                            goto    noadd1
                        add1
01A6 6A1C                            movfp   AARG+B0,wreg
01A7 0F1A                            addwf   DPX+B2         ;add lsb
01A8 6A1D                            movfp   AARG+B1,wreg
01A9 111B                            addwfc  DPX+B3         ;add msb
                        noadd1
                                  if i < 8
01AA 1A1D                            rlcf    AARG+B1,W      ; rotate sign into carry bit
01AB 191B                            rrcf    DPX+B3         ; for i < 8, no meaningful bits
01AC 191A                            rrcf    DPX+B2         ; are in DPX+B0
01AD 1919                            rrcf    DPX+B1
                                  else
                                     rlcf    AARG+B1,W      ; rotate sign into carry bit
                                     rrcf    DPX+B3
                                     rrcf    DPX+B2
                                     rrcf    DPX+B1
                                     rrcf    DPX+B0
                                  endif
0002                              i = i+1
                                  if i < 8
01AE 921E                            btfss      BARG+B0,i           ; test low byte
                                  else
                                     btfss      BARG+B1,i-8         ; test high byte
                                  endif
01AF C1B4                            goto    noadd2                      add2
01B0 6A1C                            movfp   AARG+B0,wreg
01B1 0F1A                            addwf   DPX+B2                 ;add lsb
01B2 6A1D                            movfp   AARG+B1,wreg
01B3 111B                            addwfc  DPX+B3                 ;add msb
                        noadd2
                                  if i < 8
01B4 1A1D                            rlcf    AARG+B1,W              ; rotate sign into carry bit
01B5 191B                            rrcf    DPX+B3                 ; for i < 8, no meaningful
01B6 191A                            rrcf    DPX+B2                 ; are in DPX+B0
01B7 1919                            rrcf    DPX+B1
                                  else
                                     rlcf    AARG+B1,W              ; rotate sign into carry bit
                                     rrcf    DPX+B3
                                     rrcf    DPX+B2
                                     rrcf    DPX+B1
                                     rrcf    DPX+B0
                               endif
0003                              i = i+1
                                  if i < 8
01B8 931E                            btfss      BARG+B0,i           ; test low byte
                                  else
                                     btfss      BARG+B1,i-8         ; test high byte
                                  endif
01B9 C1BE                            goto    noadd3
                        add3
01BA 6A1C                            movfp   AARG+B0,wreg
01BB 0F1A                            addwf   DPX+B2                 ;add lsb
01BC 6A1D                            movfp   AARG+B1,wreg
01BD 111B                            addwfc  DPX+B3                 ;add msb
                        noadd3
                                  if i < 8
01BE 1A1D                            rlcf    AARG+B1,W              ; rotate sign into carry bit
01BF 191B                            rrcf    DPX+B3                 ; for i < 8, no meaningful
01C0 191A                            rrcf    DPX+B2                 ; are in DPX+B0
01C1 1919                            rrcf    DPX+B1
                                  else
                                     rlcf    AARG+B1,W              ; rotate sign into carry bit
                                     rrcf    DPX+B3
                                     rrcf    DPX+B2
                                     rrcf    DPX+B1
                                     rrcf    DPX+B0
```

```
                    endif
0004                          i = i+1
                              if i < 8
01C2 941E                        btfss      BARG+B0,i          ; test low byte
                              else
                                 btfss      BARG+B1,i-8        ; test high byte
                              endif
01C3 C1C8                        goto    noadd4
                    add4
01C4 6A1C                        movfp   AARG+B0,wreg
01C5 0F1A                        addwf   DPX+B2              ;add lsb
01C6 6A1D                        movfp   AARG+B1,wreg
01C7 111B                        addwfc  DPX+B3              ;add msb
                    noadd4
                              if i < 8
01C8 1A1D                        rlcf    AARG+B1,W           ; rotate sign into carry bit
01C9 191B                        rrcf    DPX+B3              ; for i < 8, no meaningful
01CA 191A                        rrcf    DPX+B2              ; are in DPX+B0
01CB 1919                        rrcf    DPX+B1
                              else
                                 rlcf    AARG+B1,W           ; rotate sign into carry bit
                                 rrcf    DPX+B3
                                 rrcf    DPX+B2
                                 rrcf    DPX+B1
                                 rrcf    DPX+B0
                           endif
0005                          i = i+1
                              if i < 8
01CC 951E                        btfss      BARG+B0,i          ; test low byte
                              else
                                 btfss      BARG+B1,i-8        ; test high byte
                              endif
01CD C1D2                        goto    noadd5
                    add5
01CE 6A1C                        movfp   AARG+B0,wreg
01CF 0F1A                        addwf   DPX+B2  ;add lsb
01D0 6A1D                        movfp   AARG+B1,wreg
01D1 111B                        addwfc  DPX+B3  ;add msb
                    noadd5
                              if i < 8
01D2 1A1D                        rlcf    AARG+B1,W           ; rotate sign into carry bit
01D3 191B                        rrcf    DPX+B3              ; for i < 8, no meaningful
01D4 191A                        rrcf    DPX+B2              ; are in DPX+B0
01D5 1919                        rrcf    DPX+B1
                              else
                                 rlcf    AARG+B1,W           ; rotate sign into carry bit
                                 rrcf    DPX+B3
                                 rrcf    DPX+B2
                                 rrcf    DPX+B1

                                 rrcf    DPX+B0



                              endif


0006                          i = i+1

                              if i < 8
01D6 961E                        btfss      BARG+B0,i          ; test low byte
                              else
                                 btfss      BARG+B1,i-8        ; test high byte

                              endif

01D7 C1DC                        goto    noadd6
                    add6
01D8 6A1C                        movfp   AARG+B0,wreg
```

```
01D9 0F1A                           addwf   DPX+B2  ;add lsb
01DA 6A1D                           movfp   AARG+B1,wreg
01DB 111B                           addwfc  DPX+B3  ;add msb
                    noadd6
                                if i < 8
01DC 1A1D                           rlcf    AARG+B1,W             ; rotate sign into carry bit
01DD 191B                           rrcf    DPX+B3                ; for i < 8, no meaningful
01DE 191A                           rrcf    DPX+B2                ; are in DPX+B0
01DF 1919                           rrcf    DPX+B1
                                else
                                    rlcf    AARG+B1,W             ; rotate sign into carry bit
                                    rrcf    DPX+B3
                                    rrcf    DPX+B2
                                    rrcf    DPX+B1
                                    rrcf    DPX+B0
                                endif
0007                            i = i+1
                                if i < 8
01E0 971E                           btfss       BARG+B0,i         ; test low byte
                                else
                                    btfss       BARG+B1,i-8       ; test high byte
                                endif
01E1 C1E6                           goto    noadd7
                    add7
01E2 6A1C                           movfp   AARG+B0,wreg
01E3 0F1A                           addwf   DPX+B2  ;add lsb
01E4 6A1D                           movfp   AARG+B1,wreg
01E5 111B                           addwfc  DPX+B3  ;add msb
                    noadd7
                                if i < 8

01E6 1A1D                           rlcf    AARG+B1,W             ; rotate sign into carry bit
01E7 191B                           rrcf    DPX+B3                ; for i < 8, no meaningful
01E8 191A                           rrcf    DPX+B2                ; are in DPX+B0
01E9 1919                           rrcf    DPX+B1
                                else
                                    rlcf    AARG+B1,W             ; rotate sign into carry bit
                                    rrcf    DPX+B3
                                    rrcf    DPX+B2
                                    rrcf    DPX+B1
                                    rrcf    DPX+B0
                                endif
0008                            i = i+1
                                if i < 8
                                    btfss       BARG+B0,i         ; test low byte
                                else
01EA 901F                           btfss       BARG+B1,i-8       ; test high byte
                                endif
01EB C1F0                           goto    noadd8
                    add8
01EC 6A1C                           movfp   AARG+B0,wreg
01ED 0F1A                           addwf   DPX+B2  ;add lsb
01EE 6A1D                           movfp   AARG+B1,wreg
01EF 111B                           addwfc  DPX+B3  ;add msb
                    noadd8
                                if i < 8
                                    rlcf    AARG+B1,W             ; rotate sign into carry bit
                                    rrcf    DPX+B3                ; for i < 8, no meaningful
                                    rrcf    DPX+B2                ; are in DPX+B0
                                    rrcf    DPX+B1
                                else
01F0 1A1D                           rlcf    AARG+B1,W             ; rotate sign into carry bit
01F1 191B                           rrcf    DPX+B3
01F2 191A                           rrcf    DPX+B2
01F3 1919                           rrcf    DPX+B1
01F4 1918                           rrcf    DPX+B0
                                endif
0009                            i = i+1
```

```
                                 if i < 8
                                    btfss      BARG+B0,i          ; test low byte
                                 else
01F5 911F                           btfss      BARG+B1,i-8        ; test high byte
                                 endif
01F6 C1FB                           goto    noadd9
                       add9
01F7 6A1C                           movfp   AARG+B0,wreg
01F8 0F1A                           addwf   DPX+B2  ;add lsb
01F9 6A1D                           movfp   AARG+B1,wreg
01FA 111B                           addwfc  DPX+B3  ;add msb
                       noadd9
                                 if i < 8
                                    rlcf    AARG+B1,W          ; rotate sign into carry bit
                                    rrcf    DPX+B3             ; for i < 8, no meaningful
                                    rrcf    DPX+B2             ; are in DPX+B0
                                    rrcf    DPX+B1
                                 else
01FB 1A1D                           rlcf    AARG+B1,W          ; rotate sign into carry bit
01FC 191B                           rrcf    DPX+B3
01FD 191A                           rrcf    DPX+B2
01FE 1919                           rrcf    DPX+B1
01FF 1918                           rrcf    DPX+B0
                                 endif
000A                             i = i+1
                                if i < 8
                                    btfss      BARG+B0,i          ; test low byte
                                 else
0200 921F                           btfss      BARG+B1,i-8        ; test high byte
                                 endif
0201 C206                           goto    noadd10
                       add10
0202 6A1C                           movfp   AARG+B0,wreg
0203 0F1A                           addwf   DPX+B2  ;add lsb
0204 6A1D                           movfp   AARG+B1,wreg
0205 111B                           addwfc  DPX+B3  ;add msb
                       noadd10
                                 if i < 8
                                    rlcf    AARG+B1,W          ; rotate sign into carry bit
                                    rrcf    DPX+B3             ; for i < 8, no meaningful
                                    rrcf    DPX+B2             ; are in DPX+B0
                                    rrcf    DPX+B1
                                 else
0206 1A1D                           rlcf    AARG+B1,W          ; rotate sign into carry bit
0207 191B                           rrcf    DPX+B3
0208 191A                           rrcf    DPX+B2
0209 1919                           rrcf    DPX+B1
020A 1918                           rrcf    DPX+B0
                                 endif
000B                             i = i+1
                                 if i < 8
                                    btfss      BARG+B0,i          ; test low byte
                                 else
020B 931F                           btfss      BARG+B1,i-8        ; test high byte
                                 endif
020C C211                           goto    noadd11
                       add11
020D 6A1C                           movfp   AARG+B0,wreg
020E 0F1A                           addwf   DPX+B2  ;add lsb
020F 6A1D                           movfp   AARG+B1,wreg
0210 111B                           addwfc  DPX+B3  ;add msb
                       noadd11
                                 if i < 8
                                    rlcf    AARG+B1,W          ; rotate sign into carry bit
                                    rrcf    DPX+B3             ; for i < 8, no meaningful
                                    rrcf    DPX+B2             ; are in DPX+B0
                                    rrcf    DPX+B1
                                 else
```

```
0211 1A1D                          rlcf    AARG+B1,W             ; rotate sign into carry bit
0212 191B                          rrcf    DPX+B3
0213 191A                          rrcf    DPX+B2
0214 1919                          rrcf    DPX+B1
0215 1918                          rrcf    DPX+B0
                                endif
000C                            i = i+1
                                if i < 8
                                   btfss      BARG+B0,i          ; test low byte
                                else
0216 941F                          btfss      BARG+B1,i-8        ; test high byte
                                endif
0217 C21C                          goto    noadd12
                     add12
0218 6A1C                          movfp   AARG+B0,wreg
0219 0F1A                          addwf   DPX+B2  ;add lsb
021A 6A1D                          movfp   AARG+B1,wreg
021B 111B                          addwfc  DPX+B3  ;add msb
                     noadd12
                                if i < 8
                                   rlcf    AARG+B1,W             ; rotate sign into carry bit
                                   rrcf    DPX+B3                ; for i < 8, no meaningful
                                   rrcf    DPX+B2                ; are in DPX+B0
                                   rrcf    DPX+B1
                                else
021C 1A1D                          rlcf    AARG+B1,W             ; rotate sign into carry bit
021D 191B                          rrcf    DPX+B3
021E 191A                          rrcf    DPX+B2
021F 1919                          rrcf    DPX+B1
0220 1918                          rrcf    DPX+B0
                                endif
000D                            i = i+1
                                if i < 8
                                   btfss      BARG+B0,i          ; test low byte
                                else
221 951F                          btfss      BARG+B1,i-8         ; test high byte
                                endif
0222 C227                          goto    noadd13
                     add13
0223 6A1C                          movfp   AARG+B0,wreg
0224 0F1A                          addwf   DPX+B2  ;add lsb
0225 6A1D                          movfp   AARG+B1,wreg
0226 111B                          addwfc  DPX+B3  ;add msb
                     noadd13
                                if i < 8
                                   rlcf    AARG+B1,W             ; rotate sign into carry bit
                                   rrcf    DPX+B3                ; for i < 8, no meaningful
                                   rrcf    DPX+B2                ; are in DPX+B0
                                   rrcf    DPX+B1
                                else
0227 1A1D                          rlcf    AARG+B1,W             ; rotate sign into carry bit
0228 191B                          rrcf    DPX+B3
0229 191A                          rrcf    DPX+B2
022A 1919                          rrcf    DPX+B1
022B 1918                          rrcf    DPX+B0
                                endif
000E                            i = i+1
                                if i < 8
                                   btfss      BARG+B0,i          ; test low byte
                                else
022C 961F                          btfss      BARG+B1,i-8        ; test high byte
                                endif
022D C232                          goto    noadd14
                     add14
022E 6A1C                          movfp   AARG+B0,wreg
022F 0F1A                          addwf   DPX+B2                ;add lsb
0230 6A1D                          movfp   AARG+B1,wreg
0231 111B                          addwfc  DPX+B3                ;add msb
```

```
                    noadd14
                                if i < 8
                                   rlcf    AARG+B1,W             ; rotate sign into carry bit
                                   rrcf    DPX+B3                ; for i < 8, no meaningful
                                   rrcf    DPX+B2                ; are in DPX+B0
                                   rrcf    DPX+B1
                                else
0232 1A1D                          rlcf    AARG+B1,W             ; rotate sign into carry bit
0233 191B                          rrcf    DPX+B3
0234 191A                          rrcf    DPX+B2
0235 1919                          rrcf    DPX+B1
0236 1918                          rrcf    DPX+B0
                                endif
000F                            i = i+1
                              if   SIGNED
0237 1A1D                          rlcf    AARG+B1,W             ; since BARG is always made
0238 191B                          rrcf    DPX+B3                ; the last bit is known to be
0239 191A                          rrcf    DPX+B2
023A 1919                          rrcf    DPX+B1
023B 1918                          rrcf    DPX+B0
                              endif

023C 0002                   return

                    ;*****************************************************************************

                                        include “traject.asm”  ;
                    Trajectory Generation
                    ;*****************************************************************************
                    ;
                    ;
                    ;                                  Trajectory Generation Routines
                    ;
                    ;*****************************************************************************


                    ;*****************************************************************************
                    ; NAME:         doPreMove
                    ;
                    ; DESCRIPTION:

                    doPreMove:

                    CLR16   INTEGRAL

023D 2996                   CLRF    INTEGRAL+B0
023E 2997                   CLRF    INTEGRAL+B1

                    MOV24   NMOVVAL,MOVVAL                      ; move buffer to MOVVAL

023F 6A5B                   MOVFP   NMOVVAL+B0,wreg             ; get byte of NMOVVAL into w
0240 4A5F                   MOVPF   wreg,MOVVAL+B0              ; move to MOVVAL(B0)
0241 6A5C                   MOVFP   NMOVVAL+B1,wreg             ; get byte of NMOVVAL into w
0242 4A60                   MOVPF   wreg,MOVVAL+B1              ; move to MOVVAL(B1)
0243 6A5D                   MOVFP   NMOVVAL+B2,wreg             ; get byte of NMOVVAL into w
0244 4A61                   MOVPF   wreg,MOVVAL+B2              ; move to MOVVAL(B2)

0245 8F93                   bcf     MOVSTAT,bit7                ; clear buffer flag
0246 8693                   bsf     MOVSTAT,bit6                ; set motion status flag
0247 8593                   bsf     MOVSTAT,bit5                ; set move in progress flag
0248 6AC2                   movfp   ONE,wreg
0249 4A94                   movpf   wreg,MOVFLAG                ; initialize MOVEFLAG to 1


024A 2951                   clrf    OPOSITION+B0                ; initialize buffers
                    MOV24   POSITION,OPOSITION+B1

024B 6A55                   MOVFP   POSITION+B0,wreg            ; get byte of POSITION into w
024C 4A52                   MOVPF   wreg,OPOSITION+B1+B0        ; move to OPOSITION+B1(B0)
024D 6A56                   MOVFP   POSITION+B1,wreg            ; get byte of POSITION into w
```

```
024E 4A53                MOVPF   wreg,OPOSITION+B1+B1       ; move to OPOSITION+B1(B1)
024F 6A57                MOVFP   POSITION+B2,wreg           ; get byte of POSITION into w
0250 4A54                MOVPF   wreg,OPOSITION+B1+B2       ; move to OPOSITION+B1(B2)


                 MOV32   OPOSITION,MOVPBUF

0251 6A51                MOVFP   OPOSITION+B0,wreg          ; get byte of OPOSITION into w
0252 4AA4                MOVPF   wreg,MOVPBUF+B0            ; move to MOVPBUF(B0)
0253 6A52                MOVFP   OPOSITION+B1,wreg          ; get byte of OPOSITION into w
0254 4AA5                MOVPF   wreg,MOVPBUF+B1            ; move to MOVPBUF(B1)
0255 6A53                MOVFP   OPOSITION+B2,wreg          ; get byte of OPOSITION into w
0256 4AA6                MOVPF   wreg,MOVPBUF+B2            ; move to MOVPBUF(B2)
0257 6A54                MOVFP   OPOSITION+B3,wreg          ; get byte of OPOSITION into w
0258 4AA7                MOVPF   wreg,MOVPBUF+B3            ; move to MOVPBUF(B3)


0259 2995                clrf    SATFLAG
                 CLR16   MOVTIME                            ; clear move times

025A 2967                CLRF    MOVTIME+B0
025B 2968                CLRF    MOVTIME+B1


                 CLR16   T1                                 ; 0 used as flag for no maximum

025C 296A                CLRF    T1+B0
025D 296B                CLRF    T1+B1


                 CLR16   T2

025E 296C                CLRF    T2+B0
025F 296D                CLRF    T2+B1


                 CLR16   TAU

0260 296E                CLRF    TAU+B0
0261 296F                CLRF    TAU+B1


                 CLR32   MOVDEL                             ; clear move discretization error

0262 29B0                CLRF    MOVDEL+B0
0263 29B1                CLRF    MOVDEL+B1
0264 29B2                CLRF    MOVDEL+B2
0265 29B3                CLRF    MOVDEL+B3


                  CLR16   PH2FLAT                          ; clear phase 2 flat counter

0266 29B4                CLRF    PH2FLAT+B0
0267 29B5                CLRF    PH2FLAT+B1



0268 3391                tstfsz  MODETYPE
0269 C2C5                goto    vmode
                 pmode
                 MOVFP24  MOVVAL,TMP

026A 785F                MOVFP   MOVVAL+B0,TMP+B0           ; move MOVVAL(B0) to TMP(B0)
026B 7960                MOVFP   MOVVAL+B1,TMP+B1           ; move MOVVAL(B1) to TMP(B1)
026C 7A61                MOVFP   MOVVAL+B2,TMP+B2           ; move MOVVAL(B2) to TMP(B2)


026D 971A                btfss   TMP+B2,MSB
026E C276                goto    mvpos
```

```
                    NEG24   TMP

026F 1318                   COMF    TMP+B0
0270 1319                   COMF    TMP+B1
0271 131A                   COMF    TMP+B2
0272 290A                   CLRF    wreg
0273 1518                   INCF    TMP+B0
0274 1119                   ADDWFC  TMP+B1
0275 111A                   ADDWFC  TMP+B2


                    mvpos
0276 291C                   clrf    MOVTMP+B0                   ; calculate  abs(MOVVAL) - 3
0277 291D                   clrf    MOVTMP+B1                   ; do immediate move if nega-
tive
0278 291E                   clrf    MOVTMP+B2
0279 801C                   bsf     MOVTMP+B0,bit0
027A 811C                   bsf     MOVTMP+B0,bit1
                    SUB24   MOVTMP,TMP

027B 6A1C                   MOVFP   MOVTMP+B0,wreg              ; get lowest byte of MOVTMP
027C 0518                   SUBWF   TMP+B0                      ; sub lowest byte of TMP, save
027D 6A1D                   MOVFP   MOVTMP+B1,wreg              ; get 2nd byte of MOVTMP into
027E 0319                   SUBWFB  TMP+B1                      ; sub 2nd byte of TMP, save in
027F 6A1E                   MOVFP   MOVTMP+B2,wreg              ; get 3rd byte of MOVTMP into
0280 031A                   SUBWFB  TMP+B2                      ; sub 3rd byte of TMP, save in



0281 971A                   btfss   TMP+B2,MSB                  ; check for zero move
0282 C28E                   goto    nonzero
0283 2B90                   setf    SERVOFLAG                   ; set servoflag to restore
0284 2994                   clrf    MOVFLAG
0285 8D93                   bcf     MOVSTAT,bit5
0286 8E93                   bcf     MOVSTAT,bit6
                    ADD24   MOVVAL,POSITION

0287 6A5F                   MOVFP   MOVVAL+B0,wreg              ; get lowest byte of MOVVAL
0288 0F55                   ADDWF   POSITION+B0                 ; add lowest byte of POSITION,
0289 6A60                   MOVFP   MOVVAL+B1,wreg              ; get 2nd byte of MOVVAL into
028A 1156                   ADDWFC  POSITION+B1                 ; add 2nd byte of POSITION,
028B 6A61                   MOVFP   MOVVAL+B2,wreg              ; get 3rd byte of MOVVAL into
028C 1157                   ADDWFC  POSITION+B2                 ; add 3rd byte of POSITION,


028D 0002           return
                    nonzero
                    CLR32   MOVVBUF

028E 29A8                   CLRF    MOVVBUF+B0
028F 29A9                   CLRF    MOVVBUF+B1
0290 29AA                   CLRF    MOVVBUF+B2
0291 29AB                   CLRF    MOVVBUF+B3



0292 6A61                   movfp   MOVVAL+B2,wreg              ; move sign
0293 B580                   andlw   0x80
0294 4A69                   movpf   wreg,MOVSIGN

0295 29A3                   clrf    V+B3                        ; create appropriate velocity
                    MOV24   VL,V                                ; acceleration limits from

0296 6A20                   MOVFP   VL+B0,wreg                  ; get byte of VL into w
0297 4AA0                   MOVPF   wreg,V+B0                   ; move to V(B0)
0298 6A21                   MOVFP   VL+B1,wreg                  ; get byte of VL into w
0299 4AA1                   MOVPF   wreg,V+B1                   ; move to V(B1)
029A 6A22                   MOVFP   VL+B2,wreg                  ; get byte of VL into w
029B 4AA2                   MOVPF   wreg,V+B2                   ; move to V(B2)
```

```
029C 299F                clrf    A+B3
                 MOV24   AL,A

029D 6A23                MOVFP   AL+B0,wreg              ; get byte of AL into w
029E 4A9C                MOVPF   wreg,A+B0               ; move to A(B0)
029F 6A24                MOVFP   AL+B1,wreg              ; get byte of AL into w
02A0 4A9D                MOVPF   wreg,A+B1               ; move to A(B1)
02A1 6A25                MOVFP   AL+B2,wreg              ; get byte of AL into w
02A2 4A9E                MOVPF   wreg,A+B2                   ; move to A(B2)


02A3 290A                clrf    wreg
02A4 3269                cpfsgt  MOVSIGN
02A5 C2B8                goto    minc
                 NEG32   V

02A6 13A0                COMF    V+B0
02A7 13A1                COMF    V+B1
02A8 13A2                COMF    V+B2
02A9 13A3                COMF    V+B3
02AA 290A                CLRF    wreg
02AB 15A0                INCF    V+B0
02AC 11A1                ADDWFC  V+B1
02AD 11A2                ADDWFC  V+B2
02AE 11A3                ADDWFC  V+B3


                 NEG32   A

02AF 139C                COMF    A+B0
02B0 139D                COMF    A+B1
02B1 139E                COMF    A+B2
02B2 139F                COMF    A+B3
02B3 290A                CLRF    wreg
02B4 159C                INCF    A+B0
02B5 119D                ADDWFC  A+B1
02B6 119E                ADDWFC  A+B2
02B7 119F                ADDWFC  A+B3


                 minc
02B8 2963                clrf    HMOVVAL+B0              ; evaluate MOVVAL/2
                         MOV24   MOVVAL,HMOVVAL+B1

02B9 6A5F                MOVFP   MOVVAL+B0,wreg          ; get byte of MOVVAL into w
02BA 4A64                MOVPF   wreg,HMOVVAL+B1+B0      ; move to HMOVVAL+B1(B0)
02BB 6A60                MOVFP   MOVVAL+B1,wreg          ; get byte of MOVVAL into w
02BC 4A65                MOVPF   wreg,HMOVVAL+B1+B1      ; move to HMOVVAL+B1(B1)
02BD 6A61                MOVFP   MOVVAL+B2,wreg          ; get byte of MOVVAL into w
02BE 4A66                MOVPF   wreg,HMOVVAL+B1+B2      ; move to HMOVVAL+B1(B2)


                 RRC32   HMOVVAL                         ; half move in Q8

02BF 1A66                RLCF    HMOVVAL+B3,W            ; move sign into carry bit
02C0 1966                RRCF    HMOVVAL+B3
02C1 1965                RRCF    HMOVVAL+B2
02C2 1964                RRCF    HMOVVAL+B1
02C3 1963                RRCF    HMOVVAL+B0


02C4 C2FE                goto    modeready

                 vmode
02C5 9F91                btfsc   MODETYPE,MSB            ; is it torque move?
02C6 C306                goto    tmode
```

```
02C7 2966                   clrf    HMOVVAL+B3                  ; compute final minus initial
                    MOV24   MOVVAL,HMOVVAL

02C8 6A5F                   MOVFP   MOVVAL+B0,wreg              ; get byte of MOVVAL into w
02C9 4A63                   MOVPF   wreg,HMOVVAL+B0             ; move to HMOVVAL(B0)
02CA 6A60                   MOVFP   MOVVAL+B1,wreg              ; get byte of MOVVAL into w
02CB 4A64                   MOVPF   wreg,HMOVVAL+B1             ; move to HMOVVAL(B1)
02CC 6A61                   MOVFP   MOVVAL+B2,wreg              ; get byte of MOVVAL into w
02CD 4A65                   MOVPF   wreg,HMOVVAL+B2             ; move to HMOVVAL(B2)


02CE 9F61                   btfsc   MOVVAL+B2,MSB
02CF 2B66                   setf    HMOVVAL+B3
                    SUB32   MOVVBUF,HMOVVAL

02D0 6AA8                   MOVFP   MOVVBUF+B0,wreg         ; get lowest byte of MOVVBUF into w
02D1 0563                   SUBWF   HMOVVAL+B0              ; sub lowest byte of HMOVVAL, save in
02D2 6AA9                   MOVFP   MOVVBUF+B1,wreg         ; get 2nd byte of MOVVBUF into w
02D3 0364                   SUBWFB  HMOVVAL+B1              ; sub 2nd byte of HMOVVAL, save in
02D4 6AAA                   MOVFP   MOVVBUF+B2,wreg         ; get 3rd byte of MOVVBUF into w
02D5 0365                   SUBWFB  HMOVVAL+B2              ; sub 3rd byte of HMOVVAL, save in
02D6 6AAB                   MOVFP   MOVVBUF+B3,wreg         ; get 4th byte of MOVVBUF into w
02D7 0366                   SUBWFB  HMOVVAL+B3              ; sub 4th byte of HMOVVAL, save in
02D8 6A66                   movfp   HMOVVAL+B3,wreg
02D9 B580                   andlw   0x80
02DA 4A69                   movpf   wreg,MOVSIGN

02DB 29A3                   clrf    V+B3                    ; create appropriate velocity and
                    MOV24   VL,V                            ; acceleration limits from move sign

02DC 6A20                   MOVFP   VL+B0,wreg              ; get byte of VL into w
02DD 4AA0                   MOVPF   wreg,V+B0               ; move to V(B0)
02DE 6A21                   MOVFP   VL+B1,wreg              ; get byte of VL into w
02DF 4AA1                   MOVPF   wreg,V+B1               ; move to V(B1)
02E0 6A22                   MOVFP   VL+B2,wreg              ; get byte of VL into w
02E1 4AA2                   MOVPF   wreg,V+B2               ; move to V(B2)


02E2 299F                   clrf    A+B3
                    MOV24   AL,A

02E3 6A23                   MOVFP   AL+B0,wreg              ; get byte of AL into w
02E4 4A9C                   MOVPF   wreg,A+B0               ; move to A(B0)
02E5 6A24                   MOVFP   AL+B1,wreg              ; get byte of AL into w
02E6 4A9D                   MOVPF   wreg,A+B1               ; move to A(B1)
02E7 6A25                   MOVFP   AL+B2,wreg              ; get byte of AL into w
02E8 4A9E                   MOVPF   wreg,A+B2               ; move to A(B2)


02E9 290A                   clrf    wreg
02EA 3269                   cpfsgt  MOVSIGN
02EB C2FE                   goto    modeready
                    NEG32   V

02EC 13A0                   COMF    V+B0
02ED 13A1                   COMF    V+B1
02EE 13A2                   COMF    V+B2
02EF 13A3                   COMF    V+B3
02F0 290A                   CLRF    wreg
02F1 15A0                   INCF    V+B0
02F2 11A1                   ADDWFC  V+B1
02F3 11A2                   ADDWFC  V+B2
02F4 11A3                   ADDWFC  V+B3


                    NEG32   A

02F5 139C                   COMF    A+B0
02F6 139D                   COMF    A+B1
```

```
02F7 139E                   COMF    A+B2
02F8 139F                   COMF    A+B3
02F9 290A                   CLRF    wreg
02FA 159C                   INCF    A+B0
02FB 119D                   ADDWFC  A+B1
02FC 119E                   ADDWFC  A+B2
02FD 119F                   ADDWFC  A+B3
                            modeready
02FE 2962                   clrf    MOVVAL+B3
02FF 9F61                   btfsc   MOVVAL+B2,MSB
0300 2B62                   setf    MOVVAL+B3

0301 2B90                   setf    SERVOFLAG               ; set servoflag to restore servo
                                                            ; if stopped

                           if     _PICMASTER_DEBUG

                          ;*****************************************************************
                          ;       For PICMASTER Debug/servo tuning puporses only Purposes Only
                          ;
                          testCapCount
0302 6ABC                         movfp   CAPCOUNT+B0,wreg
0303 08BD                         iorwf   CAPCOUNT+B1,W
0304 4ABB                         movpf   wreg,CAPFLAG
                          ;*****************************************************************************
                           endif

0305 0002                         return

                          tmode                                             ; torque/voltage mode
                         MOV16   MOVVAL+B1,YPWM                             ; set new commanded value

0306 6A60                         MOVFP   MOVVAL+B1+B0,wreg                  ; get byte of MOVVAL+B1 into w
0307 0188                         MOVWF   YPWM+B0                            ; move to YPWM(B0)
0308 6A61                         MOVFP   MOVVAL+B1+B1,wreg                  ; get byte of MOVVAL+B1 into w
0309 0189                         MOVWF   YPWM+B1                            ; move to YPWM(B1)


030A 2990                         clrf    SERVOFLAG                        ; disable servo
030B E50B                         call    doTorque                         ; set pwm duty cycle
030C 2994                         clrf    MOVFLAG
030D 8D93                         bcf     MOVSTAT,bit5
                           if     _PICMASTER_DEBUG
030E C302                         goto    testCapCount
                           else
                                  return
                           endif

                          ;*****************************************************************************

                          ;*****************************************************************************
                          ; NAME:         doMove
                          ;
                          ; DESCRIPTION:  In position mode, trapezoidal moves are performed. Phase1
                          ;               and phase2 respectively, are the periods for the first and
                          ;               second halves of the move. The move time is defined as zero
                          ;               at the beginning of the move,T2 is the time at half the
                                                        move, T1 is the time when c
                          ;               begins,(the region of constant velocity reduces to a point
                          ;               in the case where maximum speed is not realized, and the
                          ;               trapezoidal move degenerates into a trianglular move,
                          ;               together with T1=T2), and TAU is the total time of the move.
                          ;               The accelerations are +-AL or 0.
                          ;
                          ;
                          ;                triangle speed                  trapezoidal speed
                          ;
                          ;                                                 _________________
                          ;                                               /                 \
```

```
;                  /\                      /                \
;                 /  \                    /                  \
;                /    \                  /                    \
;               /      \                /                      \
;              /        \              /                        \
;
;             0   T1=T2  TAU            0     T1      T2          TAU
;
;
;
;               Let x denote the undershoot and y the overshoot commanded
;               at adjacent sample times as half the move is crossed.
;               In the case of a triangular move, the discretization error
;               is given by
;
;                        error = min (2x,2y)
;
;               For a trapezoidal move, the discretization error is
;
;                       error = min (2x,y-x) <= .5*(maximum commanded speed)
;
;               This discretization error is resolved in the final sample
;               time of the move by executing a step to the final position
;               at zero speed. The method employed here the best possible
;               performance with regard to discretization error without
;               dynamically modifying velocity and acceleration limits.
;
;
;
;               In velocity mode, ramp moves are performed.
;
;
;                                       / final velocity
;                                      /
;                                     /
;                                    /
;                                   /
;            initial velocity      /
;
;                                  0     TAU
;
;
;
doMove

 INC16   MOVTIME                        ; increment move time

030F 290A                CLRF    wreg
0310 1567                INCF    (MOVTIME)+B0
0311 1168                ADDWFC  (MOVTIME)+B1

0312 E468                call    doPosVel            ; evaluate iterative equations
0313 3391                tstfsz  MODETYPE
0314 C429                goto    vmove
                         pmove
0315 6AC2                movfp   ONE,wreg            ; test if in phase1
0316 3194                cpfseq  MOVFLAG
0317 C3DB                goto    phase2
 phase1
                         MOVFP32  MOVDEL,MOVTMP      ; save previous discretization error
0318 7CB0                MOVFP   MOVDEL+B0,MOVTMP+B0 ; move MOVDEL(B0) to MOVTMP(B0)
0319 7DB1                MOVFP   MOVDEL+B1,MOVTMP+B1 ; move MOVDEL(B1) to MOVTMP(B1)
031A 7EB2                MOVFP   MOVDEL+B2,MOVTMP+B2  ; move MOVDEL(B2) to MOVTMP(B2)
031B 7FB3                MOVFP   MOVDEL+B3,MOVTMP+B3 ; move MOVDEL(B3) to MOVTMP(B3)
                         MOV32   OPOSITION,MOVDEL    ; test if half move
031C 6A51                MOVFP   OPOSITION+B0,wreg   ; get byte of OPOSITION into w
031D 4AB0                MOVPF   wreg,MOVDEL+B0      ; move to MOVDEL(B0)
031E 6A52                MOVFP   OPOSITION+B1,wreg   ; get byte of OPOSITION into w
031F 4AB1                MOVPF   wreg,MOVDEL+B1      ; move to MOVDEL(B1)
```

```
0320 6A53                MOVFP   OPOSITION+B2,wreg      ; get byte of OPOSITION into w
0321 4AB2                MOVPF   wreg,MOVDEL+B2         ; move to MOVDEL(B2)
0322 6A54                MOVFP   OPOSITION+B3,wreg      ; get byte of OPOSITION into w
0323 4AB3                MOVPF   wreg,MOVDEL+B3         ; move to MOVDEL(B3)
                         ADD32   HMOVVAL,MOVDEL
0324 6A63                MOVFP   HMOVVAL+B0,wreg        ; get lowest byte of HMOVVAL into w
0325 0FB0                ADDWF   MOVDEL+B0              ; add lowest byte of MOVDEL, save in
0326 6A64                MOVFP   HMOVVAL+B1,wreg        ; get 2nd byte of HMOVVAL into w
0327 11B1                ADDWFC  MOVDEL+B1              ; add 2nd byte of MOVDEL, save in
0328 6A65                MOVFP   HMOVVAL+B2,wreg        ; get 3rd byte of HMOVVAL into w
0329 11B2                ADDWFC  MOVDEL+B2              ; add 3rd byte of MOVDEL, save in
032A 6A66                MOVFP   HMOVVAL+B3,wreg        ; get 4th byte of HMOVVAL into w
032B 11B3                ADDWFC  MOVDEL+B3              ; add 4th byte of MOVDEL, save in
                         SUB32   MOVPBUF,MOVDEL
032C 6AA4                MOVFP   MOVPBUF+B0,wreg        ; get lowest byte of MOVPBUF into
032D 05B0                SUBWF   MOVDEL+B0              ; sub lowest byte of MOVDEL, save
032E 6AA5                MOVFP   MOVPBUF+B1,wreg        ; get 2nd byte of MOVPBUF into w
032F 03B1                SUBWFB  MOVDEL+B1              ; sub 2nd byte of MOVDEL, save in
0330 6AA6                MOVFP   MOVPBUF+B2,wreg        ; get 3rd byte of MOVPBUF into w
0331 03B2                SUBWFB  MOVDEL+B2              ; sub 3rd byte of MOVDEL, save in
0332 6AA7                MOVFP   MOVPBUF+B3,wreg        ; get 4th byte of MOVPBUF into w
0333 03B3                SUBWFB  MOVDEL+B3              ; sub 4th byte of MOVDEL, save in
0334 9769                btfss   MOVSIGN,MSB
0335 C33F                goto    mpos1
                         NEG32   MOVDEL
0336 13B0                COMF    MOVDEL+B0
0337 13B1                COMF    MOVDEL+B1
0338 13B2                COMF    MOVDEL+B2
0339 13B3                COMF    MOVDEL+B3
033A 290A                CLRF    wreg
033B 15B0                INCF    MOVDEL+B0
033C 11B1                ADDWFC  MOVDEL+B1
033D 11B2                ADDWFC  MOVDEL+B2
033E 11B3                ADDWFC  MOVDEL+B3
                         mpos1
033F 97B3                btfss   MOVDEL+B3,MSB
0340 C3A5                goto    speedup                ; continue to speed up if in
                         TFSZ16  T1                     ; if T1=0, maximum velocity not
0341 6A6A                MOVFP   T1+B0,wreg
0342 086B                IORWF   T1+B1,W
0343 330A                TSTFSZ  wreg
                                                        ; reached,
                                                        ; has been set in speedup
0344 C378                goto    t2net1
                         NEG32   A                      ; negate A for speeddown
0345 139C                COMF    A+B0
0346 139D                COMF    A+B1
0347 139E                COMF    A+B2
0348 139F                COMF    A+B3
0349 290A                CLRF    wreg
034A 159C                INCF    A+B0
034B 119D                ADDWFC  A+B1
034C 119E                ADDWFC  A+B2
034D 119F                ADDWFC  A+B3
                         ADD32   MOVDEL,MOVTMP          ; test x-y < 0
034E 6AB0                MOVFP   MOVDEL+B0,wreg         ; get lowest byte of MOVDEL into
034F 0F1C                ADDWF   MOVTMP+B0              ; add lowest byte of MOVTMP, save
0350 6AB1                MOVFP   MOVDEL+B1,wreg         ; get 2nd byte of MOVDEL into w
0351 111D                ADDWFC  MOVTMP+B1              ; add 2nd byte of MOVTMP, save in
0352 6AB2                MOVFP   MOVDEL+B2,wreg         ; get 3rd byte of MOVDEL into w
0353 111E                ADDWFC  MOVTMP+B2              ; add 3rd byte of MOVTMP, save in
0354 6AB3                MOVFP   MOVDEL+B3,wreg         ; get 4th byte of MOVDEL into w
0355 111F                ADDWFC  MOVTMP+B3              ; add 4th byte of MOVTMP, save in
0356 971F                btfss   MOVTMP+B3,MSB          ; if new discretization error larger,
0357 C36E                goto    triok                  ; backup to define T2, otherwise ok
0358 2B6C                setf    T2+B0                  ; set T2=-1 for backup
0359 2B6D                setf    T2+B1
                         NEG32   A                      ; negate A to undo
```

```
035A 139C                    COMF    A+B0
035B 139D                    COMF    A+B1
035C 139E                    COMF    A+B2
035D 139F                    COMF    A+B3
035E 290A                    CLRF    wreg
035F 159C                    INCF    A+B0
0360 119D                    ADDWFC  A+B1
0361 119E                    ADDWFC  A+B2
0362 119F                    ADDWFC  A+B3
0363 E48A                    call    undoPosVel
                             NEG32   A                       ; negate A again for speeddown
0364 139C                    COMF    A+B0
0365 139D                    COMF    A+B1
0366 139E                    COMF    A+B2
0367 139F                    COMF    A+B3
0368 290A                    CLRF    wreg
0369 159C                    INCF    A+B0
036A 119D                    ADDWFC  A+B1
036B 119E                    ADDWFC  A+B2
036C 119F                    ADDWFC  A+B3
036D E468                    call    doPosVel                ; and reevaluate iterative equations
                               triok
                             ADD16   MOVTIME,T2                          ; add time to T2
036E 6A67                    MOVFP   MOVTIME+B0,wreg         ; get lowest byte of MOVTIME into w
036F 0F6C                    ADDWF   T2+B0                   ; add lowest byte of T2, save in
0370 6A68                    MOVFP   MOVTIME+B1,wreg         ; get 2nd byte of MOVTIME into w
0371 116D                    ADDWFC  T2+B1                   ; add 2nd byte of T2, save in T2(B1)
                             MOV16   T2,T1
0372 6A6C                    MOVFP   T2+B0,wreg              ; get byte of T2 into w
0373 016A                    MOVWF   T1+B0                   ; move to T1(B0)
0374 6A6D                    MOVFP   T2+B1,wreg              ; get byte of T2 into w
0375 016B                    MOVWF   T1+B1                   ; move to T1(B1)
0376 1594                    incf    MOVFLAG                 ; increment move flag for
0377 C3CE                    goto    mvok                    ; execute last phase1 move
               t2net1
0378 2B6C                    setf    T2+B0                   ; set T2=-1 for backup
0379 2B6D                    setf    T2+B1
                             ADD16   MOVTIME,T2              ; add time to T2
037A 6A67                    MOVFP   MOVTIME+B0,wreg         ; get lowest byte of MOVTIME
037B 0F6C                    ADDWF   T2+B0                   ; add lowest byte of T2, save
037C 6A68                    MOVFP   MOVTIME+B1,wreg         ; get 2nd byte of MOVTIME into
037D 116D                    ADDWFC  T2+B1                   ; add 2nd byte of T2, save in T2(B1)
                             MOVFP32  MOVTMP,TMP             ; test if 3x-y < 0
037E 781C                    MOVFP   MOVTMP+B0,TMP+B0        ; move MOVTMP(B0) to TMP(B0)
037F 791D                    MOVFP   MOVTMP+B1,TMP+B1        ; move MOVTMP(B1) to TMP(B1)
0380 7A1E                    MOVFP   MOVTMP+B2,TMP+B2        ; move MOVTMP(B2) to TMP(B2)
0381 7B1F                    MOVFP   MOVTMP+B3,TMP+B3        ; move MOVTMP(B3) to TMP(B3)
                             RLC32   MOVTMP
0382 8804                    BCF     _carry
0383 1B1C                    RLCF    MOVTMP+B0
0384 1B1D                    RLCF    MOVTMP+B1
0385 1B1E                    RLCF    MOVTMP+B2
0386 1B1F                    RLCF    MOVTMP+B3
                             ADD32   TMP,MOVTMP
0387 6A18                    MOVFP   TMP+B0,wreg             ; get lowest byte of TMP into
0388 0F1C                    ADDWF   MOVTMP+B0               ; add lowest byte of MOVTMP,
0389 6A19                    MOVFP   TMP+B1,wreg             ; get 2nd byte of TMP into w
038A 111D                    ADDWFC  MOVTMP+B1               ; add 2nd byte of MOVTMP, save
038B 6A1A                    MOVFP   TMP+B2,wreg             ; get 3rd byte of TMP into w
038C 111E                    ADDWFC  MOVTMP+B2               ; add 3rd byte of MOVTMP, save
038D 6A1B                    MOVFP   TMP+B3,wreg             ; get 4th byte of TMP into w
038E 111F                    ADDWFC  MOVTMP+B3               ; add 4th byte of MOVTMP, save
                             ADD32   MOVDEL,MOVTMP
038F 6AB0                    MOVFP   MOVDEL+B0,wreg          ; get lowest byte of MOVDEL
0390 0F1C                    ADDWF   MOVTMP+B0               ; add lowest byte of MOVTMP,
0391 6AB1                    MOVFP   MOVDEL+B1,wreg          ; get 2nd byte of MOVDEL into
0392 111D                    ADDWFC  MOVTMP+B1               ; add 2nd byte of MOVTMP, save
```

```
0393 6AB2                   MOVFP   MOVDEL+B2,wreg          ; get 3rd byte of MOVDEL into
0394 111E                   ADDWFC  MOVTMP+B2               ; add 3rd byte of MOVTMP, save
0395 6AB3                   MOVFP   MOVDEL+B3,wreg          ; get 4th byte of MOVDEL into
0396 111F                   ADDWFC  MOVTMP+B3               ; add 4th byte of MOVTMP, save
0397 971F                   btfss   MOVTMP+B3,MSB           ; if new discretization error
0398 C39B                   goto    trapok                  ; take one more flat step
0399 2BB4                   setf    PH2FLAT+B0
039A 2BB5                   setf    PH2FLAT+B1
             trapok
                            ADD16   T2,PH2FLAT
039B 6A6C                   MOVFP   T2+B0,wreg              ; get lowest byte of T2 into w
039C 0FB4                   ADDWF   PH2FLAT+B0              ; add lowest byte of PH2FLAT,
039D 6A6D                   MOVFP   T2+B1,wreg              ; get 2nd byte of T2 into w
039E 11B5                   ADDWFC  PH2FLAT+B1              ; add 2nd byte of PH2FLAT,
                            SUB16   T1,PH2FLAT
039F 6A6A                   MOVFP   T1+B0,wreg              ; get lowest byte of T1 into w
03A0 05B4                   SUBWF   PH2FLAT+B0              ; sub lowest byte of PH2FLAT,
03A1 6A6B                   MOVFP   T1+B1,wreg              ; get 2nd byte of T1 into w
03A2 03B5                   SUBWFB  PH2FLAT+B1              ; sub 2nd byte of PH2FLAT,
03A3 1594                   incf    MOVFLAG                 ; increment move flag for
03A4 C3CE                   goto    mvok                    ; execute last phase1 move
                            speedup
                             MOVFP32  V,MOVTMP              ; test if maximum velocity
03A5 7CA0                   MOVFP   V+B0,MOVTMP+B0          ; move V(B0) to MOVTMP(B0)
03A6 7DA1                   MOVFP   V+B1,MOVTMP+B1          ; move V(B1) to MOVTMP(B1)
03A7 7EA2                   MOVFP   V+B2,MOVTMP+B2          ; move V(B2) to MOVTMP(B2)
03A8 7FA3                   MOVFP   V+B3,MOVTMP+B3          ; move V(B3) to MOVTMP(B3)
                            SUB32   MOVVBUF,MOVTMP
03A9 6AA8                   MOVFP   MOVVBUF+B0,wreg         ; get lowest byte of MOVVBUF
03AA 051C                   SUBWF   MOVTMP+B0               ; sub lowest byte of MOVTMP,
03AB 6AA9                   MOVFP   MOVVBUF+B1,wreg         ; get 2nd byte of MOVVBUF into
03AC 031D                   SUBWFB  MOVTMP+B1               ; sub 2nd byte of MOVTMP, save
03AD 6AAA                   MOVFP   MOVVBUF+B2,wreg         ; get 3rd byte of MOVVBUF into
03AE 031E                   SUBWFB  MOVTMP+B2               ; sub 3rd byte of MOVTMP, save
03AF 6AAB                   MOVFP   MOVVBUF+B3,wreg         ; get 4th byte of MOVVBUF into
03B0 031F                   SUBWFB  MOVTMP+B3               ; sub 4th byte of MOVTMP, save in
03B1 9769                   btfss   MOVSIGN,MSB
03B2 C3BC                   goto    mpos
                            NEG32   MOVTMP
03B3 131C                   COMF    MOVTMP+B0
03B4 131D                   COMF    MOVTMP+B1
03B5 131E                   COMF    MOVTMP+B2
03B6 131F                   COMF    MOVTMP+B3
03B7 290A                   CLRF    wreg
03B8 151C                   INCF    MOVTMP+B0
03B9 111D                   ADDWFC  MOVTMP+B1
03BA 111E                   ADDWFC  MOVTMP+B2
03BB 111F                   ADDWFC  MOVTMP+B3
             mpos
03BC 971F                   btfss   MOVTMP+B3,MSB
03BD C3CE                   goto    mvok                    ; if not, execute move
                            TFSZ16  T1                      ; if so, check to see if T1
03BE 6A6A                   MOVFP   T1+B0,wreg
03BF 086B                   IORWF   T1+B1,W
03C0 330A                   TSTFSZ  wreg
                                                            ; already been set
03C1 C3CE                   goto    mvok
03C2 E48A                   call    undoPosVel              ; if not, backup and redo
                            CLR32   A                       ; equations, resulting in an
03C3 299C                   CLRF    A+B0
03C4 299D                   CLRF    A+B1
03C5 299E                   CLRF    A+B2
03C6 299F                   CLRF    A+B3
03C7 E468                   call    doPosVel                ; maximum speed <= VL
03C8 2B6A                   setf    T1+B0                   ; evaluate T1
03C9 2B6B                   setf    T1+B1
                            ADD16   MOVTIME,T1
03CA 6A67                   MOVFP   MOVTIME+B0,wreg         ; get lowest byte of MOVTIME
03CB 0F6A                   ADDWF   T1+B0                   ; add lowest byte of T1, save
```

```
03CC 6A68                    MOVFP   MOVTIME+B1,wreg        ; get 2nd byte of MOVTIME into
03CD 116B                    ADDWFC  T1+B1                 ; add 2nd byte of T1, save in
                    mvok
                             MOV24   MOVPBUF+B1,POSITION   ; move Q8 calculated position
03CE 6AA5                    MOVFP   MOVPBUF+B1+B0,wreg    ; get byte of MOVPBUF+B1 into
03CF 4A55                    MOVPF   wreg,POSITION+B0      ; move to POSITION(B0)
03D0 6AA6                    MOVFP   MOVPBUF+B1+B1,wreg    ; get byte of MOVPBUF+B1 into
03D1 4A56                    MOVPF   wreg,POSITION+B1      ; move to POSITION(B1)
03D2 6AA7                    MOVFP   MOVPBUF+B1+B2,wreg    ; get byte of MOVPBUF+B1 into
03D3 4A57                    MOVPF   wreg,POSITION+B2      ; move to POSITION(B2)
                             MOV24   MOVVBUF+B0,VELOCITY   ; move Q0 calculated velocity
03D4 6AA8                    MOVFP   MOVVBUF+B0+B0,wreg    ; get byte of MOVVBUF+B0 into
03D5 4A58                    MOVPF   wreg,VELOCITY+B0      ; move to VELOCITY(B0)
03D6 6AA9                    MOVFP   MOVVBUF+B0+B1,wreg    ; get byte of MOVVBUF+B0 into
03D7 4A59                    MOVPF   wreg,VELOCITY+B1      ; move to VELOCITY(B1)
03D8 6AAA                    MOVFP   MOVVBUF+B0+B2,wreg    ; get byte of MOVVBUF+B0 into
03D9 4A5A                    MOVPF   wreg,VELOCITY+B2      ; move to VELOCITY(B2)
03DA 0002                    return

                             phase2
                             TFSZ16  PH2FLAT               ; is flat section finished?
03DB 6AB4                    MOVFP   PH2FLAT+B0,wreg
03DC 08B5                    IORWF   PH2FLAT+B1,W
03DD 330A                    TSTFSZ  wreg
03DE C3FF                    goto    flat
                             TFSZ32  MOVVBUF               ; is velocity zero?
03DF 6AA8                    MOVFP   MOVVBUF+B0,wreg
03E0 08A9                    IORWF   MOVVBUF+B1,W
03E1 08AA                    IORWF   MOVVBUF+B2,W
03E2 08AB                    IORWF   MOVVBUF+B3,W
03E3 330A                    TSTFSZ  wreg
03E4 C41C                    goto    mready                ; if not, execute move
03E5 2994                    clrf    MOVFLAG               ; if so, clear MOVFLAG
03E6 8E93                    bcf     MOVSTAT,bit6          ; clear motion status flag
03E7 8D93                    bcf     MOVSTAT,bit5          ; clear move in progress flag
                             CLR32   A                     ; set zero velocity and acceleration,
03E8 299C                    CLRF    A+B0
03E9 299D                    CLRF    A+B1
03EA 299E                    CLRF    A+B2
03EB 299F                    CLRF    A+B3
                             MOV16   MOVTIME,TAU
03EC 6A67                    MOVFP   MOVTIME+B0,wreg       ; get byte of MOVTIME into w
03ED 016E                    MOVWF   TAU+B0                ; move to TAU(B0)
03EE 6A68                    MOVFP   MOVTIME+B1,wreg       ; get byte of MOVTIME into w
03EF 016F                    MOVWF   TAU+B1                ; move to TAU(B1)
                             MOV32   OPOSITION,MOVPBUF     ; execute last move to P(0)+MOVVAL
03F0 6A51                    MOVFP   OPOSITION+B0,wreg     ; get byte of OPOSITION into w
03F1 4AA4                    MOVPF   wreg,MOVPBUF+B0       ; move to MOVPBUF(B0)
03F2 6A52                    MOVFP   OPOSITION+B1,wreg     ; get byte of OPOSITION into w
03F3 4AA5                    MOVPF   wreg,MOVPBUF+B1       ; move to MOVPBUF(B1)
03F4 6A53                    MOVFP   OPOSITION+B2,wreg     ; get byte of OPOSITION into w
03F5 4AA6                    MOVPF   wreg,MOVPBUF+B2       ; move to MOVPBUF(B2)
03F6 6A54                    MOVFP   OPOSITION+B3,wreg     ; get byte of OPOSITION into w
03F7 4AA7                    MOVPF   wreg,MOVPBUF+B3       ; move to MOVPBUF(B3)
                             ADD24   MOVVAL,MOVPBUF+B1
03F8 6A5F                    MOVFP   MOVVAL+B0,wreg        ; get lowest byte of MOVVAL into w
03F9 0FA5                    ADDWF   MOVPBUF+B1+B0         ; add lowest byte of MOVPBUF+B1, save
03FA 6A60                    MOVFP   MOVVAL+B1,wreg        ; get 2nd byte of MOVVAL into w
03FB 11A6                    ADDWFC  MOVPBUF+B1+B1         ; add 2nd byte of MOVPBUF+B1, save in
03FC 6A61                    MOVFP   MOVVAL+B2,wreg        ; get 3rd byte of MOVVAL into w
03FD 11A7                    ADDWFC  MOVPBUF+B1+B2         ; add 3rd byte of MOVPBUF+B1, save in
03FE C41C                    goto    mready

                    flat
03FF 2B1C                    setf    MOVTMP+B0
0400 2B1D                    setf    MOVTMP+B1
                             ADD16   MOVTMP,PH2FLAT        ; decrement by one use DEC16
0401 6A1C                    MOVFP   MOVTMP+B0,wreg        ; get lowest byte of MOVTMP into w
0402 0FB4                    ADDWF   PH2FLAT+B0            ; add lowest byte of PH2FLAT, save in
```

```
0403 6A1D                   MOVFP   MOVTMP+B1,wreg         ; get 2nd byte of MOVTMP into w
0404 11B5                   ADDWFC  PH2FLAT+B1             ; add 2nd byte of PH2FLAT, save in
                            TFSZ16  PH2FLAT
0405 6AB4                   MOVFP   PH2FLAT+B0,wreg
0406 08B5                   IORWF   PH2FLAT+B1,W
0407 330A                   TSTFSZ  wreg
0408 C41C                   goto    mready
0409 299F                   clrf    A+B3                   ; begin speed down section
                            MOV24   AL,A
040A 6A23                   MOVFP   AL+B0,wreg             ; get byte of AL into w
040B 4A9C                   MOVPF   wreg,A+B0              ; move to A(B0)
040C 6A24                   MOVFP   AL+B1,wreg             ; get byte of AL into w
040D 4A9D                   MOVPF   wreg,A+B1              ; move to A(B1)
040E 6A25                   MOVFP   AL+B2,wreg             ; get byte of AL into w
040F 4A9E                   MOVPF   wreg,A+B2              ; move to A(B2)
0410 290A                   clrf    wreg
0411 3169                   cpfseq  MOVSIGN
0412 C41C                   goto    mready
                            NEG32   A
0413 139C                   COMF    A+B0
0414 139D                   COMF    A+B1
0415 139E                   COMF    A+B2
0416 139F                   COMF    A+B3
0417 290A                   CLRF    wreg
0418 159C                   INCF    A+B0
0419 119D                   ADDWFC  A+B1
041A 119E                   ADDWFC  A+B2
041B 119F                   ADDWFC  A+B3
            mready
                            MOV24   MOVPBUF+B1,POSITION
041C 6AA5                   MOVFP   MOVPBUF+B1+B0,wreg     ; get byte of MOVPBUF+B1 into w
041D 4A55                   MOVPF   wreg,POSITION+B0       ; move to POSITION(B0)
041E 6AA6                   MOVFP   MOVPBUF+B1+B1,wreg     ; get byte of MOVPBUF+B1 into w
041F 4A56                   MOVPF   wreg,POSITION+B1       ; move to POSITION(B1)
0420 6AA7                   MOVFP   MOVPBUF+B1+B2,wreg     ; get byte of MOVPBUF+B1 into w
0421 4A57                   MOVPF   wreg,POSITION+B2       ; move to POSITION(B2)
                            MOV24   MOVVBUF+B0,VELOCITY
0422 6AA8                   MOVFP   MOVVBUF+B0+B0,wreg     ; get byte of MOVVBUF+B0 into w
0423 4A58                   MOVPF   wreg,VELOCITY+B0       ; move to VELOCITY(B0)
0424 6AA9                   MOVFP   MOVVBUF+B0+B1,wreg     ; get byte of MOVVBUF+B0 into w
0425 4A59                   MOVPF   wreg,VELOCITY+B1       ; move to VELOCITY(B1)
0426 6AAA                   MOVFP   MOVVBUF+B0+B2,wreg     ; get byte of MOVVBUF+B0 into w
0427 4A5A                   MOVPF   wreg,VELOCITY+B2       ; move to VELOCITY(B2)
0428 0002                   return
            vmove
                            MOVFP32  MOVVAL,MOVTMP         ; test if final velocity reached
0429 7C5F                   MOVFP   MOVVAL+B0,MOVTMP+B0    ; move MOVVAL(B0) to MOVTMP(B0)
042A 7D60                   MOVFP   MOVVAL+B1,MOVTMP+B1    ; move MOVVAL(B1) to MOVTMP(B1)
042B 7E61                   MOVFP   MOVVAL+B2,MOVTMP+B2    ; move MOVVAL(B2) to MOVTMP(B2)
042C 7F62                   MOVFP   MOVVAL+B3,MOVTMP+B3    ; move MOVVAL(B3) to MOVTMP(B3)
                            SUB32   MOVVBUF,MOVTMP
042D 6AA8                   MOVFP   MOVVBUF+B0,wreg        ; get lowest byte of MOVVBUF into w
042E 051C                   SUBWF   MOVTMP+B0              ; sub lowest byte of MOVTMP, save in
042F 6AA9                   MOVFP   MOVVBUF+B1,wreg        ; get 2nd byte of MOVVBUF into w
0430 031D                   SUBWFB  MOVTMP+B1              ; sub 2nd byte of MOVTMP, save in
0431 6AAA                   MOVFP   MOVVBUF+B2,wreg        ; get 3rd byte of MOVVBUF into w
0432 031E                   SUBWFB  MOVTMP+B2              ; sub 3rd byte of MOVTMP, save in
0433 6AAB                   MOVFP   MOVVBUF+B3,wreg        ; get 4th byte of MOVVBUF into w
0434 031F                   SUBWFB  MOVTMP+B3              ; sub 4th byte of MOVTMP, save in
0435 9769                   btfss   MOVSIGN,MSB
0436 C440                   goto    vmpos
                            NEG32   MOVTMP
0437 131C                   COMF    MOVTMP+B0
0438 131D                   COMF    MOVTMP+B1
0439 131E                   COMF    MOVTMP+B2
043A 131F                   COMF    MOVTMP+B3
043B 290A                   CLRF    wreg
043C 151C                   INCF    MOVTMP+B0
```

```
043D 111D                   ADDWFC  MOVTMP+B1
043E 111E                   ADDWFC  MOVTMP+B2
043F 111F                   ADDWFC  MOVTMP+B3
             vmpos
0440 971F                   btfss   MOVTMP+B3,MSB
0441 C45B                   goto    vmoveok              ; if not, continue
                            CLR32   A                    ; if so, set A=0 and continue with
0442 299C                   CLRF    A+B0
0443 299D                   CLRF    A+B1
0444 299E                   CLRF    A+B2
0445 299F                   CLRF    A+B3
                            MOV32   MOVVAL,MOVVBUF       ; move unless the final velocity
0446 6A5F                   MOVFP   MOVVAL+B0,wreg       ; get byte of MOVVAL into w
0447 4AA8                   MOVPF   wreg,MOVVBUF+B0      ; move to MOVVBUF(B0)
0448 6A60                   MOVFP   MOVVAL+B1,wreg       ; get byte of MOVVAL into w
0449 4AA9                   MOVPF   wreg,MOVVBUF+B1      ; move to MOVVBUF(B1)
044A 6A61                   MOVFP   MOVVAL+B2,wreg       ; get byte of MOVVAL into w
044B 4AAA                   MOVPF   wreg,MOVVBUF+B2      ; move to MOVVBUF(B2)
044C 6A62                   MOVFP   MOVVAL+B3,wreg       ; get byte of MOVVAL into w
044D 4AAB                   MOVPF   wreg,MOVVBUF+B3      ; move to MOVVBUF(B3)
                                                         ; is zero.
044E 2994                   clrf    MOVFLAG              ; clear MOVFLAG
044F 8D93                   bcf     MOVSTAT,bit5         ; clear move in progress flag
                            MOV16   MOVTIME,TAU
0450 6A67                   MOVFP   MOVTIME+B0,wreg      ; get byte of MOVTIME into w
0451 016E                   MOVWF   TAU+B0               ; move to TAU(B0)
0452 6A68                   MOVFP   MOVTIME+B1,wreg      ; get byte of MOVTIME into w
0453 016F                   MOVWF   TAU+B1               ; move to TAU(B1)
                            TFSZ32  MOVVAL
0454 6A5F                   MOVFP   MOVVAL+B0,wreg
0455 0860                   IORWF   MOVVAL+B1,W
0456 0861                   IORWF   MOVVAL+B2,W
0457 0862                   IORWF   MOVVAL+B3,W
0458 330A                   TSTFSZ  wreg
0459 C45B                   goto    vmoveok
045A 8E93                   bcf     MOVSTAT,bit6         ; if final velocity is zero, clear
                                                          ; motion status flag
             vmoveok
                            MOV24   MOVPBUF+B1,POSITION
045B 6AA5                   MOVFP   MOVPBUF+B1+B0,wreg   ; get byte of MOVPBUF+B1 into w
045C 4A55                   MOVPF   wreg,POSITION+B0     ; move to POSITION(B0)
045D 6AA6                   MOVFP   MOVPBUF+B1+B1,wreg   ; get byte of MOVPBUF+B1 into w
045E 4A56                   MOVPF   wreg,POSITION+B1     ; move to POSITION(B1)
045F 6AA7                   MOVFP   MOVPBUF+B1+B2,wreg   ; get byte of MOVPBUF+B1 into w
0460 4A57                   MOVPF   wreg,POSITION+B2     ; move to POSITION(B2)
                            MOV24   MOVVBUF+B0,VELOCITY
0461 6AA8                   MOVFP   MOVVBUF+B0+B0,wreg   ; get byte of MOVVBUF+B0 into w
0462 4A58                   MOVPF   wreg,VELOCITY+B0     ; move to VELOCITY(B0)
0463 6AA9                   MOVFP   MOVVBUF+B0+B1,wreg   ; get byte of MOVVBUF+B0 into w
0464 4A59                   MOVPF   wreg,VELOCITY+B1     ; move to VELOCITY(B1)
0465 6AAA                   MOVFP   MOVVBUF+B0+B2,wreg   ; get byte of MOVVBUF+B0 into w
0466 4A5A                   MOVPF   wreg,VELOCITY+B2     ; move to VELOCITY(B2)
0467 0002                   return
```

```
                ;*****************************************************************

                ;*****************************************************************
                ; NAME:         doPosVel
                ;
                ; DESCRIPTION:  Evaluates the iterative equations for trapezoidal
                ;               generation
                ;
                ;               V(k)=V(k-1)+A,          P(k)=P(k-1)+V(k-1)+A/2,
                ;
                ;               where abs(A)={AL,0} depending on the region of the
                ;               being executed.
                ;

                doPosVel

                            ADD32   MOVVBUF,MOVPBUF         ; P(k-1)+V(k-1)
0468 6AA8                   MOVFP   MOVVBUF+B0,wreg         ; get lowest byte of MOVVBUF into w
0469 0FA4                   ADDWF   MOVPBUF+B0              ; add lowest byte of MOVPBUF, save in
046A 6AA9                   MOVFP   MOVVBUF+B1,wreg         ; get 2nd byte of MOVVBUF into w
046B 11A5                   ADDWFC  MOVPBUF+B1              ; add 2nd byte of MOVPBUF, save in
046C 6AAA                   MOVFP   MOVVBUF+B2,wreg         ; get 3rd byte of MOVVBUF into w
046D 11A6                   ADDWFC  MOVPBUF+B2              ; add 3rd byte of MOVPBUF, save in
046E 6AAB                   MOVFP   MOVVBUF+B3,wreg         ; get 4th byte of MOVVBUF into w
046F 11A7                   ADDWFC  MOVPBUF+B3              ; add 4th byte of MOVPBUF, save in

                            ADD32   A,MOVVBUF                           ; V(k)=V(k-1)+A
0470 6A9C                   MOVFP   A+B0,wreg               ; get lowest byte of A into w
0471 0FA8                   ADDWF   MOVVBUF+B0              ; add lowest byte of MOVVBUF, save in
0472 6A9D                   MOVFP   A+B1,wreg               ; get 2nd byte of A into w
0473 11A9                   ADDWFC  MOVVBUF+B1              ; add 2nd byte of MOVVBUF, save in
0474 6A9E                   MOVFP   A+B2,wreg               ; get 3rd byte of A into w
0475 11AA                   ADDWFC  MOVVBUF+B2              ; add 3rd byte of MOVVBUF, save in
0476 6A9F                   MOVFP   A+B3,wreg               ; get 4th byte of A into w
0477 11AB                   ADDWFC  MOVVBUF+B3              ; add 4th byte of MOVVBUF, save in

                            MOVFP32  A,MOVTMP               ; compute A/2
0478 7C9C                   MOVFP   A+B0,MOVTMP+B0          ; move A(B0) to MOVTMP(B0)
0479 7D9D                   MOVFP   A+B1,MOVTMP+B1          ; move A(B1) to MOVTMP(B1)
047A 7E9E                   MOVFP   A+B2,MOVTMP+B2          ; move A(B2) to MOVTMP(B2)
047B 7F9F                   MOVFP   A+B3,MOVTMP+B3          ; move A(B3) to MOVTMP(B3)

                            RRC32   MOVTMP
047C 1A1F                   RLCF    MOVTMP+B3,W             ; move sign into carry bit
047D 191F                   RRCF    MOVTMP+B3
047E 191E                   RRCF    MOVTMP+B2
047F 191D                   RRCF    MOVTMP+B1
0480 191C                   RRCF    MOVTMP+B0

                            ADD32   MOVTMP,MOVPBUF          ; P(k)=P(k-1)+V(k-1)+A/2,
0481 6A1C                   MOVFP   MOVTMP+B0,wreg  ; get lowest byte of MOVTMP into w
0482 0FA4                   ADDWF   MOVPBUF+B0      ; add lowest byte of MOVPBUF, save in
0483 6A1D                   MOVFP   MOVTMP+B1,wreg  ; get 2nd byte of MOVTMP into w
0484 11A5                   ADDWFC  MOVPBUF+B1      ; add 2nd byte of MOVPBUF, save in
0485 6A1E                   MOVFP   MOVTMP+B2,wreg  ; get 3rd byte of MOVTMP into w
0486 11A6                   ADDWFC  MOVPBUF+B2      ; add 3rd byte of MOVPBUF, save in
0487 6A1F                   MOVFP   MOVTMP+B3,wreg  ; get 4th byte of MOVTMP into w
0488 11A7                   ADDWFC  MOVPBUF+B3      ; add 4th byte of MOVPBUF, save in )

0489 0002                   return
```

```
                ;*****************************************************************
                ;
                ;*****************************************************************
                ; NAME: undoPosVel
                ;
                ; DESCRIPTION:  Backward iteration of the equations for trapezoidal
                ;               generation
                ;
                ;               V(k-1)=V(k)-A,          P(k-1)=P(k)-V(k-1)-A/2,
                ;
                ;               where abs(A)={AL,0} depending on the region of the
                ;               being executed. This routine is used to reverse a
                ;               to be made beyond a decision point.
                ;

                           undoPosVel

                          SUB32   A,MOVVBUF       ; V(k-1)=V(k)-A
048A 6A9C                 MOVFP   A+B0,wreg       ; get lowest byte of A into w
048B 05A8                 SUBWF   MOVVBUF+B0      ; sub lowest byte of MOVVBUF, save in
048C 6A9D                 MOVFP   A+B1,wreg       ; get 2nd byte of A into w
048D 03A9                 SUBWFB  MOVVBUF+B1      ; sub 2nd byte of MOVVBUF, save in
048E 6A9E                 MOVFP   A+B2,wreg       ; get 3rd byte of A into w
048F 03AA                 SUBWFB  MOVVBUF+B2      ; sub 3rd byte of MOVVBUF, save in
0490 6A9F                 MOVFP   A+B3,wreg       ; get 4th byte of A into w
0491 03AB                 SUBWFB  MOVVBUF+B3      ; sub 4th byte of MOVVBUF, save in

                          SUB32   MOVVBUF,MOVPBUF ; P(k)-V(k-1)
0492 6AA8                 MOVFP   MOVVBUF+B0,wreg ; get lowest byte of MOVVBUF into w
0493 05A4                 SUBWF   MOVPBUF+B0      ; sub lowest byte of MOVPBUF, save in
0494 6AA9                 MOVFP   MOVVBUF+B1,wreg ; get 2nd byte of MOVVBUF into w
0495 03A5                 SUBWFB  MOVPBUF+B1      ; sub 2nd byte of MOVPBUF, save in
0496 6AAA                 MOVFP   MOVVBUF+B2,wreg ; get 3rd byte of MOVVBUF into w
0497 03A6                 SUBWFB  MOVPBUF+B2      ; sub 3rd byte of MOVPBUF, save in
0498 6AAB                 MOVFP   MOVVBUF+B3,wreg ; get 4th byte of MOVVBUF into w
0499 03A7                 SUBWFB  MOVPBUF+B3      ; sub 4th byte of MOVPBUF, save in

                          MOVFP32  A,MOVTMP       ; compute A/2
049A 7C9C                 MOVFP   A+B0,MOVTMP+B0  ; move A(B0) to MOVTMP(B0)
049B 7D9D                 MOVFP   A+B1,MOVTMP+B1  ; move A(B1) to MOVTMP(B1)
049C 7E9E                 MOVFP   A+B2,MOVTMP+B2  ; move A(B2) to MOVTMP(B2)
049D 7F9F                 MOVFP   A+B3,MOVTMP+B3  ; move A(B3) to MOVTMP(B3)

                          RRC32   MOVTMP
049E 1A1F                 RLCF    MOVTMP+B3,W     ; move sign into carry bit
049F 191F                 RRCF    MOVTMP+B3
04A0 191E                 RRCF    MOVTMP+B2
04A1 191D                 RRCF    MOVTMP+B1
04A2 191C                 RRCF    MOVTMP+B0

                          SUB32   MOVTMP,MOVPBUF  ; P(k-1)=P(k)-V(k-1)-A/2,
04A3 6A1C                 MOVFP   MOVTMP+B0,wreg ; get lowest byte of MOVTMP into w
04A4 05A4                 SUBWF   MOVPBUF+B0      ; sub lowest byte of MOVPBUF, save in
04A5 6A1D                 MOVFP   MOVTMP+B1,wreg  ; get 2nd byte of MOVTMP into w
04A6 03A5                 SUBWFB  MOVPBUF+B1      ; sub 2nd byte of MOVPBUF, save in
04A7 6A1E                 MOVFP   MOVTMP+B2,wreg  ; get 3rd byte of MOVTMP into w
04A8 03A6                 SUBWFB  MOVPBUF+B2      ; sub 3rd byte of MOVPBUF, save in
04A9 6A1F                 MOVFP   MOVTMP+B3,wreg  ; get 4th byte of MOVTMP into w
04AA 03A7                 SUBWFB  MOVPBUF+B3      ; sub 4th byte of MOVPBUF, save in

04AB 0002                 return
```

```
                ;*****************************************************************

                if _SERVO_PID
                include “pid.asm”       ; PID Algorithm
                ;*****************************************************************
                ;                           PID Servo Implementation
                ;
                ;  Implement Y=KP*U0+KI*INTEGRAL+KV*(U0-U1)
                ;
                ;*****************************************************************



                ;*****************************************************************
                ; NAME:         doServo
                ;
                ; DESCRIPTION:  Performs the servo loop calculations.
                ;

                             doServo:

                             MOV16   POSERROR,U0            ; save new position error in
04AC 6A79                    MOVFP   POSERROR+B0,wreg       ; get byte of POSERROR into w
04AD 0184                    MOVWF   U0+B0                  ;move to U0(B0)
04AE 6A7A                    MOVFP   POSERROR+B1,wreg       ; get byte of POSERROR into w
04AF 0185                    MOVWF   U0+B1                  ; move to U0(B1)

                             LOADAB  U0,KP                  ; compute KP*U0
04B0 7C84                    MOVFP   U0+B0,AARG+B0          ; load lo byte of U0 to AARG
04B1 7D85                    MOVFP   U0+B1,AARG+B1          ; load hi byte of U0 to AARG
04B2 7E26                    MOVFP   KP+B0,BARG+B0          ; load lo byte of KP to BARG
04B3 7F27                    MOVFP   KP+B1,BARG+B1          ; load hi byte of KP to BARG
04B4 E12B                    call    Dmult
                             MOVPF32  DPX,Y                 ; Y=KP*U0
04B5 5880                    MOVPF   DPX+B0,Y+B0            ; move DPX(B0) to Y(B0)
04B6 5981                    MOVPF   DPX+B1,Y+B1            ; move DPX(B1) to Y(B1)
04B7 5A82                    MOVPF   DPX+B2,Y+B2            ; move DPX(B2) to Y(B2)
04B8 5B83                    MOVPF   DPX+B3,Y+B3            ; move DPX(B3) to Y(B3)
04B9 290A                    clrf    wreg                   ; if previous output saturated, do
04BA 3295                    cpfsgt  SATFLAG                ; not accumulate integrator
04BB E552                    call    doIntegral

                             LOADAB  INTEGRAL,KI            ; compute KI*INTEGRAL
04BC 7C96                    MOVFP   INTEGRAL+B0,AARG+B0    ; load lo byte of INTEGRAL to AARG
04BD 7D97                    MOVFP   INTEGRAL+B1,AARG+B1    ; load hi byte of INTEGRAL to AARG
04BE 7E2A                    MOVFP   KI+B0,BARG+B0          ; load lo byte of KI to BARG
04BF 7F2B                    MOVFP   KI+B1,BARG+B1          ; load hi byte of KI to BARG
04C0 E12B                    call    Dmult

                             ADD32   DPX,Y                  ; Y=KP*U0+KI*INTEGRAL
04C1 6A18                    MOVFP   DPX+B0,wreg            ; get lowest byte of DPX into w
04C2 0F80                    ADDWF   Y+B0                   ; add lowest byte of Y, save in Y(B0)
04C3 6A19                    MOVFP   DPX+B1,wreg            ; get 2nd byte of DPX into w
04C4 1181                    ADDWFC  Y+B1                   ; add 2nd byte of Y, save in Y(B1)
04C5 6A1A                    MOVFP   DPX+B2,wreg            ; get 3rd byte of DPX into w
04C6 1182                    ADDWFC  Y+B2                   ; add 3rd byte of Y, save in Y(B2)
04C7 6A1B                    MOVFP   DPX+B3,wreg            ; get 4th byte of DPX into w
04C8 1183                    ADDWFC  Y+B3                   ; add 4th byte of Y, save in Y(B3)

                             MOVFP16  U0,AARG               ; compute KV*(U0-U1)
04C9 7C84                    MOVFP   U0+B0,AARG+B0          ; move U0(B0) to AARG(B0)
04CA 7D85                    MOVFP   U0+B1,AARG+B1          ; move U0(B1) to AARG(B1)

                             SUB16   U1,AARG
04CB 6A86                    MOVFP   U1+B0,wreg             ; get lowest byte of U1 into w
04CC 051C                    SUBWF   AARG+B0                ; sub lowest byte of AARG, save in
04CD 6A87                    MOVFP   U1+B1,wreg             ; get 2nd byte of U1 into w
04CE 031D                    SUBWFB  AARG+B1                ; sub 2nd byte of AARG, save in
```

```
                        MOVFP16 KV,BARG
04CF 7E28               MOVFP   KV+B0,BARG+B0          ; move KV(B0) to BARG(B0)
04D0 7F29               MOVFP   KV+B1,BARG+B1          ; move KV(B1) to BARG(B1)
04D1 E12B               call    Dmult
                        ADD32   DPX,Y           ; Y=KP*U0+KI*INTEGRAL+KV*(U0-U1)
04D2 6A18               MOVFP   DPX+B0,wreg     ; get lowest byte of DPX into w
04D3 0F80               ADDWF   Y+B0            ; add lowest byte of Y, save in Y(B0)
04D4 6A19               MOVFP   DPX+B1,wreg     ; get 2nd byte of DPX into w
04D5 1181               ADDWFC  Y+B1            ; add 2nd byte of Y, save in Y(B1)
04D6 6A1A               MOVFP   DPX+B2,wreg     ; get 3rd byte of DPX into w
04D7 1182               ADDWFC  Y+B2            ; add 3rd byte of Y, save in Y(B2)
04D8 6A1B               MOVFP   DPX+B3,wreg     ; get 4th byte of DPX into w
04D9 1183               ADDWFC  Y+B3            ; add 4th byte of Y, save in Y(B3)
                        MOV16   U0,U1           ; push errors into U(k-1)
04DA 6A84               MOVFP   U0+B0,wreg      ; get byte of U0 into w
04DB 0186               MOVWF   U1+B0           ; move to U1(B0)
04DC 6A85               MOVFP   U0+B1,wreg      ; get byte of U0 into w
04DD 0187               MOVWF   U1+B1           ; move to U1(B1)
04DE 290A               clrf    wreg
04DF 32B9               cpfsgt  SHIFTNUM
04E0 C4E9               goto    grabok
04E1 78B9               movfp   SHIFTNUM,TMP
            grabloop
                        RLC32   Y
04E2 8804               BCF     _carry
04E3 1B80               RLCF    Y+B0
04E4 1B81               RLCF    Y+B1
04E5 1B82               RLCF    Y+B2
04E6 1B83               RLCF    Y+B3
04E7 1718               decfsz  TMP
04E8 C4E2               goto    grabloop
            grabok
04E9 2995               clrf    SATFLAG
04EA 9F83               btfsc   Y+B3,MSB        ; saturate to middle 16 bits,
04EB C4F9               goto    negs            ; keeping top 10 bits for pw1dcH
            poss                                ; and pw1dcL
04EC 6A82               movfp   Y+B2,wreg       ; check if Y >= 2**23
04ED B580               andlw   0x80
04EE 0983               iorwf   Y+B3
04EF 290A               clrf    wreg
04F0 3283               cpfsgt  Y+B3
04F1 C505               goto    zero6bits       ; if not, zero 6 bits
04F2 1595               incf    SATFLAG         ; if so, set Y=0x007FFFFF
04F3 2983               clrf    Y+B3            ; clear for debug purposes
04F4 B07F               movlw   0x7F
04F5 4A82               movpf   wreg,Y+B2
04F6 2B81               setf    Y+B1
04F7 2B80               setf    Y+B0
04F8 C505               goto    zero6bits
            negs
04F9 6A82               movfp   Y+B2,wreg       ; check if Y <= -2**23
04FA B37F               iorlw   0x7F
04FB 0B83               andwf   Y+B3
04FC 2B0A               setf    wreg
04FD 3083               cpfslt  Y+B3
04FE C505               goto    zero6bits       ; if not, zero 6 bits

04FF 2B95               setf    SATFLAG         ; if so, set Y = 0xFF800000
0500 2B83               setf    Y+B3
0501 2982               clrf    Y+B2
0502 8782               bsf     Y+B2,MSB
0503 2981               clrf    Y+B1
0504 2980               clrf    Y+B0
            zero6bits
                        MOV24   Y+B1,YPWM+B0    ; move Y to YPWM and zero 6 bits
0505 6A81               MOVFP   Y+B1+B0,wreg    ; get byte of Y+B1 into w
0506 4A88               MOVPF   wreg,YPWM+B0+B0 ; move to YPWM+B0(B0)
0507 6A82               MOVFP   Y+B1+B1,wreg    ; get byte of Y+B1 into w
0508 4A89               MOVPF   wreg,YPWM+B0+B1 ; move to YPWM+B0(B1)
```

```
0509 6A83                   MOVFP   Y+B1+B2,wreg    ; get byte of Y+B1 into w
050A 4A8A                   MOVPF   wreg,YPWM+B0+B2 ; move to YPWM+B0(B2)
             doTorque                               ; entry point for torque mode
050B B0C0                   movlw   0xC0
050C 0B88                   andwf   YPWM+B0
050D 9F89                   btfsc   YPWM+B1,MSB
050E C516                   goto    tmlimit
             tplimit
050F 9692                   btfss   EXTSTAT,bit6
0510 C51C                   goto    mplimitok
                            CLR32   YPWM
0511 2988                   CLRF    YPWM+B0
0512 2989                   CLRF    YPWM+B1
0513 298A                   CLRF    YPWM+B2
0514 298B                   CLRF    YPWM+B3
0515 C51C                   goto    mplimitok
             tmlimit
0516 9592                   btfss   EXTSTAT,bit5
0517 C51C                   goto    mplimitok
                            CLR32   YPWM
0518 2988                   CLRF    YPWM+B0
0519 2989                   CLRF    YPWM+B1
051A 298A                   CLRF    YPWM+B2
051B 298B                   CLRF    YPWM+B3
             mplimitok
051C B07F                   movlw   PW1DCH_INIT     ; adjustment from bipolar to unipolar
051D 4A19                   movpf   wreg,TMP+B1     ; for 50% duty cycle
051E B0C0                   movlw   PW1DCL_INIT
051F 4A18                   movpf   wreg,TMP+B0
                            ADD16   TMP,YPWM
0520 6A18                   MOVFP   TMP+B0,wreg     ; get lowest byte of TMP into w
0521 0F88                   ADDWF   YPWM+B0         ; add lowest byte of YPWM, save in YPWM(B0)
0522 6A19                   MOVFP   TMP+B1,wreg     ; get 2nd byte of TMP into w
0523 1189                   ADDWFC  YPWM+B1         ; add 2nd byte of YPWM, save in YPWM(B1)
0524 2919                   clrf    TMP+B1          ; correct by 1 LSB
0525 B040                   movlw   0x40            ; add one to bit5 of pw1dcL
0526 4A18                   movpf   wreg,TMP+B0
                            ADD16   TMP,YPWM
0527 6A18                   MOVFP   TMP+B0,wreg     ; get lowest byte of TMP into w
0528 0F88                   ADDWF   YPWM+B0         ; add lowest byte of YPWM, save in YPWM(B0)
0529 6A19                   MOVFP   TMP+B1,wreg     ; get 2nd byte of TMP into w
052A 1189                   ADDWFC  YPWM+B1         ; add 2nd byte of YPWM, save in YPWM(B1)

             testmax
052B 291A                   clrf    TMP+B2          ; check pwm maximum limit
052C 298A                   clrf    YPWM+B2         ; LMD18200 must have a minimum pulse
052D 298B                   clrf    YPWM+B3         ; so duty cycle must not be 0 or 100%
                            MOVFP16  YPWMAX,TMP
052E 788E                   MOVFP   YPWMAX+B0,TMP+B0; move YPWMAX(B0) to TMP(B0)
052F 798F                   MOVFP   YPWMAX+B1,TMP+B1; move YPWMAX(B1) to TMP(B1)
                            SUB24   YPWM,TMP
0530 6A88                   MOVFP   YPWM+B0,wreg    ; get lowest byte of YPWM into w
0531 0518                   SUBWF   TMP+B0          ; sub lowest byte of TMP, save in TMP(B0)
0532 6A89                   MOVFP   YPWM+B1,wreg    ; get 2nd byte of YPWM into w
0533 0319                   SUBWFB  TMP+B1          ; sub 2nd byte of TMP, save in TMP(B1)
0534 6A8A                   MOVFP   YPWM+B2,wreg    ; get 3rd byte of YPWM into w
0535 031A                   SUBWFB  TMP+B2          ; sub 3rd byte of TMP, save in TMP(B2)
0536 971A                   btfss   TMP+B2,MSB
0537 C53D                   goto    testmin
                            MOV16   YPWMAX,YPWM     ; saturate to max
0538 6A8E                   MOVFP   YPWMAX+B0,wreg  ; get byte of YPWMAX into w
0539 0188                   MOVWF   YPWM+B0         ; move to YPWM(B0)
053A 6A8F                   MOVFP   YPWMAX+B1,wreg  ; get byte of YPWMAX into w
053B 0189                   MOVWF   YPWM+B1         ; move to YPWM(B1)
053C C54E                   goto    limitok
            testmin
053D 291A                   clrf    TMP+B2          ;check pwm minimum limit
053E 298A                   clrf    YPWM+B2
053F 298B                   clrf    YPWM+B3
```

```
                        MOVFP16  YPWMIN,TMP
0540 788C               MOVFP   YPWMIN+B0,TMP+B0 ; move YPWMIN(B0) to TMP(B0)
0541 798D               MOVFP   YPWMIN+B1,TMP+B1 ; move YPWMIN(B1) to TMP(B1)
                        SUB24   YPWM,TMP
0542 6A88               MOVFP   YPWM+B0,wreg    ; get lowest byte of YPWM into w
0543 0518               SUBWF   TMP+B0          ; sub lowest byte of TMP, save in TMP(B0)
0544 6A89               MOVFP   YPWM+B1,wreg    ; get 2nd byte of YPWM into w
0545 0319               SUBWFB  TMP+B1          ; sub 2nd byte of TMP, save in TMP(B1)
0546 6A8A               MOVFP   YPWM+B2,wreg    ; get 3rd byte of YPWM into w
0547 031A               SUBWFB  TMP+B2          ; sub 3rd byte of TMP, save in TMP(B2)
0548 9F1A               btfsc   TMP+B2,MSB
0549 C54E               goto    limitok
                        MOV16   YPWMIN,YPWM     ; saturate to min
054A 6A8C               MOVFP   YPWMIN+B0,wreg  ; get byte of YPWMIN into w
054B 0188               MOVWF   YPWM+B0         ; move to YPWM(B0)
054C 6A8D               MOVFP   YPWMIN+B1,wreg  ; get byte of YPWMIN into w
054D 0189               MOVWF   YPWM+B1         ; move to YPWM(B1)
            limitok
054E B803               movlb   bank3           ; set new duty cycle
054F 7088               movfp   YPWM+B0,pw1dcl
0550 7289               movfp   YPWM+B1,pw1dch

0551 0002               return
            ;*****************************************************************

            ;*****************************************************************
            ; NAME:         doIntegral
            ;
            ; DESCRIPTION:  Evaluates the integral for the servo calculations.
            ;
                          doIntegral

                          ADD16   U0,INTEGRAL    ; do integral

0552 6A84               MOVFP   U0+B0,wreg      ; get lowest byte of U0 into w
0553 0F96               ADDWF   INTEGRAL+B0    ; add lowest byte of INTEGRAL, save in
0554 6A85               MOVFP   U0+B1,wreg      ; get 2nd byte of U0 into w
0555 1197               ADDWFC  INTEGRAL+B1    ; add 2nd byte of INTEGRAL, save in

0556 0002           return

                    ;*****************************************************************

                    endif

                    if _SERIAL_IO
                    include “serial.asm”              ; Serial I/O Routines
                    ;*****************************************************************
                    ;
                    ;                           Serial I/O & Utility Functions
                    ;
                    ;*****************************************************************


                    ;*****************************************************************
                    ; NAME:         IdleFunction

                    ; DESCRIPTION:  This routine will perform work while doing waits in
                    ;               serial I/O functions.
                    ;

                    IdleFunction

0557 0004           CLRWDT
0558 0002           return

                    ;*****************************************************************

                    ;*****************************************************************
                    ; NAME:         DoCommand
```

```
                ;
                ; DESCRIPTION:  Search command table for command and execute it.
                ;

                DoCommand

0559 B059                   movlw   (CMD_TABLE & 0xff)     ; CMD_TABLE LSB
055A 4A0D                   movpf   wreg,tblptrl
055B B007                   movlw   page CMD_TABLE         ; CMD_TABLE MSB
055C 4A0E                   movpf   wreg,tblptrh

055D AB3A                   tablrd  1,1,CMDTEMP
                tryNextCmd

055E A93A                   tablrd  0,1,CMDTEMP            ; read entry from table
055F A23B                   tlrd    1,CMDPTRH
0560 A93C                   tablrd  0,1,CMDPTRL

0561 6A3A                   movfp   CMDTEMP,wreg
0562 30C1                   cpfslt  ZERO

0563 C56E                   goto    noCommand              ; error if end of table

0564 3139                   cpfseq  CMDCHAR
0565 C55E                   goto    tryNextCmd

0566 E679                   call    PutChar                ; echo command

0567 633B                   movfp   CMDPTRH,pclath         ; indirect jump to command routine
0568 623C                   movfp   CMDPTRL,pcl
0569 0000                   NOP
                cmdFinish
056A E679                   call    PutChar                ; send response character from
                                                           ; command routine followed by CR
056B B00D                   movlw   CR
056C E679                   call    PutChar

056D C124                   goto    PollingLoop

                             noCommand
056E B03F                   movlw   CMD_BAD                 ; send error character
056F C56A                   goto    cmdFinish

                ;*****************************************************************************

                ;*****************************************************************************
                ; NAME:         do_null
                ;
                ; DESCRIPTION:  The do nothing command used to determine if the chip is
                ;               working. Initiated by a carriage return.

                do_null
0570 B021               movlw   CMD_OK
0571 C56A               goto    cmdFinish

                ;*****************************************************************************

                ;*****************************************************************************
                ; NAME:         do_move

                ; DESCRIPTION:  Commands the axis to move to a new position or velocity.
                ;               Position data is relative, and in encoder counts. Velocity
                ;               data is absolute, and in encoder counts/sample time multi-
                ;               plied by 256. All moves are performed by the controller such
                ;               that velocity and acceleration limits set into parameter
                ;               memory will not be violated. All move commands are kept in a
                ;               one deep FIFO buffer. The command in the buffer is executed
                ;               as soon as the currently executed command is complete.
                ;
```

```
                        ;
                        ; ARGUMENTS:    M [800000,7FFFFF]
                        ;
                        do_move

                                if     DECIO

0572 E6CC                       call    GetDecVal

                                else

                                call    GetVal

                                endif

0573 9F93                       btfsc   MOVSTAT,bit7           ; test if buffer available
0574 C57E                       goto    bufoverflow
                                MOV24   VALBUF,NMOVVAL         ; if so, accept value into NMOVVAL
0575 6A31                       MOVFP   VALBUF+B0,wreg         ; get byte of VALBUF into w
0576 4A5B                       MOVPF   wreg,NMOVVAL+B0        ; move to NMOVVAL(B0)
0577 6A32                       MOVFP   VALBUF+B1,wreg         ; get byte of VALBUF into w
0578 4A5C                       MOVPF   wreg,NMOVVAL+B1        ; move to NMOVVAL(B1)
0579 6A33                       MOVFP   VALBUF+B2,wreg         ; get byte of VALBUF into w
057A 4A5D                       MOVPF   wreg,NMOVVAL+B2        ; move to NMOVVAL(B2)
057B 8793                       bsf     MOVSTAT,bit7           ; set buffer full flag
057C B021                       movlw   CMD_OK
057D C56A                       goto    cmdFinish

                        bufoverflow
057E B03F                       movlw   CMD_BAD ; else, return error
057F C56A                       goto    cmdFinish

                        ;*************************************************************************

                        ;*************************************************************************
                        ; NAME:         do_mode
                        ;
                        ; DESCRIPTION:  An argument of “P” will cause all subsequent move commands
                        ;               to be incremental position moves. A “V” argument will cause
                        ;               all subsequent moves to be absolute velocity moves.
                        ;
                        ; ARGUMENTS:    O [P,V]
                        ;

                        do_mode

0580 E557                       call    IdleFunction   ; get single character loop
0581 E681                       call    GetChk
0582 31C2                       cpfseq  ONE
0583 C580                       goto    do_mode
0584 E676                       call    GetChar
0585 4A4D                       movpf   wreg,STRVALL
0586 2991                       clrf    MODETYPE       ; MODETYPE=0 for position moves
                        testP
0587 B050                       movlw   ‘P’            ; position moves for type P
0588 314D                       cpfseq  STRVALL
0589 C58B                       goto    testV
058A C598                       goto    modeok
                        testV
058B B056                       movlw   ‘V’            ; velocity moves for type V
058C 314D                       cpfseq  STRVALL
058D C590                       goto    testT
058E 1591                       incf    MODETYPE       ; MODETYPE=1 for velocity moves
058F C598                       goto    modeok
                        testT
0590 B054                       movlw   ‘T’            ; TORQUE Moves for type ‘T’
0591 314D                       cpfseq  STRVALL
0592 C596                       goto    modeerror
0593 2B91                       setf    MODETYPE       ; MODETYPE=-1 for torque moves
```

```
0594 2990                       clrf    SERVOFLAG       ; disable servo
0595 C598                       goto    modeok
                        modeerror
0596 B03F                       movlw   CMD_BAD         ; mode error
0597 C56A                       goto    cmdFinish
                        modeok
0598 6A4D                       movfp   STRVALL,wreg    ; echo type character
0599 E679                       call    PutChar

059A B021                       movlw   CMD_OK
059B C56A                       goto    cmdFinish

                        ;*****************************************************************************

                        ;*****************************************************************************
                        ; NAME:         do_setparameter
                        ;
                        ; DESCRIPTION:  Sets controller parameters to the value given.
                        ;
                        ;       Parameter               #               Range
                        ;
                        ;       VL=velocity limit       0               [0,7FFFFF]
                        ;       AL=acceleration limit   1               [0,7FFFFF]
                        ;
                        ;       KP=proportional gain    2               [8000,7FFF]
                        ;       KP=velocity gain        3               [8000,7FFF]
                        ;       KP=integral gain        4               [8000,7FFF]
                        ;
                        ;       IM=integrator mode      5               [0,3]
                        ;
                        ;       FV=velocity FF          6               [8000,7FFF] : Not Imple
                        ;       FA=acceleration FF      7               [8000,7FFF] : Not Imple
                        ;
                        ;
                        ; ARGUMENTS:    S [0,FF] [800000,7FFFFF]
                        ;

                        do_setparameter

059C E669                       call    GetPar                  ; get parameter number

059D B008                       movlw   NUMPAR                  ; check if in range [0,NUMPAR]
059E 3031                       cpfslt  VALBUF+B0
059F C5C1                       goto    Serror

05A0 B07C                       movlw   (PAR_TABLE & 0xff)      ; PAR_TABLE LSB
05A1 4A0D                       movpf   wreg,tblptrl
05A2 B007                       movlw   page PAR_TABLE          ; PAR_TABLE MSB
05A3 4A0E                       movpf   wreg,tblptrh

05A4 AB3D                       tablrd  1,1,PARTEMP

                        setNextPar

05A5 A23D                       tlrd    1,PARTEMP               ; read entry from table
05A6 A93E                       tablrd  0,1,PARLEN
05A7 A93F                       tablrd  0,1,PARPTR

05A8 B008                       movlw   NUMPAR                  ; error if end of table
05A9 303D                       cpfslt  PARTEMP
05AA C5C1                       goto    Serror

05AB 6A3D                       movfp   PARTEMP,wreg
05AC 3131                       cpfseq  VALBUF+B0
05AD C5A5                       goto    setNextPar

05AE 6A3F                       movfp   PARPTR,wreg             ; pointer to parameter in fsr1
05AF 690A                       movfp   wreg,fsr1
```

```
                            if     DECIO                    ; get new value in VALBUF

05B0 E6CC                   call    GetDecVal

                            else

                            call    GetVal

                            endif

05B1 B031                   movlw   VALBUF                   ; pointer to VALBUF in fsr0
05B2 610A                   movfp   wreg,fsr0
                             AUTOINC                         ; set autoincrement

05B3 8404                   BSF     _fs0
05B4 8D04                   BCF     _fs1
05B5 8604                   BSF     _fs2
05B6 8F04                   BCF     _fs3


                        setGetMore
05B7 6800                   movfp   indf0,indf1              ; move new value to parameter
05B8 073E                   decf    PARLEN
05B9 333E                   tstfsz  PARLEN
05BA C5B7                   goto    setGetMore

                             AUTONO                          ; no autoincrement

05BB 8404                   BSF     _fs0
05BC 8504                   BSF     _fs1
05BD 8604                   BSF     _fs2
05BE 8704                   BSF     _fs3

05BF B021                   movlw   CMD_OK
05C0 C56A                   goto    cmdFinish

                        Serror
05C1 B03F                   movlw   CMD_BAD
05C2 C56A                   goto    cmdFinish

                        ;*************************************************************************

                        ;*************************************************************************
                        ; NAME:         do_readparameter
                        ;
                        ; DESCRIPTION:  Returns the present value of a parameter.
                        ;
                        ; ARGUMENTS:    R [0,FF]
                        ;
                        ; RETURNS:      The present value of the requested parameter is returned.

                        do_readparameter

05C3 E669                   call    GetPar                   ; get parameter number

05C4 B008                   movlw   NUMPAR                   ; check if in range [0,NUMPAR]
05C5 3031                   cpfslt  VALBUF+B0
05C6 C5EB                   goto    Rerror

05C7 B07C                   movlw   (PAR_TABLE & 0xff)       ; PAR_TABLE LSB
05C8 4A0D                   movpf   wreg,tblptrl
05C9 B007                   movlw   page PAR_TABLE           ; PAR_TABLE MSB
05CA 4A0E                   movpf   wreg,tblptrh

05CB AB3D                   tablrd  1,1,PARTEMP

                        readNextPar

05CC A23D                   tlrd    1,PARTEMP                ; read entry from table
```

```
05CD A93E                   tablrd  0,1,PARLEN
05CE A93F                   tablrd  0,1,PARPTR

05CF B008                   movlw   NUMPAR                  ; error if end of table
05D0 303D                   cpfslt  PARTEMP
05D1 C5EB                   goto    Rerror

05D2 6A3D                   movfp   PARTEMP,wreg
05D3 3131                   cpfseq  VALBUF+B0
05D4 C5CC                   goto    readNextPar

05D5 6A3F                   movfp   PARPTR,wreg             ; pointer to parameter in fsr1
05D6 690A                   movfp   wreg,fsr1

05D7 B031                   movlw   VALBUF                  ; pointer to VALBUF in fsr1
05D8 610A                   movfp   wreg,fsr0
                             AUTOINC                        ; set autoincrement

05D9 8404                   BSF     _fs0
05DA 8D04                   BCF     _fs1
05DB 8604                   BSF     _fs2
05DC 8F04                   BCF     _fs3

                            CLR24   VALBUF                  ; clear old VALBUF

05DD 2931                   CLRF    VALBUF+B0
05DE 2932                   CLRF    VALBUF+B1
05DF 2933                   CLRF    VALBUF+B2


                  readGetMore
05E0 6008                   movfp   indf1,indf0     ; read parameter into VALBUF
05E1 073E                   decf    PARLEN
05E2 333E                   tstfsz  PARLEN
05E3 C5E0                   goto    readGetMore

                AUTONO                              ; no autoincrement

05E4 8404                   BSF     _fs0
05E5 8504                   BSF     _fs1
05E6 8604                   BSF     _fs2
05E7 8704                   BSF     _fs3

                            if     DECIO            ; send parameter value

05E8 E728                   call    PutDecVal

                            else

                            call    PutVal

                            endif

05E9 B021                   movlw   CMD_OK
05EA C56A                   goto    cmdFinish

                  Rerror
05EB B03F                   movlw   CMD_BAD
05EC C56A                   goto    cmdFinish

                  ;*****************************************************************************

                  ;*****************************************************************************
                  ; NAME: do_shutter
                  ;
                  ; DESCRIPTION:  Returns the time (in sample time counts [0,FFFF]) since the
                  ;               start of the present move and captures the commanded and
                  ;               measured values of position and velocity at the time of the
                  ;               command.
```

```
                        ;
                        ;
                        ; ARGUMENTS:    C
                        ;
                        ; RETURNS:      The time since the start of the present move is returned.
                        ;
                        do_shutter

                                MOV24   POSITION,CPOSITION    ; capture commanded position

05ED 6A55                       MOVFP   POSITION+B0,wreg      ; get byte of POSITION into w
05EE 4A40                       MOVPF   wreg,CPOSITION+B0     ; move to CPOSITION(B0)
05EF 6A56                       MOVFP   POSITION+B1,wreg      ; get byte of POSITION into w
05F0 4A41                       MOVPF   wreg,CPOSITION+B1     ; move to CPOSITION(B1)
05F1 6A57                       MOVFP   POSITION+B2,wreg      ; get byte of POSITION into w
05F2 4A42                       MOVPF   wreg,CPOSITION+B2     ; move to CPOSITION(B2)


                                MOV24   VELOCITY,CVELOCITY    ; capture commanded velocity

05F3 6A58                       MOVFP   VELOCITY+B0,wreg      ; get byte of VELOCITY into w
05F4 4A43                       MOVPF   wreg,CVELOCITY+B0     ; move to CVELOCITY(B0)
05F5 6A59                       MOVFP   VELOCITY+B1,wreg      ; get byte of VELOCITY into w
05F6 4A44                       MOVPF   wreg,CVELOCITY+B1     ; move to CVELOCITY(B1)
05F7 6A5A                       MOVFP   VELOCITY+B2,wreg      ; get byte of VELOCITY into w
05F8 4A45                       MOVPF   wreg,CVELOCITY+B2     ; move to CVELOCITY(B2)


                                MOV24   MPOSITION,CMPOSITION  ; capture measured position

05F9 6A72                       MOVFP   MPOSITION+B0,wreg     ; get byte of MPOSITION into w
05FA 4A46                       MOVPF   wreg,CMPOSITION+B0    ; move to CMPOSITION(B0)
05FB 6A73                       MOVFP   MPOSITION+B1,wreg     ; get byte of MPOSITION into w
05FC 4A47                       MOVPF   wreg,CMPOSITION+B1    ; move to CMPOSITION(B1)
05FD 6A74                       MOVFP   MPOSITION+B2,wreg     ; get byte of MPOSITION into w
05FE 4A48                       MOVPF   wreg,CMPOSITION+B2    ; move to CMPOSITION(B2)


                                MOV24   MVELOCITY,CMVELOCITY  ; capture measured velocity

05FF 6A75                       MOVFP   MVELOCITY+B0,wreg     ; get byte of MVELOCITY into w
0600 4A49                       MOVPF   wreg,CMVELOCITY+B0    ; move to CMVELOCITY(B0)
0601 6A76                       MOVFP   MVELOCITY+B1,wreg     ; get byte of MVELOCITY into w
0602 4A4A                       MOVPF   wreg,CMVELOCITY+B1    ; move to CMVELOCITY(B1)
0603 6A77                       MOVFP   MVELOCITY+B2,wreg     ; get byte of MVELOCITY into w
0604 4A4B                       MOVPF   wreg,CMVELOCITY+B2    ; move to CMVELOCITY(B2)

0605 2933                       clrf    VALBUF+B2
                                MOV16   MOVTIME,VALBUF        ; capture move time, move to VALBUF

0606 6A67                       MOVFP   MOVTIME+B0,wreg       ; get byte of MOVTIME into w
0607 0131                       MOVWF   VALBUF+B0             ; move to VALBUF(B0)
0608 6A68                       MOVFP   MOVTIME+B1,wreg       ; get byte of MOVTIME into w
0609 0132                       MOVWF   VALBUF+B1             ; move to VALBUF(B1)

                                if     DECIO

060A E728                       call    PutDecVal

                                else

                                call    PutVal

                                endif

060B B021                       movlw   CMD_OK
060C C56A                       goto    cmdFinish
```

```
                    ;*******************************************************************************

                    ;*******************************************************************************
                    ; NAME: do_readcomposition
                    ;
                    ; DESCRIPTION:  Returns the commanded position count which was captured
                    ;               during the last shutter command.
                    ;
                    ; ARGUMENTS:    P
                    ;
                    ; RETURNS:      The last captured position count is returned. [800000,7FFFFF]
                    ;

                    do_readcomposition

                            MOV24   CPOSITION,VALBUF       ; move CPOSITION to VALBUF

060D 6A40                   MOVFP   CPOSITION+B0,wreg      ; get byte of CPOSITION into w
060E 4A31                   MOVPF   wreg,VALBUF+B0         ; move to VALBUF(B0)
060F 6A41                   MOVFP   CPOSITION+B1,wreg      ; get byte of CPOSITION into w
0610 4A32                   MOVPF   wreg,VALBUF+B1         ; move to VALBUF(B1)
0611 6A42                   MOVFP   CPOSITION+B2,wreg      ; get byte of CPOSITION into w
0612 4A33                   MOVPF   wreg,VALBUF+B2         ; move to VALBUF(B2)

                            if     DECIO

0613 E728                   call    PutDecVal

                            else

                            call    PutVal

                            endif

0614 B021                   movlw   CMD_OK
0615 C56A                   goto    cmdFinish

                    ;*******************************************************************************

                    ;*******************************************************************************
                    ; NAME: do_readcomvelocity
                    ;
                    ; DESCRIPTION:  Returns the commanded velocity multiplied by 256 which was
                    ;               captured during the last shutter command.
                    ;
                    ; ARGUMENTS:    V
                    ;
                    ; RETURNS:      The last captured commanded velocity times 256 is returned.
                    ;               [800000,7FFFFF]
                    ;

                    do_readcomvelocity

                            MOV24   CVELOCITY,VALBUF       ; move commanded velocity to VALBUF

0616 6A43                   MOVFP   CVELOCITY+B0,wreg      ; get byte of CVELOCITY into w
0617 4A31                   MOVPF   wreg,VALBUF+B0         ; move to VALBUF(B0)
0618 6A44                   MOVFP   CVELOCITY+B1,wreg      ; get byte of CVELOCITY into w
0619 4A32                   MOVPF   wreg,VALBUF+B1         ; move to VALBUF(B1)
061A 6A45                   MOVFP   CVELOCITY+B2,wreg      ; get byte of CVELOCITY into w
061B 4A33                   MOVPF   wreg,VALBUF+B2         ; move to VALBUF(B2)

                            if     DECIO

061C E728                   call    PutDecVal

                            else

                            call    PutVal
```

```
                        endif

061D B021               movlw   CMD_OK
061E C56A               goto    cmdFinish

                ;*************************************************************************
                ;*************************************************************************
                ; NAME: do_readactposition
                ;
                ; DESCRIPTION:  Returns the measured position count which was captured
                ;               during the last shutter command.
                ;
                ; ARGUMENTS:    p
                ;
                ; RETURNS:      The last captured measured position count is returned.
                ;               [800000,7FFFFF]
                ;
                do_readactposition

                MOV24   CMPOSITION,VALBUF               ; move measured position to

061F 6A46               MOVFP   CMPOSITION+B0,wreg      ; get byte of CMPOSITION into w
0620 4A31               MOVPF   wreg,VALBUF+B0          ; move to VALBUF(B0)
0621 6A47               MOVFP   CMPOSITION+B1,wreg      ; get byte of CMPOSITION into w
0622 4A32               MOVPF   wreg,VALBUF+B1          ; move to VALBUF(B1)
0623 6A48               MOVFP   CMPOSITION+B2,wreg      ; get byte of CMPOSITION into w
0624 4A33               MOVPF   wreg,VALBUF+B2          ; move to VALBUF(B2)

                        if     DECIO

0625 E728               call    PutDecVal

                        else

                        call    PutVal

                        endif

0626 B021               movlw   CMD_OK
0627 C56A               goto    cmdFinish

                ;*************************************************************************
                ;*************************************************************************
                ; NAME: do_readactvelocity
                ;
                ; DESCRIPTION:  Returns the measured velocity multiplied by 256 which was
                ;               captured during the last shutter command.
                ;
                ; ARGUMENTS:    v
                ;
                ; RETURNS:      The last captured measured velocity times 256 is returned.
                ;               [800000,7FFFFF]
                ;
                do_readactvelocity

                        MOV24   CMVELOCITY,VALBUF       ; move measured velocity to

0628 6A49               MOVFP   CMVELOCITY+B0,wreg      ; get byte of CMVELOCITY into w
0629 4A31               MOVPF   wreg,VALBUF+B0          ; move to VALBUF(B0)
062A 6A4A               MOVFP   CMVELOCITY+B1,wreg      ; get byte of CMVELOCITY into w
062B 4A32               MOVPF   wreg,VALBUF+B1          ; move to VALBUF(B1)
062C 6A4B               MOVFP   CMVELOCITY+B2,wreg      ; get byte of CMVELOCITY into w
062D 4A33               MOVPF   wreg,VALBUF+B2          ; move to VALBUF(B2)
```

```
                        if     DECIO

062E E728               call    PutDecVal

                        else

                        call    PutVal

                        endif

062F B021               movlw   CMD_OK
0630 C56A               goto    cmdFinish

                ;*****************************************************************************

                ;*****************************************************************************
                ; NAME: do_externalstatus
                ;
                ; DESCRIPTION:  Returns a two digit hex number which defines the state of
                ;               the bits in the external status register. Issuing this
                ;               command will clear all the bits in the external status
                ;               register unless the event which set the bit is still true.
                ;
                ;
                ; ARGUMENTS:    X
                ;
                ; RETURNS:      The external status register is returned.
                ;

                do_externalstatus

0631 8406               bsf     _glintd
0632 6A92               movfp   EXTSTAT,wreg
0633 2992               clrf    EXTSTAT
0634 8C06               bcf     _glintd
0635 E688               call    PutHex

0636 B021               movlw   CMD_OK
0637 C56A               goto    cmdFinish

                ;*****************************************************************************

                ;*****************************************************************************
                ; NAME: do_movestatus
                ;
                ; DESCRIPTION:  Returns a two digit hex number which defines the state of
                ;               the bits in the move status register. Issuing this command
                ;               will clear all the bits in the move status register unless
                ;               the event which set the bit is still true.
                ;
                ; ARGUMENTS:    Y
                ;
                ; RETURNS:      The move status register is returned.
                ;

                do_movestatus

0638 6A93               movfp   MOVSTAT,wreg
0639 E688               call    PutHex

063A B021               movlw   CMD_OK
063B C56A               goto    cmdFinish
```

```
                    ;*****************************************************************************
                    ;
                    ;*****************************************************************************
                    ; NAME:         do_readindposition
                    ;
                    ; DESCRIPTION:  Returns the last index position captured in position counts.
                    ;
                    ; ARGUMENTS:    I
                    ;
                    ; RETURNS:      The last captured index position is returned.
                    ;
                    do_readindposition

                            MOV24   INDEXPOS,VALBUF        ; move measured velocity to VALBUF

063C 6AB6                   MOVFP   INDEXPOS+B0,wreg       ; get byte of INDEXPOS into w
063D 4A31                   MOVPF   wreg,VALBUF+B0         ; move to VALBUF(B0)
063E 6AB7                   MOVFP   INDEXPOS+B1,wreg       ; get byte of INDEXPOS into w
063F 4A32                   MOVPF   wreg,VALBUF+B1         ; move to VALBUF(B1)
0640 6AB8                   MOVFP   INDEXPOS+B2,wreg       ; get byte of INDEXPOS into w
0641 4A33                   MOVPF   wreg,VALBUF+B2         ; move to VALBUF(B2)

                            if     DECIO

0642 E728                   call    PutDecVal

                            else

                            call    PutVal

                            endif

0643 B021                   movlw   CMD_OK
0644 C56A                   goto    cmdFinish

                    ;*****************************************************************************
                    ;
                    ;*****************************************************************************
                    ; NAME:         do_setposition
                    ;
                    ; DESCRIPTION:  Sets the measured and commanded position to the value given.
                    ;               This command should not be sent unless the move FIFO buffer
                    ;
                    ; ARGUMENTS:    H [800000,7FFFFF]
                    ;

                    do_setposition

                            if     DECIO

0645 E6CC                   call    GetDecVal

                            else

                            call    GetVal

                            endif

                            MOV24   VALBUF,POSITION

0646 6A31                   MOVFP   VALBUF+B0,wreg         ; get byte of VALBUF into w
0647 4A55                   MOVPF   wreg,POSITION+B0       ; move to POSITION(B0)
0648 6A32                   MOVFP   VALBUF+B1,wreg         ; get byte of VALBUF into w
0649 4A56                   MOVPF   wreg,POSITION+B1       ; move to POSITION(B1)
064A 6A33                   MOVFP   VALBUF+B2,wreg         ; get byte of VALBUF into w
064B 4A57                   MOVPF   wreg,POSITION+B2       ; move to POSITION(B2)
```

```
                        MOV24   VALBUF,MPOSITION

064C 6A31               MOVFP   VALBUF+B0,wreg          ; get byte of VALBUF into w
064D 4A72               MOVPF   wreg,MPOSITION+B0       ; move to MPOSITION(B0)
064E 6A32               MOVFP   VALBUF+B1,wreg          ; get byte of VALBUF into w
064F 4A73               MOVPF   wreg,MPOSITION+B1       ; move to MPOSITION(B1)
0650 6A33               MOVFP   VALBUF+B2,wreg          ; get byte of VALBUF into w
0651 4A74               MOVPF   wreg,MPOSITION+B2       ; move to MPOSITION(B2)

                        CLR32   Y

0652 2980               CLRF    Y+B0
0653 2981               CLRF    Y+B1
0654 2982               CLRF    Y+B2
0655 2983               CLRF    Y+B3

0656 B021               movlw   CMD_OK
0657 C56A               goto    cmdFinish

                ;*************************************************************************

                ;*************************************************************************
                ; NAME:         do_reset
                ;
                ; DESCRIPTION:  Performs a software reset.
                ;
                ; ARGUMENTS:    Z
                ;

                do_reset

0658 B021               movlw   CMD_OK
0659 E679               call    PutChar
065A C021               goto    Startup


                ;*************************************************************************
                ; NAME:         do_stop
                ;
                ; DESCRIPTION:  Stops servo by clearing SERVOFLAG.
                ;
                do_stop

065B 2990               clrf    SERVOFLAG

065C B021               movlw   CMD_OK
065D C56A               goto    cmdFinish

                ;*************************************************************************

                ;*************************************************************************
                ; NAME:         do_capture
                ;

                do_capture

                  if     ((_PICMASTER_DEBUG == 1) && (DECIO == 1))

065E E6CC               call    GetDecVal

                  endif

                  if    ((_PICMASTER_DEBUG == 1) && (DECIO == 0))

                        call    GetVal

                  endif

                  if    _PICMASTER_DEBUG == 1
```

```
                        MOV16   VALBUF,CAPCOUNT

065F 6A31               MOVFP   VALBUF+B0,wreg  ; get byte of VALBUF into w
0660 01BC               MOVWF   CAPCOUNT+B0     ; move to CAPCOUNT(B0)
0661 6A32               MOVFP   VALBUF+B1,wreg  ; get byte of VALBUF into w
0662 01BD               MOVWF   CAPCOUNT+B1     ; move to CAPCOUNT(B1)

                        MOV16   VALBUF,CAPTMP

0663 6A31               MOVFP   VALBUF+B0,wreg  ; get byte of VALBUF into w
0664 01BE               MOVWF   CAPTMP+B0       ; move to CAPTMP(B0)
0665 6A32               MOVFP   VALBUF+B1,wreg  ; get byte of VALBUF into w
0666 01BF               MOVWF   CAPTMP+B1       ; move to CAPTMP(B1)

0667 B021               movlw   CMD_OK
0668 C56A               goto    cmdFinish
                  endif

                ;*************************************************************************
                ; NAME:         GetPar
                ;
                ; DESCRIPTION:  Get a parameter number [0,FF] from the serial port and place
                ;               it in VALBUF+B0.
                ;

                GetPar

                         CLR24   VALBUF

0669 2931               CLRF    VALBUF+B0
066A 2932               CLRF    VALBUF+B1
066B 2933               CLRF    VALBUF+B2

066C E6B2               call    GetHex
066D 6A4E               movfp   HEXVAL,wreg
066E B50F               andlw   0x0F
066F 4A31               movpf   wreg,VALBUF+B0
0670 1D31               swapf   VALBUF+B0

0671 E6B2               call    GetHex
0672 6A31               movfp   VALBUF+B0,wreg

0673 0E4E               addwf   HEXVAL,W
0674 4A31               movpf   wreg,VALBUF+B0

0675 0002               return

                ;*************************************************************************

                ;*************************************************************************
                ; NAME:         GetChar
                ;
                ; DESCRIPTION:  Get character from receive buffer.
                ;
                GetChar

0676 B800               movlb   bank0           ; set bank0
0677 540A               movpf   rcreg,wreg      ; receive character

0678 0002               return

                ;*************************************************************************

                ;*************************************************************************
                ; NAME:         PutChar
                ;
                ; DESCRIPTION:  send character out the serial port
                ;
```

```
                        ; ARGUMENTS:    wreg contains byte to be transmitted
                        ;
                        PutChar

0679 B801                       movlb   bank1           ; set bank1
                        bufwait                         ; is transmit buffer empty?
067A 9116                       btfss   _tbmt
067B C67A                       goto    bufwait

067C B800                       movlb   bank0           ; set bank0
                        shfwait
067D 9115                       btfss   _trmt           ; is transmit shift register empty?
067E C67D                       goto    shfwait

067F 4A16                       movpf   wreg,txreg      ; if so, send character

0680 0002                       return

                        ;*****************************************************************************

                        ;*****************************************************************************
                        ; NAME:         GetChk
                        ;
                        ; DESCRIPTION:  Check if character is in receive buffer.
                        ;

                        GetChk
0681 B801                       movlb   bank1           ; set bank1
0682 560A                       movpf   pir,wreg
0683 B501                       andlw   CHARREADY       ; return status in wreg
0684 0002                       return

                        ;*****************************************************************************

                        ;*****************************************************************************
                        ; NAME: PutDec
                        ;
                        ; DESCRIPTION:  Converts a hex value [0,F] in wreg to its ASCII equivalent.
                        ;               The upper nibble of wreg is assumed to be zero.
                        ;
                        ; ENTRY CONDITIONS: wreg = value to be converted and sent in ASCII decimal
                        ;

                                if     DECIO

                        PutDec
0685 B130                       addlw   0x30            ; convert to ASCII
0686 E679                       call    PutChar
0687 0002                       return

                                endif

                        ;*****************************************************************************

                        ;*****************************************************************************
                        ; NAME: PutHex
                        ;
                        ; DESCRIPTION:  Convert the wreg value to ASCII hexadecimal. The output
                        ;               format is two digits with the A-F parts in upper case and
                        ;               leading zeros. The result is sent out the serial port with
                        ;               PutChar.
                        ;
                        ; ENTRY CONDITIONS: wreg = value to be converted and sent in ASCII hex
                        ;

                        PutHex
```

```
0688 4A4E                       movpf   wreg,HEXVAL
0689 1D0A                       swapf   wreg
068A B50F                       andlw   0x0F
068B 4A4F                       movpf   wreg,HEXTMP
068C 2D0A                       negw    wreg
068D B109                       addlw   0x09
068E 970A                       btfss   wreg,MSB
068F C693                       goto    puth20
0690 B037                       movlw   'A'-0x0A
0691 0E4F                       addwf   HEXTMP,W
0692 C695                       goto    puth25
                        puth20
0693 B030                       movlw   '0'
0694 0E4F                       addwf   HEXTMP,W
                        puth25
0695 E679                       call    PutChar

0696 6A4E                       movfp   HEXVAL,wreg
0697 B50F                       andlw   0x0F
0698 4A4F                       movpf   wreg,HEXTMP
0699 2D0A                       negw    wreg
069A B109                       addlw   0x09
069B 970A                       btfss   wreg,MSB
069C C6A0                       goto    putl20
069D B037                       movlw   'A'-0x0A
069E 0E4F                       addwf   HEXTMP,W
069F C6A2                       goto    putl25
                        putl20
06A0 B030                       movlw   '0'
06A1 0E4F                       addwf   HEXTMP,W
                        putl25
06A2 E679                       call    PutChar

06A3 0002                       return

                        ;*****************************************************************************

                        ;*****************************************************************************
                        ; NAME:         PutStr
                        ;
                        ; DESCRIPTION:  Sends a character string out the serial port.
                        ;

                        PutStr
06A4 AB4C                       tablrd  1,1,STRVALH
                        GetNextPair

06A5 A24C                       tlrd    1,STRVALH
06A6 A94D                       tablrd  0,1,STRVALL

06A7 6A4C                       movfp   STRVALH,wreg
06A8 31C1                       cpfseq  ZERO
06A9 C6AB                       goto    putH
06AA 0002                       return
                        putH
06AB E679                       call    PutChar

06AC 6A4D                       movfp   STRVALL,wreg
06AD 31C1                       cpfseq  ZERO
06AE C6B0                       goto    putL
06AF 0002                       return
                        putL
06B0 E679                       call    PutChar

06B1 C6A5                       goto    GetNextPair

                        ;*****************************************************************************

                        ;*****************************************************************************
```

```
                        ; NAME:         GetHex
                        ;
                        ; DESCRIPTION:  Receive an ASCII hex character from the serial port and
                        ;               convert to numerical value.
                        ;
                        ; RETURNS:      numerical value in HEXVAL

                        GetHex

                        getnxt
06B2 E557                       call    IdleFunction
06B3 E681                       call    GetChk
06B4 31C2                       cpfseq  ONE
06B5 C6B2                       goto    getnxt

06B6 2950                       clrf    HEXSTAT
06B7 E676                       call    GetChar
06B8 4A4E                       movpf   wreg,HEXVAL
06B9 E679                       call    PutChar
06BA B00D                       movlw   CR
06BB 044E                       subwf   HEXVAL,W
06BC 330A                       tstfsz  wreg
06BD C6BF                       goto    gth10
06BE C6C9                       goto    gthCR
                        gth10

06BF 6A4E                       movfp   HEXVAL,wreg
06C0 B239                       sublw   '9'
06C1 970A                       btfss   wreg,MSB
06C2 C6C5                       goto    gth20

06C3 B009                       movlw   0x09
06C4 0F4E                       addwf   HEXVAL

                        gth20
06C5 B00F                       movlw   0x0F
06C6 0B4E                       andwf   HEXVAL
06C7 2950                       clrf    HEXSTAT
06C8 0002                       return

                        gthCR
06C9 B001                       movlw   0x01
06CA 4A50                       movpf   wreg,HEXSTAT
06CB 0002                       return

                        ;*****************************************************************************


                        ;*****************************************************************************
                        ; NAME:         getval
                        ;
                        ; DESCRIPTION:  Get a value [800000,7FFFFF] from the serial port and place
                        ;               it in VALBUF.
                        ;
                                if     DECIO
                                else

                        GetVal
                                CLR24   VALBUF
                        getnext
                                call    GetHex

                                movlw   0x01
                                cpfseq  HEXSTAT
                                goto    shift
                                return
                        shift
                                swapf   VALBUF+B2
                                movfp   VALBUF+B2,wreg
```

```
                        andlw   0xF0
                        movpf   wreg,VALBUF+B2
                        swapf   VALBUF+B1
                        movfp   VALBUF+B1,wreg
                        andlw   0x0F
                        addwf   VALBUF+B2
                        movfp   VALBUF+B1,wreg
                        andlw   0xF0
                        movpf   wreg,VALBUF+B1
                        swapf   VALBUF+B0
                        movfp   VALBUF+B0,wreg
                        andlw   0x0F
                        addwf   VALBUF+B1
                        movfp   VALBUF+B0,wreg
                        andlw   0xF0
                        addwf   HEXVAL,W
                        movpf   wreg,VALBUF+B0

                        goto    getnext

                        endif

                    ;*****************************************************************************

                    ;*****************************************************************************
                    ; NAME:         GetDecVal
                    ;
                    ; DESCRIPTION:  Get a value [-8388608,8388607] from the serial port and
                    ;               place it in VALBUF
                    ;
                    ; RETURNS:      numerical value is returned in VALBUF

                        if     DECIO

                    GetDecVal
                             CLR24   VALBUF

06CC 2931                   CLRF    VALBUF+B0
06CD 2932                   CLRF    VALBUF+B1
06CE 2933                   CLRF    VALBUF+B2

06CF E708                   call    GetDec
06D0 2B9B                   setf    DECSIGN
06D1 B001                   movlw   DEC_MN
06D2 3199                   cpfseq  DECSTAT
06D3 299B                   clrf    DECSIGN

                    getdecnext
06D4 E708                   call    GetDec

06D5 B002                   movlw   DEC_CR
06D6 3199                   cpfseq  DECSTAT
06D7 C6D9                   goto    mul10
06D8 C6FD                   goto    fixsign
                    mul10

                            RLC24   VALBUF                  ; multiply VALBUF by two

06D9 8804                   BCF     _carry
06DA 1B31                   RLCF    VALBUF+B0
06DB 1B32                   RLCF    VALBUF+B1
06DC 1B33                   RLCF    VALBUF+B2

                            MOV24   VALBUF,DVALBUF          ; save in DVALBUF

06DD 6A31                   MOVFP   VALBUF+B0,wreg          ; get byte of VALBUF into w
06DE 4A34                   MOVPF   wreg,DVALBUF+B0         ; move to DVALBUF(B0)
06DF 6A32                   MOVFP   VALBUF+B1,wreg          ; get byte of VALBUF into w
06E0 4A35                   MOVPF   wreg,DVALBUF+B1         ; move to DVALBUF(B1)
```

```
06E1 6A33                   MOVFP   VALBUF+B2,wreg         ; get byte of VALBUF into w
06E2 4A36                   MOVPF   wreg,DVALBUF+B2        ; move to DVALBUF(B2)

                            RLC24   VALBUF

06E3 8804                   BCF     _carry
06E4 1B31                   RLCF    VALBUF+B0
06E5 1B32                   RLCF    VALBUF+B1
06E6 1B33                   RLCF    VALBUF+B2

                            RLC24   VALBUF                 ; VALBUF now multiplied by eight

06E7 8804                   BCF     _carry
06E8 1B31                   RLCF    VALBUF+B0
06E9 1B32                   RLCF    VALBUF+B1
06EA 1B33                   RLCF    VALBUF+B2

                            ADD24   DVALBUF,VALBUF         ; VALBUF now multiplied by ten

06EB 6A34                   MOVFP   DVALBUF+B0,wreg        ; get lowest byte of DVALBUF into w
06EC 0F31                   ADDWF   VALBUF+B0              ; add lowest byte of VALBUF, save in
06ED 6A35                   MOVFP   DVALBUF+B1,wreg        ; get 2nd byte of DVALBUF into w
06EE 1132                   ADDWFC  VALBUF+B1              ; add 2nd byte of VALBUF, save in
06EF 6A36                   MOVFP   DVALBUF+B2,wreg        ; get 3rd byte of DVALBUF into w
06F0 1133                   ADDWFC  VALBUF+B2              ; add 3rd byte of VALBUF, save in

                            CLR24   DVALBUF

06F1 2934                   CLRF    DVALBUF+B0
06F2 2935                   CLRF    DVALBUF+B1
06F3 2936                   CLRF    DVALBUF+B2
06F4 6A98                   movfp   DECVAL,wreg
06F5 4A34                   movpf   wreg,DVALBUF+B0
                            ADD24   DVALBUF,VALBUF
06F6 6A34                   MOVFP   DVALBUF+B0,wreg ; get lowest byte of DVALBUF into w
06F7 0F31                   ADDWF   VALBUF+B0       ; add lowest byte of VALBUF, save in
06F8 6A35                   MOVFP   DVALBUF+B1,wreg ; get 2nd byte of DVALBUF into w
06F9 1132                   ADDWFC  VALBUF+B1       ; add 2nd byte of VALBUF, save in VALBUF(B1)
06FA 6A36                   MOVFP   DVALBUF+B2,wreg ; get 3rd byte of DVALBUF into w
06FB 1133                   ADDWFC  VALBUF+B2       ; add 3rd byte of VALBUF, save in VALBUF(B2)

06FC C6D4                   goto    getdecnext
                    fixsign
06FD 290A                   clrf    wreg
06FE 329B                   cpfsgt  DECSIGN
06FF 0002                   return
                            NEG24   VALBUF

0700 1331                   COMF    VALBUF+B0
0701 1332                   COMF    VALBUF+B1
0702 1333                   COMF    VALBUF+B2
0703 290A                   CLRF    wreg
0704 1531                   INCF    VALBUF+B0
0705 1132                   ADDWFC  VALBUF+B1
0706 1133                   ADDWFC  VALBUF+B2

0707 0002                   return

                            endif
```

```
                        ;*****************************************************************************

                        ;*****************************************************************************
                        ; NAME:         GetDec
                        ;
                        ; DESCRIPTION:  Receive an ASCII decimal character from the serial port and
                        ;               convert to its numerical value.
                        ;
                        ; ARGUMENTS:    numerical value is returned in DECVAL
                        ;

                                if     DECIO

                        GetDec

                        getdecnxt
0708 E557                       call    IdleFunction
0709 E681                       call    GetChk
070A 31C2                       cpfseq  ONE
070B C708                       goto    getdecnxt

070C E676                       call    GetChar
070D 4A98                       movpf   wreg,DECVAL
070E E679                       call    PutChar

070F B00D                       movlw   CR
0710 0498                       subwf   DECVAL,W
0711 30C1                       cpfslt  ZERO
0712 C71F                       goto    gtdCR
0713 B02D                       movlw   MN
0714 0498                       subwf   DECVAL,W
0715 30C1                       cpfslt  ZERO
0716 C722                       goto    gtdMN
0717 B020                       movlw   SP
0718 0498                       subwf   DECVAL,W
0719 30C1                       cpfslt  ZERO
071A C725                       goto    gtdSP
                        gtd09
071B B00F                       movlw   0x0F
071C 0B98                       andwf   DECVAL
071D 2999                       clrf    DECSTAT
071E 0002                       return
                        gtdCR
071F B002                       movlw   DEC_CR
0720 4A99                       movpf   wreg,DECSTAT
0721 0002                       return
                        gtdMN
0722 B001                       movlw   DEC_MN
0723 4A99                       movpf   wreg,DECSTAT
0724 0002                       return
                        gtdSP
0725 B000                       movlw   DEC_SP
0726 4A99                       movpf   wreg,DECSTAT
0727 0002                       return

                                endif

                        ;*****************************************************************************


                        ;*****************************************************************************
                        ; NAME:         PutVal
                        ;
                        ; DESCRIPTION:  Sends the value in VALBUF [800000,7FFFFF] out the serial
                        ;

                                if     DECIO
                                else

                        PutVal
```

```
                        movfp   VALBUF+B2,wreg
                        call    PutHex
                        movfp   VALBUF+B1,wreg
                        call    PutHex
                        movfp   VALBUF+B0,wreg
                        call    PutHex

                        return

                        endif

                ;*************************************************************************

                ;*************************************************************************
                ; NAME:         PutDecVal
                ;
                ; DESCRIPTION:  Send the value in VALBUF [-8388608,8388607] out the serial
                ;

                        if     DECIO

                PutDecVal

0728 9733               btfss   VALBUF+B2,MSB
0729 C734               goto    pdpos
                        NEG24   VALBUF

072A 1331               COMF    VALBUF+B0
072B 1332               COMF    VALBUF+B1
072C 1333               COMF    VALBUF+B2
072D 290A               CLRF    wreg
072E 1531               INCF    VALBUF+B0
072F 1132               ADDWFC  VALBUF+B1
0730 1133               ADDWFC  VALBUF+B2


0731 B02D               movlw   MN
0732 E679               call    PutChar
0733 C736               goto    pddigits
                pdpos
0734 B020               movlw   SP
0735 E679               call    PutChar

                pddigits
0736 B08D               movlw   (DEC_TABLE & 0xff)     ; DEC_TABLE LSB
0737 4A0D               movpf   wreg,tblptrl
0738 B007               movlw   page DEC_TABLE         ; DEC_TABLE MSB
0739 4A0E               movpf   wreg,tblptrh

073A A934               tablrd  0,1,DVALBUF+B0
                readNextDec

073B A034               tlrd    0,DVALBUF+B0           ; read entry from table
073C AB35               tablrd  1,1,DVALBUF+B1
073D A936               tablrd  0,1,DVALBUF+B2

073E 2B0A               setf    wreg                   ; unitsposition if end of table
073F 3134               cpfseq  DVALBUF+B0
0740 C742               goto    getdigit
0741 C756               goto    unitsposition
                getdigit
0742 1534               incf    DVALBUF+B0             ; restore to power of 10
0743 2B98               setf    DECVAL                 ; set DECVAL to -1
                inc
0744 1598               incf    DECVAL                 ; increment DECVAL
                        SUB24   DVALBUF,VALBUF         ; check if in range

0745 6A34               MOVFP   DVALBUF+B0,wreg        ; get lowest byte of DVALBUF into w
0746 0531               SUBWF   VALBUF+B0              ; sub lowest byte of VALBUF, save in
```

```
0747 6A35                   MOVFP   DVALBUF+B1,wreg ; get 2nd byte of DVALBUF into w
0748 0332                   SUBWFB  VALBUF+B1       ; sub 2nd byte of VALBUF, save in VALBUF(B1)
0749 6A36                   MOVFP   DVALBUF+B2,wreg ; get 3rd byte of DVALBUF into w
074A 0333                   SUBWFB  VALBUF+B2       ; sub 3rd byte of VALBUF, save in VALBUF(B2)

074B 9733                   btfss   VALBUF+B2,MSB
074C C744                   goto    inc

                            ADD24   DVALBUF,VALBUF  ; if so, correct VALBUF for next digit

074D 6A34                   MOVFP   DVALBUF+B0,wreg ; get lowest byte of DVALBUF into w
074E 0F31                   ADDWF   VALBUF+B0       ; add lowest byte of VALBUF, save in
074F 6A35                   MOVFP   DVALBUF+B1,wreg ; get 2nd byte of DVALBUF into w
0750 1132                   ADDWFC  VALBUF+B1       ; add 2nd byte of VALBUF, save in VALBUF(B1)
0751 6A36                   MOVFP   DVALBUF+B2,wreg ; get 3rd byte of DVALBUF into w
0752 1133                   ADDWFC  VALBUF+B2       ; add 3rd byte of VALBUF, save in VALBUF(B2)

0753 6A98                   movfp   DECVAL,wreg     ; send DECVAL
0754 E685                   call    PutDec

0755 C73B                   goto    readNextDec     ; get next table entry

                    unitsposition
0756 6A31                   movfp   VALBUF+B0,wreg  ; units position value now in VALBUF
0757 E685                   call    PutDec

0758 0002                   return


                            endif

                    ;*****************************************************************************


                    ;*****************************************************************************
                    ;
                    ;
                    ;
                    ;       TABLES:


                              CMD_START CMD_TABLE

                              CMD_TABLE

                              CMD_DEF do_null,DO_NULL

0759 000D                   DATA    DO_NULL
075A 0570                   DATA    do_null

                              CMD_DEF do_move,DO_MOVE

075B 004D                   DATA    DO_MOVE
075C 0572                   DATA    do_move

                              CMD_DEF do_mode,DO_MODE

075D 004F                   DATA    DO_MODE
075E 0580                   DATA    do_mode

                              CMD_DEF do_setparameter,DO_SETPARAMETER

075F 0053                   DATA    DO_SETPARAMETER
0760 059C                   DATA    do_setparameter

                              CMD_DEF do_readparameter,DO_READPARAMETER

0761 0052                   DATA    DO_READPARAMETER
0762 05C3                   DATA    do_readparameter
```

```
                            CMD_DEF do_shutter,DO_SHUTTER

0763 0043                   DATA    DO_SHUTTER
0764 05ED                   DATA    do_shutter

                            CMD_DEF do_readcomposition,DO_READCOMPOSITION

0765 0050                   DATA    DO_READCOMPOSITION
0766 060D                   DATA    do_readcomposition

                            CMD_DEF do_readcomvelocity,DO_READCOMVELOCITY

0767 0056                   DATA    DO_READCOMVELOCITY
0768 0616                   DATA    do_readcomvelocity

                            CMD_DEF do_readactposition,DO_READACTPOSITION

0769 0070                   DATA    DO_READACTPOSITION
076A 061F                   DATA    do_readactposition

                            CMD_DEF do_readactvelocity,DO_READACTVELOCITY

076B 0076                   DATA    DO_READACTVELOCITY
076C 0628                   DATA    do_readactvelocity

                            CMD_DEF do_externalstatus,DO_EXTERNALSTATUS

076D 0058                   DATA    DO_EXTERNALSTATUS
076E 0631                   DATA    do_externalstatus

                            CMD_DEF do_movestatus,DO_MOVESTATUS

076F 0059                   DATA    DO_MOVESTATUS
0770 0638                   DATA    do_movestatus

                            CMD_DEF do_readindposition,DO_READINDPOSITION

0771 0049                   DATA    DO_READINDPOSITION
0772 063C                   DATA    do_readindposition

                            CMD_DEF do_setposition,DO_SETPOSITION

0773 0048                   DATA    DO_SETPOSITION
0774 0645                   DATA    do_setposition

                            CMD_DEF do_reset,DO_RESET

0775 005A                   DATA    DO_RESET
0776 0658                   DATA    do_reset

                            CMD_DEF do_stop,DO_STOP

0777 0073                   DATA    DO_STOP
0778 065B                   DATA    do_stop

                    if      _PICMASTER_DEBUG
                            CMD_DEF do_capture,DO_CAPTURE

0779 0063                   DATA    DO_CAPTURE
077A 065E                   DATA    do_capture

                    endif
                            CMD_END
                    ;                                       ;
077B 0000                   DATA    0x00


                    ;
                    ; In PAR_TABLE, the code word is as follows :
```

```
                    ;        Low Byte is # of bytes, Hi Byte is function code
                    ;

077C 0003           PAR_TABLE       DATA    0x0003
077D 0020                           DATA    VL
077E 0103                           DATA    0x0103
077F 0023                           DATA    AL
0780 0202                           DATA    0x0202
0781 0026                           DATA    KP
0782 0302                           DATA    0x0302
0783 0028                           DATA    KV
0784 0402                           DATA    0x0402
0785 002A                           DATA    KI
0786 0501                           DATA    0x0501
0787 002C                           DATA    IM
0788 0602                           DATA    0x0602
0789 002D                           DATA    FV
078A 0702                           DATA    0x0702
078B 002F                           DATA    FA
078C 0008                           DATA    NUMPAR

                            if     DECIO

078D 423F           DEC_TABLE       DATA    0x423F
078E 000F                           DATA    0x000F
078F 869F                           DATA    0x869F
0790 0001                           DATA    0x0001
0791 270F                           DATA    0x270F
0792 0000                           DATA    0x0000
0793 03E7                           DATA    0x03E7
0794 0000                           DATA    0x0000
0795 0063                           DATA    0x0063
0796 0000                           DATA    0x0000
0797 0009                           DATA    0x0009
0798 0000                           DATA    0x0000
0799 FFFF                           DATA    0xFFFF
                            endif


                     endif

                     if     _PICMASTER_DEBUG
                             include “picmastr.asm” ; PIC-MASTER Debug (data capture) File

0060                 #define CaptureAddr     0x8000 ; addr for dummy Table Writes (to TRACE

                    ;*************************************************************************
                    ; NAME:         doCaptureRegs
                    ;
                    ; DESCRIPTION:  Captures Desired Register Values To PIC-MASTER Trace Buffer
                    ;               Intended for PICMASTER Demo/debug/servo tuning Purposes Only
                    ;               Capture The following registers to Trace Buffer by putting
                    ;               A Trace point on a TABLW instruction. Trace only 2nd Cycle
                    ;
                    ;               (a) POSERROR (position error : 16 bits)
                    ;               (b) VELERROR (velocity error : 16 bits)
                    ;               (c) MPOSITION (measured position value : 24 bits)
                    ;               (d) MVELOCITY (measured velocity value : 24 bits)
                    ;               (e) POSITION (commanded position : 24 bits)
                    ;               (f) VELOCITY (commanded velocity : 24 bits)
                    ;               (g) Y (output of servo loop : 32 bits)
                    ;               (h) YPWM (output value written to PWM : 10 bits)
                    ;
                    ;
                    ;
                    doCaptureRegs
                                                          ; !end! hdr !skip start!
079A B000                   movlw   (CaptureAddr & 0xff)
079B 010D                   movwf   tblptrl
079C B080                   movlw   CaptureAddr/256
```

```
079D 010E                       movwf   tblptrh                 ; setup table pointer address

079E AC79                       tablwt  0,0,POSERROR+B0         ; dummy tablwt
079F A67A                       tlwt    1,POSERROR+B1           ; now table latch = 16 bits contents
of POSERROR
                        capPerr
07A0 AC79                       tablwt  0,0,POSERROR+B0         ; perform actual table write of
POSERROR

07A1 AC7C                       tablwt  0,0,VELERROR+B0
07A2 A67D                       tlwt    1,VELERROR+B1           ; capture Velocity error
                        capVerr
07A3 AC7C                       tablwt  0,0,VELERROR+B0

07A4 AC72                       tablwt  0,0,MPOSITION+B0
07A5 A673                       tlwt    1,MPOSITION+B1          ; capture measured position
                        capMpos
07A6 AC72                       tablwt  0,0,MPOSITION+B0

07A7 AC55                       tablwt  0,0,POSITION+B0
07A8 A656                       tlwt    1,POSITION+B1           ; capture commanded position
                        capPos
07A9 AC55                       tablwt  0,0,POSITION+B0

07AA AC75                       tablwt  0,0,MVELOCITY+B0
07AB A676                       tlwt    1,MVELOCITY+B1          ; capture measured velocity
                        capMvel
07AC AC75                       tablwt  0,0,MVELOCITY+B0

07AD AC58                       tablwt  0,0,VELOCITY+B0
07AE A659                       tlwt    1,VELOCITY+B1           ; capture commanded velocity
                        capVel
07AF AC58                       tablwt  0,0,VELOCITY+B0

07B0 AC88                       tablwt  0,0,YPWM+B0
07B1 A689                       tlwt    1,YPWM+B1               ; capture commanded velocity
                        capPwm
07B2 AC88                       tablwt  0,0,YPWM+B0

                                DEC16   CAPTMP

07B3 290A                       CLRF    wreg
07B4 07BE                       DECF    CAPTMP+B0
07B5 03BF                       SUBWFB  CAPTMP+B1

                                TFSZ16  CAPTMP

07B6 6ABE                       MOVFP   CAPTMP+B0,wreg
07B7 08BF                       IORWF   CAPTMP+B1,W
07B8 330A                       TSTFSZ  wreg

07B9 0002                       return

07BA 0001                       DATA    0x0001                  ; HALT instruction (avail only in

                        HaltTrace
07BB 29BB                       clrf    CAPFLAG
                                MOV16   CAPCOUNT,CAPTMP

07BC 6ABC                       MOVFP   CAPCOUNT+B0,wreg        ; get byte of CAPCOUNT into w
07BD 01BE                       MOVWF   CAPTMP+B0               ; move to CAPTMP(B0)
07BE 6ABD                       MOVFP   CAPCOUNT+B1,wreg        ; get byte of CAPCOUNT into w
07BF 01BF                       MOVWF   CAPTMP+B1                   ; move to CAPTMP(B1)
07C0 0002                       return
                        ;*****************************************************************************

                         end
                                END
Errors   :    0
Warnings :    0
```

NOTES:

WORLDWIDE SALES & SERVICE

AMERICAS

Corporate Office
Microchip Technology Inc.
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 602 786-7200 Fax: 602 786-7277
Technical Support: 602 786-7627
Web: http://www.mchip.com/microhip

Atlanta
Microchip Technology Inc.
500 Sugar Mill Road, Suite 200B
Atlanta, GA 30350
Tel: 770 640-0034 Fax: 770 640-0307

Boston
Microchip Technology Inc.
5 Mount Royal Avenue
Marlborough, MA 01752
Tel: 508 480-9990 Fax: 508 480-8575

Chicago
Microchip Technology Inc.
333 Pierce Road, Suite 180
Itasca, IL 60143
Tel: 708 285-0071 Fax: 708 285-0075

Dallas
Microchip Technology Inc.
14651 Dallas Parkway, Suite 816
Dallas, TX 75240-8809
Tel: 214 991-7177 Fax: 214 991-8588

Dayton
Microchip Technology Inc.
35 Rockridge Road
Englewood, OH 45322
Tel: 513 832-2543 Fax: 513 832-2841

Los Angeles
Microchip Technology Inc.
18201 Von Karman, Suite 455
Irvine, CA 92715
Tel: 714 263-1888 Fax: 714 263-1338

New York
Microchip Technology Inc.
150 Motor Parkway, Suite 416
Hauppauge, NY 11788
Tel: 516 273-5305 Fax: 516 273-5335

AMERICAS (continued)

San Jose
Microchip Technology Inc.
2107 North First Street, Suite 590
San Jose, CA 95131
Tel: 408 436-7950 Fax: 408 436-7955

ASIA/PACIFIC

Hong Kong
Microchip Technology
Unit No. 3002-3004, Tower 1
Metroplaza
223 Hing Fong Road
Kwai Fong, N.T. Hong Kong
Tel: 852 2 401 1200 Fax: 852 2 401 3431

Korea
Microchip Technology
168-1, Youngbo Bldg. 3 Floor
Samsung-Dong, Kangnam-Ku,
Seoul, Korea
Tel: 82 2 554 7200 Fax: 82 2 558 5934

Singapore
Microchip Technology
200 Middle Road
#10-03 Prime Centre
Singapore 188980
Tel: 65 334 8870 Fax: 65 334 8850

Taiwan
Microchip Technology
10F-1C 207
Tung Hua North Road
Taipei, Taiwan, ROC
Tel: 886 2 717 7175 Fax: 886 2 545 0139

EUROPE

United Kingdom
Arizona Microchip Technology Ltd.
Unit 6, The Courtyard
Meadow Bank, Furlong Road
Bourne End, Buckinghamshire SL8 5AJ
Tel: 44 0 1628 851077 Fax: 44 0 1628 850259

France
Arizona Microchip Technology SARL
2 Rue du Buisson aux Fraises
91300 Massy - France
Tel: 33 1 69 53 63 20 Fax: 33 1 69 30 90 79

Germany
Arizona Microchip Technology GmbH
Gustav-Heinemann-Ring 125
D-81739 Muenchen, Germany
Tel: 49 89 627 144 0 Fax: 49 89 627 144 44

Italy
Arizona Microchip Technology SRL
Centro Direzionale Colleoni
Palazzo Pegaso Ingresso No. 2
Via Paracelso 23, 20041
Agrate Brianza (MI) Italy
Tel: 39 039 689 9939 Fax: 39 039 689 9883

JAPAN

Microchip Technology Intl. Inc.
Benex S-1 6F
3-18-20, Shin Yokohama
Kohoku-Ku, Yokohama
Kanagawa 222 Japan
Tel: 81 45 471 6166 Fax: 81 45 471 6122

9/22/95