AN532

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 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 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 (Digital Signal Processors). 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 discussed in this application note 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 by offering a number of advantages:

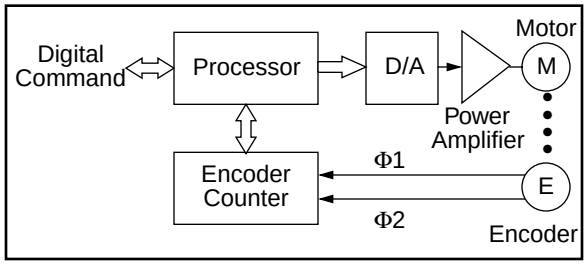
- Increased Acceleration, Velocity
- Improved Efficiency
- Reduced Audible Noise
- True Disturbance Rejection

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 D/A (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 a simple PID algorithm (as implemented in this application note). The combination of servo compensator and trajectory calculations can place significant demands on the processor.

The D/A conversion can be handled by a conventional DAC or by using the PIC17C42's 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 its simpler and cheaper.

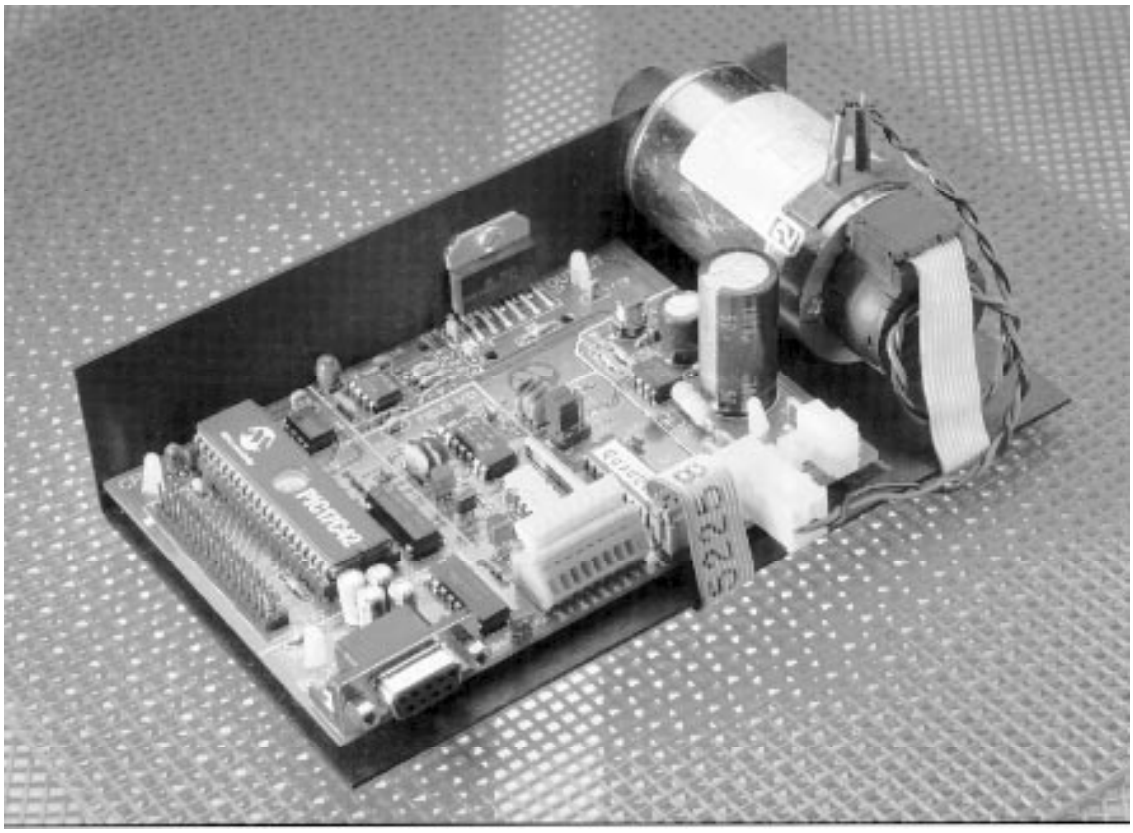
Sequential encoders produce quadrature pulse trains, from which position, speed, and direction of the motor rotation can be derived. The frequency is proportional to speed and each transition of F1 and F2 represents an increment of position. The phase of the signals is used to determine direction of rotation.

These encoder signals are usually decoded into Count Up and Count Down pulses, using a small state machine. 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 2. 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.

FIGURE 2: THE PIC17C42 BASED SERVO CONTROL BOARD



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 D/A 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 (Figure 3 and Figure 4).

FIGURE 3: SEQUENTIAL ENCODER SIGNALS

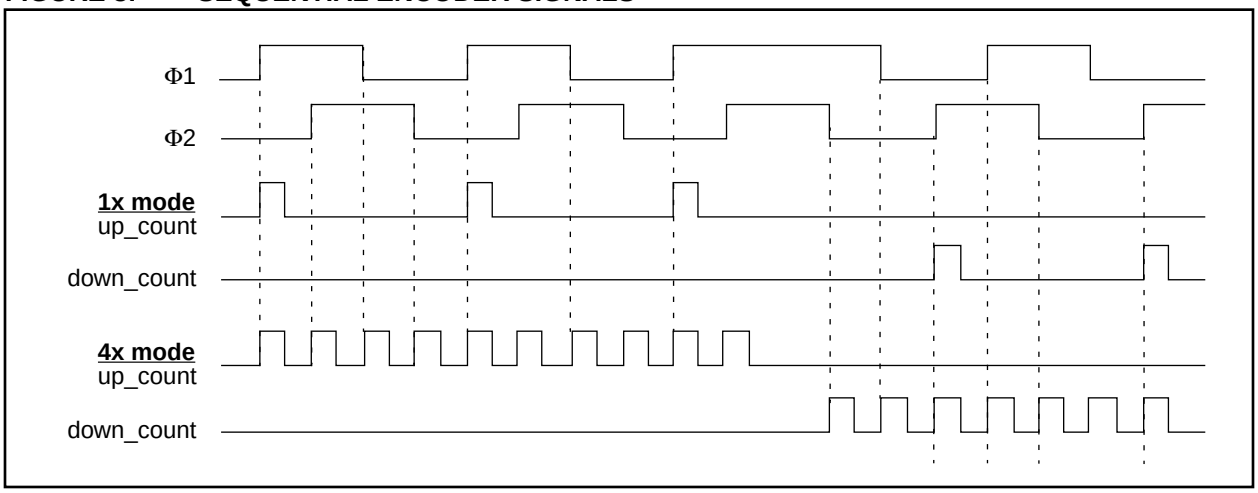
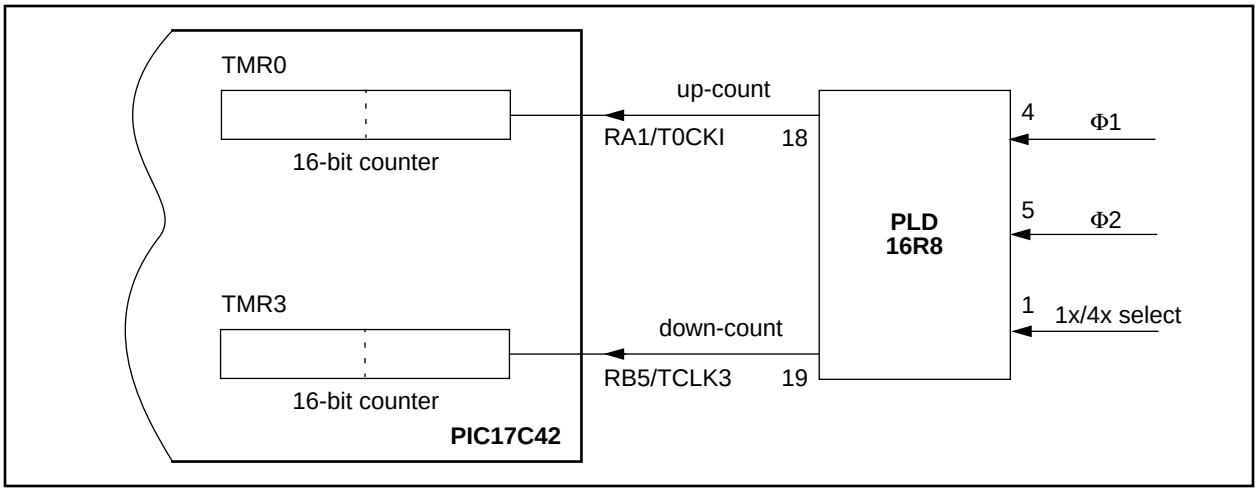


FIGURE 4: ENCODER INTERFACE SCHEME



THE COMPENSATOR

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

The standard digital PID algorithm's form is shown in Figure 5. *U(k)* is the position or velocity error and *Y(k)* is the output.

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

Integrator windup 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 in Figure 5 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 62.5 ns (at 16 MHz). This translates into 10-bit resolution at a 15.6 kHz rate or 1 part in 800 (9 1/2-bit) resolution at 20 kHz. 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 0.5 ms or more and a mechanical time constant of 20.0 ms or more. A 15 kHz PWM output is effectively equivalent to that of a linear amplifier.

In the system shown in Figure 6, 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.

FIGURE 5: DIGITAL PID IMPLEMENTATION

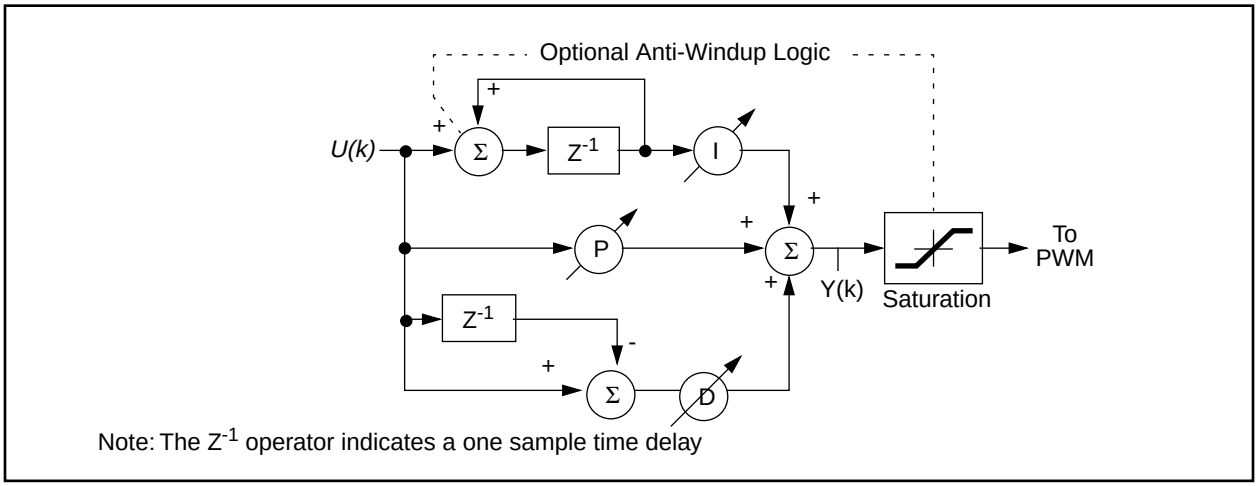
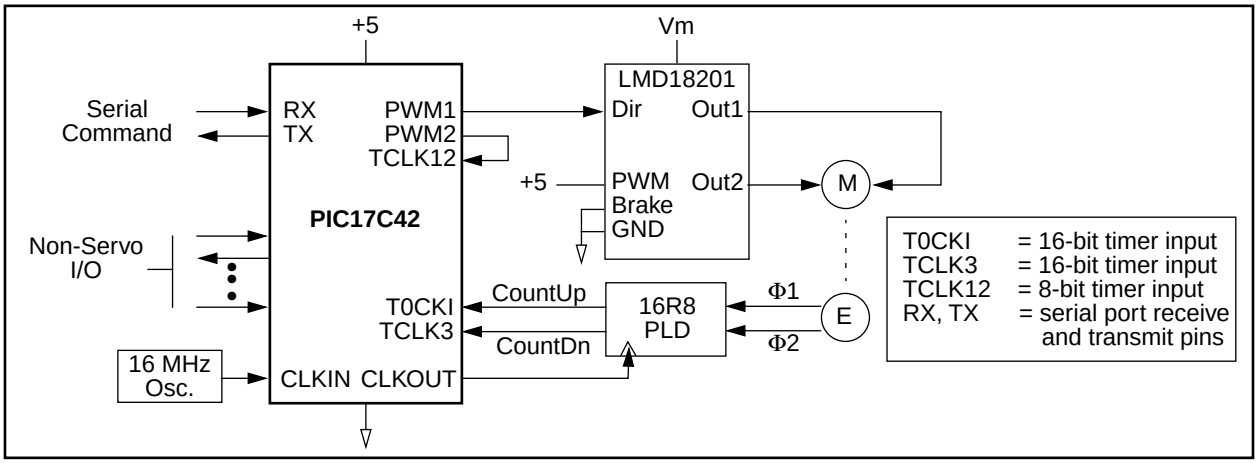


FIGURE 6: THE PIC17C42 SERVO SYSTEM



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 6. The PLD converts the quadrature pulses into two pulse streams: Count Up and Count Down (Figure 3). These signals are then fed to two 16-bit timers of the PIC17C42 (Timer3 and Timer0). 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, 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 1 ms, the maximum encoder rate would be 3.2767 MHz.**

Counter wraparound 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.

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 for a short move where maximum velocity was not reached (Figure 7).

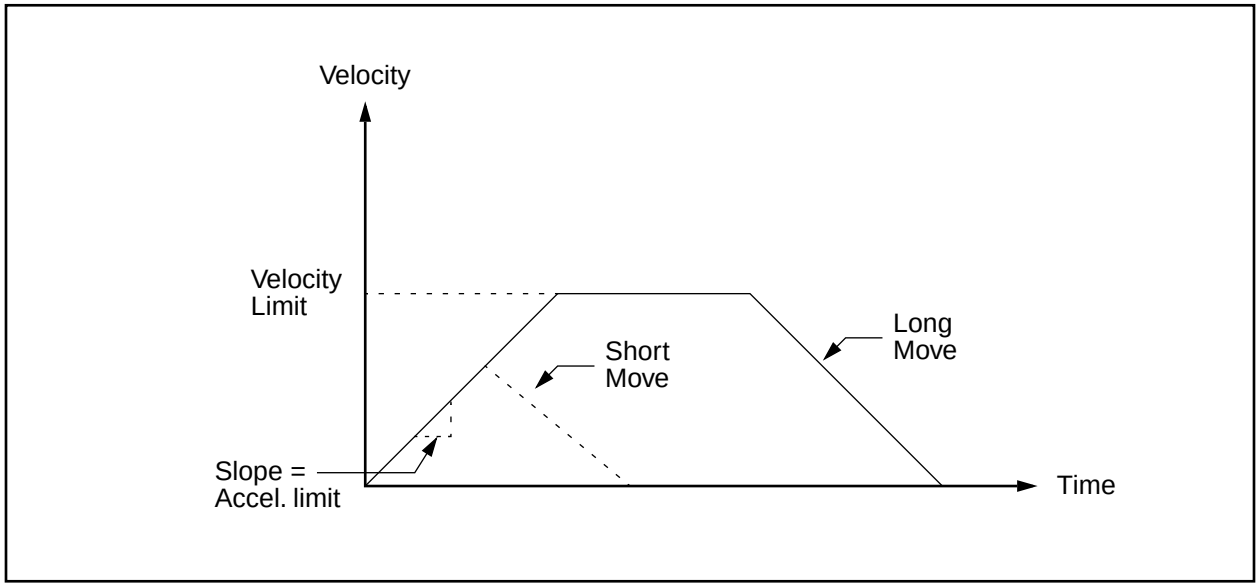
The `doPreMove` subroutine is invoked once at the beginning of a move to calculate the trajectory limits. The `doMove` routine is then invoked at every sample time to calculate new "desired" velocity and position values as follows:

$$VK = VK\text{-}1 + A \qquad (A = \text{Acceleration})$$

$$PK = PK\text{-}1 + VK\text{-}1 + A/2$$

For more details on trajectory generation, see Appendix E.

FIGURE 7: VELOCITY RAMP SEGMENTS FOR POSITION MOVES



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 (Timer2) 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 8).

FIGURE 8: SAMPLING SCHEME

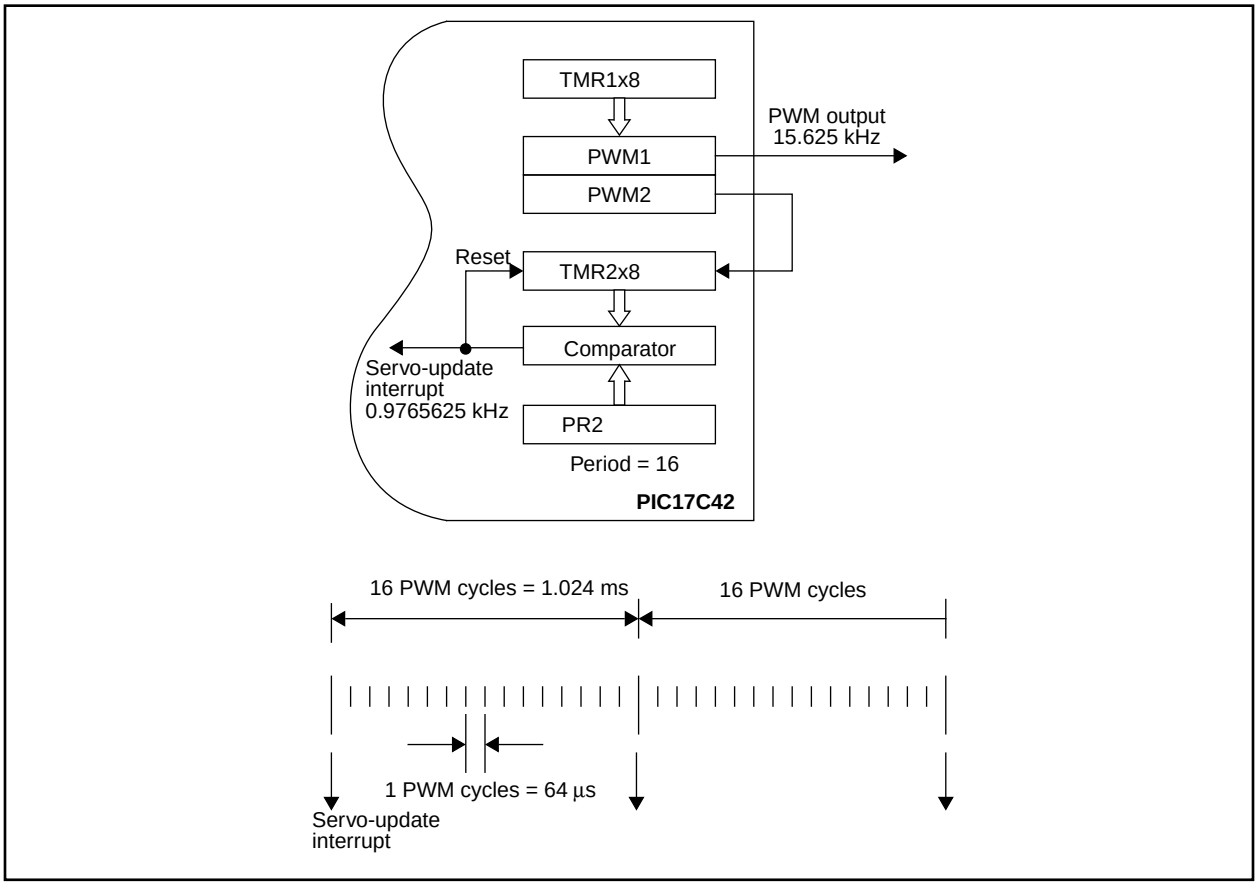


FIGURE 9: FLOWCHART FOR FOREGROUND PROCESSING

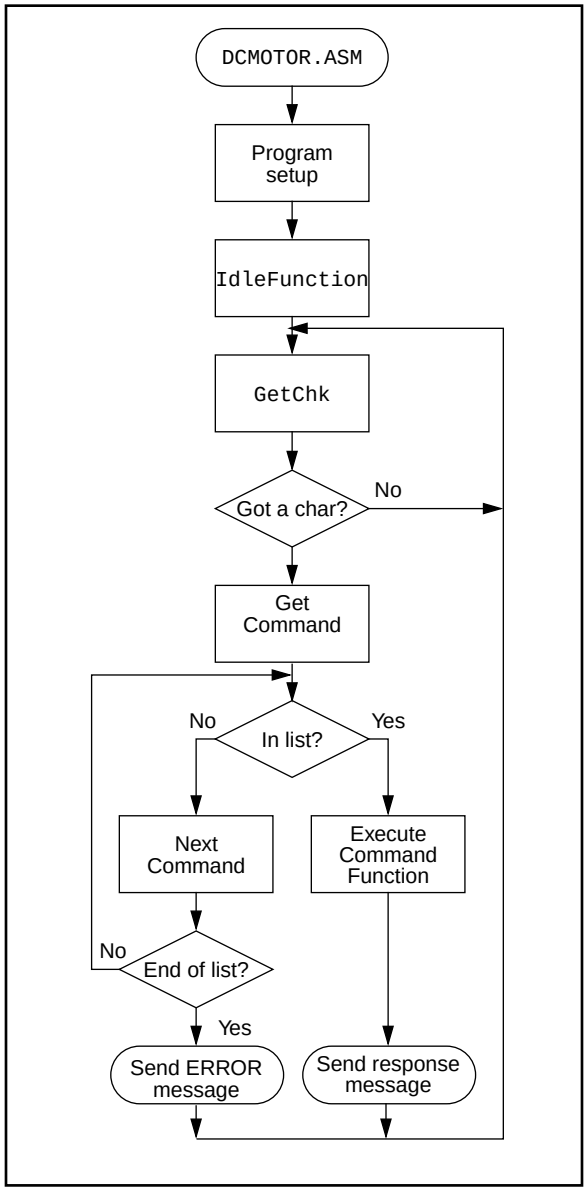
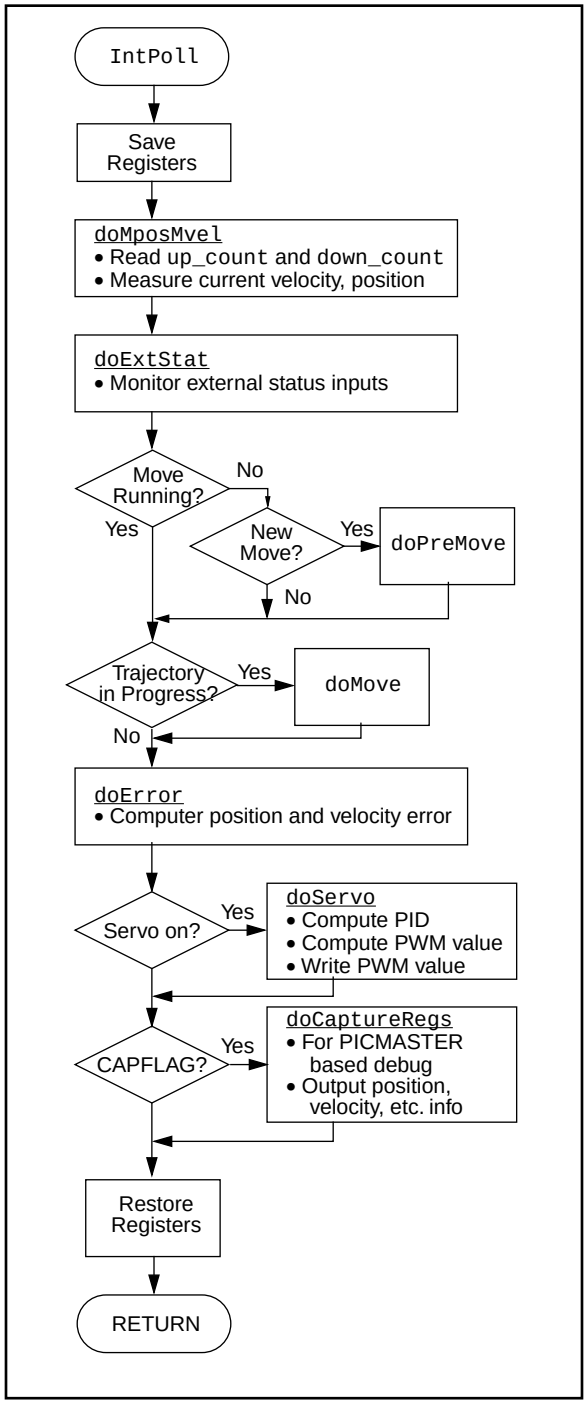


FIGURE 10: FLOWCHART FOR INTERRUPT SERVICE ROUTINE



The following events must occur in the interrupt service routine:

- Read Timers (TMR0 & 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 0.250 ms to execute with 16 MHz 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)
[00h to FFh, -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	#	Range
Velocity Limit	00h	0 to 8,388,607$_{10}$ *
Acceleration Limit	01h	0 to 8,388,607$_{10}$ **
Kp: Proportional Gain	02h	-32768$_{10}$ to 32767$_{10}$
Kd: Differential Gain	03h	-32768$_{10}$ to 32767$_{10}$
Ki: Integral Gain	04h	-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, (#) [00h to FFh]

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:

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, [-8,388,60810 to 8,388,60710]

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

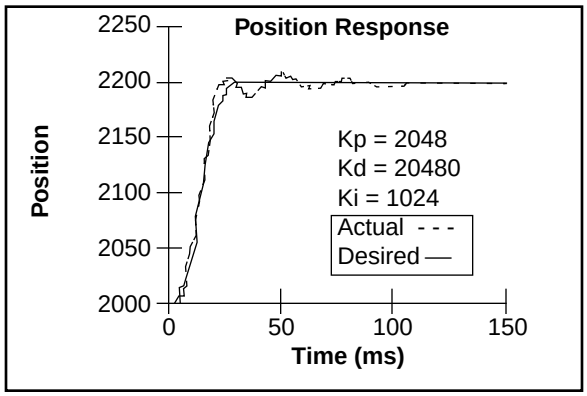
Once the PID loop is successfully implemented, the next challenge is to tune it. This was made simple through extensive use of the PICMASTER™ In-Circuit Emulator for the PIC17C42.

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

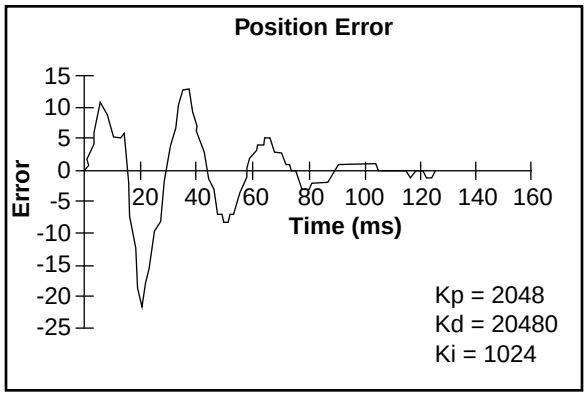
To tune the PID, first a small amount of diagnostics code is added in the servo routine (`doCaptureRegs`). This code simply outputs, at every sample point, the actual and desired position values, actual and desired velocity values, position error and velocity error by using a `TABLWT` instruction. These 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 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.

FIGURE 11: TYPICAL SERVO RESPONSE

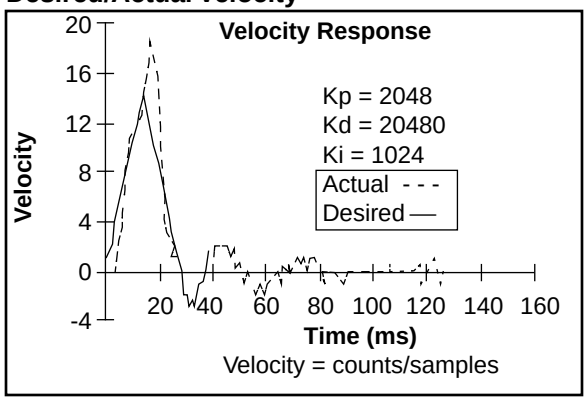
Desired/Actual Position



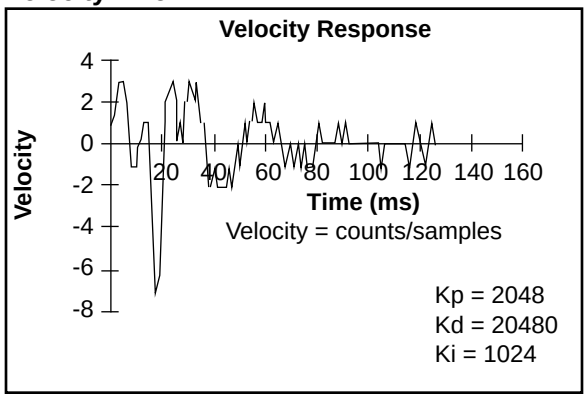
Position Error



Desired/Actual Velocity



Velocity Error



Under Windows multi-tasking environment, using a 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 setups 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 @8 MHz	PIC17C42 @16 MHz	PIC17C42 @25 MHz
Max Encoder Rate	1 MHz	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.

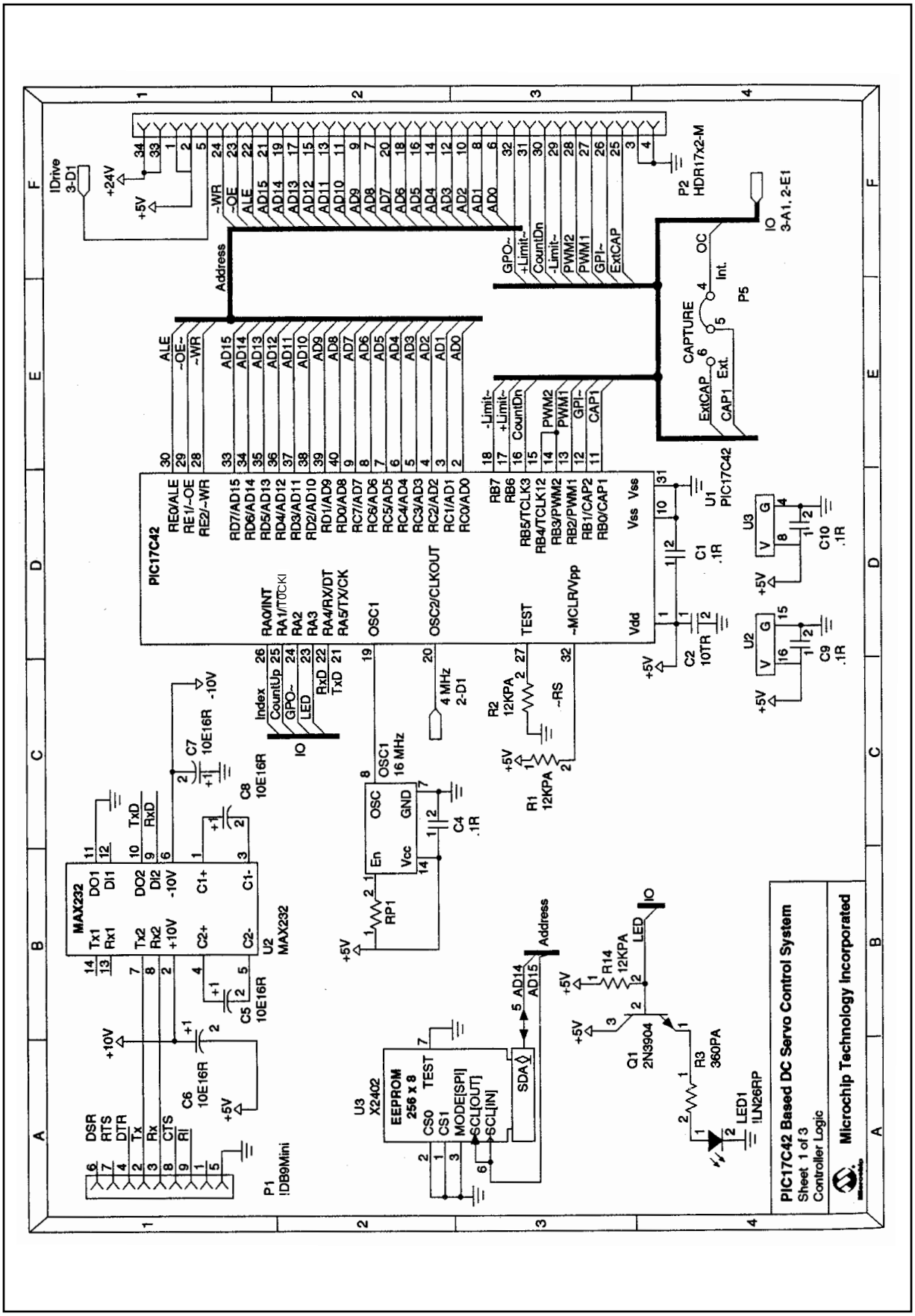
References

1.Thomas Bucella, “Comparing DSPs to Microprocessors in Motion Control Systems-Some Real World Data”, PCIM conference proceedings © 1990 Intertec Communications, Inc.

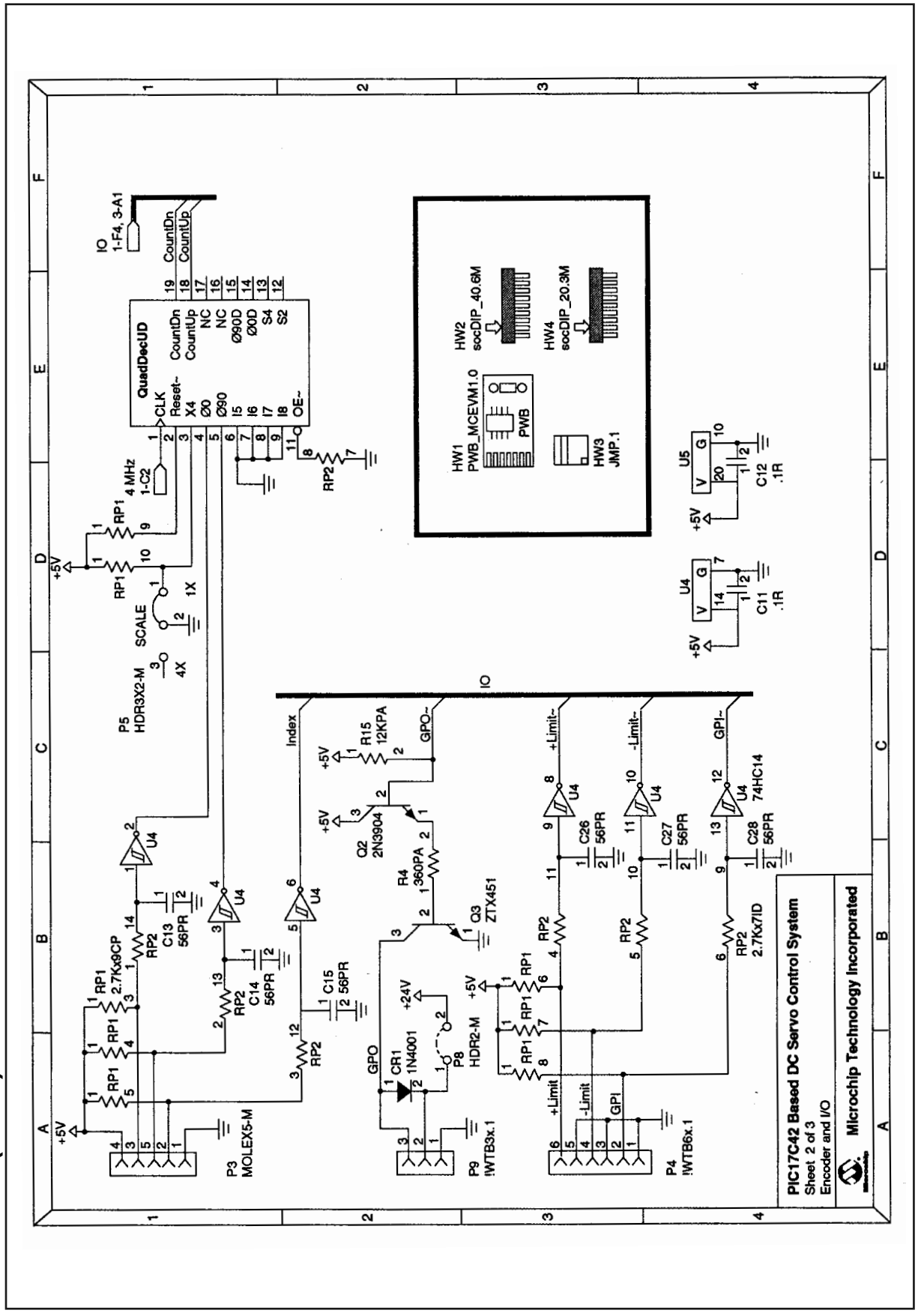
2.David M. Auslander, Cheng H. Tham, “Real-Time Software for Control” © 1990 Prentice-Hall, Inc., Englewood Cliffs, NJ

3.“DC Motors, Speed Controls, Servo Systems” Fifth Edition © 1980 Electro-Craft Corporation, Hopkins, MN

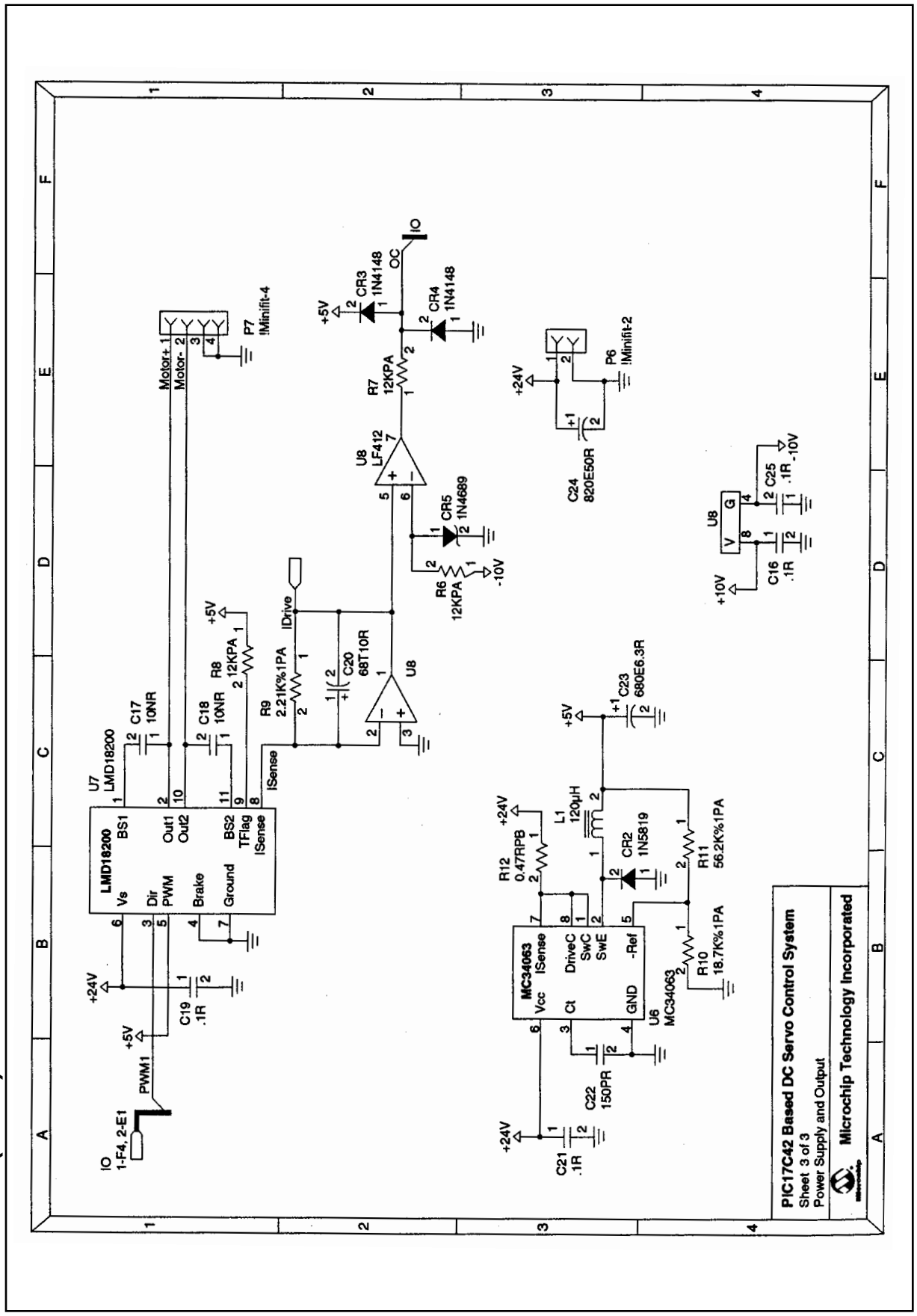
APPENDIX A: SCHEMATIC DIAGRAM



APPENDIX A (CONT.): SCHEMATIC DIAGRAM



APPENDIX A (CONT.): SCHEMATIC DIAGRAM

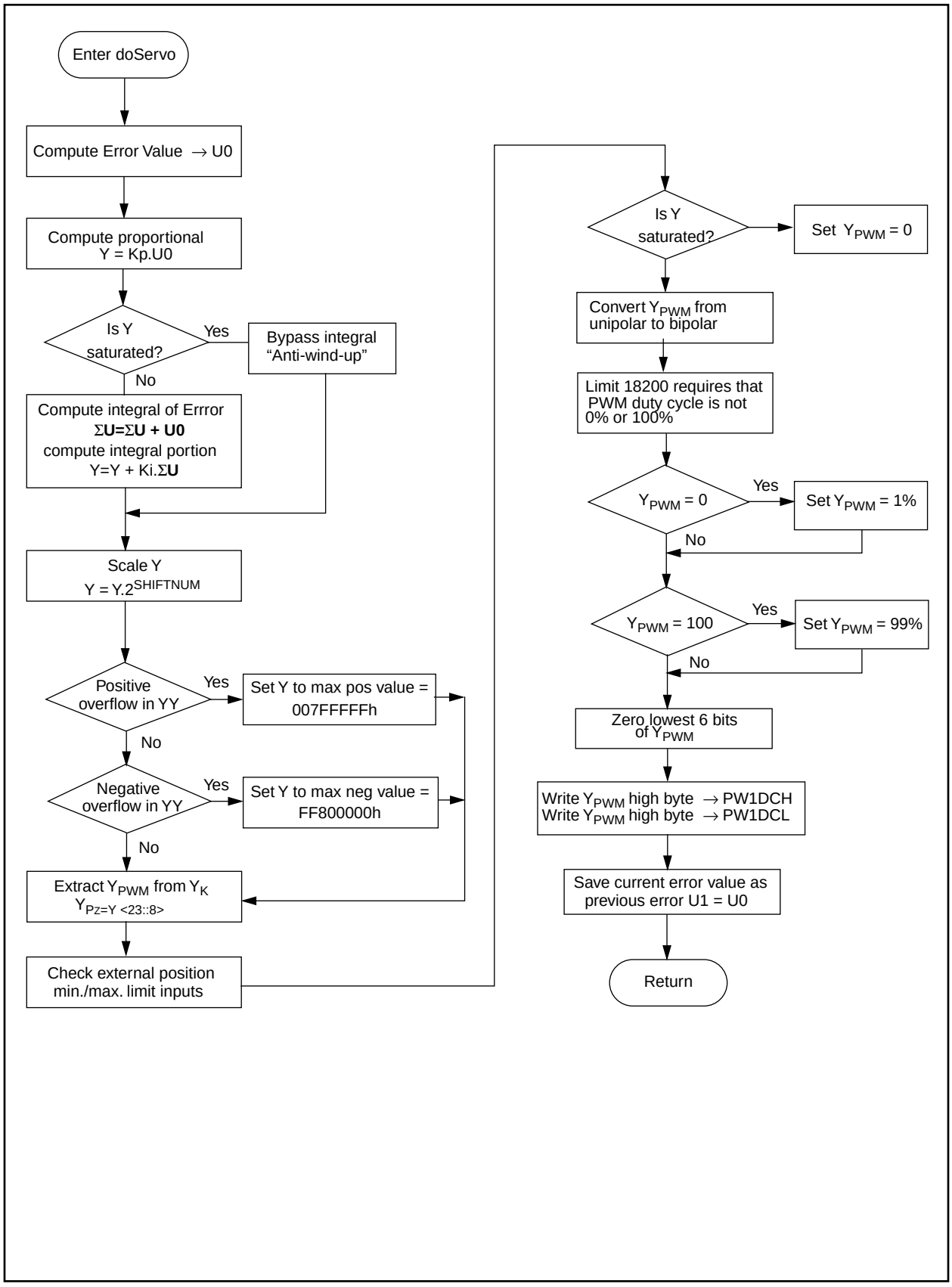


APPENDIX B:

```
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:

C.1 PID ALGORITHM FLOWCHART



C.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  ┐    │
       CPFSGT     SATFLAG           ; not accumulate integrator         │anti-
       CALL       doIntegral                                            │wind-
                                                                        │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                    ┐
       ANDLW      0x80                                                       │ If positive
       IORWF      Y+B3                                                       │ overflow, saturate
       CLRF       WREG                                                       │ y to maximum
       CPFSGT     Y+B3                                                       │ positive number
       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                   ┐
       IORLW      0x7F                                                       │ If negative
       ANDWF      Y+B3                                                       │ overflow, saturate
       SETF       WREG                                                       │ y to maximum
       CPFSLT     Y+B3                                                       │ negative number
       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
        CLR32   YPWM                                     been reached
        GOTO    mplimitok                                then zero PWM
tmlimit                                                  output
        BTFSS   EXTSTAT,BIT5
        GOTO    mplimitok
        CLR32   YPWM
mplimitok
        MOVLW   PW1DCH_INIT     ; adjustment from bipolar to unipolar   Convert PWM
        MOVPF   WREG,TMP+B1     ; for 50% duty cycle                    from unnipolar
        MOVLW   PW1DCL_INIT                                             to bipolar
        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

testmax
        CLRF    TMP+B2          ; check pwm maximum limit
        CLRF    YPWM+B2         ; LMD18200 must have a minimum pulse    PWM cycle must
        CLRF    YPWM+B3         ; so duty cycle must not be 0 or 100%   not be 0% of
        MVFP16  YPWMAX,TMP                                              100%
        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
        MOVFP   YPWM+B0,PW1DCL                                          values to PWM
        MOVFP   YPWM+B1,PW1DCH                                          registers

        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   TMR0H,WREG
       MOVPF   TMR0L,UPCOUNT+B0
       CPFSEQ  TMR0H                    ; Skip next if HI hasn't changed
       GOTO    readUp                   ; HI changed, re-read LO
       MOVPF   WREG,UPCOUNT+B1          ; OK to store HI now

       CLRFM   VELOCITY+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 position
       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 modeready, 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:

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

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:

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

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:

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

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 12:

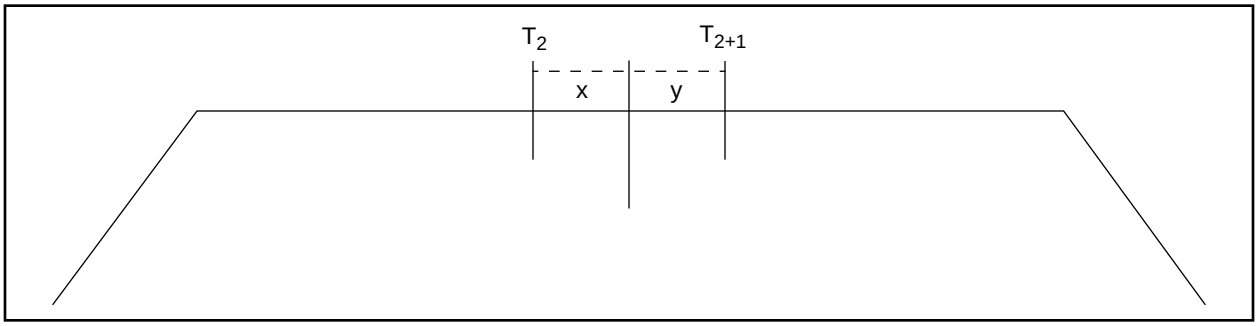


FIGURE 13:

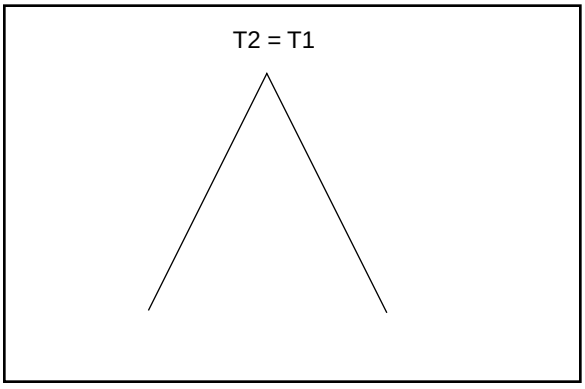
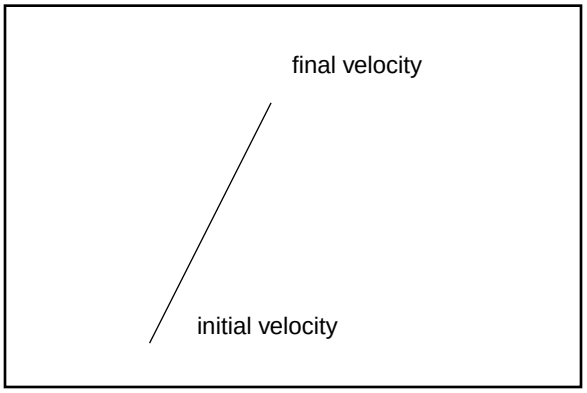


FIGURE 14:



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.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX F: COMPLETE CODE LISTING (DCMOTOR.LST)

```
MPASM 01.40 Released          DCMOTOR.ASM   1-16-1997  13:20:16         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001         TITLE “DCMOTOR SERVO CONTROL:  Revision:   1.9
                     00002 ;       Revised: 8/5/92
                     00003 ;
                     00004 ;       Program:          DCMOTOR.ASM
                     00005 ;       Revision Date:
                     00006 ;                         1-13-97      Compatibility with MPASMWIN 1.40
                     00007 ;
                     00008 ;       CREDIT:  Developed by Teknic Inc. 1992
                     00009 ;
                     00010 ;*************************************************************************
                     00011
                     00012 ;       PROCESSOR       PIC17C42
                     00013         LIST            P = 17C42, COLUMNS=120, XREF=YES, NOWRAP, LINES=255, R=DEC
                     00014
                     00015         #include “dcmotor.h17”
                     00001 ;*************************************************************************
                     00002 ;
                     00003 ; Header file for dcmotor.asm:
                     00004 ; Revised: 8/5/92
                     00005 ;*************************************************************************
                     00006 ;
                     00007 ; hardware constants
                     00008 ;
                     00009 ;
  00F42400           00010 MASTER_CLOCK    set    16000000        ; 16 MHz: change for diff clock speed
  003D0900           00011 CLKOUT          set    MASTER_CLOCK/4
  000003E8           00012 SAMPLE_RATE     set    1000
                     00013
  0000000C           00014 BAUD19200       set    (MASTER_CLOCK/((32*19200)-1)/2-1)
  00000019           00015 BAUD9600        set    (MASTER_CLOCK/((32*9600)-1)/2-1)
  00000067           00016 BAUD2400        set    (MASTER_CLOCK/((32*2400)-1)/2-1)
  000000CF           00017 BAUD1200        set    (MASTER_CLOCK/((32*1200)-1)/2-1)
  000000FF           00018 BAUD_MIN        set    0xFF
  00000019           00019 BAUD_DEFAULT    set    BAUD9600
                     00020
```

```
00000006          00021 TCON1_INIT      set    0x06
0000003F          00022 TCON2_INIT      set    0x3F
000000FF          00023 PR1_INIT        set    0xFF            ; set pwm frequency to CLKOUT/256 khz
0000000F          00024 PR2_INIT        set    (CLKOUT/(PR1_INIT+1)+SAMPLE_RATE/2)/SAMPLE_RATE-1
0000007F          00025 PW1DCH_INIT     set    (PR1_INIT/2)    ; set duty cycle to 50%, PW1DCH = PR1_INIT/2
000000C0          00026 PW1DCL_INIT     set    0xC0            ; and PW1DCL = 0xC0
00000080          00027 RTCSTA_INIT     set    0x80
00000090          00028 RCSTA_INIT      set    0x90
00000020          00029 TXSTA_INIT      set    0x20
00000019          00030 SPBRG_INIT      set    BAUD_DEFAULT
                  00031
000000F3          00032 DDRB_INIT       set    0xF3
00000000          00033 DDRD_INIT       set    0x00
                  00034 ;
                  00035 ;
                  00036 ;       max and min pwm values
                  00037 ;
00000040          00038 PWMINL          set    0x40
00000001          00039 PWMINH          set    0x01            ; 0x0000 + 0x0140 (min 10 bit pwm +5)
00000080          00040 PWMAXL          set    0x80
000000FE          00041 PWMAXH          set    0xFE            ; 0xFFC0 - 0x0140 ( max 10 bit pwm -5)
                  00042 ;
                  00043 ;
                  00044 ;
                  00045 ;     17c42 constants
                  00046 ;
                  00047 ;
                  00048 ;
00000000          00049 LO      EQU     0
00000001          00050 HI      EQU     1
00000000          00051 B0      EQU     0
00000001          00052 B1      EQU     1
00000002          00053 B2      EQU     2
00000003          00054 B3      EQU     3
00000007          00055 MSB     EQU     7
00000000          00056 LSB     EQU     0
                  00057 ;
                  00058 ; define special function registers:
                  00059
                  00060         #define W 0
                  00061         #define true    1
                  00062         #define false   0
                  00063         #define TRUE    1
                  00064         #define FALSE   0
                  00065
                  00066         cblock 0x00
00000000          00067                BIT0,BIT1,BIT2,BIT3,BIT4,BIT5,BIT6,BIT7
```

```
                 endc
                 cblock 0x00     ; define banks
00000000                BANK0,BANK1,BANK2,BANK3
                 endc
                 cblock 0x00             ; unbanked registers
00000000                INDF0,FSR0,PCL,PCLATH,ALUSTA,RTCSTA,CPUSTA,INTSTA
00000008           INDF1,FSR1,WREG,TMR0L,TMR0H,TBLPTRL,TBLPTRH,BSR
                 endc
                 cblock 0x10             ; bank0 registers
00000010                PORTA,DDRB,PORTB,RCSTA,RCREG,TXSTA,TXREG,SPBRG
                 endc
                 cblock 0x10             ; bank1 registers
00000010                         DDRC,PORTC,DDRD,PORTD,DDRE,PORTE,PIR,PIE
                 endc
                 cblock 0x10             ; bank2 registers
00000010                     TMR1,TMR2,TMR3L,TMR3H,PR1,PR2,PR3L,PR3H
                 endc
00000016   CA1L    equ     0x16            ; alternate function def
00000017   CA1H    equ     0x17
                 cblock  0x10     ; define bank3 variables
00000010                      PW1DCL,PW2DCL,PW1DCH,PW2DCH,CA2L,CA2H,TCON1,TCON2
                 endc
           ;****************************************************************
           ; define commonly used bits:
           ; ALUSTA bit definitions
                 #define _carry  ALUSTA,0
                 #define _c      ALUSTA,0
                 #define _cy     ALUSTA,0
                 #define _dc     ALUSTA,1
                 #define _z      ALUSTA,2
                 #define _ov     ALUSTA,3
                 #define _fs0    ALUSTA,4
                 #define _fs1    ALUSTA,5
                 #define _fs2    ALUSTA,6
                 #define _fs3    ALUSTA,7
           ; T0STA bit definitions
```

```
        #define _ps0    T0STA,1
        #define _ps1    T0STA,2
        #define _ps2    T0STA,3
        #define _ps3    T0STA,4
        #define _tosc   T0STA,5
        #define _tose   T0STA,6
        #define _intedg T0STA,7
; CPUSTA bit definitions
        #define _npd    CPUSTA,2
        #define _nto    CPUSTA,3
        #define _gint   CPUSTA,4
        #define _glintd CPUSTA,4
        #define _stkav  CPUSTA,5
; INTSTA bit definitions
        #define _inte   INTSTA,0
        #define _toie   INTSTA,1
        #define _t0ckie INTSTA,2
        #define _peie   INTSTA,3
        #define _intf   INTSTA,4
        #define _t0if   INTSTA,5
        #define _t0ckif INTSTA,6
        #define _peif   INTSTA,7
; PIR Bit definitions
        #define _rcif   PIR,0
        #define _txif   PIR,1
        #define _ca1if  PIR,2
        #define _ca2if  PIR,3
        #define _tmr1if PIR,4
        #define _tmr2if PIR,5
        #define _tmr3if PIR,6
        #define _rbif   PIR,7
; PIE Bit definitions
        #define _rcie   PIE,0
        #define _txie   PIE,1
        #define _ca1ie  PIE,2
        #define _ca2ie  PIE,3
        #define _tmr1ie  PIE,4
```

```
        #define _tmr2ie PIE,5
        #define _tmr3ie PIE,6
        #define _rbie    PIE,7
; RCSTA bit definitions
        #define _rx9d   RCVSTA,0
        #define _oerr   RCVSTA,1
        #define _ferr   RCVSTA,2
        #define _cren   RCVSTA,4
        #define _cren   RCVSTA,5
        #define _rx9    RCVSTA,6
        #define _spen   RCVSTA,7
; TXSTA bit definitions
        #define _tx9d   TXSTA,0
        #define _trmt   TXSTA,1
        #define _sync   TXSTA,4
        #define _txen   TXSTA,5
        #define _tx9    TXSTA,6
        #define _csrc   TXSTA,7
; TCON1 bit definitions
        #define _tmr1cs TCON1,0
        #define _tmr2cs TCON1,1
        #define _tmr3cs TCON1,2
        #define _t16    TCON1,3
        #define _ca1ed0 TCON1,4
        #define _ca1ed1 TCON1,5
        #define _ca2ed0 TCON1,6
        #define _ca2ed1 TCON1,7
; TCON2 bit definitions
        #define _tmr1on  TCON2,0
        #define _tmr2on  TCON2,1
        #define _tmr3on  TCON2,2
        #define _ca1pr3  TCON2,3
        #define _pwm1on  TCON2,4
        #define _pwm2on  TCON2,5
        #define _ca1ovf  TCON2,6
        #define _ca2ovf  TCON2,7
;
;
```

```
                    00209 ;
                    00210 ;     ascii constants
                    00211 ;
                    00212 ;
0000000D            00213 CR      set    0x0D
00000018            00214 CAN     set    0x18
00000008            00215 BS      set    0x08
00000020            00216 SP      set    0x20
0000000A            00217 LF      set    0x0A
0000002D            00218 MN      set    '-'
                    00219 ;
                    00220 ;
                    00221 ;*****************************************************************************
                    00222 ;
00000001            00223 DECIO   EQU     TRUE                    ; true for decimal, false for hex
                    00224 ;
                    00225 ;     cmds constants and macros
                    00226 ;
                    00227 ;
00000001            00228 CHARREADY       set    0x01
                    00229 ;
                    00230 ;
00000008            00231 NUMPAR          set    0x08
                    00232 ;
                    00233 ; Response characters
                    00234 ;
00000021            00235 CMD_OK          set    '!'
0000003F            00236 CMD_BAD         set    '?'
                    00237 ;
                    00238 ; Exit values
                    00239 ;
                    00240 ;
00000000            00241 HEX_SP          set    0x00
00000001            00242 HEX_MN          set    0x01
00000002            00243 HEX_CR          set    0x02
00000003            00244 HEX_CAN         set    0x03
                    00245 ;
00000000            00246 DEC_SP          set    0x00
00000001            00247 DEC_MN          set    0x01
00000002            00248 DEC_CR          set    0x02
00000003            00249 DEC_CAN         set    0x03
                    00250 ;
                    00251 ;
                    00252 ; Command characters
                    00253 ;
0000000D            00254 DO_NULL                set    CR
0000004D            00255 DO_MOVE                set    'M'     ; M
```

```
0000004F          00256 DO_MODE                 set    'O'     ; O
00000053          00257 DO_SETPARAMETER         set    'S'     ; S
00000052          00258 DO_READPARAMETER        set    'R'     ; R
00000043          00259 DO_SHUTTER              set    'C'     ; C
00000050          00260 DO_READCOMPOSITION      set    'P'     ; P
00000056          00261 DO_READCOMVELOCITY      set    'V'     ; V
00000070          00262 DO_READACTPOSITION      set    'p'     ; p
00000076          00263 DO_READACTVELOCITY      set    'v'     ; v
00000058          00264 DO_EXTERNALSTATUS       set    'X'     ; X
00000059          00265 DO_MOVESTATUS           set    'Y'     ; Y
00000049          00266 DO_READINDPOSITION      set    'I'     ; I
00000048          00267 DO_SETPOSITION          set    'H'     ; H
0000005A          00268 DO_RESET                set    'Z'     ; Z
00000073          00269 DO_STOP                 set    's'     ; s
00000063          00270 DO_CAPTURE              set    'c'     ; c
                  00271
                  00272 ;*****************************************************************************
                  00273 ; NAME:         CMD_DEF
                  00274 ;
                  00275 ; DESCRIPTION:  Creates all the definitions for a command table data struc-
                  00276 ;               ture. The first word is at the command character used, and
                  00277 ;               the second word is a pointer to the function that handles
                  00278 ;               this command function.
                  00279 ;
                  00280 ; ENTRY CONDITIONS:     Must be contiguous with the other entries for the
                  00281 ;                       function to work.
                  00282 ;
                  00283 ; ARGUMENTS:            FUNC    command execution function
                  00284 ;                       ROOT    NAME ROOT
                  00285 ;
                  00286
                  00287 CMD_DEF MACRO   FUNC,ROOT
                  00288
                  00289         DATA    ROOT
                  00290         DATA    FUNC
                  00291         ENDM
                  00292
00000002          00293 CMD_ENTRY_LENGTH        set    2
                  00294
                  00295 ;*****************************************************************************
                  00296
                  00297 ;*****************************************************************************
                  00298 ; NAME:         CMD_START
                  00299 ;
                  00300 ; DESCRIPTION:  Labels the start of the command table.
                  00301 ;
                  00302
```

```
00303 CMD_START MACRO LABEL
00304
00305 LABEL
00306         ENDM                                    ;
00307
00308 ;*****************************************************************************
00309
00310 ;*****************************************************************************
00311 ; NAME:         CMD_END
00312 ;
00313 ; DESCRIPTION:  Marks the end of the command table with an entry of 0x00
00314 ;
00315
00316 CMD_END MACRO
00317 ;                                               ;
00318         DATA    0x00
00319         ENDM                                    ;
00320
00321 ;*****************************************************************************
00322
00323 ;*****************************************************************************
00324 ; NAME:         CLR32
00325 ;
00326 ; DESCRIPTION:  Clear 4 consecutive bytes of data memory
00327 ;
00328 ; ARGUMENTS:    0 => a
00329 ;
00330 ; TIMING (cycles): 4
00331 ;
00332
00333 CLR32 MACRO     a
00334         CLRF    a+B0, F
00335         CLRF    a+B1, F
00336         CLRF    a+B2, F
00337         CLRF    a+B3, F
00338
00339         ENDM                                    ;
00340
00341 ;*****************************************************************************
00342
00343 ;*****************************************************************************
00344 ; NAME:         CLR24
00345 ;
00346 ; DESCRIPTION:  Clear 3 consecutive bytes of data memory
00347 ;
00348 ; ARGUMENTS:    0 => a
00349 ;
```

```
00350 ; TIMING (cycles): 3
00351 ;
00352
00353 CLR24 MACRO     a
00354         CLRF    a+B0, F
00355         CLRF    a+B1, F
00356         CLRF    a+B2, F
00357
00358         ENDM
00359
00360 ;*****************************************************************************
00361
00362 ;*****************************************************************************
00363 ; NAME:         CLR16
00364 ;
00365 ; DESCRIPTION:  Clear 2 consecutive bytes of data memory
00366 ;
00367 ; ARGUMENTS:    0 => a
00368 ;
00369 ; TIMING(cycles): 2
00370 ;
00371
00372 CLR16 MACRO     a
00373         CLRF    a+B0, F
00374         CLRF    a+B1, F
00375
00376         ENDM
00377
00378 ;*****************************************************************************
00379
00380 ;*****************************************************************************
00381 ; NAME:         MOV32
00382 ;
00383 ; DESCRIPTION:  32 bit move
00384 ;
00385 ; ARGUMENTS:    a => b
00386 ;
00387 ; TIMING (cycles):8
00388 ;
00389
00390 MOV32 MACRO     a,b
00391
00392         MOVFP   a+B0,WREG               ; get byte of a into w
00393         MOVPF   WREG,b+B0               ; move to b(B0)
00394         MOVFP   a+B1,WREG               ; get byte of a into w
00395         MOVPF   WREG,b+B1               ; move to b(B1)
00396         MOVFP   a+B2,WREG               ; get byte of a into w
```

```
00397         MOVPF   WREG,b+B2               ; move to b(B2)
00398         MOVFP   a+B3,WREG               ; get byte of a into w
00399         MOVPF   WREG,b+B3               ; move to b(B3)
00400
00401         ENDM
00402
00403 ;*****************************************************************************
00404
00405 ;*****************************************************************************
00406 ; NAME:         MOV24
00407 ;
00408 ; DESCRIPTION:  24 bit move
00409 ;
00410 ; ARGUMENTS:    a => b
00411 ;
00412 ; TIMING (cycles): 6
00413 ;
00414
00415 MOV24 MACRO     a,b
00416
00417         MOVFP   a+B0,WREG               ; get byte of a into w
00418         MOVPF   WREG,b+B0               ; move to b(B0)
00419         MOVFP   a+B1,WREG               ; get byte of a into w
00420         MOVPF   WREG,b+B1               ; move to b(B1)
00421         MOVFP   a+B2,WREG               ; get byte of a into w
00422         MOVPF   WREG,b+B2               ; move to b(B2)
00423
00424         ENDM
00425
00426 ;*****************************************************************************
00427
00428 ;*****************************************************************************
00429 ; NAME:         MOV16
00430 ;
00431 ; DESCRIPTION:  16 bit move
00432 ;
00433 ; ARGUMENTS:    a => b
00434 ;
00435 ; TIMING (in cycles): 4
00436 ;
00437
00438 MOV16 MACRO     a,b
00439
00440         MOVFP   a+B0,WREG               ; get byte of a into w
00441         MOVPF   WREG,b+B0               ; move to b(B0)
00442         MOVFP   a+B1,WREG               ; get byte of a into w
00443         MOVPF   WREG,b+B1               ; move to b(B1)
```

```
00444
00445         ENDM
00446
00447 ;*******************************************************************************
00448
00449 ;*******************************************************************************
00450 ; NAME:         MVPF32
00451 ;
00452 ; DESCRIPTION:  32 bit move from P data memory to F data memory
00453 ;
00454 ; ARGUMENTS:    A => B
00455 ;
00456 ; TIMING (cycles): 4
00457 ;
00458
00459 MVPF32 MACRO    A,B
00460
00461         MOVPF   A+B0,B+B0               ; move A(B0) to B(B0)
00462         MOVPF   A+B1,B+B1               ; move A(B1) to B(B1)
00463         MOVPF   A+B2,B+B2               ; move A(B2) to B(B2)
00464         MOVPF   A+B3,B+B3               ; move A(B3) to B(B3)
00465
00466         ENDM
00467
00468 ;*******************************************************************************
00469
00470 ;*******************************************************************************
00471 ; NAME:         MVPF24
00472 ;
00473 ; DESCRIPTION:  24 bit move from P data memory to F data memory
00474 ;
00475 ; ARGUMENTS:    A => B
00476 ;
00477 ;
00478 ; TIMING (cycles): 3
00479 ;
00480
00481 MVPF24 MACRO    A,B
00482
00483         MOVPF   A+B0,B+B0               ; move A(B0) to B(B0)
00484         MOVPF   A+B1,B+B1               ; move A(B1) to B(B1)
00485         MOVPF   A+B2,B+B2               ; move A(B2) to B(B2)
00486
00487         ENDM
00488
00489 ;*******************************************************************************
00490
```

```
00491 ;*****************************************************************************
00492 ; NAME:         MVPF16
00493 ;
00494 ; DESCRIPTION:  16 bit move from P data memory to F data memory
00495 ;
00496 ; ARGUMENTS:    A => B
00497 ;
00498 ; TIMING (cycles): 2
00499 ;
00500
00501 MVPF16 MACRO    A,B
00502
00503         MOVPF   A+B0,B+B0               ; move A(B0) to B(B0)
00504         MOVPF   A+B1,B+B1               ; move A(B1) to B(B1)
00505
00506         ENDM
00507
00508 ;*****************************************************************************
00509
00510
00511 ;*****************************************************************************
00512 ; NAME:         MVFP32
00513 ;
00514 ; DESCRIPTION:  32 bit move from F data memory to P data memory
00515 ;
00516 ; ARGUMENTS:    A => B
00517 ;
00518 ; TIMING (cycles): 4
00519
00520 MVFP32 MACRO    A,B
00521
00522         MOVFP   A+B0,B+B0               ; move A(B0) to B(B0)
00523         MOVFP   A+B1,B+B1               ; move A(B1) to B(B1)
00524         MOVFP   A+B2,B+B2               ; move A(B2) to B(B2)
00525         MOVFP   A+B3,B+B3               ; move A(B3) to B(B3)
00526
00527         ENDM
00528
00529 ;*****************************************************************************
00530
00531 ;*****************************************************************************
00532 ; NAME:         MVFP24
00533 ;
00534 ; DESCRIPTION:  24 bit move from F data memory to P data memory
00535 ;
00536 ; ARGUMENTS:    A => B
00537 ;
```

```
00538 ; TIMING (cycles): 3
00539 ;
00540
00541 MVFP24 MACRO    A,B
00542
00543         MOVFP   A+B0,B+B0               ; move A(B0) to B(B0)
00544         MOVFP   A+B1,B+B1               ; move A(B1) to B(B1)
00545         MOVFP   A+B2,B+B2               ; move A(B2) to B(B2)
00546
00547         ENDM
00548
00549 ;*******************************************************************************
00550
00551 ;*******************************************************************************
00552 ; NAME:         MVFP16
00553 ;
00554 ; DESCRIPTION:  16 bit move from F data memory to P data memory
00555 ;
00556 ; ARGUMENTS:    A => B
00557 ;
00558 ; TIMING (cycles): 2
00559 ;
00560
00561 MVFP16 MACRO    A,B
00562
00563         MOVFP   A+B0,B+B0               ; move A(B0) to B(B0)
00564         MOVFP   A+B1,B+B1               ; move A(B1) to B(B1)
00565
00566         ENDM
00567
00568 ;*******************************************************************************
00569
00570 ;*******************************************************************************
00571 ; NAME:         LOADAB
00572 ;
00573 ; DESCRIPTION:  Loads extended math library AARG and BARG
00574 ;
00575 ; ARGUMENTS:    A => AARG
00576 ;               B => BARG
00577 ;
00578 ; TIMING (cycles): 4
00579
00580 LOADAB MACRO    A,B
00581
00582         MOVFP   A+B0,AARG+B0            ; load lo byte of A to AARG
00583         MOVFP   A+B1,AARG+B1            ; load hi byte of A to AARG
00584         MOVFP   B+B0,BARG+B0            ; load lo byte of B to BARG
```

```
00585         MOVFP   B+B1,BARG+B1            ; load hi byte of B to BARG
00586
00587         ENDM
00588
00589 ;*****************************************************************************
00590
00591 ;*****************************************************************************
00592 ; NAME:         ADD32
00593 ;
00594 ; DESCRIPTION:  32 bit add
00595 ;
00596 ; ARGUMENTS:    a + b => b
00597 ;
00598 ; TIMING (cycles): 8
00599 ;
00600
00601 ADD32 MACRO     a,b
00602
00603         MOVFP   a+B0,WREG               ; get lowest byte of a into w
00604         ADDWF   b+B0, F                 ; add lowest byte of b, save in b(B0)
00605         MOVFP   a+B1,WREG               ; get 2nd byte of a into w
00606         ADDWFC  b+B1, F                 ; add 2nd byte of b, save in b(B1)
00607         MOVFP   a+B2,WREG               ; get 3rd byte of a into w
00608         ADDWFC  b+B2, F                 ; add 3rd byte of b, save in b(B2)
00609         MOVFP   a+B3,WREG               ; get 4th byte of a into w
00610         ADDWFC  b+B3, F                 ; add 4th byte of b, save in b(B3)
00611
00612         ENDM
00613
00614 ;*****************************************************************************
00615
00616 ;*****************************************************************************
00617 ; NAME:         ADD24
00618 ;
00619 ; DESCRIPTION:  24 bit add
00620 ;
00621 ; ARGUMENTS:    a + b => b
00622 ;
00623 ; TIMING (cycles): 6
00624 ;
00625
00626 ADD24 MACRO     a,b
00627
00628         MOVFP   a+B0,WREG               ; get lowest byte of a into w
00629         ADDWF   b+B0, F                 ; add lowest byte of b, save in b(B0)
00630         MOVFP   a+B1,WREG               ; get 2nd byte of a into w
00631         ADDWFC  b+B1, F                 ; add 2nd byte of b, save in b(B1)
```

```
        MOVFP   a+B2,WREG              ; get 3rd byte of a into w
        ADDWFC  b+B2, F                ; add 3rd byte of b, save in b(B2)

        ENDM

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

;*****************************************************************************
; NAME:         ADD16
;
; DESCRIPTION:  16 bit add
;
; ARGUMENTS:    a + b => b
;
;
; TIMING (cycles): 4
;

ADD16 MACRO     a,b

        MOVFP   a+B0,WREG              ; get lowest byte of a into w
        ADDWF   b+B0, F                ; add lowest byte of b, save in b(B0)
        MOVFP   a+B1,WREG              ; get 2nd byte of a into w
        ADDWFC  b+B1, F                ; add 2nd byte of b, save in b(B1)

        ENDM

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

;*****************************************************************************
; NAME:         SUB32
;
; DESCRIPTION:  32 bit subtract
;
;
; ARGUMENTS:    b - a => b
;
; TIMING (cycles): 8
;

SUB32 MACRO     a,b

        MOVFP   a+B0,WREG              ; get lowest byte of a into w
        SUBWF   b+B0, F                ; sub lowest byte of b, save in b(B0)
        MOVFP   a+B1,WREG              ; get 2nd byte of a into w
        SUBWFB  b+B1, F                ; sub 2nd byte of b, save in b(B1)
        MOVFP   a+B2,WREG              ; get 3rd byte of a into w
```

```
00679         SUBWFB  b+B2, F                 ; sub 3rd byte of b, save in b(B2)
00680         MOVFP   a+B3,WREG               ; get 4th byte of a into w
00681         SUBWFB  b+B3, F                 ; sub 4th byte of b, save in b(B3)
00682
00683         ENDM
00684
00685 ;*****************************************************************************
00686
00687 ;*****************************************************************************
00688 ; NAME:         SUB24
00689 ;
00690 ; DESCRIPTION:  24 bit subtract
00691 ;
00692 ; ARGUMENTS:    b - a => b
00693 ;
00694 ; TIMING (in cycles): 6
00695 ;
00696
00697 SUB24 MACRO     a,b
00698
00699         MOVFP   a+B0,WREG               ; get lowest byte of a into w
00700         SUBWF   b+B0, F                 ; sub lowest byte of b, save in b(B0)
00701         MOVFP   a+B1,WREG               ; get 2nd byte of a into w
00702         SUBWFB  b+B1, F                 ; sub 2nd byte of b, save in b(B1)
00703         MOVFP   a+B2,WREG               ; get 3rd byte of a into w
00704         SUBWFB  b+B2, F                 ; sub 3rd byte of b, save in b(B2)
00705
00706         ENDM
00707
00708 ;*****************************************************************************
00709
00710 ;*****************************************************************************
00711 ; NAME:         SUB16
00712 ;
00713 ; DESCRIPTION:  16 bit subtract
00714 ;
00715 ; ARGUMENTS:    b - a => b
00716 ;
00717 ; TIMING (cycles): 4
00718 ;
00719
00720 SUB16 MACRO     a,b
00721
00722         MOVFP   a+B0,WREG               ; get lowest byte of a into w
00723         SUBWF   b+B0, F                 ; sub lowest byte of b, save in b(B0)
00724         MOVFP   a+B1,WREG               ; get 2nd byte of a into w
00725         SUBWFB  b+B1, F                 ; sub 2nd byte of b, save in b(B1)
```

```

        ENDM

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

;*****************************************************************************
; NAME: RLC32
;
; DESCRIPTION: 32 bit rotate left
;
; ARGUMENTS: 2*a => a
;
; TIMING (cycles): 5
;

RLC32 MACRO     a

        BCF     _carry
        RLCF    a+B0, F
        RLCF    a+B1, F
        RLCF    a+B2, F
        RLCF    a+B3, F

        ENDM

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

;*****************************************************************************
; NAME:         RLC24
;
; DESCRIPTION:  24 bit rotate left
;
; ARGUMENTS:    2*a => a
;
; TIMING (cycles): 4
;

RLC24 MACRO     a

        BCF     _carry
        RLCF    a+B0, F
        RLCF    a+B1, F
        RLCF    a+B2, F

        ENDM

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

```
;*****************************************************************************
; NAME:         RLC16
;
; DESCRIPTION:  16 bit rotate left
;
; ARGUMENTS:    2*a => a
;
;
; TIMING (cycles): 3
;

RLC16 MACRO     a
        BCF     _carry
        RLCF    a+B0, F
        RLCF    a+B1, F

        ENDM

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

;*****************************************************************************
; NAME:         RRC32
;
; DESCRIPTION:  32 bit rotate right
;
; ARGUMENTS:    a/2 => a
;
; TIMING (cycles): 5
;

RRC32 MACRO     a

        RLCF    a+B3,W                  ; move sign into carry bit
        RRCF    a+B3, F
        RRCF    a+B2, F
        RRCF    a+B1, F
        RRCF    a+B0, F

        ENDM

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

;*****************************************************************************
; NAME:         RRC24
;
; DESCRIPTION:  24 bit rotate right
```

```
;
; ARGUMENTS:    a/2 => a
;
; TIMING (cycles): 4
;

RRC24 MACRO     a

        RLCF    a+B2,W                  ; move sign into carry bit
        RRCF    a+B2, F
        RRCF    a+B1, F
        RRCF    a+B0, F

        ENDM

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

;*****************************************************************************
; NAME:         RRC16
;
; DESCRIPTION:  16 bit rotate right
;
; ENTRY CONDITIONS: a/2 => a
;
; TIMING (cycles): 3
;

RRC16 MACRO     a

        RLCF    a+B1,W                  ; move sign into carry bit
        RRCF    a+B1, F
        RRCF    a+B0, F

        ENDM

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

;*****************************************************************************
; NAME:         INC24
;
; DESCRIPTION:  24 bit increment
;
; ARGUMENTS:    a+1 => a
;
; TIMING (cycles): 4
;

```

```
00867 INC24 MACRO     a
00868
00869         CLRF    WREG, F
00870         INCF    a+B0, F
00871         ADDWFC  a+B1, F
00872         ADDWFC  a+B2, F
00873
00874         ENDM
00875
00876 ;*****************************************************************************
00877
00878 ;*****************************************************************************
00879 ; NAME:         INC16
00880 ;
00881 ; DESCRIPTION:  16 bit increment
00882 ;
00883 ; ARGUMENTS:    a+1 => a
00884 ;
00885 ; TIMING (cycles): 3
00886 ;
00887
00888 INC16 MACRO     a
00889
00890         CLRF    WREG, F
00891         INCF    a+B0, F
00892         ADDWFC  a+B1, F
00893
00894         ENDM
00895
00896 ;*****************************************************************************
00897
00898 ;*****************************************************************************
00899 ; NAME:         DEC24
00900 ;
00901 ; DESCRIPTION:  Decrement A 24 Bit Number
00902 ;
00903 ; ARGUMENTS:    a-1 => a
00904 ;
00905 ; TIMING (cycles): 4
00906 ;
00907
00908 DEC24 MACRO     a
00909
00910         CLRF    WREG, F
00911         DECF    a+B0, F
00912         SUBWFB  a+B1, F
00913         SUBWFB  a+B2, F
```

```

        ENDM

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

;*****************************************************************************
; DESCRIPTION:  Decrement A 16 Bit Number
;
; ARGUMENTS:    a-1 => a
;
; TIMING (cycles): 3
;

DEC16 MACRO     a

        CLRF    WREG, F
        DECF    a+B0, F
        SUBWFB  a+B1, F

        ENDM

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

;*****************************************************************************
; NAME:         NEG32
;
; DESCRIPTION:  32 bit negate
;
; ARGUMENTS:    -A => A
;
; TIMING (cycles): 9
;

NEG32 MACRO     A

        COMF    A+B0, F
        COMF    A+B1, F
        COMF    A+B2, F
        COMF    A+B3, F
        CLRF    WREG, F
        INCF    A+B0, F
        ADDWFC  A+B1, F
        ADDWFC  A+B2, F
        ADDWFC  A+B3, F

        ENDM

```

```
00961 ;*************************************************************************
00962
00963 ;*************************************************************************
00964 ; NAME:         NEG24
00965 ;
00966 ; DESCRIPTION:  24 bit negate
00967 ;
00968 ; ARGUMENTS:    -A => A
00969 ;
00970 ; TIMING (cycles): 7
00971 ;
00972
00973 NEG24 MACRO     A
00974
00975         COMF    A+B0, F
00976         COMF    A+B1, F
00977         COMF    A+B2, F
00978         CLRF    WREG, F
00979         INCF    A+B0, F
00980         ADDWFC  A+B1, F
00981         ADDWFC  A+B2, F
00982
00983         ENDM
00984
00985 ;*************************************************************************
00986
00987 ;*************************************************************************
00988 ; NAME:         NEG16
00989 ;
00990 ; DESCRIPTION:  16 bit negate
00991 ;
00992 ; ARGUMENTS:    -A => A
00993 ;
00994 ; TIMING (cycles): 5
00995 ;
00996
00997 NEG16 MACRO     A
00998
00999         COMF    A+B0, F
01000         COMF    A+B1, F
01001         CLRF    WREG, F
01002         INCF    A+B0, F
01003         ADDWFC  A+B1, F
01004
01005         ENDM
01006
01007 ;*************************************************************************
```

```
01008
01009 ;*******************************************************************************
01010 ; NAME:         AUTONO
01011 ;
01012 ; DESCRIPTION:  Sets no auto increment or decrement
01013 ;
01014 ; TIMING (cycles): 4
01015
01016 AUTONO MACRO
01017
01018         BSF     _fs0
01019         BSF     _fs1
01020         BSF     _fs2
01021         BSF     _fs3
01022
01023         ENDM
01024
01025 ;*******************************************************************************
01026
01027 ;*******************************************************************************
01028 ; NAME:         AUTOINC
01029 ;
01030 ; DESCRIPTION:  Set auto increment
01031 ;
01032 ; TIMING (cycles): 4
01033 ;
01034
01035 AUTOINC MACRO
01036
01037         BSF     _fs0
01038         BCF     _fs1
01039         BSF     _fs2
01040         BCF     _fs3
01041
01042         ENDM
01043
01044 ;*******************************************************************************
01045
01046 ;*******************************************************************************
01047 ; NAME:         AUTODEC
01048 ;
01049 ; DESCRIPTION:  Sets auto decrement
01050 ;
01051 ; TIMING (cycles): 4
01052 ;
01053
01054 AUTODEC MACRO
```

```
01055
01056         BCF     _fs0
01057         BCF     _fs1
01058         BCF     _fs2
01059         BCF     _fs3
01060
01061         ENDM
01062
01063 ;*****************************************************************************
01064
01065 ;*****************************************************************************
01066 ; NAME:         TFSZ32
01067 ;
01068 ; DESCRIPTION:  32 bit test and skip if zero
01069 ;
01070 ; TIMING (cycles): 6
01071 ;
01072
01073 TFSZ32 MACRO    A
01074
01075         MOVFP   A+B0,WREG
01076         IORWF   A+B1,W
01077         IORWF   A+B2,W
01078         IORWF   A+B3,W
01079         TSTFSZ  WREG
01080         ENDM
01081
01082 ;*****************************************************************************
01083
01084 ;*****************************************************************************
01085 ; NAME:         TFSZ24
01086 ;
01087 ; DESCRIPTION:  24 bit test and skip if zero
01088 ;
01089 ; TIMING (cycles): 5
01090
01091 TFSZ24 MACRO    A
01092
01093         MOVFP   A+B0,WREG
01094         IORWF   A+B1,W
01095         IORWF   A+B2,W
01096         TSTFSZ  WREG
01097         ENDM
01098
01099 ;*****************************************************************************
01100
01101 ;*****************************************************************************
```

```
                    01102 ; NAME:         TFSZ16
                    01103 ;
                    01104 ; DESCRIPTION:  16 bit test and skip if zero
                    01105 ;
                    01106 ; TIMING (cycles): 4
                    01107 ;
                    01108
                    01109 TFSZ16 MACRO    A
                    01110
                    01111         MOVFP   A+B0,WREG
                    01112         IORWF   A+B1,W
                    01113         TSTFSZ  WREG
                    01114         ENDM
                    01115
                    01116 ;*****************************************************************************
                    00016
                    00017 ;*****************************************************************************
                    00018 ;       global  variables
                    00019 ;
                    00020         CBLOCK  0x18
00000018            00021         DPX,DPX1,DPX2,DPX3              ; arithmetic accumulator
0000001C            00022         AARG,AARG1,BARG,BARG1           ; multiply arguments
                    00023         ENDC
                    00024
                    00025         CBLOCK  0x18
00000018            00026         TMP,TMP1,TMP2,TMP3              ; temporary variables
0000001C            00027         MOVTMP,MOVTMP1,MOVTMP2,MOVTMP3  ; move temporary storage
                    00028         ENDC
                    00029
                    00030         CBLOCK  0x20
00000020            00031         VL,VL1,VL2                      ; velocity limit
00000023            00032         AL,AL1,AL2                      ; acceleration limit
                    00033
00000026            00034         KP,KP1                          ; proportional gain
00000028            00035         KV,KV1                          ; velocity gain
0000002A            00036         KI,KI1                          ; integral gain
0000002C            00037         IM                              ; integrator mode
0000002D            00038         FV,FV1                          ; velocity feedforward
0000002F            00039         FA,FA1                          ; acceleration feedforward
                    00040
00000031            00041         VALBUF,VALBUF1,VALBUF2          ; iovalue buffer
00000034            00042         CVALBUF,CVALBUF1,CVALBUF2       ; iovalue buffer
00000037            00043         DVALBUF,DVALBUF1,DVALBUF2       ; iovalue buffer
0000003A            00044         ISRBSR,ISRWREG                  ; isr save storage
0000003C            00045         CMDCHAR,CMDTEMP,CMDPTRH,CMDPTRL ; command interface variables
00000040            00046         PARTEMP,PARLEN,PARPTR           ; parameter variables
                    00047
```

```
00000043          00048         CPOSITION,CPOSITION1,CPOSITION2         ; shutter commanded position
00000046          00049         CVELOCITY,CVELOCITY1,CVELOCITY2         ; shutter commanded velocity
00000049          00050         CMPOSITION,CMPOSITION1,CMPOSITION2      ; shutter measured position
0000004C          00051         CMVELOCITY,CMVELOCITY1,CMVELOCITY2      ; shutter measured velocity
                  00052
0000004F          00053         STRVALH,STRVALL                 ; string io variables
00000051          00054         HEXVAL,HEXTMP,HEXSTAT           ; hex io variables
                  00055
00000054          00056         OPOSITION,OPOSITION1
00000056          00057         OPOSITION2,OPOSITION3           ; original commanded position
00000058          00058         POSITION,POSITION1,POSITION2    ; commanded position
0000005B          00059         VELOCITY,VELOCITY1,VELOCITY2    ; commanded velocity
                  00060
0000005E          00061         NMOVVAL,NMOVVAL1,NMOVVAL2,NMOVVAL3      ; move value
00000062          00062         MOVVAL,MOVVAL1,MOVVAL2,MOVVAL3          ; move value
00000066          00063         HMOVVAL,HMOVVAL1,HMOVVAL2,HMOVVAL3      ; half move value
0000006A          00064         MOVTIME,MOVTIME1                        ; move time in sample counts
                  00065
0000006C          00066         MOVSIGN                         ; 0x00 for positive, 0x80 for negative
0000006D          00067         T1,T11                          ; time to maximum velocity
0000006F          00068         T2,T21                          ; time for half the move
00000071          00069         TAU,TAU1                        ; total move time
00000073          00070         NMODE                           ; next move modetype
00000074          00071         MODE                            ; move modetype
                  00072
00000075          00073         MPOSITION,MPOSITION1,MPOSITION2 ; measured position
00000078          00074         MVELOCITY,MVELOCITY1
0000007A          00075         MVELOCITY2,MVELOCITY3           ; measured velocity
0000007C          00076         POSERROR,POSERROR1,POSERROR2    ; position error
0000007F          00077         VELERROR,VELERROR1,VELERROR2    ; velocity error
                  00078
00000082          00079         SIGN                            ; multiply sign
                  00080
00000083          00081         Y,Y1,Y2,Y3                      ; Y(k) before pwm conversion
00000087          00082         YPWM,YPWM1,YPWM2,YPWM3          ; pwm input
0000008B          00083         YPWMIN,YPWMIN1,YPWMAX,YPWMAX1   ; pwm input limits
                  00084
0000008F          00085         U0,U01,U1,U11,U2,U21            ; saturated error at successive times
                  00086
00000095          00087         SERVOFLAG                       ; servoflag = 0 => no servo
00000096          00088         MODETYPE                        ; mode flag(0=position,1=velocity,-1=torque)
00000097          00089         EXTSTAT                         ; external status register
00000098          00090         MOVSTAT                         ; move status register
00000099          00091         MOVFLAG                         ; move flag
0000009A          00092         SATFLAG                         ; saturation flag (1=pos,-1=neg)
0000009B          00093         INTEGRAL,INTEGRAL1              ; integrator
                  00094
```

```
0000009D          00095         C0,C01,C1,C11,C2,C21            ; difference equation coefficients
                  00096
000000A3          00097         DECVAL,DECSTAT,DECTMP,DECSIGN   ; decimal io variables
                  00098
000000A7          00099         A,A1,A2,A3                              ; commanded acceleration = +-AL, 0
000000AB          00100         V,V1,V2,V3                              ; commanded velocity = +-VL, 0
000000AF          00101         MOVPBUF,MOVPBUF1,MOVPBUF2,MOVPBUF3      ; commanded position buffer
000000B3          00102         MOVVBUF,MOVVBUF1,MOVVBUF2,MOVVBUF3      ; commanded velocity buffer
                  00103
000000B7          00104         UPCOUNT,UPCOUNT1                ; running up counter
000000B9          00105         DOWNCOUNT,DOWNCOUNT1            ; running down counter
                  00106
000000BB          00107         MOVDEL,MOVDEL1,MOVDEL2,MOVDEL3  ; move discretization delta
000000BF          00108         PH2FLAT,PH2FLAT1                ; phase 2 flat iteration counter
000000C1          00109         INDEXPOS,INDEXPOS1,INDEXPOS2    ; position at last index pulse
                  00110
000000C4          00111         SHIFTNUM                        ; # of bit shifts from middle 16
                  00112
                  00113 ;*************************************************************************
                  00114 ;       For PICMASTER Debug/servo tuning Purposes Only
                  00115
000000C5          00116         CAPFLAG                         ; trace capture flag
000000C6          00117         CAPCOUNT,CAPCOUNT1              ; PICMASTER trace capture counter
000000C8          00118         CAPTMP,CAPTMP1                  ; trace capture temporary storage
                  00119
                  00120 ;*************************************************************************
                  00121
000000CA          00122         ZERO,ONE                        ; constants
                  00123
                  00124         ENDC
                  00125
                  00126 ;*****************************************************************************
                  00127
                  00128
0000              00129         ORG     0x0                     ; reset vector
0000 C021         00130         GOTO    Startup                 ; startup vector
                  00131
0020              00132         ORG     0x20
0020 C070         00133         GOTO    InterruptPoll           ; interrupt vector
                  00134
                  00135
                  00136 ;*****************************************************************************
                  00137 ; NAME:         Startup
                  00138 ;
                  00139 ; DESCRIPTION:  This routine is called on the hardware reset or when the
                  00140 ;               program wishes to restore initial conditions. Initiali-
                  00141 ;               zation of run-time constants takes place here.
```

```
                    00142 ;
                    00143 ; RETURNS:      restart to safe and initial state
                    00144 ;
                    00145 ; STACK UTILIZATION: none
                    00146 ; TIMING (in cycles): X
                    00147
0021                00148 Startup
                    00149
0021 8406           00150         BSF     _glintd                 ; disable all interrupts
                    00151         AUTONO                          ; no auto increment or decrement
                        M
0022 8404               M         BSF     _fs0
0023 8504               M         BSF     _fs1
0024 8604               M         BSF     _fs2
0025 8704               M         BSF     _fs3
                        M
0026 B018           00152         MOVLW   0x18                    ; clear all memory locations [18,FF]
0027 4A01           00153         MOVPF   WREG,FSR0
                    00154
0028                00155 memloop
0028 2900           00156         CLRF    INDF0, F
0029 1F01           00157         INCFSZ  FSR0, F
002A C028           00158         GOTO    memloop
                    00159
002B 15CB           00160         INCF    ONE, F
                    00161
002C B803           00162         MOVLB   BANK3                   ; BANK3 initialization
002D B03F           00163         MOVLW   TCON2_INIT
002E 770A           00164         MOVFP   WREG,TCON2
                    00165
002F B07F           00166         MOVLW   PW1DCH_INIT             ; set duty cycle to midpoint
0030 720A           00167         MOVFP   WREG,PW1DCH
0031 730A           00168         MOVFP   WREG,PW2DCH
                    00169
0032 B0C0           00170         MOVLW   PW1DCL_INIT
0033 700A           00171         MOVFP   WREG,PW1DCL
0034 710A           00172         MOVFP   WREG,PW2DCL
                    00173
0035 B006           00174         MOVLW   TCON1_INIT              ; set organization of timers
0036 760A           00175         MOVFP   WREG,TCON1
                    00176
0037 B802           00177         MOVLB   BANK2                   ; BANK2 initialization
                    00178
0038 B0FF           00179         MOVLW   PR1_INIT
0039 740A           00180         MOVFP   WREG,PR1                ; initialize timer1 period
                    00181
003A B00F           00182         MOVLW   PR2_INIT
```

```
003B 750A          00183        MOVFP   WREG,PR2              ; initialize timer2 period
                   00184
                   00185
003C B800          00186        MOVLB   BANK0                 ; BANK0 initialization
                   00187
003D B080          00188        MOVLW   T0STA_INIT
003E 650A          00189        MOVFP   WREG,RTCSTA           ; sets T0 for external input
                   00190
003F B090          00191        MOVLW   RCSTA_INIT
0040 730A          00192        MOVFP   WREG,RCSTA            ; set receive status
                   00193
0041 B020          00194        MOVLW   TXSTA_INIT            ; set transmit status
0042 750A          00195        MOVFP   WREG,TXSTA
                   00196
0043 B019          00197        MOVLW   SPBRG_INIT            ; set baud rate
0044 770A          00198        MOVFP   WREG,SPBRG
                   00199
0045 B0F3          00200        MOVLW   DDRB_INIT
0046 710A          00201        MOVFP   WREG,DDRB             ; set port B for whatever
                   00202
0047 B801          00203        MOVLB   BANK1                 ; BANK1 initialization
                   00204
0048 B008          00205        MOVLW   0x08                  ; initialize some parameters
0049 4A27          00206        MOVPF   WREG,KP+B1            ; proportional gain
                   00207
004A B050          00208        MOVLW   0x50
004B 4A29          00209        MOVPF   WREG,KV+B1            ; derivative gain
                   00210
004C B004          00211        MOVLW   0x04
004D 4A2B          00212        MOVPF   WREG,KI+B1            ; integral gain
                   00213
004E B001          00214        MOVLW   0x01
004F 4A24          00215        MOVPF   WREG,AL+B1            ; acceleration limit
                   00216
0050 B0F0          00217        MOVLW   0xF0
0051 4A21          00218        MOVPF   WREG,VL+B1            ; velocity limit
                   00219
0052 82C4          00220        BSF     SHIFTNUM,2            ; set shift number
                   00221
0053 5288          00222        MOVPF   PW1DCH,YPWM+B1
                   00223
0054 B080          00224        MOVLW   PWMAXL                ; initialize pwm limits
0055 4A8D          00225        MOVPF   WREG,YPWMAX+B0
0056 B0FE          00226        MOVLW   PWMAXH
0057 4A8E          00227        MOVPF   WREG,YPWMAX+B1
0058 B040          00228        MOVLW   PWMINL
0059 4A8B          00229        MOVPF   WREG,YPWMIN+B0
```

```
005A B001           00230         MOVLW   PWMINH
005B 4A8C           00231         MOVPF   WREG,YPWMIN+B1
                    00232
005C 2916           00233         CLRF    PIR, F                  ; clear flags, set indiviual interrupts
005D 2907           00234         CLRF    INTSTA, F
005E 8517           00235         BSF     _tm2ie
005F 8307           00236         BSF     _peie
                    00237
0060 8C06           00238         BCF     _glintd                 ; enable interrupts
                    00239
0061 B802           00240         MOVLB   BANK2
                    00241
0062                00242 zeroctrs
0062 290B           00243         CLRF    TMR0L, F                ; clear up counter
0063 290C           00244         CLRF    TMR0H, F
                    00245
0064 2912           00246         CLRF    TMR3L, F                ; clear down counter
0065 2913           00247         CLRF    TMR3H, F
                    00248
0066 B0FF           00249         MOVLW   0xFF
0067 170A           00250 delay   DECFSZ  WREG, F
0068 C067           00251         GOTO    delay
                    00252
0069 6A0B           00253         MOVFP   TMR0L,WREG
006A 080C           00254         IORWF   TMR0H,W
006B 0812           00255         IORWF   TMR3L,W
006C 0813           00256         IORWF   TMR3H,W
006D 330A           00257         TSTFSZ  WREG
006E C062           00258         GOTO    zeroctrs                ; motor still moving
                    00259
006F C086           00260         GOTO    PollingLoop
                    00261
                    00262 ;*****************************************************************************
                    00263
                    00264 ;*****************************************************************************
                    00265 ; NAME:         InterruptPoll
                    00266
                    00267
0070                00268 InterruptPoll
                    00269
0070 4F3A           00270         MOVPF   BSR,ISRBSR              ; save BSR,WREG
0071 4A3B           00271         MOVPF   WREG,ISRWREG
                    00272
0072 B801           00273         MOVLB   BANK1
                    00274
0073 E5EC           00275         CALL    doMPosMVel              ; calculate measured position and
                    00276                                         ; velocity
```

```
                        00277
0074 E61D               00278         CALL    doExtstat               ; evaluate external status
                        00279
0075 2298               00280         RLNCF   MOVSTAT,W               ; if MOVFLAG=0 and MOVSTAT,BIT7=1
0076 B501               00281         ANDLW   0x01                    ; then do premove. This is only
0077 0499               00282         SUBWF   MOVFLAG,W               ; executed once at the beginning of
0078 9F0A               00283         BTFSC   WREG,MSB                ; each move
0079 E37E               00284         CALL    doPreMove
                        00285
007A 9E98               00286         BTFSC   MOVSTAT,BIT6            ; is motion continuing?
007B E44F               00287         CALL    doMove                  ; if so, do move
                        00288
007C E291               00289         CALL    doError                 ; calculate position and velocity
                        00290                                         ; error
007D 3395               00291         TSTFSZ  SERVOFLAG               ; test servoflag, if 0 then no servo
007E E2D8               00292         CALL    doServo                 ; do servo
                        00293
007F 33C5               00294         TSTFSZ  CAPFLAG
0080 E742               00295         CALL    doCaptureRegs           ; for PIC-MASTER Trace Capture, demo purposes
                        00296
0081 B801               00297         MOVLB   BANK1
                        00298
0082 2916               00299         CLRF    PIR, F                  ; clear all interrupt request flags
                        00300
0083 6F3A               00301         MOVFP   ISRBSR,BSR              ; restore BSR,WREG
0084 6A3B               00302         MOVFP   ISRWREG,WREG
                        00303
0085 0005               00304         RETFIE
                        00305
                        00306 ;*****************************************************************************
                        00307
                        00308 ;*****************************************************************************
                        00309 ; NAME:                 PollingLoop
                        00310 ;
                        00311 ; DESCRIPTION:          The actual polling loop called after the board's
                        00312 ;                       initialization
                        00313 ;
                        00314 ; ENTRY CONDITIONS:     System globals and hardware initialized and the
                        00315 ;                       interrupt processes started.
                        00316 ;
                        00317
0086                    00318 PollingLoop
                        00319
0086 E08D               00320         CALL    IdleFunction
0087 E1AC               00321         CALL    GetChk
0088 31CB               00322         CPFSEQ  ONE                     ; GetChk, is receive buffer full?
0089 C086               00323         GOTO    PollingLoop
```

```
                    00324
008A E1A1           00325         CALL    GetChar                 ; if so, get character
008B 4A3C           00326         MOVPF   WREG,CMDCHAR            ; put in CMDCHAR
008C C08F           00327         GOTO    DoCommand
                    00328
                    00329 ;*****************************************************************************
                    00330
                    00331 ;*****************************************************************************
                    00332 ; NAME:         IdleFunction
                    00333
                    00334 ; DESCRIPTION:  This routine will perform work while doing waits in serial
                    00335 ;               I/O functions.
                    00336 ;
                    00337
008D                00338 IdleFunction
                    00339
008D 0004           00340         CLRWDT
008E 0002           00341         RETURN
                    00342
                    00343 ;*****************************************************************************
                    00344
                    00345 ;*****************************************************************************
                    00346 ; NAME:         DoCommand
                    00347 ;
                    00348 ; DESCRIPTION:  Search command table for command and execute it.
                    00349 ;
                    00350
008F                00351 DoCommand
                    00352
008F B066           00353         MOVLW   LOW     CMD_TABLE         ; CMD_TABLE LSB
0090 4A0D           00354         MOVPF   WREG,TBLPTRL
0091 B007           00355         MOVLW   HIGH    CMD_TABLE         ; CMD_TABLE MSB
0092 4A0E           00356         MOVPF   WREG,TBLPTRH
                    00357
0093 AB3D           00358         TABLRD  1,1,CMDTEMP
0094                00359 tryNextCmd
0094 A93D           00360         TABLRD  0,1,CMDTEMP             ; read entry from table
0095 A23E           00361         TLRD    1,CMDPTRH
0096 A93F           00362         TABLRD  0,1,CMDPTRL
                    00363
0097 6A3D           00364         MOVFP   CMDTEMP,WREG
0098 30CA           00365         CPFSLT  ZERO
                    00366
0099 C0A4           00367         GOTO    noCommand               ; error if end of table
                    00368
009A 313C           00369         CPFSEQ  CMDCHAR
009B C094           00370         GOTO    tryNextCmd
```

```
                    00371
009C E1A4           00372         CALL    PutChar                 ; echo command
                    00373
009D 633E           00374         MOVFP   CMDPTRH,PCLATH          ; indirect jump to command routine
009E 623F           00375         MOVFP   CMDPTRL,PCL
009F 0000           00376         NOP
00A0                00377 cmdFinish
00A0 E1A4           00378         CALL    PutChar                 ; send response character from
                    00379                                         ; command routine followed by CR
00A1 B00D           00380         MOVLW   CR
00A2 E1A4           00381         CALL    PutChar
                    00382
00A3 C086           00383         GOTO    PollingLoop
                    00384
00A4                00385 noCommand
00A4 B03F           00386         MOVLW   CMD_BAD                 ; send error character
00A5 C0A0           00387         GOTO    cmdFinish
                    00388
                    00389 ;*****************************************************************************
                    00390
                    00391 ;*****************************************************************************
                    00392 ; NAME:         do_null
                    00393 ;
                    00394 ; DESCRIPTION:  The do nothing command used to determine if the chip is
                    00395 ;               working. Initiated by a carriage return.
                    00396
00A6                00397 do_null
00A6 B021           00398         MOVLW   CMD_OK
00A7 C0A0           00399         GOTO    cmdFinish
                    00400
                    00401 ;*****************************************************************************
                    00402
                    00403 ;*****************************************************************************
                    00404 ; NAME:         do_move
                    00405
                    00406 ; DESCRIPTION:  Commands the axis to move to a new position or velocity.
                    00407 ;               Position data is relative, and in encoder counts. Velocity
                    00408 ;               data is absolute, and in encoder counts/sample time multi-
                    00409 ;               plied by 256. All moves are performed by the controller such
                    00410 ;               that velocity and acceleration limits set into parameter
                    00411 ;               memory will not be violated. All move commands are kept in a
                    00412 ;               one deep FIFO buffer. The command in the buffer is executed
                    00413 ;               as soon as the currently executed command is complete.
                    00414 ;
                    00415 ;
                    00416 ; ARGUMENTS:    M [800000,7FFFFF]
                    00417 ;
```

```
00A8                00418 do_move
                    00419
                    00420         #if     DECIO
                    00421
00A8 E217           00422         CALL    GetDecVal
                    00423
                    00424         #else
                    00425
                    00426         CALL    GetVal
                    00427
                    00428         #endif
                    00429
00A9 9F98           00430         BTFSC   MOVSTAT,BIT7            ; test if buffer available
00AA C0B4           00431         GOTO    bufoverflow
                    00432
                    00433         MOV24   VALBUF,NMOVVAL          ; if so, accept value into NMOVVAL and
                        M
00AB 6A31               M         MOVFP   VALBUF+B0,WREG          ; get byte of a into w
00AC 4A5E               M         MOVPF   WREG,NMOVVAL+B0         ; move to b(B0)
00AD 6A32               M         MOVFP   VALBUF+B1,WREG          ; get byte of a into w
00AE 4A5F               M         MOVPF   WREG,NMOVVAL+B1         ; move to b(B1)
00AF 6A33               M         MOVFP   VALBUF+B2,WREG          ; get byte of a into w
00B0 4A60               M         MOVPF   WREG,NMOVVAL+B2         ; move to b(B2)
                        M
00B1 8798           00434         BSF     MOVSTAT,BIT7            ; set buffer full flag
                    00435
00B2 B021           00436         MOVLW   CMD_OK
00B3 C0A0           00437         GOTO    cmdFinish
                    00438
00B4                00439 bufoverflow
00B4 B03F           00440         MOVLW   CMD_BAD                 ; else, return error
00B5 C0A0           00441         GOTO    cmdFinish
                    00442
                    00443 ;*************************************************************************
                    00444
                    00445 ;*************************************************************************
                    00446 ; NAME:         do_mode
                    00447 ;
                    00448 ; DESCRIPTION:  An argument of “P” will cause all subsequent move commands
                    00449 ;               to be incremental position moves. A “V” argument will cause
                    00450 ;               all subsequent moves to be absolute velocity moves.
                    00451 ;
                    00452 ; ARGUMENTS:    O [P,V]
                    00453 ;
                    00454
00B6                00455 do_mode
                    00456
```

```
00B6 E08D          00457         CALL    IdleFunction            ; get single character loop
00B7 E1AC          00458         CALL    GetChk
00B8 31CB          00459         CPFSEQ  ONE
00B9 C0B6          00460         GOTO    do_mode
00BA E1A1          00461         CALL    GetChar
00BB 4A50          00462         MOVPF   WREG,STRVALL
00BC 2996          00463         CLRF    MODETYPE, F             ; MODETYPE=0 for position moves
00BD               00464 testP
00BD B050          00465         MOVLW   'P'                     ; position moves for type P
00BE 3150          00466         CPFSEQ  STRVALL
00BF C0C1          00467         GOTO    testV
00C0 C0CE          00468         GOTO    modeok
00C1               00469 testV
00C1 B056          00470         MOVLW   'V'                     ; velocity moves for type V
00C2 3150          00471         CPFSEQ  STRVALL
00C3 C0C6          00472         GOTO    testT
00C4 1596          00473         INCF    MODETYPE, F             ; MODETYPE=1 for velocity moves
00C5 C0CE          00474         GOTO    modeok
00C6               00475 testT
00C6 B054          00476         MOVLW   'T'                     ; TORQUE Moves for type 'T'
00C7 3150          00477         CPFSEQ  STRVALL
00C8 C0CC          00478         GOTO    modeerror
00C9 2B96          00479         SETF    MODETYPE, F             ; MODETYPE=-1 for torque moves
00CA 2995          00480         CLRF    SERVOFLAG, F            ; disable servo
00CB C0CE          00481         GOTO    modeok
00CC               00482 modeerror
00CC B03F          00483         MOVLW   CMD_BAD                 ; mode error
00CD C0A0          00484         GOTO    cmdFinish
00CE               00485 modeok
00CE 6A50          00486         MOVFP   STRVALL,WREG            ; echo type character
00CF E1A4          00487         CALL    PutChar
                   00488
00D0 B021          00489         MOVLW   CMD_OK
00D1 C0A0          00490         GOTO    cmdFinish
                   00491
                   00492 ;*****************************************************************************
                   00493
                   00494 ;*****************************************************************************
                   00495 ; NAME:         do_setparameter
                   00496 ;
                   00497 ; DESCRIPTION:  Sets controller parameters to the value given.
                   00498 ;
                   00499 ;       Parameter               #               Range
                   00500 ;
                   00501 ;       VL=velocity limit       0               [0,7FFFFF]
                   00502 ;       AL=acceleration limit   1               [0,7FFFFF]
                   00503 ;
```

```
                        00504 ;       KP=proportional gain    2             [8000,7FFF]
                        00505 ;       KP=velocity gain        3             [8000,7FFF]
                        00506 ;       KP=integral gain        4             [8000,7FFF]
                        00507 ;
                        00508 ;       IM=integrator mode      5             [0,3]
                        00509 ;
                        00510 ;       FV=velocity FF          6             [8000,7FFF] : Not Implemented
                        00511 ;       FA=acceleration FF      7             [8000,7FFF] : Not Implemented
                        00512 ;
                        00513 ;
                        00514 ; ARGUMENTS:    S [0,FF] [800000,7FFFFF]
                        00515 ;
                        00516
00D2                    00517 do_setparameter
                        00518
00D2 E253               00519         CALL    GetPar                        ; get parameter number
                        00520
00D3 B008               00521         MOVLW   NUMPAR                        ; check if in range [0,NUMPAR]
00D4 3031               00522         CPFSLT  VALBUF+B0
00D5 C0F7               00523         GOTO    Serror
                        00524
00D6 B089               00525         MOVLW   LOW    PAR_TABLE              ; PAR_TABLE LSB
00D7 4A0D               00526         MOVPF   WREG,TBLPTRL
00D8 B007               00527         MOVLW   HIGH   PAR_TABLE              ; PAR_TABLE MSB
00D9 4A0E               00528         MOVPF   WREG,TBLPTRH
                        00529
00DA AB40               00530         TABLRD  1,1,PARTEMP
                        00531
00DB                    00532 setNextPar
00DB A240               00533         TLRD    1,PARTEMP                     ; read entry from table
00DC A941               00534         TABLRD  0,1,PARLEN
00DD A942               00535         TABLRD  0,1,PARPTR
                        00536
00DE B008               00537         MOVLW   NUMPAR                        ; error if end of table
00DF 3040               00538         CPFSLT  PARTEMP
00E0 C0F7               00539         GOTO    Serror
                        00540
00E1 6A40               00541         MOVFP   PARTEMP,WREG
00E2 3131               00542         CPFSEQ  VALBUF+B0
00E3 C0DB               00543         GOTO    setNextPar
                        00544
00E4 6A42               00545         MOVFP   PARPTR,WREG                   ; pointer to parameter in FSR1
00E5 690A               00546         MOVFP   WREG,FSR1
                        00547
                        00548         #if     DECIO                         ; get new value in VALBUF
                        00549
00E6 E217               00550         CALL    GetDecVal
```

```
                    00551
                    00552         #else
                    00553
                    00554         CALL    GetVal
                    00555
                    00556         #endif
                    00557
00E7 B031           00558         MOVLW   VALBUF                  ; pointer to VALBUF in FSR0
00E8 610A           00559         MOVFP   WREG,FSR0
                    00560         AUTOINC                         ; set autoincrement
                        M
00E9 8404               M         BSF     _fs0
00EA 8D04               M         BCF     _fs1
00EB 8604               M         BSF     _fs2
00EC 8F04               M         BCF     _fs3
                        M
00ED                00561 setGetMore
00ED 6800           00562         MOVFP   INDF0,INDF1             ; move new value to parameter
00EE 0741           00563         DECF    PARLEN, F
00EF 3341           00564         TSTFSZ  PARLEN
00F0 C0ED           00565         GOTO    setGetMore
                    00566
                    00567         AUTONO                          ; no autoincrement
                        M
00F1 8404               M         BSF     _fs0
00F2 8504               M         BSF     _fs1
00F3 8604               M         BSF     _fs2
00F4 8704               M         BSF     _fs3
                        M
                    00568
                    00569
00F5 B021           00570         MOVLW   CMD_OK
00F6 C0A0           00571         GOTO    cmdFinish
                    00572
00F7                00573 Serror
00F7 B03F           00574         MOVLW   CMD_BAD
00F8 C0A0           00575         GOTO    cmdFinish
                    00576
                    00577 ;*****************************************************************************
                    00578
                    00579 ;*****************************************************************************
                    00580 ; NAME:         do_readparameter
                    00581 ;
                    00582 ; DESCRIPTION:  Returns the present value of a parameter.
                    00583 ;
                    00584 ; ARGUMENTS:    R [0,FF]
                    00585 ;
```

```
                    00586 ; RETURNS:      The present value of the requested parameter is returned.
                    00587
00F9                00588 do_readparameter
                    00589
00F9 E253           00590         CALL    GetPar                  ; get parameter number
                    00591
00FA B008           00592         MOVLW   NUMPAR                  ; check if in range [0,NUMPAR]
00FB 3031           00593         CPFSLT  VALBUF+B0
00FC C121           00594         GOTO    Rerror
                    00595
00FD B089           00596         MOVLW   LOW     PAR_TABLE       ; PAR_TABLE LSB
00FE 4A0D           00597         MOVPF   WREG,TBLPTRL
00FF B007           00598         MOVLW   HIGH    PAR_TABLE       ; PAR_TABLE MSB
0100 4A0E           00599         MOVPF   WREG,TBLPTRH
                    00600
0101 AB40           00601         TABLRD  1,1,PARTEMP
                    00602
0102                00603 readNextPar
0102 A240           00604         TLRD    1,PARTEMP               ; read entry from table
0103 A941           00605         TABLRD  0,1,PARLEN
0104 A942           00606         TABLRD  0,1,PARPTR
                    00607
0105 B008           00608         MOVLW   NUMPAR                  ; error if end of table
0106 3040           00609         CPFSLT  PARTEMP
0107 C121           00610         GOTO    Rerror
                    00611
0108 6A40           00612         MOVFP   PARTEMP,WREG
0109 3131           00613         CPFSEQ  VALBUF+B0
010A C102           00614         GOTO    readNextPar
                    00615
010B 6A42           00616         MOVFP   PARPTR,WREG             ; pointer to parameter in FSR1
010C 690A           00617         MOVFP   WREG,FSR1
                    00618
010D B031           00619         MOVLW   VALBUF                  ; pointer to VALBUF in FSR1
010E 610A           00620         MOVFP   WREG,FSR0
                    00621         AUTOINC                         ; set autoincrement
                        M
010F 8404               M         BSF     _fs0
0110 8D04               M         BCF     _fs1
0111 8604               M         BSF     _fs2
0112 8F04               M         BCF     _fs3
                        M
                    00622
                    00623         CLR24   VALBUF                  ; clear old VALBUF
0113 2931               M         CLRF    VALBUF+B0, F
0114 2932               M         CLRF    VALBUF+B1, F
0115 2933               M         CLRF    VALBUF+B2, F
```

```
                        M
0116                00624 readGetMore
0116 6008           00625         MOVFP   INDF1,INDF0             ; read parameter into VALBUF
0117 0741           00626         DECF    PARLEN, F
0118 3341           00627         TSTFSZ  PARLEN
0119 C116           00628         GOTO    readGetMore
                    00629
                    00630         AUTONO                          ; no autoincrement
                        M
011A 8404               M         BSF     _fs0
011B 8504               M         BSF     _fs1
011C 8604               M         BSF     _fs2
011D 8704               M         BSF     _fs3
                        M
                    00631
                    00632
                    00633         #if     DECIO                   ; send parameter value
                    00634
011E E260           00635         CALL    PutDecVal
                    00636
                    00637         #else
                    00638
                    00639         CALL    PutVal
                    00640
                    00641         #endif
                    00642
011F B021           00643         MOVLW   CMD_OK
0120 C0A0           00644         GOTO    cmdFinish
                    00645
0121                00646 Rerror
0121 B03F           00647         MOVLW   CMD_BAD
0122 C0A0           00648         GOTO    cmdFinish
                    00649
                    00650 ;*************************************************************************
                    00651
                    00652 ;*************************************************************************
                    00653 ; NAME: do_shutter
                    00654 ;
                    00655 ; DESCRIPTION:  Returns the time (in sample time counts [0,FFFF]) since the
                    00656 ;               start of the present move and captures the commanded and
                    00657 ;               measured values of position and velocity at the time of the
                    00658 ;               command.
                    00659 ;
                    00660 ;
                    00661 ; ARGUMENTS:    C
                    00662 ;
                    00663 ; RETURNS:      The time since the start of the present move is returned.
```

```
                    00664 ;
                    00665
0123                00666 do_shutter
                    00667
0123 8406           00668         BSF     _glintd                 ; disable all interrupts
                    00669
                    00670         MOV24   POSITION,CPOSITION      ; capture commanded position
                        M
0124 6A58               M         MOVFP   POSITION+B0,WREG        ; get byte of a into w
0125 4A43               M         MOVPF   WREG,CPOSITION+B0       ; move to b(B0)
0126 6A59               M         MOVFP   POSITION+B1,WREG        ; get byte of a into w
0127 4A44               M         MOVPF   WREG,CPOSITION+B1       ; move to b(B1)
0128 6A5A               M         MOVFP   POSITION+B2,WREG        ; get byte of a into w
0129 4A45               M         MOVPF   WREG,CPOSITION+B2       ; move to b(B2)
                        M
                    00671         MOV24   VELOCITY,CVELOCITY      ; capture commanded velocity
                        M
012A 6A5B               M         MOVFP   VELOCITY+B0,WREG        ; get byte of a into w
012B 4A46               M         MOVPF   WREG,CVELOCITY+B0       ; move to b(B0)
012C 6A5C               M         MOVFP   VELOCITY+B1,WREG        ; get byte of a into w
012D 4A47               M         MOVPF   WREG,CVELOCITY+B1       ; move to b(B1)
012E 6A5D               M         MOVFP   VELOCITY+B2,WREG        ; get byte of a into w
012F 4A48               M         MOVPF   WREG,CVELOCITY+B2       ; move to b(B2)
                        M
                    00672         MOV24   MPOSITION,CMPOSITION    ; capture measured position
                        M
0130 6A75               M         MOVFP   MPOSITION+B0,WREG       ; get byte of a into w
0131 4A49               M         MOVPF   WREG,CMPOSITION+B0      ; move to b(B0)
0132 6A76               M         MOVFP   MPOSITION+B1,WREG       ; get byte of a into w
0133 4A4A               M         MOVPF   WREG,CMPOSITION+B1      ; move to b(B1)
0134 6A77               M         MOVFP   MPOSITION+B2,WREG       ; get byte of a into w
0135 4A4B               M         MOVPF   WREG,CMPOSITION+B2      ; move to b(B2)
                        M
                    00673         MOV24   MVELOCITY,CMVELOCITY    ; capture measured velocity
                        M
0136 6A78               M         MOVFP   MVELOCITY+B0,WREG       ; get byte of a into w
0137 4A4C               M         MOVPF   WREG,CMVELOCITY+B0      ; move to b(B0)
0138 6A79               M         MOVFP   MVELOCITY+B1,WREG       ; get byte of a into w
0139 4A4D               M         MOVPF   WREG,CMVELOCITY+B1      ; move to b(B1)
013A 6A7A               M         MOVFP   MVELOCITY+B2,WREG       ; get byte of a into w
013B 4A4E               M         MOVPF   WREG,CMVELOCITY+B2      ; move to b(B2)
                        M
                    00674
013C 2933           00675         CLRF    VALBUF+B2, F
                    00676         MOV16   MOVTIME,VALBUF          ; capture move time, move to VALBUF
                        M
013D 6A6A               M         MOVFP   MOVTIME+B0,WREG         ; get byte of a into w
```

```
013E 4A31               M         MOVPF   WREG,VALBUF+B0          ; move to b(B0)
013F 6A6B               M         MOVFP   MOVTIME+B1,WREG         ; get byte of a into w
0140 4A32               M         MOVPF   WREG,VALBUF+B1          ; move to b(B1)
                        M
                    00677
0141 8C06           00678         BCF     _glintd                 ; enable all interrupts
                    00679
                    00680         #if     DECIO
                    00681
0142 E260           00682         CALL    PutDecVal
                    00683
                    00684         #else
                    00685
                    00686         CALL    PutVal
                    00687
                    00688         #endif
                    00689
0143 B021           00690         MOVLW   CMD_OK
0144 C0A0           00691         GOTO    cmdFinish
                    00692
                    00693 ;*************************************************************************
                    00694
                    00695 ;*************************************************************************
                    00696 ; NAME: do_readcomposition
                    00697 ;
                    00698 ; DESCRIPTION:  Returns the commanded position count which was captured
                    00699 ;               during the last shutter command.
                    00700 ;
                    00701 ; ARGUMENTS:    P
                    00702 ;
                    00703 ; RETURNS:      The last captured position count is returned. [800000,7FFFFF]
                    00704 ;
                    00705
0145                00706 do_readcomposition
                    00707
                    00708         MOV24   CPOSITION,VALBUF        ; move CPOSITION to VALBUF
                        M
0145 6A43               M         MOVFP   CPOSITION+B0,WREG       ; get byte of a into w
0146 4A31               M         MOVPF   WREG,VALBUF+B0          ; move to b(B0)
0147 6A44               M         MOVFP   CPOSITION+B1,WREG       ; get byte of a into w
0148 4A32               M         MOVPF   WREG,VALBUF+B1          ; move to b(B1)
0149 6A45               M         MOVFP   CPOSITION+B2,WREG       ; get byte of a into w
014A 4A33               M         MOVPF   WREG,VALBUF+B2          ; move to b(B2)
                        M
                    00709
                    00710         #if     DECIO
                    00711
```

```
014B E260           00712         CALL    PutDecVal
                    00713
                    00714         #else
                    00715
                    00716         CALL    PutVal
                    00717
                    00718         #endif
                    00719
014C B021           00720         MOVLW   CMD_OK
014D C0A0           00721         GOTO    cmdFinish
                    00722
                    00723 ;*****************************************************************************
                    00724
                    00725 ;*****************************************************************************
                    00726 ; NAME: do_readcomvelocity
                    00727 ;
                    00728 ; DESCRIPTION:  Returns the commanded velocity multiplied by 256 which was
                    00729 ;               captured during the last shutter command.
                    00730 ;
                    00731 ; ARGUMENTS:    V
                    00732 ;
                    00733 ; RETURNS:      The last captured commanded velocity times 256 is returned.
                    00734 ;               [800000,7FFFFF]
                    00735 ;
                    00736
014E                00737 do_readcomvelocity
                    00738
                    00739         MOV24   CVELOCITY,VALBUF        ; move commanded velocity to VALBUF
                        M
014E 6A46               M         MOVFP   CVELOCITY+B0,WREG       ; get byte of a into w
014F 4A31               M         MOVPF   WREG,VALBUF+B0          ; move to b(B0)
0150 6A47               M         MOVFP   CVELOCITY+B1,WREG       ; get byte of a into w
0151 4A32               M         MOVPF   WREG,VALBUF+B1          ; move to b(B1)
0152 6A48               M         MOVFP   CVELOCITY+B2,WREG       ; get byte of a into w
0153 4A33               M         MOVPF   WREG,VALBUF+B2          ; move to b(B2)
                        M
                    00740
                    00741         #if     DECIO
                    00742
0154 E260           00743         CALL    PutDecVal
                    00744
                    00745         #else
                    00746
                    00747         CALL    PutVal
                    00748
                    00749         #endif
                    00750
```

```
0155 B021           00751         MOVLW   CMD_OK
0156 C0A0           00752         GOTO    cmdFinish
                    00753
                    00754 ;*****************************************************************************
                    00755
                    00756 ;*****************************************************************************
                    00757 ; NAME: do_readactposition
                    00758 ;
                    00759 ; DESCRIPTION:  Returns the measured position count which was captured during
                    00760 ;               the last shutter command.
                    00761 ;
                    00762 ; ARGUMENTS:    p
                    00763 ;
                    00764 ; RETURNS:      The last captured measured position count is returned.
                    00765 ;               [800000,7FFFFF]
                    00766 ;
                    00767
0157                00768 do_readactposition
                    00769
                    00770         MOV24   CMPOSITION,VALBUF       ; move measured position to VALBUF
                        M
0157 6A49               M         MOVFP   CMPOSITION+B0,WREG      ; get byte of a into w
0158 4A31               M         MOVPF   WREG,VALBUF+B0          ; move to b(B0)
0159 6A4A               M         MOVFP   CMPOSITION+B1,WREG      ; get byte of a into w
015A 4A32               M         MOVPF   WREG,VALBUF+B1          ; move to b(B1)
015B 6A4B               M         MOVFP   CMPOSITION+B2,WREG      ; get byte of a into w
015C 4A33               M         MOVPF   WREG,VALBUF+B2          ; move to b(B2)
                        M
                    00771
                    00772         #if     DECIO
                    00773
015D E260           00774         CALL    PutDecVal
                    00775
                    00776         #else
                    00777
                    00778         CALL    PutVal
                    00779
                    00780         #endif
                    00781
015E B021           00782         MOVLW   CMD_OK
015F C0A0           00783         GOTO    cmdFinish
                    00784
                    00785 ;*****************************************************************************
                    00786
                    00787 ;*****************************************************************************
                    00788 ; NAME: do_readactvelocity
                    00789 ;
```

```
                    00790 ; DESCRIPTION:  Returns the measured velocity multiplied by 256 which was
                    00791 ;               captured during the last shutter command.
                    00792 ;
                    00793 ; ARGUMENTS:    v
                    00794 ;
                    00795 ; RETURNS:      The last captured measured velocity times 256 is returned.
                    00796 ;               [800000,7FFFFF]
                    00797 ;
                    00798
0160                00799 do_readactvelocity
                    00800
                    00801         MOV24   CMVELOCITY,VALBUF       ; move measured velocity to VALBUF
                        M
0160 6A4C               M         MOVFP   CMVELOCITY+B0,WREG      ; get byte of a into w
0161 4A31               M         MOVPF   WREG,VALBUF+B0          ; move to b(B0)
0162 6A4D               M         MOVFP   CMVELOCITY+B1,WREG      ; get byte of a into w
0163 4A32               M         MOVPF   WREG,VALBUF+B1          ; move to b(B1)
0164 6A4E               M         MOVFP   CMVELOCITY+B2,WREG      ; get byte of a into w
0165 4A33               M         MOVPF   WREG,VALBUF+B2          ; move to b(B2)
                        M
                    00802
                    00803         #if     DECIO
                    00804
0166 E260           00805         CALL    PutDecVal
                    00806
                    00807         #else
                    00808
                    00809         CALL    PutVal
                    00810
                    00811         #endif
                    00812
0167 B021           00813         MOVLW   CMD_OK
0168 C0A0           00814         GOTO    cmdFinish
                    00815
                    00816 ;*****************************************************************************
                    00817
                    00818 ;*****************************************************************************
                    00819 ; NAME: do_externalstatus
                    00820 ;
                    00821 ; DESCRIPTION:  Returns a two digit hex number which defines the state of
                    00822 ;               the bits in the external status register. Issuing this
                    00823 ;               command will clear all the bits in the external status
                    00824 ;               register unless the event which set the bit is still true.
                    00825 ;
                    00826 ;
                    00827 ; ARGUMENTS:    X
                    00828 ;
```

```
                    00829 ; RETURNS:      The external status register is returned.
                    00830 ;
                    00831
0169                00832 do_externalstatus
                    00833
0169 8406           00834         BSF     _glintd
016A 6A97           00835         MOVFP   EXTSTAT,WREG
016B 2997           00836         CLRF    EXTSTAT, F
016C 8C06           00837         BCF     _glintd
016D E1B3           00838         CALL    PutHex
                    00839
016E B021           00840         MOVLW   CMD_OK
016F C0A0           00841         GOTO    cmdFinish
                    00842
                    00843 ;*****************************************************************************
                    00844
                    00845 ;*****************************************************************************
                    00846 ; NAME: do_movestatus
                    00847 ;
                    00848 ; DESCRIPTION:  Returns a two digit hex number which defines the state of
                    00849 ;               the bits in the move status register. Issuing this command
                    00850 ;               will clear all the bits in the move status register unless
                    00851 ;               the event which set the bit is still true.
                    00852 ;
                    00853 ; ARGUMENTS:    Y
                    00854 ;
                    00855 ; RETURNS:      The move status register is returned.
                    00856 ;
                    00857
0170                00858 do_movestatus
                    00859
0170 6A98           00860         MOVFP   MOVSTAT,WREG
0171 E1B3           00861         CALL    PutHex
                    00862
0172 B021           00863         MOVLW   CMD_OK
0173 C0A0           00864         GOTO    cmdFinish
                    00865
                    00866 ;*****************************************************************************
                    00867
                    00868 ;*****************************************************************************
                    00869 ; NAME:         do_readindposition
                    00870 ;
                    00871 ; DESCRIPTION:  Returns the last index position captured in position counts.
                    00872 ;
                    00873 ; ARGUMENTS:    I
                    00874 ;
                    00875 ; RETURNS:      The last captured index position is returned.
```

```
                    00876 ;
                    00877
0174                00878 do_readindposition
                    00879
                    00880         MOV24   INDEXPOS,VALBUF         ; move measured velocity to VALBUF
                        M
0174 6AC1               M         MOVFP   INDEXPOS+B0,WREG        ; get byte of a into w
0175 4A31               M         MOVPF   WREG,VALBUF+B0          ; move to b(B0)
0176 6AC2               M         MOVFP   INDEXPOS+B1,WREG        ; get byte of a into w
0177 4A32               M         MOVPF   WREG,VALBUF+B1          ; move to b(B1)
0178 6AC3               M         MOVFP   INDEXPOS+B2,WREG        ; get byte of a into w
0179 4A33               M         MOVPF   WREG,VALBUF+B2          ; move to b(B2)
                        M
                    00881
                    00882         #if     DECIO
                    00883
017A E260           00884         CALL    PutDecVal
                    00885
                    00886         #else
                    00887
                    00888         CALL    PutVal
                    00889
                    00890         #endif
                    00891
017B B021           00892         MOVLW   CMD_OK
017C C0A0           00893         GOTO    cmdFinish
                    00894
                    00895 ;*****************************************************************************
                    00896
                    00897 ;*****************************************************************************
                    00898 ; NAME:         do_setposition
                    00899 ;
                    00900 ; DESCRIPTION:  Sets the measured and commanded position to the value given.
                    00901 ;               This command should not be sent unless the move FIFO buffer is empty.
                    00902 ;
                    00903 ; ARGUMENTS:    H [800000,7FFFFF]
                    00904 ;
                    00905
017D                00906 do_setposition
                    00907
                    00908         #if     DECIO
                    00909
017D E217           00910         CALL    GetDecVal
                    00911
                    00912         #else
                    00913
                    00914         CALL    GetVal
```

```
                    00915
                    00916         #endif
                    00917
                    00918         MOV24   VALBUF,POSITION
                        M
017E 6A31               M         MOVFP   VALBUF+B0,WREG          ; get byte of a into w
017F 4A58               M         MOVPF   WREG,POSITION+B0        ; move to b(B0)
0180 6A32               M         MOVFP   VALBUF+B1,WREG          ; get byte of a into w
0181 4A59               M         MOVPF   WREG,POSITION+B1        ; move to b(B1)
0182 6A33               M         MOVFP   VALBUF+B2,WREG          ; get byte of a into w
0183 4A5A               M         MOVPF   WREG,POSITION+B2        ; move to b(B2)
                        M
                    00919         MOV24   VALBUF,MPOSITION
                        M
0184 6A31               M         MOVFP   VALBUF+B0,WREG          ; get byte of a into w
0185 4A75               M         MOVPF   WREG,MPOSITION+B0       ; move to b(B0)
0186 6A32               M         MOVFP   VALBUF+B1,WREG          ; get byte of a into w
0187 4A76               M         MOVPF   WREG,MPOSITION+B1       ; move to b(B1)
0188 6A33               M         MOVFP   VALBUF+B2,WREG          ; get byte of a into w
0189 4A77               M         MOVPF   WREG,MPOSITION+B2       ; move to b(B2)
                        M
                    00920
                    00921         CLR32   Y
018A 2983               M         CLRF    Y+B0, F
018B 2984               M         CLRF    Y+B1, F
018C 2985               M         CLRF    Y+B2, F
018D 2986               M         CLRF    Y+B3, F
                        M
                    00922
018E B021           00923         MOVLW   CMD_OK
018F C0A0           00924         GOTO    cmdFinish
                    00925
                    00926 ;*****************************************************************************
                    00927
                    00928 ;*****************************************************************************
                    00929 ; NAME:         do_reset
                    00930 ;
                    00931 ; DESCRIPTION:  Performs a software reset.
                    00932 ;
                    00933 ; ARGUMENTS:    Z
                    00934 ;
                    00935
0190                00936 do_reset
                    00937
0190 B021           00938         MOVLW   CMD_OK
0191 E1A4           00939         CALL    PutChar
0192 C021           00940         GOTO    Startup
```

```
                    00941
                    00942
                    00943 ;*****************************************************************************
                    00944 ; NAME:         do_stop
                    00945 ;
                    00946 ; DESCRIPTION:  Stops servo by clearing SERVOFLAG.
                    00947 ;
0193                00948 do_stop
                    00949
0193 2995           00950         CLRF    SERVOFLAG, F
                    00951
0194 B021           00952         MOVLW   CMD_OK
0195 C0A0           00953         GOTO    cmdFinish
                    00954
                    00955 ;*****************************************************************************
                    00956
                    00957 ;*****************************************************************************
                    00958 ; NAME:         do_capture
                    00959 ;
                    00960
0196                00961 do_capture
                    00962
                    00963         #if     DECIO
                    00964
0196 E217           00965         CALL    GetDecVal
                    00966
                    00967         #else
                    00968
                    00969         CALL    GetVal
                    00970
                    00971         #endif
                    00972
                    00973         MOV16   VALBUF,CAPCOUNT
                        M
0197 6A31               M         MOVFP   VALBUF+B0,WREG          ; get byte of a into w
0198 4AC6               M         MOVPF   WREG,CAPCOUNT+B0        ; move to b(B0)
0199 6A32               M         MOVFP   VALBUF+B1,WREG          ; get byte of a into w
019A 4AC7               M         MOVPF   WREG,CAPCOUNT+B1        ; move to b(B1)
                        M
                    00974         MOV16   VALBUF,CAPTMP
                        M
019B 6A31               M         MOVFP   VALBUF+B0,WREG          ; get byte of a into w
019C 4AC8               M         MOVPF   WREG,CAPTMP+B0          ; move to b(B0)
019D 6A32               M         MOVFP   VALBUF+B1,WREG          ; get byte of a into w
019E 4AC9               M         MOVPF   WREG,CAPTMP+B1          ; move to b(B1)
                        M
                    00975
```

```
019F B021           00976         MOVLW   CMD_OK
01A0 C0A0           00977         GOTO    cmdFinish
                    00978
                    00979 ;*****************************************************************************
                    00980 ; NAME:         GetChar
                    00981 ;
                    00982 ; DESCRIPTION:  Get character from receive buffer.
                    00983 ;
01A1                00984 GetChar
                    00985
01A1 B800           00986         MOVLB   BANK0                   ; set bank0
01A2 540A           00987         MOVPF   RCREG,WREG              ; receive character
                    00988
01A3 0002           00989         RETURN
                    00990
                    00991 ;*****************************************************************************
                    00992
                    00993 ;*****************************************************************************
                    00994 ; NAME:         PutChar
                    00995 ;
                    00996 ; DESCRIPTION:  send character out the serial port
                    00997 ;
                    00998 ; ARGUMENTS:    WREG contains byte to be transmitted
                    00999 ;
                    01000
01A4                01001 PutChar
                    01002
01A4 B801           01003         MOVLB   BANK1                   ; set bank1
01A5                01004 bufwait                                 ; is transmit buffer empty?
01A5 9116           01005         BTFSS   _tbmt
01A6 C1A5           01006         GOTO    bufwait
                    01007
01A7 B800           01008         MOVLB   BANK0                   ; set bank0
01A8                01009 shfwait
01A8 9115           01010         BTFSS   _trmt                   ; is transmit shift register empty?
01A9 C1A8           01011         GOTO    shfwait
                    01012
01AA 4A16           01013         MOVPF   WREG,TXREG              ; if so, send character
                    01014
01AB 0002           01015         RETURN
                    01016
                    01017 ;*****************************************************************************
                    01018
                    01019 ;*****************************************************************************
                    01020 ; NAME:         GetChk
                    01021 ;
                    01022 ; DESCRIPTION:  Check if character is in receive buffer.
```

```
                    01023 ;
                    01024
01AC                01025 GetChk
01AC B801           01026         MOVLB   BANK1                   ; set bank1
01AD 560A           01027         MOVPF   PIR,WREG
01AE B501           01028         ANDLW   CHARREADY               ; return status in WREG
01AF 0002           01029         RETURN
                    01030
                    01031 ;*****************************************************************************
                    01032
                    01033 ;*****************************************************************************
                    01034 ; NAME: PutDec
                    01035 ;
                    01036 ; DESCRIPTION:  Converts a hex value [0,F] in WREG to its ASCII equivalent.
                    01037 ;               The upper nibble of WREG is assumed to be zero.
                    01038 ;
                    01039 ; ENTRY CONDITIONS: WREG = value to be converted and sent in ASCII decimal
                    01040 ;
                    01041
                    01042         #if     DECIO
                    01043
01B0                01044 PutDec
01B0 B130           01045         ADDLW   0x30                    ; convert to ASCII
01B1 E1A4           01046         CALL    PutChar
01B2 0002           01047         RETURN
                    01048
                    01049         #endif
                    01050
                    01051 ;*****************************************************************************
                    01052
                    01053 ;*****************************************************************************
                    01054 ; NAME: PutHex
                    01055 ;
                    01056 ; DESCRIPTION:  Convert the WREG value to ASCII hexadecimal. The output
                    01057 ;               format is two digits with the A-F parts in upper case and
                    01058 ;               leading zeros. The result is sent out the serial port with
                    01059 ;               PutChar.
                    01060 ;
                    01061 ; ENTRY CONDITIONS: WREG = value to be converted and sent in ASCII hex
                    01062 ;
                    01063
01B3                01064 PutHex
                    01065
01B3 4A51           01066         MOVPF   WREG,HEXVAL
01B4 1D0A           01067         SWAPF   WREG, F
01B5 B50F           01068         ANDLW   0x0F
01B6 4A52           01069         MOVPF   WREG,HEXTMP
```

```
01B7 2D0A                  NEGW    WREG, F
01B8 B109                  ADDLW   0x09
01B9 970A                  BTFSS   WREG,MSB
01BA C1BE                  GOTO    puth20
01BB B037                  MOVLW   'A'-0x0A
01BC 0E52                  ADDWF   HEXTMP,W
01BD C1C0                  GOTO    puth25
01BE               puth20
01BE B030                  MOVLW   '0'
01BF 0E52                  ADDWF   HEXTMP,W
01C0               puth25
01C0 E1A4                  CALL    PutChar

01C1 6A51                  MOVFP   HEXVAL,WREG
01C2 B50F                  ANDLW   0x0F
01C3 4A52                  MOVPF   WREG,HEXTMP
01C4 2D0A                  NEGW    WREG, F
01C5 B109                  ADDLW   0x09
01C6 970A                  BTFSS   WREG,MSB
01C7 C1CB                  GOTO    putl20
01C8 B037                  MOVLW   'A'-0x0A
01C9 0E52                  ADDWF   HEXTMP,W
01CA C1CD                  GOTO    putl25
01CB               putl20
01CB B030                  MOVLW   '0'
01CC 0E52                  ADDWF   HEXTMP,W
01CD               putl25
01CD E1A4                  CALL    PutChar

01CE 0002                  RETURN

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

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

01CF               PutStr
01CF AB4F                  TABLRD  1,1,STRVALH
01D0               GetNextPair
01D0 A24F                  TLRD    1,STRVALH
01D1 A950                  TABLRD  0,1,STRVALL

01D2 6A4F                  MOVFP   STRVALH,WREG
01D3 31CA                  CPFSEQ  ZERO
```

```
01D4 C1D6          01117         GOTO    putH
01D5 0002          01118         RETURN
01D6               01119 putH
01D6 E1A4          01120         CALL    PutChar
                   01121
01D7 6A50          01122         MOVFP   STRVALL,WREG
01D8 31CA          01123         CPFSEQ  ZERO
01D9 C1DB          01124         GOTO    putL
01DA 0002          01125         RETURN
01DB               01126 putL
01DB E1A4          01127         CALL    PutChar
                   01128
01DC C1D0          01129         GOTO    GetNextPair
                   01130
                   01131 ;*******************************************************************************
                   01132
                   01133 ;*******************************************************************************
                   01134 ; NAME:         GetHex
                   01135 ;
                   01136 ; DESCRIPTION:  Receive an ASCII hex character from the serial port and
                   01137 ;               convert to numerical value.
                   01138 ;
                   01139 ; RETURNS:      numerical value in HEXVAL
                   01140
01DD               01141 GetHex
                   01142
01DD               01143 getnxt
01DD E08D          01144         CALL    IdleFunction
01DE E1AC          01145         CALL    GetChk
01DF 31CB          01146         CPFSEQ  ONE
01E0 C1DD          01147         GOTO    getnxt
                   01148
01E1 2953          01149         CLRF    HEXSTAT, F
01E2 E1A1          01150         CALL    GetChar
01E3 4A51          01151         MOVPF   WREG,HEXVAL
01E4 E1A4          01152         CALL    PutChar
01E5 B00D          01153         MOVLW   CR
01E6 0451          01154         SUBWF   HEXVAL,W
01E7 330A          01155         TSTFSZ  WREG
01E8 C1EA          01156         GOTO    gth10
01E9 C1F4          01157         GOTO    gthCR
01EA               01158 gth10
                   01159
01EA 6A51          01160         MOVFP   HEXVAL,WREG
01EB B239          01161         SUBLW   '9'
01EC 970A          01162         BTFSS   WREG,MSB
01ED C1F0          01163         GOTO    gth20
```

```
                        01164
01EE B009               01165         MOVLW   0x09
01EF 0F51               01166         ADDWF   HEXVAL, F
                        01167
01F0                    01168 gth20
01F0 B00F               01169         MOVLW   0x0F
01F1 0B51               01170         ANDWF   HEXVAL, F
01F2 2953               01171         CLRF    HEXSTAT, F
01F3 0002               01172         RETURN
                        01173
01F4                    01174 gthCR
01F4 B001               01175         MOVLW   0x01
01F5 4A53               01176         MOVPF   WREG,HEXSTAT
01F6 0002               01177         RETURN
                        01178
                        01179 ;*************************************************************************
                        01180
                        01181 ;*************************************************************************
                        01182 ; NAME:         GetDec
                        01183 ;
                        01184 ; DESCRIPTION:  Receive an ASCII decimal character from the serial port and
                        01185 ;               convert to its numerical value.
                        01186 ;
                        01187 ; ARGUMENTS:    numerical value is returned in DECVAL
                        01188 ;
                        01189
                        01190         #if     DECIO
                        01191
01F7                    01192 GetDec
                        01193
01F7                    01194 getdecnxt
01F7 E08D               01195         CALL    IdleFunction
01F8 E1AC               01196         CALL    GetChk
01F9 31CB               01197         CPFSEQ  ONE
01FA C1F7               01198         GOTO    getdecnxt
                        01199
01FB E1A1               01200         CALL    GetChar
01FC 4AA3               01201         MOVPF   WREG,DECVAL
01FD E1A4               01202         CALL    PutChar
                        01203
01FE B00D               01204         MOVLW   CR
01FF 04A3               01205         SUBWF   DECVAL,W
0200 30CA               01206         CPFSLT  ZERO
0201 C20E               01207         GOTO    gtdCR
0202 B02D               01208         MOVLW   MN
0203 04A3               01209         SUBWF   DECVAL,W
0204 30CA               01210         CPFSLT  ZERO
```

```
0205 C211           01211         GOTO    gtdMN
0206 B020           01212         MOVLW   SP
0207 04A3           01213         SUBWF   DECVAL,W
0208 30CA           01214         CPFSLT  ZERO
0209 C214           01215         GOTO    gtdSP
020A                01216 gtd09
020A B00F           01217         MOVLW   0x0F
020B 0BA3           01218         ANDWF   DECVAL, F
020C 29A4           01219         CLRF    DECSTAT, F
020D 0002           01220         RETURN
020E                01221 gtdCR
020E B002           01222         MOVLW   DEC_CR
020F 4AA4           01223         MOVPF   WREG,DECSTAT
0210 0002           01224         RETURN
0211                01225 gtdMN
0211 B001           01226         MOVLW   DEC_MN
0212 4AA4           01227         MOVPF   WREG,DECSTAT
0213 0002           01228         RETURN
0214                01229 gtdSP
0214 B000           01230         MOVLW   DEC_SP
0215 4AA4           01231         MOVPF   WREG,DECSTAT
0216 0002           01232         RETURN
                    01233
                    01234         #endif
                    01235
                    01236 ;*****************************************************************************
                    01237
                    01238 ;*****************************************************************************
                    01239 ; NAME:         getval
                    01240 ;
                    01241 ; DESCRIPTION:  Get a value [800000,7FFFFF] from the serial port and place
                    01242 ;               it in VALBUF.
                    01243 ;
                    01244         #if     DECIO
                    01245         #else
                    01246
                    01247 GetVal
                    01248         CLR24   VALBUF
                    01249 getnext
                    01250         CALL    GetHex
                    01251
                    01252         MOVLW   0x01
                    01253         CPFSEQ  HEXSTAT
                    01254         GOTO    shift
                    01255         RETURN
                    01256 shift
                    01257         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, F
                  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

0217      GetDecVal
                  CLR24   VALBUF
0217 2931     M           CLRF    VALBUF+B0, F
0218 2932     M           CLRF    VALBUF+B1, F
0219 2933     M           CLRF    VALBUF+B2, F
              M
021A E1F7         CALL    GetDec
021B 2BA6         SETF    DECSIGN, F
021C B001         MOVLW   DEC_MN
021D 31A4         CPFSEQ  DECSTAT
021E 29A6         CLRF    DECSIGN, F
```

```
021F                 01301 getdecnext
021F E1F7            01302         CALL    GetDec
                     01303
0220 B002            01304         MOVLW   DEC_CR
0221 31A4            01305         CPFSEQ  DECSTAT
0222 C224            01306         GOTO    mul10
0223 C248            01307         GOTO    fixsign
0224                 01308 mul10
                     01309
                     01310         RLC24   VALBUF                  ; multiply VALBUF by two
                         M
0224 8804                M         BCF     _carry
0225 1B31                M         RLCF    VALBUF+B0, F
0226 1B32                M         RLCF    VALBUF+B1, F
0227 1B33                M         RLCF    VALBUF+B2, F
                         M
                     01311         MOV24   VALBUF,DVALBUF          ; save in DVALBUF
                         M
0228 6A31                M         MOVFP   VALBUF+B0,WREG          ; get byte of a into w
0229 4A37                M         MOVPF   WREG,DVALBUF+B0         ; move to b(B0)
022A 6A32                M         MOVFP   VALBUF+B1,WREG          ; get byte of a into w
022B 4A38                M         MOVPF   WREG,DVALBUF+B1         ; move to b(B1)
022C 6A33                M         MOVFP   VALBUF+B2,WREG          ; get byte of a into w
022D 4A39                M         MOVPF   WREG,DVALBUF+B2         ; move to b(B2)
                         M
                     01312         RLC24   VALBUF
                         M
022E 8804                M         BCF     _carry
022F 1B31                M         RLCF    VALBUF+B0, F
0230 1B32                M         RLCF    VALBUF+B1, F
0231 1B33                M         RLCF    VALBUF+B2, F
                         M
                     01313         RLC24   VALBUF                  ; VALBUF now multiplied by eight
                         M
0232 8804                M         BCF     _carry
0233 1B31                M         RLCF    VALBUF+B0, F
0234 1B32                M         RLCF    VALBUF+B1, F
0235 1B33                M         RLCF    VALBUF+B2, F
                         M
                     01314         ADD24   DVALBUF,VALBUF          ; VALBUF now multiplied by ten
                         M
0236 6A37                M         MOVFP   DVALBUF+B0,WREG         ; get lowest byte of a into w
0237 0F31                M         ADDWF   VALBUF+B0, F            ; add lowest byte of b, save in b(B0)
0238 6A38                M         MOVFP   DVALBUF+B1,WREG         ; get 2nd byte of a into w
0239 1132                M         ADDWFC  VALBUF+B1, F            ; add 2nd byte of b, save in b(B1)
023A 6A39                M         MOVFP   DVALBUF+B2,WREG         ; get 3rd byte of a into w
023B 1133                M         ADDWFC  VALBUF+B2, F            ; add 3rd byte of b, save in b(B2)
```

```
                        M
                    01315         CLR24   DVALBUF
023C 2937               M         CLRF    DVALBUF+B0, F
023D 2938               M         CLRF    DVALBUF+B1, F
023E 2939               M         CLRF    DVALBUF+B2, F
                        M
023F 6AA3           01316         MOVFP   DECVAL,WREG
0240 4A37           01317         MOVPF   WREG,DVALBUF+B0
                    01318         ADD24   DVALBUF,VALBUF
                        M
0241 6A37               M         MOVFP   DVALBUF+B0,WREG         ; get lowest byte of a into w
0242 0F31               M         ADDWF   VALBUF+B0, F            ; add lowest byte of b, save in b(B0)
0243 6A38               M         MOVFP   DVALBUF+B1,WREG         ; get 2nd byte of a into w
0244 1132               M         ADDWFC  VALBUF+B1, F            ; add 2nd byte of b, save in b(B1)
0245 6A39               M         MOVFP   DVALBUF+B2,WREG         ; get 3rd byte of a into w
0246 1133               M         ADDWFC  VALBUF+B2, F            ; add 3rd byte of b, save in b(B2)
                        M
0247 C21F           01319         GOTO    getdecnext
0248                01320 fixsign
0248 290A           01321         CLRF    WREG, F
0249 32A6           01322         CPFSGT  DECSIGN
024A 0002           01323         RETURN
                    01324         NEG24   VALBUF
                        M
024B 1331               M         COMF    VALBUF+B0, F
024C 1332               M         COMF    VALBUF+B1, F
024D 1333               M         COMF    VALBUF+B2, F
024E 290A               M         CLRF    WREG, F
024F 1531               M         INCF    VALBUF+B0, F
0250 1132               M         ADDWFC  VALBUF+B1, F
0251 1133               M         ADDWFC  VALBUF+B2, F
                        M
0252 0002           01325         RETURN
                    01326
                    01327         #endif
                    01328
                    01329 ;*****************************************************************************
                    01330
                    01331 ;*****************************************************************************
                    01332 ; NAME:         GetPar
                    01333 ;
                    01334 ; DESCRIPTION:  Get a parameter number [0,FF] from the serial port and place
                    01335 ;               it in VALBUF+B0.
                    01336 ;
                    01337
0253                01338 GetPar
                    01339
```

```
                    01340         CLR24   VALBUF
0253 2931               M         CLRF    VALBUF+B0, F
0254 2932               M         CLRF    VALBUF+B1, F
0255 2933               M         CLRF    VALBUF+B2, F
                        M
                    01341
0256 E1DD           01342         CALL    GetHex
0257 6A51           01343         MOVFP   HEXVAL,WREG
0258 B50F           01344         ANDLW   0x0F
0259 4A31           01345         MOVPF   WREG,VALBUF+B0
025A 1D31           01346         SWAPF   VALBUF+B0, F
                    01347
025B E1DD           01348         CALL    GetHex
025C 6A31           01349         MOVFP   VALBUF+B0,WREG
                    01350
025D 0E51           01351         ADDWF   HEXVAL,W
025E 4A31           01352         MOVPF   WREG,VALBUF+B0
                    01353
025F 0002           01354         RETURN
                    01355
                    01356 ;*****************************************************************************
                    01357
                    01358 ;*****************************************************************************
                    01359 ; NAME:         PutVal
                    01360 ;
                    01361 ; DESCRIPTION:  Sends the value in VALBUF [800000,7FFFFF] out the serial port.
                    01362 ;
                    01363
                    01364         #if     DECIO
                    01365         #else
                    01366
                    01367 PutVal
                    01368
                    01369         MOVFP   VALBUF+B2,WREG
                    01370         CALL    PutHex
                    01371         MOVFP   VALBUF+B1,WREG
                    01372         CALL    PutHex
                    01373         MOVFP   VALBUF+B0,WREG
                    01374         CALL    PutHex
                    01375
                    01376         RETURN
                    01377
                    01378         #endif
                    01379
                    01380 ;*****************************************************************************
                    01381
                    01382 ;*****************************************************************************
```

```
                    01383 ; NAME:         PutDecVal
                    01384 ;
                    01385 ; DESCRIPTION:  Send the value in VALBUF [-8388608,8388607] out the serial port.
                    01386 ;
                    01387
                    01388         #if     DECIO
                    01389
0260                01390 PutDecVal
                    01391
0260 9733           01392         BTFSS   VALBUF+B2,MSB
0261 C26C           01393         GOTO    pdpos
                    01394         NEG24   VALBUF
                        M
0262 1331               M         COMF    VALBUF+B0, F
0263 1332               M         COMF    VALBUF+B1, F
0264 1333               M         COMF    VALBUF+B2, F
0265 290A               M         CLRF    WREG, F
0266 1531               M         INCF    VALBUF+B0, F
0267 1132               M         ADDWFC  VALBUF+B1, F
0268 1133               M         ADDWFC  VALBUF+B2, F
                        M
0269 B02D           01395         MOVLW   MN
026A E1A4           01396         CALL    PutChar
026B C26E           01397         GOTO    pddigits
026C                01398 pdpos
026C B020           01399         MOVLW   SP
026D E1A4           01400         CALL    PutChar
                    01401
026E                01402 pddigits
026E B09A           01403         MOVLW   LOW    DEC_TABLE          ; DEC_TABLE LSB
026F 4A0D           01404         MOVPF   WREG,TBLPTRL
0270 B007           01405         MOVLW   HIGH   DEC_TABLE          ; DEC_TABLE MSB
0271 4A0E           01406         MOVPF   WREG,TBLPTRH
                    01407
0272 A937           01408         TABLRD  0,1,DVALBUF+B0
0273                01409 readNextDec
0273 A037           01410         TLRD    0,DVALBUF+B0            ; read entry from table
0274 AB38           01411         TABLRD  1,1,DVALBUF+B1
0275 A939           01412         TABLRD  0,1,DVALBUF+B2
                    01413
0276 2B0A           01414         SETF    WREG, F                 ; unitsposition if end of table
0277 3137           01415         CPFSEQ  DVALBUF+B0
0278 C27A           01416         GOTO    getdigit
0279 C28E           01417         GOTO    unitsposition
027A                01418 getdigit
027A 1537           01419         INCF    DVALBUF+B0, F           ; restore to power of 10
027B 2BA3           01420         SETF    DECVAL, F               ; set DECVAL to -1
```

```
027C                 01421 inc
027C 15A3            01422         INCF    DECVAL, F               ; increment DECVAL
                     01423         SUB24   DVALBUF,VALBUF          ; check if in range
                         M
027D 6A37                M         MOVFP   DVALBUF+B0,WREG         ; get lowest byte of a into w
027E 0531                M         SUBWF   VALBUF+B0, F            ; sub lowest byte of b, save in b(B0)
027F 6A38                M         MOVFP   DVALBUF+B1,WREG         ; get 2nd byte of a into w
0280 0332                M         SUBWFB  VALBUF+B1, F            ; sub 2nd byte of b, save in b(B1)
0281 6A39                M         MOVFP   DVALBUF+B2,WREG         ; get 3rd byte of a into w
0282 0333                M         SUBWFB  VALBUF+B2, F            ; sub 3rd byte of b, save in b(B2)
                         M
0283 9733            01424         BTFSS   VALBUF+B2,MSB
0284 C27C            01425         GOTO    inc
                     01426
                     01427         ADD24   DVALBUF,VALBUF          ; if so, correct VALBUF for next digit
                         M
0285 6A37                M         MOVFP   DVALBUF+B0,WREG         ; get lowest byte of a into w
0286 0F31                M         ADDWF   VALBUF+B0, F            ; add lowest byte of b, save in b(B0)
0287 6A38                M         MOVFP   DVALBUF+B1,WREG         ; get 2nd byte of a into w
0288 1132                M         ADDWFC  VALBUF+B1, F            ; add 2nd byte of b, save in b(B1)
0289 6A39                M         MOVFP   DVALBUF+B2,WREG         ; get 3rd byte of a into w
028A 1133                M         ADDWFC  VALBUF+B2, F            ; add 3rd byte of b, save in b(B2)
                         M
028B 6AA3            01428         MOVFP   DECVAL,WREG             ; send DECVAL
028C E1B0            01429         CALL    PutDec
                     01430
028D C273            01431         GOTO    readNextDec             ; get next table entry
                     01432
028E                 01433 unitsposition
028E 6A31            01434         MOVFP   VALBUF+B0,WREG          ; unit position value now in VALBUF
028F E1B0            01435         CALL    PutDec
                     01436
0290 0002            01437         RETURN
                     01438
                     01439
                     01440         #endif
                     01441
                     01442 ;*****************************************************************************
                     01443
                     01444 ;*****************************************************************************
                     01445 ; NAME:         doError
                     01446 ;
                     01447 ; DESCRIPTION:  Calculates the position and velocity error.
                     01448 ;
                     01449
0291                 01450 doError
                     01451
```

```
                         MOV24   POSITION,POSERROR       ; calculate position error
                    M
0291 6A58           M    MOVFP   POSITION+B0,WREG        ; get byte of a into w
0292 4A7C           M    MOVPF   WREG,POSERROR+B0        ; move to b(B0)
0293 6A59           M    MOVFP   POSITION+B1,WREG        ; get byte of a into w
0294 4A7D           M    MOVPF   WREG,POSERROR+B1        ; move to b(B1)
0295 6A5A           M    MOVFP   POSITION+B2,WREG        ; get byte of a into w
0296 4A7E           M    MOVPF   WREG,POSERROR+B2        ; move to b(B2)
                    M
                         SUB24   MPOSITION,POSERROR
                    M
0297 6A75           M    MOVFP   MPOSITION+B0,WREG       ; get lowest byte of a into w
0298 057C           M    SUBWF   POSERROR+B0, F          ; sub lowest byte of b, save in b(B0)
0299 6A76           M    MOVFP   MPOSITION+B1,WREG       ; get 2nd byte of a into w
029A 037D           M    SUBWFB  POSERROR+B1, F          ; sub 2nd byte of b, save in b(B1)
029B 6A77           M    MOVFP   MPOSITION+B2,WREG       ; get 3rd byte of a into w
029C 037E           M    SUBWFB  POSERROR+B2, F          ; sub 3rd byte of b, save in b(B2)
                    M

029D 9F7E                BTFSC   POSERROR+B2,MSB         ; saturate error to lowest 16 bits
029E C2AA                GOTO    pneg
029F                ppos
029F 6A7D                MOVFP   POSERROR+B1,WREG
02A0 B580                ANDLW   0x80
02A1 097E                IORWF   POSERROR+B2, F
02A2 290A                CLRF    WREG, F
02A3 327E                CPFSGT  POSERROR+B2
02A4 C2B4                GOTO    psatok
02A5 297E                CLRF    POSERROR+B2, F          ; clear high byte for debug purposes
02A6 B07F                MOVLW   0x7F
02A7 4A7D                MOVPF   WREG,POSERROR+B1
02A8 2B7C                SETF    POSERROR, F
02A9 C2B4                GOTO    psatok
02AA                pneg
02AA 6A7D                MOVFP   POSERROR+B1,WREG
02AB B37F                IORLW   0x7F
02AC 0B7E                ANDWF   POSERROR+B2, F
02AD 2B0A                SETF    WREG, F
02AE 307E                CPFSLT  POSERROR+B2
02AF C2B4                GOTO    psatok
02B0 2B7E                SETF    POSERROR+B2, F          ; set high byte to 0xFF for debug purposes
02B1 297D                CLRF    POSERROR+B1, F
02B2 877D                BSF     POSERROR+B1,MSB
02B3 297C                CLRF    POSERROR, F
02B4                psatok

                         MOV24   VELOCITY,VELERROR       ; calculate velocity error
```

```
                        M
02B4 6A5B               M         MOVFP   VELOCITY+B0,WREG        ; get byte of a into w
02B5 4A7F               M         MOVPF   WREG,VELERROR+B0        ; move to b(B0)
02B6 6A5C               M         MOVFP   VELOCITY+B1,WREG        ; get byte of a into w
02B7 4A80               M         MOVPF   WREG,VELERROR+B1        ; move to b(B1)
02B8 6A5D               M         MOVFP   VELOCITY+B2,WREG        ; get byte of a into w
02B9 4A81               M         MOVPF   WREG,VELERROR+B2        ; move to b(B2)
                        M
                     01483         SUB24   MVELOCITY,VELERROR
                        M
02BA 6A78               M         MOVFP   MVELOCITY+B0,WREG       ; get lowest byte of a into w
02BB 057F               M         SUBWF   VELERROR+B0, F          ; sub lowest byte of b, save in b(B0)
02BC 6A79               M         MOVFP   MVELOCITY+B1,WREG       ; get 2nd byte of a into w
02BD 0380               M         SUBWFB  VELERROR+B1, F          ; sub 2nd byte of b, save in b(B1)
02BE 6A7A               M         MOVFP   MVELOCITY+B2,WREG       ; get 3rd byte of a into w
02BF 0381               M         SUBWFB  VELERROR+B2, F          ; sub 3rd byte of b, save in b(B2)
                        M
                     01484
02C0 9F81            01485         BTFSC   VELERROR+B2,MSB         ; saturate error to lowest 16 bits
02C1 C2CD            01486         GOTO    vneg
02C2                 01487 vpos
02C2 6A80            01488         MOVFP   VELERROR+B1,WREG
02C3 B580            01489         ANDLW   0x80
02C4 0981            01490         IORWF   VELERROR+B2, F
02C5 290A            01491         CLRF    WREG, F
02C6 3281            01492         CPFSGT  VELERROR+B2
02C7 C2D7            01493         GOTO    vsatok
02C8 2981            01494         CLRF    VELERROR+B2, F
02C9 B07F            01495         MOVLW   0x7F
02CA 4A80            01496         MOVPF   WREG,VELERROR+B1
02CB 2B7F            01497         SETF    VELERROR, F
02CC C2D7            01498         GOTO    vsatok
02CD                 01499 vneg
02CD 6A80            01500         MOVFP   VELERROR+B1,WREG
02CE B37F            01501         IORLW   0x7F
02CF 0B81            01502         ANDWF   VELERROR+B2, F
02D0 2B0A            01503         SETF    WREG, F
02D1 3081            01504         CPFSLT  VELERROR+B2
02D2 C2D7            01505         GOTO    vsatok
02D3 2B81            01506         SETF    VELERROR+B2, F
02D4 2980            01507         CLRF    VELERROR+B1, F
02D5 8780            01508         BSF     VELERROR+B1,MSB
02D6 297F            01509         CLRF    VELERROR, F
02D7                 01510 vsatok
02D7 0002            01511         RETURN
                     01512
                     01513 ;*****************************************************************************
```

```
                    01514
                    01515 ;*****************************************************************************
                    01516 ; NAME:         doServo
                    01517 ;
                    01518 ; DESCRIPTION:  Performs the servo loop calculations.
                    01519 ;
02D8                01520 doServo
                    01521
                    01522         MOV16   POSERROR,U0             ; save new position error in U0
                        M
02D8 6A7C               M         MOVFP   POSERROR+B0,WREG        ; get byte of a into w
02D9 4A8F               M         MOVPF   WREG,U0+B0              ; move to b(B0)
02DA 6A7D               M         MOVFP   POSERROR+B1,WREG        ; get byte of a into w
02DB 4A90               M         MOVPF   WREG,U0+B1              ; move to b(B1)
                        M
                    01523
                    01524         LOADAB  U0,KP                   ; compute KP*U0
                        M
02DC 7C8F               M         MOVFP   U0+B0,AARG+B0           ; load lo byte of A to AARG
02DD 7D90               M         MOVFP   U0+B1,AARG+B1           ; load hi byte of A to AARG
02DE 7E26               M         MOVFP   KP+B0,BARG+B0           ; load lo byte of B to BARG
02DF 7F27               M         MOVFP   KP+B1,BARG+B1           ; load hi byte of B to BARG
                        M
02E0 E630           01525         CALL    Dmult
                    01526         MVPF32  DPX,Y                   ; Y=KP*U0
                        M
02E1 5883               M         MOVPF   DPX+B0,Y+B0             ; move A(B0) to B(B0)
02E2 5984               M         MOVPF   DPX+B1,Y+B1             ; move A(B1) to B(B1)
02E3 5A85               M         MOVPF   DPX+B2,Y+B2             ; move A(B2) to B(B2)
02E4 5B86               M         MOVPF   DPX+B3,Y+B3             ; move A(B3) to B(B3)
                        M
                    01527
02E5 290A           01528         CLRF    WREG, F                 ; if previous output saturated, do
02E6 329A           01529         CPFSGT  SATFLAG                 ; not accumulate integrator
02E7 E618           01530         CALL    doIntegral
                    01531
                    01532         LOADAB  INTEGRAL,KI             ; compute KI*INTEGRAL
                        M
02E8 7C9B               M         MOVFP   INTEGRAL+B0,AARG+B0     ; load lo byte of A to AARG
02E9 7D9C               M         MOVFP   INTEGRAL+B1,AARG+B1     ; load hi byte of A to AARG
02EA 7E2A               M         MOVFP   KI+B0,BARG+B0           ; load lo byte of B to BARG
02EB 7F2B               M         MOVFP   KI+B1,BARG+B1           ; load hi byte of B to BARG
                        M
02EC E630           01533         CALL    Dmult
                    01534         ADD32   DPX,Y                   ; Y=KP*U0+KI*INTEGRAL
                        M
02ED 6A18               M         MOVFP   DPX+B0,WREG             ; get lowest byte of a into w
```

```
02EE 0F83              M         ADDWF   Y+B0, F               ; add lowest byte of b, save in b(B0)
02EF 6A19              M         MOVFP   DPX+B1,WREG           ; get 2nd byte of a into w
02F0 1184              M         ADDWFC  Y+B1, F               ; add 2nd byte of b, save in b(B1)
02F1 6A1A              M         MOVFP   DPX+B2,WREG           ; get 3rd byte of a into w
02F2 1185              M         ADDWFC  Y+B2, F               ; add 3rd byte of b, save in b(B2)
02F3 6A1B              M         MOVFP   DPX+B3,WREG           ; get 4th byte of a into w
02F4 1186              M         ADDWFC  Y+B3, F               ; add 4th byte of b, save in b(B3)
                       M
                    01535
                    01536         MVFP16  U0,AARG               ; compute KV*(U0-U1)
                       M
02F5 7C8F              M         MOVFP   U0+B0,AARG+B0         ; move A(B0) to B(B0)
02F6 7D90              M         MOVFP   U0+B1,AARG+B1         ; move A(B1) to B(B1)
                       M
                    01537         SUB16   U1,AARG
                       M
02F7 6A91              M         MOVFP   U1+B0,WREG            ; get lowest byte of a into w
02F8 051C              M         SUBWF   AARG+B0, F            ; sub lowest byte of b, save in b(B0)
02F9 6A92              M         MOVFP   U1+B1,WREG            ; get 2nd byte of a into w
02FA 031D              M         SUBWFB  AARG+B1, F            ; sub 2nd byte of b, save in b(B1)
                       M
                    01538         MVFP16  KV,BARG
                       M
02FB 7E28              M         MOVFP   KV+B0,BARG+B0         ; move A(B0) to B(B0)
02FC 7F29              M         MOVFP   KV+B1,BARG+B1         ; move A(B1) to B(B1)
                       M
02FD E630           01539         CALL    Dmult
                    01540         ADD32   DPX,Y                 ; Y=KP*U0+KI*INTEGRAL+KV*(U0-U1)
                       M
02FE 6A18              M         MOVFP   DPX+B0,WREG           ; get lowest byte of a into w
02FF 0F83              M         ADDWF   Y+B0, F               ; add lowest byte of b, save in b(B0)
0300 6A19              M         MOVFP   DPX+B1,WREG           ; get 2nd byte of a into w
0301 1184              M         ADDWFC  Y+B1, F               ; add 2nd byte of b, save in b(B1)
0302 6A1A              M         MOVFP   DPX+B2,WREG           ; get 3rd byte of a into w
0303 1185              M         ADDWFC  Y+B2, F               ; add 3rd byte of b, save in b(B2)
0304 6A1B              M         MOVFP   DPX+B3,WREG           ; get 4th byte of a into w
0305 1186              M         ADDWFC  Y+B3, F               ; add 4th byte of b, save in b(B3)
                       M
                    01541
0306 290A           01542         CLRF    WREG, F
0307 32C4           01543         CPFSGT  SHIFTNUM
0308 C311           01544         GOTO    grabok
0309 78C4           01545         MOVFP   SHIFTNUM,TMP
030A                01546 grabloop
                    01547         RLC32   Y
                       M
030A 8804              M         BCF     _carry
```

```
030B 1B83              M         RLCF    Y+B0, F
030C 1B84              M         RLCF    Y+B1, F
030D 1B85              M         RLCF    Y+B2, F
030E 1B86              M         RLCF    Y+B3, F
                       M
030F 1718          01548         DECFSZ  TMP, F
0310 C30A          01549         GOTO    grabloop
                   01550
0311               01551 grabok
0311 299A          01552         CLRF    SATFLAG, F
0312 9F86          01553         BTFSC   Y+B3,MSB                ; saturate to middle 16 bits,
0313 C321          01554         GOTO    negs                    ; keeping top 10 bits for PW1DCH
0314               01555 poss                                     ; and PW1DCL
0314 6A85          01556         MOVFP   Y+B2,WREG               ; check if Y >= 2**23
0315 B580          01557         ANDLW   0x80
0316 0986          01558         IORWF   Y+B3, F
0317 290A          01559         CLRF    WREG, F
0318 3286          01560         CPFSGT  Y+B3
0319 C32D          01561         GOTO    zero6bits               ; if not, zero 6 bits
                   01562
031A 159A          01563         INCF    SATFLAG, F              ; if so, set Y=0x007FFFFF
031B 2986          01564         CLRF    Y+B3, F                 ; clear for debug purposes
031C B07F          01565         MOVLW   0x7F
031D 4A85          01566         MOVPF   WREG,Y+B2
031E 2B84          01567         SETF    Y+B1, F
031F 2B83          01568         SETF    Y+B0, F
0320 C32D          01569         GOTO    zero6bits
0321               01570 negs
0321 6A85          01571         MOVFP   Y+B2,WREG               ; check if Y <= -2**23
0322 B37F          01572         IORLW   0x7F
0323 0B86          01573         ANDWF   Y+B3, F
0324 2B0A          01574         SETF    WREG, F
0325 3086          01575         CPFSLT  Y+B3
0326 C32D          01576         GOTO    zero6bits               ; if not, zero 6 bits
                   01577
0327 2B9A          01578         SETF    SATFLAG, F              ; if so, set Y = 0xFF800000
0328 2B86          01579         SETF    Y+B3, F
0329 2985          01580         CLRF    Y+B2, F
032A 8785          01581         BSF     Y+B2,MSB
032B 2984          01582         CLRF    Y+B1, F
032C 2983          01583         CLRF    Y+B0, F
                   01584
032D               01585 zero6bits
                   01586         MOV24   Y+B1,YPWM+B0            ; move Y to YPWM and zero 6 bits
                       M
032D 6A84              M         MOVFP   Y+B1+B0,WREG            ; get byte of a into w
032E 4A87              M         MOVPF   WREG,YPWM+B0+B0         ; move to b(B0)
```

```
032F 6A85              M         MOVFP   Y+B1+B1,WREG          ; get byte of a into w
0330 4A88              M         MOVPF   WREG,YPWM+B0+B1       ; move to b(B1)
0331 6A86              M         MOVFP   Y+B1+B2,WREG          ; get byte of a into w
0332 4A89              M         MOVPF   WREG,YPWM+B0+B2       ; move to b(B2)
                       M
0333                01587 doTorque                            ; entry point for torque mode
0333 B0C0           01588         MOVLW   0xC0
0334 0B87           01589         ANDWF   YPWM+B0, F
                    01590
0335 9F88           01591         BTFSC   YPWM+B1,MSB
0336 C33E           01592         GOTO    tmlimit
0337                01593 tplimit
0337 9697           01594         BTFSS   EXTSTAT,BIT6
0338 C344           01595         GOTO    mplimitok
                    01596         CLR32   YPWM
0339 2987              M          CLRF    YPWM+B0, F
033A 2988              M          CLRF    YPWM+B1, F
033B 2989              M          CLRF    YPWM+B2, F
033C 298A              M          CLRF    YPWM+B3, F
                       M
033D C344           01597         GOTO    mplimitok
033E                01598 tmlimit
033E 9597           01599         BTFSS   EXTSTAT,BIT5
033F C344           01600         GOTO    mplimitok
                    01601         CLR32   YPWM
0340 2987              M          CLRF    YPWM+B0, F
0341 2988              M          CLRF    YPWM+B1, F
0342 2989              M          CLRF    YPWM+B2, F
0343 298A              M          CLRF    YPWM+B3, F
                       M
0344                01602 mplimitok
0344 B07F           01603         MOVLW   PW1DCH_INIT           ; adjustment from bipolar to unipolar
0345 4A19           01604         MOVPF   WREG,TMP+B1           ; for 50% duty cycle
0346 B0C0           01605         MOVLW   PW1DCL_INIT
0347 4A18           01606         MOVPF   WREG,TMP+B0
                    01607         ADD16   TMP,YPWM
                       M
0348 6A18              M          MOVFP   TMP+B0,WREG           ; get lowest byte of a into w
0349 0F87              M          ADDWF   YPWM+B0, F            ; add lowest byte of b, save in b(B0)
034A 6A19              M          MOVFP   TMP+B1,WREG           ; get 2nd byte of a into w
034B 1188              M          ADDWFC  YPWM+B1, F            ; add 2nd byte of b, save in b(B1)
                       M
                    01608
034C 2919           01609         CLRF    TMP+B1, F             ; correct by 1 LSB
034D B040           01610         MOVLW   0x40                  ; add one to bit5 of PW1DCL
034E 4A18           01611         MOVPF   WREG,TMP+B0
                    01612         ADD16   TMP,YPWM
```

```
                        M
034F 6A18               M         MOVFP   TMP+B0,WREG           ; get lowest byte of a into w
0350 0F87               M         ADDWF   YPWM+B0, F            ; add lowest byte of b, save in b(B0)
0351 6A19               M         MOVFP   TMP+B1,WREG           ; get 2nd byte of a into w
0352 1188               M         ADDWFC  YPWM+B1, F            ; add 2nd byte of b, save in b(B1)
                        M
                    01613
0353                01614 testmax
0353 291A           01615         CLRF    TMP+B2, F             ; check pwm maximum limit
0354 2989           01616         CLRF    YPWM+B2, F            ; LMD18200 must have a minimum pulse
0355 298A           01617         CLRF    YPWM+B3, F            ; so duty cycle must not be 0 or 100%
                    01618         MVFP16  YPWMAX,TMP
                        M
0356 788D               M         MOVFP   YPWMAX+B0,TMP+B0      ; move A(B0) to B(B0)
0357 798E               M         MOVFP   YPWMAX+B1,TMP+B1      ; move A(B1) to B(B1)
                        M
                    01619         SUB24   YPWM,TMP
                        M
0358 6A87               M         MOVFP   YPWM+B0,WREG          ; get lowest byte of a into w
0359 0518               M         SUBWF   TMP+B0, F             ; sub lowest byte of b, save in b(B0)
035A 6A88               M         MOVFP   YPWM+B1,WREG          ; get 2nd byte of a into w
035B 0319               M         SUBWFB  TMP+B1, F             ; sub 2nd byte of b, save in b(B1)
035C 6A89               M         MOVFP   YPWM+B2,WREG          ; get 3rd byte of a into w
035D 031A               M         SUBWFB  TMP+B2, F             ; sub 3rd byte of b, save in b(B2)
                        M
035E 971A           01620         BTFSS   TMP+B2,MSB
035F C365           01621         GOTO    testmin
                    01622         MOV16   YPWMAX,YPWM           ; saturate to max
                        M
0360 6A8D               M         MOVFP   YPWMAX+B0,WREG        ; get byte of a into w
0361 4A87               M         MOVPF   WREG,YPWM+B0          ; move to b(B0)
0362 6A8E               M         MOVFP   YPWMAX+B1,WREG        ; get byte of a into w
0363 4A88               M         MOVPF   WREG,YPWM+B1          ; move to b(B1)
                        M
0364 C376           01623         GOTO    limitok
0365                01624 testmin
0365 291A           01625         CLRF    TMP+B2, F             ; check pwm minimum limit
0366 2989           01626         CLRF    YPWM+B2, F
0367 298A           01627         CLRF    YPWM+B3, F
                    01628         MVFP16  YPWMIN,TMP
                        M
0368 788B               M         MOVFP   YPWMIN+B0,TMP+B0      ; move A(B0) to B(B0)
0369 798C               M         MOVFP   YPWMIN+B1,TMP+B1      ; move A(B1) to B(B1)
                        M
                    01629         SUB24   YPWM,TMP
                        M
036A 6A87               M         MOVFP   YPWM+B0,WREG          ; get lowest byte of a into w
```

```
036B 0518              M         SUBWF   TMP+B0, F               ; sub lowest byte of b, save in b(B0)
036C 6A88              M         MOVFP   YPWM+B1,WREG            ; get 2nd byte of a into w
036D 0319              M         SUBWFB  TMP+B1, F               ; sub 2nd byte of b, save in b(B1)
036E 6A89              M         MOVFP   YPWM+B2,WREG            ; get 3rd byte of a into w
036F 031A              M         SUBWFB  TMP+B2, F               ; sub 3rd byte of b, save in b(B2)
                       M
0370 9F1A          01630         BTFSC   TMP+B2,MSB
0371 C376          01631         GOTO    limitok
                   01632         MOV16   YPWMIN,YPWM             ; saturate to min
                       M
0372 6A8B              M         MOVFP   YPWMIN+B0,WREG          ; get byte of a into w
0373 4A87              M         MOVPF   WREG,YPWM+B0            ; move to b(B0)
0374 6A8C              M         MOVFP   YPWMIN+B1,WREG          ; get byte of a into w
0375 4A88              M         MOVPF   WREG,YPWM+B1            ; move to b(B1)
                       M
0376               01633 limitok
0376 B803          01634         MOVLB   BANK3                   ; set new duty cycle
0377 7087          01635         MOVFP   YPWM+B0,PW1DCL
0378 7288          01636         MOVFP   YPWM+B1,PW1DCH
                   01637
                   01638         MOV16   U0,U1                   ; push errors into U(k-1)
                       M
0379 6A8F              M         MOVFP   U0+B0,WREG              ; get byte of a into w
037A 4A91              M         MOVPF   WREG,U1+B0              ; move to b(B0)
037B 6A90              M         MOVFP   U0+B1,WREG              ; get byte of a into w
037C 4A92              M         MOVPF   WREG,U1+B1              ; move to b(B1)
                       M
                   01639
037D 0002          01640         RETURN
                   01641
                   01642 ;*************************************************************************
                   01643
                   01644 ;*************************************************************************
                   01645 ; NAME:         doPreMove
                   01646 ;
                   01647 ; DESCRIPTION:
                   01648
037E               01649 doPreMove
                   01650
                   01651         MOV24   NMOVVAL,MOVVAL          ; move buffer to MOVVAL
                       M
037E 6A5E              M         MOVFP   NMOVVAL+B0,WREG         ; get byte of a into w
037F 4A62              M         MOVPF   WREG,MOVVAL+B0          ; move to b(B0)
0380 6A5F              M         MOVFP   NMOVVAL+B1,WREG         ; get byte of a into w
0381 4A63              M         MOVPF   WREG,MOVVAL+B1          ; move to b(B1)
0382 6A60              M         MOVFP   NMOVVAL+B2,WREG         ; get byte of a into w
0383 4A64              M         MOVPF   WREG,MOVVAL+B2          ; move to b(B2)
```

```
                        M
0384 8F98       01652   BCF     MOVSTAT,BIT7            ; clear buffer flag
0385 8698       01653   BSF     MOVSTAT,BIT6            ; set motion status flag
0386 8598       01654   BSF     MOVSTAT,BIT5            ; set move in progress flag
0387 6ACB       01655   MOVFP   ONE,WREG
0388 4A99       01656   MOVPF   WREG,MOVFLAG            ; initialize MOVEFLAG to 1
                01657
                01658
0389 2954       01659   CLRF    OPOSITION+B0, F         ; initialize buffers
                01660   MOV24   POSITION,OPOSITION+B1
                    M
038A 6A58           M   MOVFP   POSITION+B0,WREG        ; get byte of a into w
038B 4A55           M   MOVPF   WREG,OPOSITION+B1+B0    ; move to b(B0)
038C 6A59           M   MOVFP   POSITION+B1,WREG        ; get byte of a into w
038D 4A56           M   MOVPF   WREG,OPOSITION+B1+B1    ; move to b(B1)
038E 6A5A           M   MOVFP   POSITION+B2,WREG        ; get byte of a into w
038F 4A57           M   MOVPF   WREG,OPOSITION+B1+B2    ; move to b(B2)
                    M
                01661   MOV32   OPOSITION,MOVPBUF
                    M
0390 6A54           M   MOVFP   OPOSITION+B0,WREG       ; get byte of a into w
0391 4AAF           M   MOVPF   WREG,MOVPBUF+B0         ; move to b(B0)
0392 6A55           M   MOVFP   OPOSITION+B1,WREG       ; get byte of a into w
0393 4AB0           M   MOVPF   WREG,MOVPBUF+B1         ; move to b(B1)
0394 6A56           M   MOVFP   OPOSITION+B2,WREG       ; get byte of a into w
0395 4AB1           M   MOVPF   WREG,MOVPBUF+B2         ; move to b(B2)
0396 6A57           M   MOVFP   OPOSITION+B3,WREG       ; get byte of a into w
0397 4AB2           M   MOVPF   WREG,MOVPBUF+B3         ; move to b(B3)
                    M
0398 299A       01662   CLRF    SATFLAG, F
                01663   CLR16   MOVTIME                 ; clear move times
0399 296A           M   CLRF    MOVTIME+B0, F
039A 296B           M   CLRF    MOVTIME+B1, F
                    M
                01664   CLR16   T1                      ; 0 used as flag for no maximum speed
039B 296D           M   CLRF    T1+B0, F
039C 296E           M   CLRF    T1+B1, F
                    M
                01665   CLR16   T2
039D 296F           M   CLRF    T2+B0, F
039E 2970           M   CLRF    T2+B1, F
                    M
                01666   CLR16   TAU
039F 2971           M   CLRF    TAU+B0, F
03A0 2972           M   CLRF    TAU+B1, F
                    M
                01667   CLR32   MOVDEL                  ; clear move discretization error
```

```
03A1 29BB               M         CLRF    MOVDEL+B0, F
03A2 29BC               M         CLRF    MOVDEL+B1, F
03A3 29BD               M         CLRF    MOVDEL+B2, F
03A4 29BE               M         CLRF    MOVDEL+B3, F
                        M
                    01668         CLR16   PH2FLAT                 ; clear phase 2 flat counter
03A5 29BF               M         CLRF    PH2FLAT+B0, F
03A6 29C0               M         CLRF    PH2FLAT+B1, F
                        M
                    01669
03A7 3396           01670         TSTFSZ  MODETYPE
03A8 C404           01671         GOTO    vmode
03A9                01672 pmode
                    01673         MVFP24  MOVVAL,TMP
                        M
03A9 7862               M         MOVFP   MOVVAL+B0,TMP+B0        ; move A(B0) to B(B0)
03AA 7963               M         MOVFP   MOVVAL+B1,TMP+B1        ; move A(B1) to B(B1)
03AB 7A64               M         MOVFP   MOVVAL+B2,TMP+B2        ; move A(B2) to B(B2)
                        M
03AC 971A           01674         BTFSS   TMP+B2,MSB
03AD C3B5           01675         GOTO    mvpos
                    01676         NEG24   TMP
                        M
03AE 1318               M         COMF    TMP+B0, F
03AF 1319               M         COMF    TMP+B1, F
03B0 131A               M         COMF    TMP+B2, F
03B1 290A               M         CLRF    WREG, F
03B2 1518               M         INCF    TMP+B0, F
03B3 1119               M         ADDWFC  TMP+B1, F
03B4 111A               M         ADDWFC  TMP+B2, F
                        M
03B5                01677 mvpos
03B5 291C           01678         CLRF    MOVTMP+B0, F            ; calculate  abs(MOVVAL) - 3
03B6 291D           01679         CLRF    MOVTMP+B1, F            ; do immediate move if negative
03B7 291E           01680         CLRF    MOVTMP+B2, F
03B8 801C           01681         BSF     MOVTMP+B0,BIT0
03B9 811C           01682         BSF     MOVTMP+B0,BIT1
                    01683         SUB24   MOVTMP,TMP
                        M
03BA 6A1C               M         MOVFP   MOVTMP+B0,WREG          ; get lowest byte of a into w
03BB 0518               M         SUBWF   TMP+B0, F               ; sub lowest byte of b, save in b(B0)
03BC 6A1D               M         MOVFP   MOVTMP+B1,WREG          ; get 2nd byte of a into w
03BD 0319               M         SUBWFB  TMP+B1, F               ; sub 2nd byte of b, save in b(B1)
03BE 6A1E               M         MOVFP   MOVTMP+B2,WREG          ; get 3rd byte of a into w
03BF 031A               M         SUBWFB  TMP+B2, F               ; sub 3rd byte of b, save in b(B2)
                        M
                    01684
```

```
03C0 971A          01685         BTFSS   TMP+B2,MSB                ; check for zero move
03C1 C3CD          01686         GOTO    nonzero
03C2 2B95          01687         SETF    SERVOFLAG, F              ; set servoflag to restore servo
03C3 2999          01688         CLRF    MOVFLAG, F
03C4 8D98          01689         BCF     MOVSTAT,BIT5
03C5 8E98          01690         BCF     MOVSTAT,BIT6
                   01691         ADD24   MOVVAL,POSITION
                       M
03C6 6A62              M         MOVFP   MOVVAL+B0,WREG          ; get lowest byte of a into w
03C7 0F58              M         ADDWF   POSITION+B0, F          ; add lowest byte of b, save in b(B0)
03C8 6A63              M         MOVFP   MOVVAL+B1,WREG          ; get 2nd byte of a into w
03C9 1159              M         ADDWFC  POSITION+B1, F          ; add 2nd byte of b, save in b(B1)
03CA 6A64              M         MOVFP   MOVVAL+B2,WREG          ; get 3rd byte of a into w
03CB 115A              M         ADDWFC  POSITION+B2, F          ; add 3rd byte of b, save in b(B2)
                       M
03CC 0002          01692         RETURN
03CD               01693 nonzero
                   01694         CLR32   MOVVBUF
03CD 29B3              M         CLRF    MOVVBUF+B0, F
03CE 29B4              M         CLRF    MOVVBUF+B1, F
03CF 29B5              M         CLRF    MOVVBUF+B2, F
03D0 29B6              M         CLRF    MOVVBUF+B3, F
                       M
                   01695
03D1 6A64          01696         MOVFP   MOVVAL+B2,WREG          ; move sign (00h=positive,80h=negative)
03D2 B580          01697         ANDLW   0x80
03D3 4A6C          01698         MOVPF   WREG,MOVSIGN
                   01699
03D4 29AE          01700         CLRF    V+B3, F                 ; create appropriate velocity and
                   01701         MOV24   VL,V                    ; acceleration limits from move sign
                       M
03D5 6A20              M         MOVFP   VL+B0,WREG              ; get byte of a into w
03D6 4AAB              M         MOVPF   WREG,V+B0               ; move to b(B0)
03D7 6A21              M         MOVFP   VL+B1,WREG              ; get byte of a into w
03D8 4AAC              M         MOVPF   WREG,V+B1               ; move to b(B1)
03D9 6A22              M         MOVFP   VL+B2,WREG              ; get byte of a into w
03DA 4AAD              M         MOVPF   WREG,V+B2               ; move to b(B2)
                       M
03DB 29AA          01702         CLRF    A+B3, F
                   01703         MOV24   AL,A
                       M
03DC 6A23              M         MOVFP   AL+B0,WREG              ; get byte of a into w
03DD 4AA7              M         MOVPF   WREG,A+B0               ; move to b(B0)
03DE 6A24              M         MOVFP   AL+B1,WREG              ; get byte of a into w
03DF 4AA8              M         MOVPF   WREG,A+B1               ; move to b(B1)
03E0 6A25              M         MOVFP   AL+B2,WREG              ; get byte of a into w
03E1 4AA9              M         MOVPF   WREG,A+B2               ; move to b(B2)
```

```
                        M
03E2 290A       01704   M       CLRF    WREG, F
03E3 326C       01705   M       CPFSGT  MOVSIGN
03E4 C3F7       01706   M       GOTO    minc
                01707   M       NEG32   V
                        M
03E5 13AB               M       COMF    V+B0, F
03E6 13AC               M       COMF    V+B1, F
03E7 13AD               M       COMF    V+B2, F
03E8 13AE               M       COMF    V+B3, F
03E9 290A               M       CLRF    WREG, F
03EA 15AB               M       INCF    V+B0, F
03EB 11AC               M       ADDWFC  V+B1, F
03EC 11AD               M       ADDWFC  V+B2, F
03ED 11AE               M       ADDWFC  V+B3, F
                        M
                01708   M       NEG32   A
                        M
03EE 13A7               M       COMF    A+B0, F
03EF 13A8               M       COMF    A+B1, F
03F0 13A9               M       COMF    A+B2, F
03F1 13AA               M       COMF    A+B3, F
03F2 290A               M       CLRF    WREG, F
03F3 15A7               M       INCF    A+B0, F
03F4 11A8               M       ADDWFC  A+B1, F
03F5 11A9               M       ADDWFC  A+B2, F
03F6 11AA               M       ADDWFC  A+B3, F
                        M
03F7            01709 minc
03F7 2966       01710   M       CLRF    HMOVVAL+B0, F           ; evaluate MOVVAL/2
                01711   M       MOV24   MOVVAL,HMOVVAL+B1
                        M
03F8 6A62               M       MOVFP   MOVVAL+B0,WREG          ; get byte of a into w
03F9 4A67               M       MOVPF   WREG,HMOVVAL+B1+B0      ; move to b(B0)
03FA 6A63               M       MOVFP   MOVVAL+B1,WREG          ; get byte of a into w
03FB 4A68               M       MOVPF   WREG,HMOVVAL+B1+B1      ; move to b(B1)
03FC 6A64               M       MOVFP   MOVVAL+B2,WREG          ; get byte of a into w
03FD 4A69               M       MOVPF   WREG,HMOVVAL+B1+B2      ; move to b(B2)
                        M
                01712   M       RRC32   HMOVVAL                 ; half move in Q8
                        M
03FE 1A69               M       RLCF    HMOVVAL+B3,W            ; move sign into carry bit
03FF 1969               M       RRCF    HMOVVAL+B3, F
0400 1968               M       RRCF    HMOVVAL+B2, F
0401 1967               M       RRCF    HMOVVAL+B1, F
0402 1966               M       RRCF    HMOVVAL+B0, F
                        M
```

```
0403 C43D          01713         GOTO    modeready
                   01714
0404               01715 vmode
0404 9F96          01716         BTFSC   MODETYPE,MSB          ; is it torque move?
0405 C445          01717         GOTO    tmode
                   01718
0406 2969          01719         CLRF    HMOVVAL+B3, F         ; compute final minus initial velocity
                   01720         MOV24   MOVVAL,HMOVVAL
                       M
0407 6A62              M         MOVFP   MOVVAL+B0,WREG        ; get byte of a into w
0408 4A66              M         MOVPF   WREG,HMOVVAL+B0       ; move to b(B0)
0409 6A63              M         MOVFP   MOVVAL+B1,WREG        ; get byte of a into w
040A 4A67              M         MOVPF   WREG,HMOVVAL+B1       ; move to b(B1)
040B 6A64              M         MOVFP   MOVVAL+B2,WREG        ; get byte of a into w
040C 4A68              M         MOVPF   WREG,HMOVVAL+B2       ; move to b(B2)
                       M
040D 9F64          01721         BTFSC   MOVVAL+B2,MSB
040E 2B69          01722         SETF    HMOVVAL+B3, F
                   01723         SUB32   MOVVBUF,HMOVVAL
                       M
040F 6AB3              M         MOVFP   MOVVBUF+B0,WREG       ; get lowest byte of a into w
0410 0566              M         SUBWF   HMOVVAL+B0, F         ; sub lowest byte of b, save in b(B0)
0411 6AB4              M         MOVFP   MOVVBUF+B1,WREG       ; get 2nd byte of a into w
0412 0367              M         SUBWFB  HMOVVAL+B1, F         ; sub 2nd byte of b, save in b(B1)
0413 6AB5              M         MOVFP   MOVVBUF+B2,WREG       ; get 3rd byte of a into w
0414 0368              M         SUBWFB  HMOVVAL+B2, F         ; sub 3rd byte of b, save in b(B2)
0415 6AB6              M         MOVFP   MOVVBUF+B3,WREG       ; get 4th byte of a into w
0416 0369              M         SUBWFB  HMOVVAL+B3, F         ; sub 4th byte of b, save in b(B3)
                       M
                   01724
0417 6A69          01725         MOVFP   HMOVVAL+B3,WREG
0418 B580          01726         ANDLW   0x80
0419 4A6C          01727         MOVPF   WREG,MOVSIGN
                   01728
041A 29AE          01729         CLRF    V+B3, F               ; create appropriate velocity and
                   01730         MOV24   VL,V                  ; acceleration limits from move sign
                       M
041B 6A20              M         MOVFP   VL+B0,WREG            ; get byte of a into w
041C 4AAB              M         MOVPF   WREG,V+B0             ; move to b(B0)
041D 6A21              M         MOVFP   VL+B1,WREG            ; get byte of a into w
041E 4AAC              M         MOVPF   WREG,V+B1             ; move to b(B1)
041F 6A22              M         MOVFP   VL+B2,WREG            ; get byte of a into w
0420 4AAD              M         MOVPF   WREG,V+B2             ; move to b(B2)
                       M
0421 29AA          01731         CLRF    A+B3, F
                   01732         MOV24   AL,A
                       M
```

```
0422 6A23              M         MOVFP   AL+B0,WREG             ; get byte of a into w
0423 4AA7              M         MOVPF   WREG,A+B0              ; move to b(B0)
0424 6A24              M         MOVFP   AL+B1,WREG             ; get byte of a into w
0425 4AA8              M         MOVPF   WREG,A+B1              ; move to b(B1)
0426 6A25              M         MOVFP   AL+B2,WREG             ; get byte of a into w
0427 4AA9              M         MOVPF   WREG,A+B2              ; move to b(B2)
                       M
0428 290A          01733         CLRF    WREG, F
0429 326C          01734         CPFSGT  MOVSIGN
042A C43D          01735         GOTO    modeready
                   01736         NEG32   V
                       M
042B 13AB              M         COMF    V+B0, F
042C 13AC              M         COMF    V+B1, F
042D 13AD              M         COMF    V+B2, F
042E 13AE              M         COMF    V+B3, F
042F 290A              M         CLRF    WREG, F
0430 15AB              M         INCF    V+B0, F
0431 11AC              M         ADDWFC  V+B1, F
0432 11AD              M         ADDWFC  V+B2, F
0433 11AE              M         ADDWFC  V+B3, F
                       M
                   01737         NEG32   A
                       M
0434 13A7              M         COMF    A+B0, F
0435 13A8              M         COMF    A+B1, F
0436 13A9              M         COMF    A+B2, F
0437 13AA              M         COMF    A+B3, F
0438 290A              M         CLRF    WREG, F
0439 15A7              M         INCF    A+B0, F
043A 11A8              M         ADDWFC  A+B1, F
043B 11A9              M         ADDWFC  A+B2, F
043C 11AA              M         ADDWFC  A+B3, F
                       M
                   01738
043D               01739 modeready
043D 2965          01740         CLRF    MOVVAL+B3, F
043E 9F64          01741         BTFSC   MOVVAL+B2,MSB
043F 2B65          01742         SETF    MOVVAL+B3, F
                   01743
0440 2B95          01744         SETF    SERVOFLAG, F           ; set servoflag to restore servo
                   01745                                        ; if stopped
                   01746 ;*************************************************************************
                   01747 ;       For PICMASTER Debug/servo tuning puporses only Purposes Only
                   01748 ;
0441               01749 testCapCount
0441 6AC6          01750         MOVFP   CAPCOUNT+B0,WREG
```

```
0442 08C7           01751         IORWF   CAPCOUNT+B1,W
0443 4AC5           01752         MOVPF   WREG,CAPFLAG
                    01753 ;*****************************************************************************
                    01754
0444 0002           01755         RETURN
                    01756
0445                01757 tmode                                   ; torque/voltage mode
                    01758         MOV16   MOVVAL+B1,YPWM          ; set new commanded value
                        M
0445 6A63               M         MOVFP   MOVVAL+B1+B0,WREG       ; get byte of a into w
0446 4A87               M         MOVPF   WREG,YPWM+B0            ; move to b(B0)
0447 6A64               M         MOVFP   MOVVAL+B1+B1,WREG       ; get byte of a into w
0448 4A88               M         MOVPF   WREG,YPWM+B1            ; move to b(B1)
                        M
0449 2995           01759         CLRF    SERVOFLAG, F            ; disable servo
044A E333           01760         CALL    doTorque                ; set pwm duty cycle
044B 2999           01761         CLRF    MOVFLAG, F
044C 8D98           01762         BCF     MOVSTAT,BIT5
044D C441           01763     goto    testCapCount
                    01764
044E 0002           01765         RETURN
                    01766
                    01767 ;*****************************************************************************
                    01768
                    01769 ;*****************************************************************************
                    01770 ; NAME:         doMove
                    01771 ;
                    01772 ; DESCRIPTION:  In position mode, trapezoidal moves are performed. Phase1
                    01773 ;               and phase2 respectively, are the periods for the first and
                    01774 ;               second halves of the move. The move time is defined as zero
                    01775 ;               at the beginning of the move,T2 is the time at half the move, T1 is the time w
                    01776 ;               begins,(the region of constant velocity reduces to a point
                    01777 ;               in the case where maximum speed is not realized, and the
                    01778 ;               trapezoidal move degenerates into a trianglular move,
                    01779 ;               together with T1=T2), and TAU is the total time of the move.
                    01780 ;               The accelerations are +-AL or 0.
                    01781 ;
                    01782 ;
                    01783 ;                triangle speed                 trapezoidal speed
                    01784 ;
                    01785 ;                                                _________________
                    01786 ;                                               /                 \
                    01787 ;                  /\                          /                   \
                    01788 ;                 /  \                        /                     \
                    01789 ;                /    \                      /                       \
                    01790 ;               /      \                    /                         \
                    01791 ;              /        \                  /                           \
```

```
                    01792 ;
                    01793 ;             0   T1=T2  TAU             0     T1      T2            TAU
                    01794 ;
                    01795 ;
                    01796 ;
                    01797 ;               Let x denote the undershoot and y the overshoot commanded
                    01798 ;               at adjacent sample times as half the move is crossed.
                    01799 ;               In the case of a triangular move, the discretization error
                    01800 ;               is given by
                    01801 ;
                    01802 ;                        error = min (2x,2y)
                    01803 ;
                    01804 ;               For a trapezoidal move, the discretization error is
                    01805 ;
                    01806 ;                       error = min (2x,y-x) <= .5*(maximum commanded speed)
                    01807 ;
                    01808 ;               This discretization error is resolved in the final sample
                    01809 ;               time of the move by executing a step to the final position
                    01810 ;               at zero speed. The method employed here the best possible
                    01811 ;               performance with regard to discretization error without
                    01812 ;               dynamically modifying velocity and acceleration limits.
                    01813 ;
                    01814 ;
                    01815 ;
                    01816 ;               In velocity mode, ramp moves are performed.
                    01817 ;
                    01818 ;
                    01819 ;                                    / final velocity
                    01820 ;                                   /
                    01821 ;                                  /
                    01822 ;                                 /
                    01823 ;                                /
                    01824 ;             initial velocity  /
                    01825 ;
                    01826 ;                                0     TAU
                    01827 ;
                    01828 ;
                    01829
044F                01830 doMove
                    01831
                    01832         INC16   MOVTIME                ; increment move time
                        M
044F 290A               M         CLRF    WREG, F
0450 156A               M         INCF    MOVTIME+B0, F
0451 116B               M         ADDWFC  MOVTIME+B1, F
                        M
                    01833
```

```
0452 E5A8           01834        CALL    doPosVel                ; evaluate iterative equations
                    01835
0453 3396           01836        TSTFSZ  MODETYPE
0454 C569           01837        GOTO    vmove
0455                01838 pmove
0455 6ACB           01839        MOVFP   ONE,WREG                ; test if in phase1
0456 3199           01840        CPFSEQ  MOVFLAG
0457 C51B           01841        GOTO    phase2
0458                01842 phase1
                    01843        MVFP32  MOVDEL,MOVTMP           ; save previous discretization error
                        M
0458 7CBB               M        MOVFP   MOVDEL+B0,MOVTMP+B0     ; move A(B0) to B(B0)
0459 7DBC               M        MOVFP   MOVDEL+B1,MOVTMP+B1     ; move A(B1) to B(B1)
045A 7EBD               M        MOVFP   MOVDEL+B2,MOVTMP+B2     ; move A(B2) to B(B2)
045B 7FBE               M        MOVFP   MOVDEL+B3,MOVTMP+B3     ; move A(B3) to B(B3)
                        M
                    01844        MOV32   OPOSITION,MOVDEL        ; test if half move
                        M
045C 6A54               M        MOVFP   OPOSITION+B0,WREG       ; get byte of a into w
045D 4ABB               M        MOVPF   WREG,MOVDEL+B0          ; move to b(B0)
045E 6A55               M        MOVFP   OPOSITION+B1,WREG       ; get byte of a into w
045F 4ABC               M        MOVPF   WREG,MOVDEL+B1          ; move to b(B1)
0460 6A56               M        MOVFP   OPOSITION+B2,WREG       ; get byte of a into w
0461 4ABD               M        MOVPF   WREG,MOVDEL+B2          ; move to b(B2)
0462 6A57               M        MOVFP   OPOSITION+B3,WREG       ; get byte of a into w
0463 4ABE               M        MOVPF   WREG,MOVDEL+B3          ; move to b(B3)
                        M
                    01845        ADD32   HMOVVAL,MOVDEL
                        M
0464 6A66               M        MOVFP   HMOVVAL+B0,WREG         ; get lowest byte of a into w
0465 0FBB               M        ADDWF   MOVDEL+B0, F            ; add lowest byte of b, save in b(B0)
0466 6A67               M        MOVFP   HMOVVAL+B1,WREG         ; get 2nd byte of a into w
0467 11BC               M        ADDWFC  MOVDEL+B1, F            ; add 2nd byte of b, save in b(B1)
0468 6A68               M        MOVFP   HMOVVAL+B2,WREG         ; get 3rd byte of a into w
0469 11BD               M        ADDWFC  MOVDEL+B2, F            ; add 3rd byte of b, save in b(B2)
046A 6A69               M        MOVFP   HMOVVAL+B3,WREG         ; get 4th byte of a into w
046B 11BE               M        ADDWFC  MOVDEL+B3, F            ; add 4th byte of b, save in b(B3)
                        M
                    01846        SUB32   MOVPBUF,MOVDEL
                        M
046C 6AAF               M        MOVFP   MOVPBUF+B0,WREG         ; get lowest byte of a into w
046D 05BB               M        SUBWF   MOVDEL+B0, F            ; sub lowest byte of b, save in b(B0)
046E 6AB0               M        MOVFP   MOVPBUF+B1,WREG         ; get 2nd byte of a into w
046F 03BC               M        SUBWFB  MOVDEL+B1, F            ; sub 2nd byte of b, save in b(B1)
0470 6AB1               M        MOVFP   MOVPBUF+B2,WREG         ; get 3rd byte of a into w
0471 03BD               M        SUBWFB  MOVDEL+B2, F            ; sub 3rd byte of b, save in b(B2)
0472 6AB2               M        MOVFP   MOVPBUF+B3,WREG         ; get 4th byte of a into w
```

```
0473 03BE              M         SUBWFB  MOVDEL+B3, F          ; sub 4th byte of b, save in b(B3)
                       M
0474 976C            01847       BTFSS   MOVSIGN,MSB
0475 C47F            01848       GOTO    mpos1
                     01849       NEG32   MOVDEL
                       M
0476 13BB              M         COMF    MOVDEL+B0, F
0477 13BC              M         COMF    MOVDEL+B1, F
0478 13BD              M         COMF    MOVDEL+B2, F
0479 13BE              M         COMF    MOVDEL+B3, F
047A 290A              M         CLRF    WREG, F
047B 15BB              M         INCF    MOVDEL+B0, F
047C 11BC              M         ADDWFC  MOVDEL+B1, F
047D 11BD              M         ADDWFC  MOVDEL+B2, F
047E 11BE              M         ADDWFC  MOVDEL+B3, F
                       M
047F                 01850 mpos1
047F 97BE            01851       BTFSS   MOVDEL+B3,MSB
0480 C4E5            01852       GOTO    speedup               ; continue to speed up if in phase1
                     01853
                     01854       TFSZ16  T1                    ; if T1=0, maximum velocity not
                       M
0481 6A6D              M         MOVFP   T1+B0,WREG
0482 086E              M         IORWF   T1+B1,W
0483 330A              M         TSTFSZ  WREG
                     01855                                     ; reached, so T1=T2, otherwise T1
                     01856                                     ; has been set in speedup
0484 C4B8            01857       GOTO    t2net1
                     01858
                     01859       NEG32   A                     ; negate A for speeddown
                       M
0485 13A7              M         COMF    A+B0, F
0486 13A8              M         COMF    A+B1, F
0487 13A9              M         COMF    A+B2, F
0488 13AA              M         COMF    A+B3, F
0489 290A              M         CLRF    WREG, F
048A 15A7              M         INCF    A+B0, F
048B 11A8              M         ADDWFC  A+B1, F
048C 11A9              M         ADDWFC  A+B2, F
048D 11AA              M         ADDWFC  A+B3, F
                       M
                     01860       ADD32   MOVDEL,MOVTMP         ; test x-y < 0
                       M
048E 6ABB              M         MOVFP   MOVDEL+B0,WREG        ; get lowest byte of a into w
048F 0F1C              M         ADDWF   MOVTMP+B0, F          ; add lowest byte of b, save in b(B0)
0490 6ABC              M         MOVFP   MOVDEL+B1,WREG        ; get 2nd byte of a into w
0491 111D              M         ADDWFC  MOVTMP+B1, F          ; add 2nd byte of b, save in b(B1)
```

```
0492 6ABD              M         MOVFP   MOVDEL+B2,WREG          ; get 3rd byte of a into w
0493 111E              M         ADDWFC  MOVTMP+B2, F            ; add 3rd byte of b, save in b(B2)
0494 6ABE              M         MOVFP   MOVDEL+B3,WREG          ; get 4th byte of a into w
0495 111F              M         ADDWFC  MOVTMP+B3, F            ; add 4th byte of b, save in b(B3)
                       M
0496 971F          01861         BTFSS   MOVTMP+B3,MSB           ; if new discretization error larger,
0497 C4AE          01862         GOTO    triok                   ; backup to define T2, otherwise ok
                   01863
0498 2B6F          01864         SETF    T2+B0, F                ; set T2=-1 for backup
0499 2B70          01865         SETF    T2+B1, F
                   01866         NEG32   A                       ; negate A to undo
                       M
049A 13A7              M         COMF    A+B0, F
049B 13A8              M         COMF    A+B1, F
049C 13A9              M         COMF    A+B2, F
049D 13AA              M         COMF    A+B3, F
049E 290A              M         CLRF    WREG, F
049F 15A7              M         INCF    A+B0, F
04A0 11A8              M         ADDWFC  A+B1, F
04A1 11A9              M         ADDWFC  A+B2, F
04A2 11AA              M         ADDWFC  A+B3, F
                       M
04A3 E5CA          01867         CALL    undoPosVel
                   01868         NEG32   A                       ; negate A again for speeddown
                       M
04A4 13A7              M         COMF    A+B0, F
04A5 13A8              M         COMF    A+B1, F
04A6 13A9              M         COMF    A+B2, F
04A7 13AA              M         COMF    A+B3, F
04A8 290A              M         CLRF    WREG, F
04A9 15A7              M         INCF    A+B0, F
04AA 11A8              M         ADDWFC  A+B1, F
04AB 11A9              M         ADDWFC  A+B2, F
04AC 11AA              M         ADDWFC  A+B3, F
                       M
04AD E5A8          01869         CALL    doPosVel                ; and reevaluate iterative equations
04AE               01870 triok
                   01871         ADD16   MOVTIME,T2              ; add time to T2
                       M
04AE 6A6A              M         MOVFP   MOVTIME+B0,WREG         ; get lowest byte of a into w
04AF 0F6F              M         ADDWF   T2+B0, F                ; add lowest byte of b, save in b(B0)
04B0 6A6B              M         MOVFP   MOVTIME+B1,WREG         ; get 2nd byte of a into w
04B1 1170              M         ADDWFC  T2+B1, F                ; add 2nd byte of b, save in b(B1)
                       M
                   01872         MOV16   T2,T1
                       M
04B2 6A6F              M         MOVFP   T2+B0,WREG              ; get byte of a into w
```

```
04B3 4A6D              M         MOVPF   WREG,T1+B0              ; move to b(B0)
04B4 6A70              M         MOVFP   T2+B1,WREG              ; get byte of a into w
04B5 4A6E              M         MOVPF   WREG,T1+B1              ; move to b(B1)
                       M
04B6 1599          01873         INCF    MOVFLAG, F              ; increment move flag for phase2
04B7 C50E          01874         GOTO    mvok                    ; execute last phase1 move
                   01875
04B8               01876 t2net1
04B8 2B6F          01877         SETF    T2+B0, F                ; set T2=-1 for backup
04B9 2B70          01878         SETF    T2+B1, F
                   01879         ADD16   MOVTIME,T2              ; add time to T2
                       M
04BA 6A6A              M         MOVFP   MOVTIME+B0,WREG         ; get lowest byte of a into w
04BB 0F6F              M         ADDWF   T2+B0, F                ; add lowest byte of b, save in b(B0)
04BC 6A6B              M         MOVFP   MOVTIME+B1,WREG         ; get 2nd byte of a into w
04BD 1170              M         ADDWFC  T2+B1, F                ; add 2nd byte of b, save in b(B1)
                       M
                   01880
                   01881         MVFP32  MOVTMP,TMP              ; test if 3x-y < 0
                       M
04BE 781C              M         MOVFP   MOVTMP+B0,TMP+B0        ; move A(B0) to B(B0)
04BF 791D              M         MOVFP   MOVTMP+B1,TMP+B1        ; move A(B1) to B(B1)
04C0 7A1E              M         MOVFP   MOVTMP+B2,TMP+B2        ; move A(B2) to B(B2)
04C1 7B1F              M         MOVFP   MOVTMP+B3,TMP+B3        ; move A(B3) to B(B3)
                       M
                   01882         RLC32   MOVTMP
                       M
04C2 8804              M         BCF     _carry
04C3 1B1C              M         RLCF    MOVTMP+B0, F
04C4 1B1D              M         RLCF    MOVTMP+B1, F
04C5 1B1E              M         RLCF    MOVTMP+B2, F
04C6 1B1F              M         RLCF    MOVTMP+B3, F
                       M
                   01883         ADD32   TMP,MOVTMP
                       M
04C7 6A18              M         MOVFP   TMP+B0,WREG             ; get lowest byte of a into w
04C8 0F1C              M         ADDWF   MOVTMP+B0, F            ; add lowest byte of b, save in b(B0)
04C9 6A19              M         MOVFP   TMP+B1,WREG             ; get 2nd byte of a into w
04CA 111D              M         ADDWFC  MOVTMP+B1, F            ; add 2nd byte of b, save in b(B1)
04CB 6A1A              M         MOVFP   TMP+B2,WREG             ; get 3rd byte of a into w
04CC 111E              M         ADDWFC  MOVTMP+B2, F            ; add 3rd byte of b, save in b(B2)
04CD 6A1B              M         MOVFP   TMP+B3,WREG             ; get 4th byte of a into w
04CE 111F              M         ADDWFC  MOVTMP+B3, F            ; add 4th byte of b, save in b(B3)
                       M
                   01884         ADD32   MOVDEL,MOVTMP
                       M
04CF 6ABB              M         MOVFP   MOVDEL+B0,WREG          ; get lowest byte of a into w
```

```
04D0 0F1C               M         ADDWF   MOVTMP+B0, F          ; add lowest byte of b, save in b(B0)
04D1 6ABC               M         MOVFP   MOVDEL+B1,WREG        ; get 2nd byte of a into w
04D2 111D               M         ADDWFC  MOVTMP+B1, F          ; add 2nd byte of b, save in b(B1)
04D3 6ABD               M         MOVFP   MOVDEL+B2,WREG        ; get 3rd byte of a into w
04D4 111E               M         ADDWFC  MOVTMP+B2, F          ; add 3rd byte of b, save in b(B2)
04D5 6ABE               M         MOVFP   MOVDEL+B3,WREG        ; get 4th byte of a into w
04D6 111F               M         ADDWFC  MOVTMP+B3, F          ; add 4th byte of b, save in b(B3)
                        M
04D7 971F            01885        BTFSS   MOVTMP+B3,MSB         ; if new discretization error larger,
04D8 C4DB            01886        GOTO    trapok                ; take one more flat step
04D9 2BBF            01887        SETF    PH2FLAT+B0, F
04DA 2BC0            01888        SETF    PH2FLAT+B1, F
04DB                 01889 trapok
                     01890        ADD16   T2,PH2FLAT
                        M
04DB 6A6F               M         MOVFP   T2+B0,WREG            ; get lowest byte of a into w
04DC 0FBF               M         ADDWF   PH2FLAT+B0, F         ; add lowest byte of b, save in b(B0)
04DD 6A70               M         MOVFP   T2+B1,WREG            ; get 2nd byte of a into w
04DE 11C0               M         ADDWFC  PH2FLAT+B1, F         ; add 2nd byte of b, save in b(B1)
                        M
                     01891        SUB16   T1,PH2FLAT
                        M
04DF 6A6D               M         MOVFP   T1+B0,WREG            ; get lowest byte of a into w
04E0 05BF               M         SUBWF   PH2FLAT+B0, F         ; sub lowest byte of b, save in b(B0)
04E1 6A6E               M         MOVFP   T1+B1,WREG            ; get 2nd byte of a into w
04E2 03C0               M         SUBWFB  PH2FLAT+B1, F         ; sub 2nd byte of b, save in b(B1)
                        M
04E3 1599            01892        INCF    MOVFLAG, F            ; increment move flag for phase2
04E4 C50E            01893        GOTO    mvok                  ; execute last phase1 move
                     01894
04E5                 01895 speedup
                     01896        MVFP32  V,MOVTMP              ; test if maximum velocity reached
                        M
04E5 7CAB               M         MOVFP   V+B0,MOVTMP+B0        ; move A(B0) to B(B0)
04E6 7DAC               M         MOVFP   V+B1,MOVTMP+B1        ; move A(B1) to B(B1)
04E7 7EAD               M         MOVFP   V+B2,MOVTMP+B2        ; move A(B2) to B(B2)
04E8 7FAE               M         MOVFP   V+B3,MOVTMP+B3        ; move A(B3) to B(B3)
                        M
                     01897        SUB32   MOVVBUF,MOVTMP
                        M
04E9 6AB3               M         MOVFP   MOVVBUF+B0,WREG       ; get lowest byte of a into w
04EA 051C               M         SUBWF   MOVTMP+B0, F          ; sub lowest byte of b, save in b(B0)
04EB 6AB4               M         MOVFP   MOVVBUF+B1,WREG       ; get 2nd byte of a into w
04EC 031D               M         SUBWFB  MOVTMP+B1, F          ; sub 2nd byte of b, save in b(B1)
04ED 6AB5               M         MOVFP   MOVVBUF+B2,WREG       ; get 3rd byte of a into w
04EE 031E               M         SUBWFB  MOVTMP+B2, F          ; sub 3rd byte of b, save in b(B2)
04EF 6AB6               M         MOVFP   MOVVBUF+B3,WREG       ; get 4th byte of a into w
```

```
04F0 031F              M         SUBWFB  MOVTMP+B3, F           ; sub 4th byte of b, save in b(B3)
                       M
04F1 976C            01898       BTFSS   MOVSIGN,MSB
04F2 C4FC            01899       GOTO    mpos
                     01900       NEG32   MOVTMP
                       M
04F3 131C              M         COMF    MOVTMP+B0, F
04F4 131D              M         COMF    MOVTMP+B1, F
04F5 131E              M         COMF    MOVTMP+B2, F
04F6 131F              M         COMF    MOVTMP+B3, F
04F7 290A              M         CLRF    WREG, F
04F8 151C              M         INCF    MOVTMP+B0, F
04F9 111D              M         ADDWFC  MOVTMP+B1, F
04FA 111E              M         ADDWFC  MOVTMP+B2, F
04FB 111F              M         ADDWFC  MOVTMP+B3, F
                       M
04FC                 01901 mpos
04FC 971F            01902       BTFSS   MOVTMP+B3,MSB
04FD C50E            01903       GOTO    mvok                   ; if not, execute move
                     01904
                     01905       TFSZ16  T1                     ; if so, check to see if T1 has
                       M
04FE 6A6D              M         MOVFP   T1+B0,WREG
04FF 086E              M         IORWF   T1+B1,W
0500 330A              M         TSTFSZ  WREG
                     01906                                      ; already been set
0501 C50E            01907       GOTO    mvok
0502 E5CA            01908       CALL    undoPosVel             ; if not, backup and redo iterative
                     01909       CLR32   A                      ; equations, resulting in an actual
0503 29A7              M         CLRF    A+B0, F
0504 29A8              M         CLRF    A+B1, F
0505 29A9              M         CLRF    A+B2, F
0506 29AA              M         CLRF    A+B3, F
                       M
0507 E5A8            01910       CALL    doPosVel               ; maximum speed <= VL
0508 2B6D            01911       SETF    T1+B0, F               ; evaluate T1
0509 2B6E            01912       SETF    T1+B1, F
                     01913       ADD16   MOVTIME,T1
                       M
050A 6A6A              M         MOVFP   MOVTIME+B0,WREG        ; get lowest byte of a into w
050B 0F6D              M         ADDWF   T1+B0, F               ; add lowest byte of b, save in b(B0)
050C 6A6B              M         MOVFP   MOVTIME+B1,WREG        ; get 2nd byte of a into w
050D 116E              M         ADDWFC  T1+B1, F               ; add 2nd byte of b, save in b(B1)
                       M
050E                 01914 mvok
                     01915       MOV24   MOVPBUF+B1,POSITION    ; move Q8 calculated position to Q0 commanded position
                       M
```

```
050E 6AB0          M         MOVFP   MOVPBUF+B1+B0,WREG       ; get byte of a into w
050F 4A58          M         MOVPF   WREG,POSITION+B0         ; move to b(B0)
0510 6AB1          M         MOVFP   MOVPBUF+B1+B1,WREG       ; get byte of a into w
0511 4A59          M         MOVPF   WREG,POSITION+B1         ; move to b(B1)
0512 6AB2          M         MOVFP   MOVPBUF+B1+B2,WREG       ; get byte of a into w
0513 4A5A          M         MOVPF   WREG,POSITION+B2         ; move to b(B2)
                   M
                             MOV24   MOVVBUF+B0,VELOCITY      ; move Q0 calculated velocity to Q0 commanded velocity
                   M
0514 6AB3          M         MOVFP   MOVVBUF+B0+B0,WREG       ; get byte of a into w
0515 4A5B          M         MOVPF   WREG,VELOCITY+B0         ; move to b(B0)
0516 6AB4          M         MOVFP   MOVVBUF+B0+B1,WREG       ; get byte of a into w
0517 4A5C          M         MOVPF   WREG,VELOCITY+B1         ; move to b(B1)
0518 6AB5          M         MOVFP   MOVVBUF+B0+B2,WREG       ; get byte of a into w
0519 4A5D          M         MOVPF   WREG,VELOCITY+B2         ; move to b(B2)
                   M
051A 0002                    RETURN


051B                   phase2
                             TFSZ16  PH2FLAT                  ; is flat section finished?
                   M
051B 6ABF          M         MOVFP   PH2FLAT+B0,WREG
051C 08C0          M         IORWF   PH2FLAT+B1,W
051D 330A          M         TSTFSZ  WREG
051E C53F                    GOTO    flat

                             TFSZ32  MOVVBUF                  ; is velocity zero?
                   M
051F 6AB3          M         MOVFP   MOVVBUF+B0,WREG
0520 08B4          M         IORWF   MOVVBUF+B1,W
0521 08B5          M         IORWF   MOVVBUF+B2,W
0522 08B6          M         IORWF   MOVVBUF+B3,W
0523 330A          M         TSTFSZ  WREG
0524 C55C                    GOTO    mready                   ; if not, execute move

0525 2999                    CLRF    MOVFLAG, F               ; if so, clear MOVFLAG
0526 8E98                    BCF     MOVSTAT,BIT6             ; clear motion status flag
0527 8D98                    BCF     MOVSTAT,BIT5             ; clear move in progress flag
                             CLR32   A                        ; set zero velocity and acceleration,
0528 29A7          M         CLRF    A+B0, F
0529 29A8          M         CLRF    A+B1, F
052A 29A9          M         CLRF    A+B2, F
052B 29AA          M         CLRF    A+B3, F
                   M
                             MOV16   MOVTIME,TAU
                   M
```

```
052C 6A6A              M         MOVFP   MOVTIME+B0,WREG         ; get byte of a into w
052D 4A71              M         MOVPF   WREG,TAU+B0             ; move to b(B0)
052E 6A6B              M         MOVFP   MOVTIME+B1,WREG         ; get byte of a into w
052F 4A72              M         MOVPF   WREG,TAU+B1             ; move to b(B1)
                       M
                   01932         MOV32   OPOSITION,MOVPBUF       ; execute last move to P(0)+MOVVAL
                       M
0530 6A54              M         MOVFP   OPOSITION+B0,WREG       ; get byte of a into w
0531 4AAF              M         MOVPF   WREG,MOVPBUF+B0         ; move to b(B0)
0532 6A55              M         MOVFP   OPOSITION+B1,WREG       ; get byte of a into w
0533 4AB0              M         MOVPF   WREG,MOVPBUF+B1         ; move to b(B1)
0534 6A56              M         MOVFP   OPOSITION+B2,WREG       ; get byte of a into w
0535 4AB1              M         MOVPF   WREG,MOVPBUF+B2         ; move to b(B2)
0536 6A57              M         MOVFP   OPOSITION+B3,WREG       ; get byte of a into w
0537 4AB2              M         MOVPF   WREG,MOVPBUF+B3         ; move to b(B3)
                       M
                   01933         ADD24   MOVVAL,MOVPBUF+B1
                       M
0538 6A62              M         MOVFP   MOVVAL+B0,WREG          ; get lowest byte of a into w
0539 0FB0              M         ADDWF   MOVPBUF+B1+B0, F        ; add lowest byte of b, save in b(B0)
053A 6A63              M         MOVFP   MOVVAL+B1,WREG          ; get 2nd byte of a into w
053B 11B1              M         ADDWFC  MOVPBUF+B1+B1, F        ; add 2nd byte of b, save in b(B1)
053C 6A64              M         MOVFP   MOVVAL+B2,WREG          ; get 3rd byte of a into w
053D 11B2              M         ADDWFC  MOVPBUF+B1+B2, F        ; add 3rd byte of b, save in b(B2)
                       M
053E C55C          01934         GOTO    mready
                   01935
053F               01936 flat
053F 2B1C          01937         SETF    MOVTMP+B0, F
0540 2B1D          01938         SETF    MOVTMP+B1, F
                   01939         ADD16   MOVTMP,PH2FLAT          ; decrement by one use DEC16
                       M
0541 6A1C              M         MOVFP   MOVTMP+B0,WREG          ; get lowest byte of a into w
0542 0FBF              M         ADDWF   PH2FLAT+B0, F           ; add lowest byte of b, save in b(B0)
0543 6A1D              M         MOVFP   MOVTMP+B1,WREG          ; get 2nd byte of a into w
0544 11C0              M         ADDWFC  PH2FLAT+B1, F           ; add 2nd byte of b, save in b(B1)
                       M
                   01940
                   01941         TFSZ16  PH2FLAT
                       M
0545 6ABF              M         MOVFP   PH2FLAT+B0,WREG
0546 08C0              M         IORWF   PH2FLAT+B1,W
0547 330A              M         TSTFSZ  WREG
0548 C55C          01942         GOTO    mready
                   01943
0549 29AA          01944         CLRF    A+B3, F                 ; begin speed down section
                   01945         MOV24   AL,A
```

```
                        M
054A 6A23               M           MOVFP   AL+B0,WREG              ; get byte of a into w
054B 4AA7               M           MOVPF   WREG,A+B0               ; move to b(B0)
054C 6A24               M           MOVFP   AL+B1,WREG              ; get byte of a into w
054D 4AA8               M           MOVPF   WREG,A+B1               ; move to b(B1)
054E 6A25               M           MOVFP   AL+B2,WREG              ; get byte of a into w
054F 4AA9               M           MOVPF   WREG,A+B2               ; move to b(B2)
                        M
0550 290A             01946         CLRF    WREG, F
0551 316C             01947         CPFSEQ  MOVSIGN
0552 C55C             01948         GOTO    mready
                      01949         NEG32   A
                        M
0553 13A7               M           COMF    A+B0, F
0554 13A8               M           COMF    A+B1, F
0555 13A9               M           COMF    A+B2, F
0556 13AA               M           COMF    A+B3, F
0557 290A               M           CLRF    WREG, F
0558 15A7               M           INCF    A+B0, F
0559 11A8               M           ADDWFC  A+B1, F
055A 11A9               M           ADDWFC  A+B2, F
055B 11AA               M           ADDWFC  A+B3, F
                        M
                      01950
055C                  01951 mready
                      01952         MOV24   MOVPBUF+B1,POSITION
                        M
055C 6AB0               M           MOVFP   MOVPBUF+B1+B0,WREG      ; get byte of a into w
055D 4A58               M           MOVPF   WREG,POSITION+B0        ; move to b(B0)
055E 6AB1               M           MOVFP   MOVPBUF+B1+B1,WREG      ; get byte of a into w
055F 4A59               M           MOVPF   WREG,POSITION+B1        ; move to b(B1)
0560 6AB2               M           MOVFP   MOVPBUF+B1+B2,WREG      ; get byte of a into w
0561 4A5A               M           MOVPF   WREG,POSITION+B2        ; move to b(B2)
                        M
                      01953         MOV24   MOVVBUF+B0,VELOCITY
                        M
0562 6AB3               M           MOVFP   MOVVBUF+B0+B0,WREG      ; get byte of a into w
0563 4A5B               M           MOVPF   WREG,VELOCITY+B0        ; move to b(B0)
0564 6AB4               M           MOVFP   MOVVBUF+B0+B1,WREG      ; get byte of a into w
0565 4A5C               M           MOVPF   WREG,VELOCITY+B1        ; move to b(B1)
0566 6AB5               M           MOVFP   MOVVBUF+B0+B2,WREG      ; get byte of a into w
0567 4A5D               M           MOVPF   WREG,VELOCITY+B2        ; move to b(B2)
                        M
0568 0002             01954         RETURN
                      01955
0569                  01956 vmove
                      01957         MVFP32  MOVVAL,MOVTMP           ; test if final velocity reached
```

```
                        M
0569 7C62               M         MOVFP   MOVVAL+B0,MOVTMP+B0     ; move A(B0) to B(B0)
056A 7D63               M         MOVFP   MOVVAL+B1,MOVTMP+B1     ; move A(B1) to B(B1)
056B 7E64               M         MOVFP   MOVVAL+B2,MOVTMP+B2     ; move A(B2) to B(B2)
056C 7F65               M         MOVFP   MOVVAL+B3,MOVTMP+B3     ; move A(B3) to B(B3)
                        M
                    01958         SUB32   MOVVBUF,MOVTMP
                        M
056D 6AB3               M         MOVFP   MOVVBUF+B0,WREG         ; get lowest byte of a into w
056E 051C               M         SUBWF   MOVTMP+B0, F            ; sub lowest byte of b, save in b(B0)
056F 6AB4               M         MOVFP   MOVVBUF+B1,WREG         ; get 2nd byte of a into w
0570 031D               M         SUBWFB  MOVTMP+B1, F            ; sub 2nd byte of b, save in b(B1)
0571 6AB5               M         MOVFP   MOVVBUF+B2,WREG         ; get 3rd byte of a into w
0572 031E               M         SUBWFB  MOVTMP+B2, F            ; sub 3rd byte of b, save in b(B2)
0573 6AB6               M         MOVFP   MOVVBUF+B3,WREG         ; get 4th byte of a into w
0574 031F               M         SUBWFB  MOVTMP+B3, F            ; sub 4th byte of b, save in b(B3)
                        M
0575 976C           01959         BTFSS   MOVSIGN,MSB
0576 C580           01960         GOTO    vmpos
                    01961         NEG32   MOVTMP
                        M
0577 131C               M         COMF    MOVTMP+B0, F
0578 131D               M         COMF    MOVTMP+B1, F
0579 131E               M         COMF    MOVTMP+B2, F
057A 131F               M         COMF    MOVTMP+B3, F
057B 290A               M         CLRF    WREG, F
057C 151C               M         INCF    MOVTMP+B0, F
057D 111D               M         ADDWFC  MOVTMP+B1, F
057E 111E               M         ADDWFC  MOVTMP+B2, F
057F 111F               M         ADDWFC  MOVTMP+B3, F
                        M
0580                01962 vmpos
0580 971F           01963         BTFSS   MOVTMP+B3,MSB
0581 C59B           01964         GOTO    vmoveok                 ; if not, continue
                    01965
                    01966         CLR32   A                       ; if so, set A=0 and continue with
0582 29A7               M         CLRF    A+B0, F
0583 29A8               M         CLRF    A+B1, F
0584 29A9               M         CLRF    A+B2, F
0585 29AA               M         CLRF    A+B3, F
                        M
                    01967         MOV32   MOVVAL,MOVVBUF          ; move unless the final velocity
                        M
0586 6A62               M         MOVFP   MOVVAL+B0,WREG          ; get byte of a into w
0587 4AB3               M         MOVPF   WREG,MOVVBUF+B0         ; move to b(B0)
0588 6A63               M         MOVFP   MOVVAL+B1,WREG          ; get byte of a into w
0589 4AB4               M         MOVPF   WREG,MOVVBUF+B1         ; move to b(B1)
```

```
058A 6A64               M         MOVFP   MOVVAL+B2,WREG          ; get byte of a into w
058B 4AB5               M         MOVPF   WREG,MOVVBUF+B2         ; move to b(B2)
058C 6A65               M         MOVFP   MOVVAL+B3,WREG          ; get byte of a into w
058D 4AB6               M         MOVPF   WREG,MOVVBUF+B3         ; move to b(B3)
                        M
                    01968                                         ; is zero.
058E 2999           01969         CLRF    MOVFLAG, F              ; clear MOVFLAG
058F 8D98           01970         BCF     MOVSTAT,BIT5            ; clear move in progress flag
                    01971         MOV16   MOVTIME,TAU
                        M
0590 6A6A               M         MOVFP   MOVTIME+B0,WREG         ; get byte of a into w
0591 4A71               M         MOVPF   WREG,TAU+B0             ; move to b(B0)
0592 6A6B               M         MOVFP   MOVTIME+B1,WREG         ; get byte of a into w
0593 4A72               M         MOVPF   WREG,TAU+B1             ; move to b(B1)
                        M
                    01972         TFSZ32  MOVVAL
                        M
0594 6A62               M         MOVFP   MOVVAL+B0,WREG
0595 0863               M         IORWF   MOVVAL+B1,W
0596 0864               M         IORWF   MOVVAL+B2,W
0597 0865               M         IORWF   MOVVAL+B3,W
0598 330A               M         TSTFSZ  WREG
0599 C59B           01973         GOTO    vmoveok
                    01974
059A 8E98           01975         BCF     MOVSTAT,BIT6            ; if final velocity is zero, clear
                    01976                                         ; motion status flag
059B                01977 vmoveok
                    01978         MOV24   MOVPBUF+B1,POSITION
                        M
059B 6AB0               M         MOVFP   MOVPBUF+B1+B0,WREG      ; get byte of a into w
059C 4A58               M         MOVPF   WREG,POSITION+B0        ; move to b(B0)
059D 6AB1               M         MOVFP   MOVPBUF+B1+B1,WREG      ; get byte of a into w
059E 4A59               M         MOVPF   WREG,POSITION+B1        ; move to b(B1)
059F 6AB2               M         MOVFP   MOVPBUF+B1+B2,WREG      ; get byte of a into w
05A0 4A5A               M         MOVPF   WREG,POSITION+B2        ; move to b(B2)
                        M
                    01979         MOV24   MOVVBUF+B0,VELOCITY
                        M
05A1 6AB3               M         MOVFP   MOVVBUF+B0+B0,WREG      ; get byte of a into w
05A2 4A5B               M         MOVPF   WREG,VELOCITY+B0        ; move to b(B0)
05A3 6AB4               M         MOVFP   MOVVBUF+B0+B1,WREG      ; get byte of a into w
05A4 4A5C               M         MOVPF   WREG,VELOCITY+B1        ; move to b(B1)
05A5 6AB5               M         MOVFP   MOVVBUF+B0+B2,WREG      ; get byte of a into w
05A6 4A5D               M         MOVPF   WREG,VELOCITY+B2        ; move to b(B2)
                        M
05A7 0002           01980         RETURN
                    01981
```

```
                    01982 ;*****************************************************************************
                    01983
                    01984 ;*****************************************************************************
                    01985 ; NAME:         doPosVel
                    01986 ;
                    01987 ; DESCRIPTION:  Evaluates the iterative equations for trapezoidal move
                    01988 ;               generation
                    01989 ;
                    01990 ;               V(k)=V(k-1)+A,          P(k)=P(k-1)+V(k-1)+A/2,
                    01991 ;
                    01992 ;               where abs(A)={AL,0} depending on the region of the trapezoid
                    01993 ;               being executed.
                    01994 ;
                    01995
05A8                01996 doPosVel
                    01997
                    01998         ADD32   MOVVBUF,MOVPBUF         ; P(k-1)+V(k-1)
                        M
05A8 6AB3               M         MOVFP   MOVVBUF+B0,WREG         ; get lowest byte of a into w
05A9 0FAF               M         ADDWF   MOVPBUF+B0, F           ; add lowest byte of b, save in b(B0)
05AA 6AB4               M         MOVFP   MOVVBUF+B1,WREG         ; get 2nd byte of a into w
05AB 11B0               M         ADDWFC  MOVPBUF+B1, F           ; add 2nd byte of b, save in b(B1)
05AC 6AB5               M         MOVFP   MOVVBUF+B2,WREG         ; get 3rd byte of a into w
05AD 11B1               M         ADDWFC  MOVPBUF+B2, F           ; add 3rd byte of b, save in b(B2)
05AE 6AB6               M         MOVFP   MOVVBUF+B3,WREG         ; get 4th byte of a into w
05AF 11B2               M         ADDWFC  MOVPBUF+B3, F           ; add 4th byte of b, save in b(B3)
                        M
                    01999         ADD32   A,MOVVBUF               ; V(k)=V(k-1)+A
                        M
05B0 6AA7               M         MOVFP   A+B0,WREG               ; get lowest byte of a into w
05B1 0FB3               M         ADDWF   MOVVBUF+B0, F           ; add lowest byte of b, save in b(B0)
05B2 6AA8               M         MOVFP   A+B1,WREG               ; get 2nd byte of a into w
05B3 11B4               M         ADDWFC  MOVVBUF+B1, F           ; add 2nd byte of b, save in b(B1)
05B4 6AA9               M         MOVFP   A+B2,WREG               ; get 3rd byte of a into w
05B5 11B5               M         ADDWFC  MOVVBUF+B2, F           ; add 3rd byte of b, save in b(B2)
05B6 6AAA               M         MOVFP   A+B3,WREG               ; get 4th byte of a into w
05B7 11B6               M         ADDWFC  MOVVBUF+B3, F           ; add 4th byte of b, save in b(B3)
                        M
                    02000
                    02001         MVFP32  A,MOVTMP                ; compute A/2
                        M
05B8 7CA7               M         MOVFP   A+B0,MOVTMP+B0          ; move A(B0) to B(B0)
05B9 7DA8               M         MOVFP   A+B1,MOVTMP+B1          ; move A(B1) to B(B1)
05BA 7EA9               M         MOVFP   A+B2,MOVTMP+B2          ; move A(B2) to B(B2)
05BB 7FAA               M         MOVFP   A+B3,MOVTMP+B3          ; move A(B3) to B(B3)
                        M
                    02002         RRC32   MOVTMP
```

```
                  M
05BC 1A1F         M         RLCF    MOVTMP+B3,W             ; move sign into carry bit
05BD 191F         M         RRCF    MOVTMP+B3, F
05BE 191E         M         RRCF    MOVTMP+B2, F
05BF 191D         M         RRCF    MOVTMP+B1, F
05C0 191C         M         RRCF    MOVTMP+B0, F
                  M

                            ADD32   MOVTMP,MOVPBUF          ; P(k)=P(k-1)+V(k-1)+A/2,
                  M
05C1 6A1C         M         MOVFP   MOVTMP+B0,WREG          ; get lowest byte of a into w
05C2 0FAF         M         ADDWF   MOVPBUF+B0, F           ; add lowest byte of b, save in b(B0)
05C3 6A1D         M         MOVFP   MOVTMP+B1,WREG          ; get 2nd byte of a into w
05C4 11B0         M         ADDWFC  MOVPBUF+B1, F           ; add 2nd byte of b, save in b(B1)
05C5 6A1E         M         MOVFP   MOVTMP+B2,WREG          ; get 3rd byte of a into w
05C6 11B1         M         ADDWFC  MOVPBUF+B2, F           ; add 3rd byte of b, save in b(B2)
05C7 6A1F         M         MOVFP   MOVTMP+B3,WREG          ; get 4th byte of a into w
05C8 11B2         M         ADDWFC  MOVPBUF+B3, F           ; add 4th byte of b, save in b(B3)
                  M

05C9 0002                   RETURN

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

                   ;*****************************************************************************
                   ; NAME: undoPosVel
                   ;
                   ; DESCRIPTION:  Backward iteration of the equations for trapezoidal move
                   ;               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 trapezoid
                   ;               being executed. This routine is used to reverse a step about
                   ;               to be made beyond a decision point.
                   ;

05CA               undoPosVel

                            SUB32   A,MOVVBUF               ; V(k-1)=V(k)-A
                  M
05CA 6AA7         M         MOVFP   A+B0,WREG               ; get lowest byte of a into w
05CB 05B3         M         SUBWF   MOVVBUF+B0, F           ; sub lowest byte of b, save in b(B0)
05CC 6AA8         M         MOVFP   A+B1,WREG               ; get 2nd byte of a into w
05CD 03B4         M         SUBWFB  MOVVBUF+B1, F           ; sub 2nd byte of b, save in b(B1)
05CE 6AA9         M         MOVFP   A+B2,WREG               ; get 3rd byte of a into w
05CF 03B5         M         SUBWFB  MOVVBUF+B2, F           ; sub 3rd byte of b, save in b(B2)
```

```
05D0 6AAA               M         MOVFP   A+B3,WREG              ; get 4th byte of a into w
05D1 03B6               M         SUBWFB  MOVVBUF+B3, F          ; sub 4th byte of b, save in b(B3)
                        M
                    02026         SUB32   MOVVBUF,MOVPBUF        ; P(k)-V(k-1)
                        M
05D2 6AB3               M         MOVFP   MOVVBUF+B0,WREG        ; get lowest byte of a into w
05D3 05AF               M         SUBWF   MOVPBUF+B0, F          ; sub lowest byte of b, save in b(B0)
05D4 6AB4               M         MOVFP   MOVVBUF+B1,WREG        ; get 2nd byte of a into w
05D5 03B0               M         SUBWFB  MOVPBUF+B1, F          ; sub 2nd byte of b, save in b(B1)
05D6 6AB5               M         MOVFP   MOVVBUF+B2,WREG        ; get 3rd byte of a into w
05D7 03B1               M         SUBWFB  MOVPBUF+B2, F          ; sub 3rd byte of b, save in b(B2)
05D8 6AB6               M         MOVFP   MOVVBUF+B3,WREG        ; get 4th byte of a into w
05D9 03B2               M         SUBWFB  MOVPBUF+B3, F          ; sub 4th byte of b, save in b(B3)
                        M
                    02027
                    02028         MVFP32  A,MOVTMP               ; compute A/2
                        M
05DA 7CA7               M         MOVFP   A+B0,MOVTMP+B0         ; move A(B0) to B(B0)
05DB 7DA8               M         MOVFP   A+B1,MOVTMP+B1         ; move A(B1) to B(B1)
05DC 7EA9               M         MOVFP   A+B2,MOVTMP+B2         ; move A(B2) to B(B2)
05DD 7FAA               M         MOVFP   A+B3,MOVTMP+B3         ; move A(B3) to B(B3)
                        M
                    02029         RRC32   MOVTMP
                        M
05DE 1A1F               M         RLCF    MOVTMP+B3,W            ; move sign into carry bit
05DF 191F               M         RRCF    MOVTMP+B3, F
05E0 191E               M         RRCF    MOVTMP+B2, F
05E1 191D               M         RRCF    MOVTMP+B1, F
05E2 191C               M         RRCF    MOVTMP+B0, F
                        M
                    02030
                    02031         SUB32   MOVTMP,MOVPBUF         ; P(k-1)=P(k)-V(k-1)-A/2,
                        M
05E3 6A1C               M         MOVFP   MOVTMP+B0,WREG         ; get lowest byte of a into w
05E4 05AF               M         SUBWF   MOVPBUF+B0, F          ; sub lowest byte of b, save in b(B0)
05E5 6A1D               M         MOVFP   MOVTMP+B1,WREG         ; get 2nd byte of a into w
05E6 03B0               M         SUBWFB  MOVPBUF+B1, F          ; sub 2nd byte of b, save in b(B1)
05E7 6A1E               M         MOVFP   MOVTMP+B2,WREG         ; get 3rd byte of a into w
05E8 03B1               M         SUBWFB  MOVPBUF+B2, F          ; sub 3rd byte of b, save in b(B2)
05E9 6A1F               M         MOVFP   MOVTMP+B3,WREG         ; get 4th byte of a into w
05EA 03B2               M         SUBWFB  MOVPBUF+B3, F          ; sub 4th byte of b, save in b(B3)
                        M
                    02032
05EB 0002           02033         RETURN
                    02034
                    02035 ;*******************************************************************************
                    02036
```

```
                    02037 ;***************************************************************************
                    02038 ; NAME:         doMPosMVel
                    02039 ;
                    02040 ; DESCRIPTION:  Calculates current position from UpCount and DownCount
                    02041 ;
                    02042
05EC                02043 doMPosMVel
                    02044
                    02045 ; Do UpCounter first
                    02046
                    02047         MVFP16  UPCOUNT,TMP+B0          ; save old upcount
                        M
05EC 78B7               M         MOVFP   UPCOUNT+B0,TMP+B0+B0    ; move A(B0) to B(B0)
05ED 79B8               M         MOVFP   UPCOUNT+B1,TMP+B0+B1    ; move A(B1) to B(B1)
                        M
05EE                02048 readUp
05EE 4C0A           02049         MOVPF   TMR0H,WREG
05EF 4BB7           02050         MOVPF   TMR0L,UPCOUNT+B0
05F0 310C           02051         CPFSEQ  TMR0H                   ; Skip next if HI hasn't changed
05F1 C5EE           02052         GOTO    readUp                  ; HI changed, re-read LO
05F2 4AB8           02053         MOVPF   WREG,UPCOUNT+B1         ; OK to store HI now
                    02054
05F3 2978           02055         CLRF    MVELOCITY+B0, F         ; clear bits below binary point
                    02056
                    02057         MOV16   UPCOUNT,MVELOCITY+B1    ; compute upcount increment
                        M
05F4 6AB7               M         MOVFP   UPCOUNT+B0,WREG         ; get byte of a into w
05F5 4A79               M         MOVPF   WREG,MVELOCITY+B1+B0    ; move to b(B0)
05F6 6AB8               M         MOVFP   UPCOUNT+B1,WREG         ; get byte of a into w
05F7 4A7A               M         MOVPF   WREG,MVELOCITY+B1+B1    ; move to b(B1)
                        M
                    02058         SUB16   TMP+B0,MVELOCITY+B1
                        M
05F8 6A18               M         MOVFP   TMP+B0+B0,WREG          ; get lowest byte of a into w
05F9 0579               M         SUBWF   MVELOCITY+B1+B0, F      ; sub lowest byte of b, save in b(B0)
05FA 6A19               M         MOVFP   TMP+B0+B1,WREG          ; get 2nd byte of a into w
05FB 037A               M         SUBWFB  MVELOCITY+B1+B1, F      ; sub 2nd byte of b, save in b(B1)
                        M
                    02059
                    02060 ; Now do DownCounter
                    02061
                    02062         MVFP16  DOWNCOUNT,TMP+B0        ; save old downcount
                        M
05FC 78B9               M         MOVFP   DOWNCOUNT+B0,TMP+B0+B0  ; move A(B0) to B(B0)
05FD 79BA               M         MOVFP   DOWNCOUNT+B1,TMP+B0+B1  ; move A(B1) to B(B1)
                        M
05FE                02063 readDown
```

```
05FE B802          02064         MOVLB   BANK2                   ; timers in Bank 2
05FF 530A          02065         MOVPF   TMR3H,WREG
0600 52B9          02066         MOVPF   TMR3L,DOWNCOUNT+B0
0601 3113          02067         CPFSEQ  TMR3H                   ; Skip next if HI hasn't changed
0602 C5FE          02068         GOTO    readDown                ; HI changed, re-read LO
0603 4ABA          02069         MOVPF   WREG,DOWNCOUNT+B1       ; OK to store HI now
                   02070
                   02071         MVFP16  DOWNCOUNT+B0,TMP+B2     ; compute downcount increment
                       M
0604 7AB9              M         MOVFP   DOWNCOUNT+B0+B0,TMP+B2+B0 ; move A(B0) to B(B0)
0605 7BBA              M         MOVFP   DOWNCOUNT+B0+B1,TMP+B2+B1 ; move A(B1) to B(B1)
                       M
                   02072         SUB16   TMP+B0,TMP+B2
                       M
0606 6A18              M         MOVFP   TMP+B0+B0,WREG          ; get lowest byte of a into w
0607 051A              M         SUBWF   TMP+B2+B0, F            ; sub lowest byte of b, save in b(B0)
0608 6A19              M         MOVFP   TMP+B0+B1,WREG          ; get 2nd byte of a into w
0609 031B              M         SUBWFB  TMP+B2+B1, F            ; sub 2nd byte of b, save in b(B1)
                       M
                   02073
                   02074         SUB16   TMP+B2,MVELOCITY+B1     ; compute new measured velocity
                       M
060A 6A1A              M         MOVFP   TMP+B2+B0,WREG          ; get lowest byte of a into w
060B 0579              M         SUBWF   MVELOCITY+B1+B0, F      ; sub lowest byte of b, save in b(B0)
060C 6A1B              M         MOVFP   TMP+B2+B1,WREG          ; get 2nd byte of a into w
060D 037A              M         SUBWFB  MVELOCITY+B1+B1, F      ; sub 2nd byte of b, save in b(B1)
                       M
                   02075
060E 297B          02076         CLRF    MVELOCITY+B3, F         ; sign extend measured velocity for
060F 9F7A          02077         BTFSC   MVELOCITY+B2,MSB        ; 24 bit addition to measured position
0610 2B7B          02078         SETF    MVELOCITY+B3, F
                   02079
                   02080
                   02081
                   02082         ADD24   MVELOCITY+B1,MPOSITION  ; compute new measured position
                       M
0611 6A79              M         MOVFP   MVELOCITY+B1+B0,WREG    ; get lowest byte of a into w
0612 0F75              M         ADDWF   MPOSITION+B0, F         ; add lowest byte of b, save in b(B0)
0613 6A7A              M         MOVFP   MVELOCITY+B1+B1,WREG    ; get 2nd byte of a into w
0614 1176              M         ADDWFC  MPOSITION+B1, F         ; add 2nd byte of b, save in b(B1)
0615 6A7B              M         MOVFP   MVELOCITY+B1+B2,WREG    ; get 3rd byte of a into w
0616 1177              M         ADDWFC  MPOSITION+B2, F         ; add 3rd byte of b, save in b(B2)
                       M
                   02083                                       ; delta position = measured velocity
                   02084
0617 0002          02085         RETURN
                   02086
```

```
                    02087 ;*******************************************************************************
                    02088
                    02089 ;*******************************************************************************
                    02090 ; NAME:         doIntegral
                    02091 ;
                    02092 ; DESCRIPTION:  Evaluates the integral for the servo calculations.
                    02093 ;
0618                02094 doIntegral
                    02095
                    02096         ADD16   U0,INTEGRAL             ; do integral
                        M
0618 6A8F               M         MOVFP   U0+B0,WREG              ; get lowest byte of a into w
0619 0F9B               M         ADDWF   INTEGRAL+B0, F          ; add lowest byte of b, save in b(B0)
061A 6A90               M         MOVFP   U0+B1,WREG              ; get 2nd byte of a into w
061B 119C               M         ADDWFC  INTEGRAL+B1, F          ; add 2nd byte of b, save in b(B1)
                        M
                    02097
061C 0002           02098         RETURN
                    02099
                    02100 ;*******************************************************************************
                    02101
                    02102 ;*******************************************************************************
                    02103 ; NAME:         doExtstat
                    02104 ;
                    02105 ; DESCRIPTION:  Get +limit,-limit,GPI from PORTB and set in EXTSTAT
                    02106 ;
061D                02107 doExtstat
061D 9407           02108         BTFSS   _intir
061E C627           02109         GOTO    otherbits
                    02110         MOV24   MPOSITION,INDEXPOS
                        M
061F 6A75               M         MOVFP   MPOSITION+B0,WREG       ; get byte of a into w
0620 4AC1               M         MOVPF   WREG,INDEXPOS+B0        ; move to b(B0)
0621 6A76               M         MOVFP   MPOSITION+B1,WREG       ; get byte of a into w
0622 4AC2               M         MOVPF   WREG,INDEXPOS+B1        ; move to b(B1)
0623 6A77               M         MOVFP   MPOSITION+B2,WREG       ; get byte of a into w
0624 4AC3               M         MOVPF   WREG,INDEXPOS+B2        ; move to b(B2)
                        M
0625 8C07           02111         BCF     _intir
0626 8797           02112         BSF     EXTSTAT,MSB
                    02113
0627                02114 otherbits
0627 B800           02115         MOVLB   BANK0                   ; get +limit,-limit and GPI
0628 6A12           02116         MOVFP   PORTB,WREG
0629 190A           02117         RRCF    WREG, F                 ; arrange in correct bit positions
062A B561           02118         ANDLW   0x61
062B 4A18           02119         MOVPF   WREG,TMP
```

```
062C 1D18          02120         SWAPF   TMP, F
062D 0818          02121         IORWF   TMP,W
062E 0997          02122         IORWF   EXTSTAT, F              ; set in EXTSTAT
                   02123
062F 0002          02124         RETURN
                   02125
                   02126 ;***************************************************************************
                   02127
                   02128 ;***************************************************************************
                   02129 ; NAME:         Dmult
                   02130 ;
                   02131 ; DESCRIPTION:  Mult: AARG (16 bits) * BARG (16 bits) -> DPX (32 bits)
                   02132 ;
                   02133 ;      (a) Load the 1st operand in locations AARG+B0 & AARG+B1 (16 bits)
                   02134 ;      (b) Load the 2nd operand in locations BARG+B0 & BARG+B1 (16 bits)
                   02135 ;      (c) CALL Dmult
                   02136 ;      (d) The 32 bit result is in locations (DPX+B0,DPX+B1,DPX+B2,DPX+B3)
                   02137 ;
                   02138 ;       In the signed case, a savings of 9 clks can be realized by choosing
                   02139 ;       BARG as the positive factor in the product when possible.
                   02140 ;
                   02141 ; TIMING (worst case):  unsigned:                    173 clks
                   02142 ;                       signed:    if BARG positive:  170 clks
                   02143 ;                                  if BARG negative:  179 clks
                   02144 ;
                   02145
                   02146 ;***************************************************************************
                   02147
  00000001         02148 SIGNED  equ     TRUE            ; Set This To 'TRUE' for signed multiply
                   02149                                 ; and 'FALSE' for unsigned.
                   02150 ;***************************************************************************
                   02151 ;               Multiplication Macro
                   02152 ;***************************************************************************
                   02153 ;
                   02154 ; TIMING:       unsigned:     11+7*10+8*11   = 169 clks
                   02155 ;(worst case)   signed:       11+7*10+7*11+5 = 163 clks
                   02156 ;
                   02157 MULTMAC  MACRO
                   02158         variable i
                   02159
                   02160           variable i = 0
                   02161           #if   SIGNED
                   02162              variable MULT_LP_CNT = 15
                   02163           #else
                   02164              variable MULT_LP_CNT = 16
                   02165           #endif
                   02166              .while i < MULT_LP_CNT
```

```
02167
02168              .if i < 8
02169                 BTFSC      BARG+B0,i        ; test low byte
02170              .else
02171                 BTFSC      BARG+B1,i-8      ; test high byte
02172              .fi
02173                 GOTO    add#v(i)
02174              .if i < 8
02175                 RLCF    DPX+B3,W        ; rotate sign into carry bit
02176                 RRCF    DPX+B3, F       ; for i < 8, no meaningful bits
02177                 RRCF    DPX+B2, F       ; are in DPX+B0
02178                 RRCF    DPX+B1, F
02179              .else
02180                 RLCF    DPX+B3,W        ; rotate sign into carry bit
02181                 RRCF    DPX+B3, F
02182                 RRCF    DPX+B2, F
02183                 RRCF    DPX+B1, F
02184                 RRCF    DPX+B0, F
02185              .fi
02186                 variable i = i+1
02187              .endw
02188
02189
02190                 CLRF    DPX+B0, F       ; if we get here, BARG = 0
02191                 RETURN
02192
02193
02194
02195 add0
02196                 MOVFP   AARG+B0,WREG
02197                 ADDWF   DPX+B2, F        ;add lsb
02198                 MOVFP   AARG+B1,WREG
02199                 ADDWFC  DPX+B3, F        ;add msb
02200                 RLCF    AARG+B1,W        ; rotate sign into carry bit
02201                 RRCF    DPX+B3, F        ; for i < 8, no meaningful bits
02202                 RRCF    DPX+B2, F        ; are in DPX+B0
02203                 RRCF    DPX+B1, F
02204
02205           variable i = 1
02206
02207
02208              .while i < MULT_LP_CNT
02209
02210              .if i < 8
02211                 BTFSS   BARG+B0,i        ; test low byte
02212              .else
02213                 BTFSS   BARG+B1,i-8      ; test high byte
```

```
02214              .fi
02215                 GOTO    noadd#v(i)
02216 add#v(i)
02217                 MOVFP   AARG+B0,WREG
02218                 ADDWF   DPX+B2, F        ;add lsb
02219                 MOVFP   AARG+B1,WREG
02220                 ADDWFC  DPX+B3, F        ;add msb
02221
02222 noadd#v(i)
02223              .if i < 8
02224
02225                 RLCF    AARG+B1,W        ; rotate sign into carry bit
02226                 RRCF    DPX+B3, F        ; for i < 8, no meaningful bits
02227                 RRCF    DPX+B2, F        ; are in DPX+B0
02228                 RRCF    DPX+B1, F
02229
02230              .else
02231
02232                 RLCF    AARG+B1,W        ; rotate sign into carry bit
02233                 RRCF    DPX+B3, F
02234                 RRCF    DPX+B2, F
02235                 RRCF    DPX+B1, F
02236                 RRCF    DPX+B0, F
02237
02238              .fi
02239
02240              variable i = i+1
02241              .endw
02242
02243           #if   SIGNED
02244
02245                 RLCF    AARG+B1,W        ; since BARG is always made positive,
02246                 RRCF    DPX+B3, F        ; the last bit is known to be zero.
02247                 RRCF    DPX+B2, F
02248                 RRCF    DPX+B1, F
02249                 RRCF    DPX+B0, F
02250
02251           #endif
02252
02253         ENDM
02254
02255 ;          Double Precision Multiply ( 16x16 -> 32 )
02256 ;         ( AARG*BARG -> : 32 bit output in DPX
02257 ;
0630  02258 Dmult
02259      #if   SIGNED
02260
```

```
0630 971F           02261         BTFSS   BARG+B1,MSB             ; test sign of BARG
0631 C63C           02262         GOTO    argsok                  ; if positive, ok
                    02263         NEG16   AARG+B0                 ; if negative, then negate both
                        M
0632 131C               M         COMF    AARG+B0+B0, F
0633 131D               M         COMF    AARG+B0+B1, F
0634 290A               M         CLRF    WREG, F
0635 151C               M         INCF    AARG+B0+B0, F
0636 111D               M         ADDWFC  AARG+B0+B1, F
                        M
                    02264         NEG16   BARG+B0                 ; AARG and BARG
                        M
0637 131E               M         COMF    BARG+B0+B0, F
0638 131F               M         COMF    BARG+B0+B1, F
0639 290A               M         CLRF    WREG, F
063A 151E               M         INCF    BARG+B0+B0, F
063B 111F               M         ADDWFC  BARG+B0+B1, F
                        M
                    02265
                    02266       #endif
063C                02267 argsok
063C 291B           02268         CLRF    DPX+B3, F               ; clear initial partial product
063D 291A           02269         CLRF    DPX+B2, F
                    02270
                    02271         MULTMAC                         ; use macro for multiplication
  0000                  M         variable i
                        M
  0000                  M           variable i = 0
                        M           #if   SIGNED
  000F                  M              variable MULT_LP_CNT = 15
                        M           #else
                        M              variable MULT_LP_CNT = 16
                        M           #endif
                        M              .while i < MULT_LP_CNT
                        M
                        M              .if i < 8
063E 981E               M                 BTFSC      BARG+B0,i        ; test low byte
                        M              .else
                        M                 BTFSC      BARG+B1,i-8      ; test high byte
                        M              .fi
063F C6A1               M                 GOTO    add0
                        M              .if i < 8
0640 1A1B               M                 RLCF    DPX+B3,W          ; rotate sign into carry bit
0641 191B               M                 RRCF    DPX+B3, F         ; for i < 8, no meaningful bits
0642 191A               M                 RRCF    DPX+B2, F         ; are in DPX+B0
0643 1919               M                 RRCF    DPX+B1, F
                        M              .else
```

```
                    M                 RLCF    DPX+B3,W          ; rotate sign into carry bit
                    M                 RRCF    DPX+B3, F
                    M                 RRCF    DPX+B2, F
                    M                 RRCF    DPX+B1, F
                    M                 RRCF    DPX+B0, F
                    M               .fi
  0001              M                 variable i = i+1
                    M
                    M               .if i < 8
0644 991E           M                 BTFSC      BARG+B0,i        ; test low byte
                    M               .else
                    M                 BTFSC      BARG+B1,i-8      ; test high byte
                    M               .fi
0645 C6AB           M                 GOTO    add1
                    M               .if i < 8
0646 1A1B           M                 RLCF    DPX+B3,W          ; rotate sign into carry bit
0647 191B           M                 RRCF    DPX+B3, F         ; for i < 8, no meaningful bits
0648 191A           M                 RRCF    DPX+B2, F         ; are in DPX+B0
0649 1919           M                 RRCF    DPX+B1, F
                    M               .else
                    M                 RLCF    DPX+B3,W          ; rotate sign into carry bit
                    M                 RRCF    DPX+B3, F
                    M                 RRCF    DPX+B2, F
                    M                 RRCF    DPX+B1, F
                    M                 RRCF    DPX+B0, F
                    M               .fi
  0002              M                 variable i = i+1
                    M
                    M               .if i < 8
064A 9A1E           M                 BTFSC   BARG+B0,i         ; test low byte
                    M               .else
                    M                 BTFSC   BARG+B1,i-8       ; test high byte
                    M               .fi
064B C6B5           M                 GOTO    add2
                    M               .if i < 8
064C 1A1B           M                 RLCF    DPX+B3,W          ; rotate sign into carry bit
064D 191B           M                 RRCF    DPX+B3, F         ; for i < 8, no meaningful bits
064E 191A           M                 RRCF    DPX+B2, F         ; are in DPX+B0
064F 1919           M                 RRCF    DPX+B1, F
                    M               .else
                    M                 RLCF    DPX+B3,W          ; rotate sign into carry bit
                    M                 RRCF    DPX+B3, F
                    M                 RRCF    DPX+B2, F
                    M                 RRCF    DPX+B1, F
                    M                 RRCF    DPX+B0, F
                    M               .fi
  0003              M                 variable i = i+1
```

```
                        M
                        M               .if i < 8
0650 9B1E               M                  BTFSC   BARG+B0,i        ; test low byte
                        M               .else
                        M                  BTFSC   BARG+B1,i-8      ; test high byte
                        M               .fi
0651 C6BF               M                  GOTO    add3
                        M               .if i < 8
0652 1A1B               M                  RLCF    DPX+B3,W         ; rotate sign into carry bit
0653 191B               M                  RRCF    DPX+B3, F        ; for i < 8, no meaningful bits
0654 191A               M                  RRCF    DPX+B2, F        ; are in DPX+B0
0655 1919               M                  RRCF    DPX+B1, F
                        M               .else
                        M                  RLCF    DPX+B3,W         ; rotate sign into carry bit
                        M                  RRCF    DPX+B3, F
                        M                  RRCF    DPX+B2, F
                        M                  RRCF    DPX+B1, F
                        M                  RRCF    DPX+B0, F
                        M               .fi
  0004                  M                  variable i = i+1
                        M
                        M               .if i < 8
0656 9C1E               M                  BTFSC   BARG+B0,i        ; test low byte
                        M               .else
                        M                  BTFSC   BARG+B1,i-8      ; test high byte
                        M               .fi
0657 C6C9               M                  GOTO    add4
                        M               .if i < 8
0658 1A1B               M                  RLCF    DPX+B3,W         ; rotate sign into carry bit
0659 191B               M                  RRCF    DPX+B3, F        ; for i < 8, no meaningful bits
065A 191A               M                  RRCF    DPX+B2, F        ; are in DPX+B0
065B 1919               M                  RRCF    DPX+B1, F
                        M               .else
                        M                  RLCF    DPX+B3,W         ; rotate sign into carry bit
                        M                  RRCF    DPX+B3, F
                        M                  RRCF    DPX+B2, F
                        M                  RRCF    DPX+B1, F
                        M                  RRCF    DPX+B0, F
                        M               .fi
  0005                  M                  variable i = i+1
                        M
                        M               .if i < 8
065C 9D1E               M                  BTFSC   BARG+B0,i        ; test low byte
                        M               .else
                        M                  BTFSC   BARG+B1,i-8      ; test high byte
                        M               .fi
065D C6D3               M                  GOTO    add5
```

```
                    M               .if i < 8
065E 1A1B           M                  RLCF    DPX+B3,W         ; rotate sign into carry bit
065F 191B           M                  RRCF    DPX+B3, F        ; for i < 8, no meaningful bits
0660 191A           M                  RRCF    DPX+B2, F        ; are in DPX+B0
0661 1919           M                  RRCF    DPX+B1, F
                    M               .else
                    M                  RLCF    DPX+B3,W         ; rotate sign into carry bit
                    M                  RRCF    DPX+B3, F
                    M                  RRCF    DPX+B2, F
                    M                  RRCF    DPX+B1, F
                    M                  RRCF    DPX+B0, F
                    M               .fi
  0006              M                  variable i = i+1
                    M
                    M               .if i < 8
0662 9E1E           M                  BTFSC   BARG+B0,i        ; test low byte
                    M               .else
                    M                  BTFSC   BARG+B1,i-8      ; test high byte
                    M               .fi
0663 C6DD           M                  GOTO    add6
                    M               .if i < 8
0664 1A1B           M                  RLCF    DPX+B3,W         ; rotate sign into carry bit
0665 191B           M                  RRCF    DPX+B3, F        ; for i < 8, no meaningful bits
0666 191A           M                  RRCF    DPX+B2, F        ; are in DPX+B0
0667 1919           M                  RRCF    DPX+B1, F
                    M               .else
                    M                  RLCF    DPX+B3,W         ; rotate sign into carry bit
                    M                  RRCF    DPX+B3, F
                    M                  RRCF    DPX+B2, F
                    M                  RRCF    DPX+B1, F
                    M                  RRCF    DPX+B0, F
                    M               .fi
  0007              M                  variable i = i+1
                    M
                    M               .if i < 8
0668 9F1E           M                  BTFSC   BARG+B0,i        ; test low byte
                    M               .else
                    M                  BTFSC   BARG+B1,i-8      ; test high byte
                    M               .fi
0669 C6E7           M                  GOTO    add7
                    M               .if i < 8
066A 1A1B           M                  RLCF    DPX+B3,W         ; rotate sign into carry bit
066B 191B           M                  RRCF    DPX+B3, F        ; for i < 8, no meaningful bits
066C 191A           M                  RRCF    DPX+B2, F        ; are in DPX+B0
066D 1919           M                  RRCF    DPX+B1, F
                    M               .else
                    M                  RLCF    DPX+B3,W         ; rotate sign into carry bit
```

```
                    M                RRCF    DPX+B3, F
                    M                RRCF    DPX+B2, F
                    M                RRCF    DPX+B1, F
                    M                RRCF    DPX+B0, F
                    M              .fi
  0008              M                variable i = i+1
                    M
                    M              .if i < 8
                    M                BTFSC   BARG+B0,i        ; test low byte
                    M              .else
066E 981F           M                BTFSC   BARG+B1,i-8      ; test high byte
                    M              .fi
066F C6F1           M                GOTO    add8
                    M              .if i < 8
                    M                RLCF    DPX+B3,W         ; rotate sign into carry bit
                    M                RRCF    DPX+B3, F        ; for i < 8, no meaningful bits
                    M                RRCF    DPX+B2, F        ; are in DPX+B0
                    M                RRCF    DPX+B1, F
                    M              .else
0670 1A1B           M                RLCF    DPX+B3,W         ; rotate sign into carry bit
0671 191B           M                RRCF    DPX+B3, F
0672 191A           M                RRCF    DPX+B2, F
0673 1919           M                RRCF    DPX+B1, F
0674 1918           M                RRCF    DPX+B0, F
                    M              .fi
  0009              M                variable i = i+1
                    M
                    M              .if i < 8
                    M                BTFSC   BARG+B0,i        ; test low byte
                    M              .else
0675 991F           M                BTFSC   BARG+B1,i-8      ; test high byte
                    M              .fi
0676 C6FC           M                GOTO    add9
                    M              .if i < 8
                    M                RLCF    DPX+B3,W         ; rotate sign into carry bit
                    M                RRCF    DPX+B3, F        ; for i < 8, no meaningful bits
                    M                RRCF    DPX+B2, F        ; are in DPX+B0
                    M                RRCF    DPX+B1, F
                    M              .else
0677 1A1B           M                RLCF    DPX+B3,W         ; rotate sign into carry bit
0678 191B           M                RRCF    DPX+B3, F
0679 191A           M                RRCF    DPX+B2, F
067A 1919           M                RRCF    DPX+B1, F
067B 1918           M                RRCF    DPX+B0, F
                    M              .fi
  000A              M                variable i = i+1
                    M
```

```
                    M               .if i < 8
                    M                  BTFSC   BARG+B0,i       ; test low byte
                    M               .else
067C 9A1F           M                  BTFSC   BARG+B1,i-8     ; test high byte
                    M               .fi
067D C707           M                  GOTO    add10
                    M               .if i < 8
                    M                  RLCF    DPX+B3,W        ; rotate sign into carry bit
                    M                  RRCF    DPX+B3, F       ; for i < 8, no meaningful bits
                    M                  RRCF    DPX+B2, F       ; are in DPX+B0
                    M                  RRCF    DPX+B1, F
                    M               .else
067E 1A1B           M                  RLCF    DPX+B3,W        ; rotate sign into carry bit
067F 191B           M                  RRCF    DPX+B3, F
0680 191A           M                  RRCF    DPX+B2, F
0681 1919           M                  RRCF    DPX+B1, F
0682 1918           M                  RRCF    DPX+B0, F
                    M               .fi
  000B              M                  variable i = i+1
                    M
                    M               .if i < 8
                    M                  BTFSC   BARG+B0,i       ; test low byte
                    M               .else
0683 9B1F           M                  BTFSC   BARG+B1,i-8     ; test high byte
                    M               .fi
0684 C712           M                  GOTO    add11
                    M               .if i < 8
                    M                  RLCF    DPX+B3,W        ; rotate sign into carry bit
                    M                  RRCF    DPX+B3, F       ; for i < 8, no meaningful bits
                    M                  RRCF    DPX+B2, F       ; are in DPX+B0
                    M                  RRCF    DPX+B1, F
                    M               .else
0685 1A1B           M                  RLCF    DPX+B3,W        ; rotate sign into carry bit
0686 191B           M                  RRCF    DPX+B3, F
0687 191A           M                  RRCF    DPX+B2, F
0688 1919           M                  RRCF    DPX+B1, F
0689 1918           M                  RRCF    DPX+B0, F
                    M               .fi
  000C              M                  variable i = i+1
                    M
                    M               .if i < 8
                    M                  BTFSC   BARG+B0,i       ; test low byte
                    M               .else
068A 9C1F           M                  BTFSC   BARG+B1,i-8     ; test high byte
                    M               .fi
068B C71D           M                  GOTO    add12
                    M               .if i < 8
```

```
                        M                 RLCF    DPX+B3,W        ; rotate sign into carry bit
                        M                 RRCF    DPX+B3, F       ; for i < 8, no meaningful bits
                        M                 RRCF    DPX+B2, F       ; are in DPX+B0
                        M                 RRCF    DPX+B1, F
                        M               .else
068C 1A1B               M                 RLCF    DPX+B3,W        ; rotate sign into carry bit
068D 191B               M                 RRCF    DPX+B3, F
068E 191A               M                 RRCF    DPX+B2, F
068F 1919               M                 RRCF    DPX+B1, F
0690 1918               M                 RRCF    DPX+B0, F
                        M               .fi
  000D                  M                 variable i = i+1
                        M
                        M               .if i < 8
                        M                 BTFSC   BARG+B0,i        ; test low byte
                        M               .else
0691 9D1F               M                 BTFSC   BARG+B1,i-8      ; test high byte
                        M               .fi
0692 C728               M                 GOTO    add13
                        M               .if i < 8
                        M                 RLCF    DPX+B3,W        ; rotate sign into carry bit
                        M                 RRCF    DPX+B3, F       ; for i < 8, no meaningful bits
                        M                 RRCF    DPX+B2, F       ; are in DPX+B0
                        M                 RRCF    DPX+B1, F
                        M               .else
0693 1A1B               M                 RLCF    DPX+B3,W        ; rotate sign into carry bit
0694 191B               M                 RRCF    DPX+B3, F
0695 191A               M                 RRCF    DPX+B2, F
0696 1919               M                 RRCF    DPX+B1, F
0697 1918               M                 RRCF    DPX+B0, F
                        M               .fi
  000E                  M                 variable i = i+1
                        M
                        M               .if i < 8
                        M                 BTFSC   BARG+B0,i        ; test low byte
                        M               .else
0698 9E1F               M                 BTFSC   BARG+B1,i-8      ; test high byte
                        M               .fi
0699 C733               M                 GOTO    add14
                        M               .if i < 8
                        M                 RLCF    DPX+B3,W        ; rotate sign into carry bit
                        M                 RRCF    DPX+B3, F       ; for i < 8, no meaningful bits
                        M                 RRCF    DPX+B2, F       ; are in DPX+B0
                        M                 RRCF    DPX+B1, F
                        M               .else
069A 1A1B               M                 RLCF    DPX+B3,W        ; rotate sign into carry bit
069B 191B               M                 RRCF    DPX+B3, F
```

```
069C 191A               M                RRCF    DPX+B2, F
069D 1919               M                RRCF    DPX+B1, F
069E 1918               M                RRCF    DPX+B0, F
                        M             .fi
  000F                  M                variable i = i+1
                        M             .endw
                        M
                        M
069F 2918               M                CLRF    DPX+B0, F      ; if we get here, BARG = 0
06A0 0002               M                RETURN
                        M
                        M
                        M
06A1                    M add0
06A1 6A1C               M                MOVFP   AARG+B0,WREG
06A2 0F1A               M                ADDWF   DPX+B2, F       ;add lsb
06A3 6A1D               M                MOVFP   AARG+B1,WREG
06A4 111B               M                ADDWFC  DPX+B3, F       ;add msb
06A5 1A1D               M                RLCF    AARG+B1,W       ; rotate sign into carry bit
06A6 191B               M                RRCF    DPX+B3, F       ; for i < 8, no meaningful bits
06A7 191A               M                RRCF    DPX+B2, F       ; are in DPX+B0
06A8 1919               M                RRCF    DPX+B1, F
                        M
  0001                  M           variable i = 1
                        M
                        M
                        M             .while i < MULT_LP_CNT
                        M
                        M             .if i < 8
06A9 911E               M                BTFSS   BARG+B0,i       ; test low byte
                        M             .else
                        M                BTFSS   BARG+B1,i-8     ; test high byte
                        M             .fi
06AA C6AF               M                GOTO    noadd1
06AB                    M add1
06AB 6A1C               M                MOVFP   AARG+B0,WREG
06AC 0F1A               M                ADDWF   DPX+B2, F       ; add lsb
06AD 6A1D               M                MOVFP   AARG+B1,WREG
06AE 111B               M                ADDWFC  DPX+B3, F       ; add msb
                        M
06AF                    M noadd1
                        M             .if i < 8
                        M
06AF 1A1D               M                RLCF    AARG+B1,W       ; rotate sign into carry bit
06B0 191B               M                RRCF    DPX+B3, F       ; for i < 8, no meaningful bits
06B1 191A               M                RRCF    DPX+B2, F       ; are in DPX+B0
06B2 1919               M                RRCF    DPX+B1, F
```

```
                    M
                    M               .else
                    M
                    M                  RLCF    AARG+B1,W       ; rotate sign into carry bit
                    M                  RRCF    DPX+B3, F
                    M                  RRCF    DPX+B2, F
                    M                  RRCF    DPX+B1, F
                    M                  RRCF    DPX+B0, F
                    M
                    M               .fi
                    M
  0002              M               variable i = i+1
                    M
                    M               .if i < 8
06B3 921E           M                  BTFSS   BARG+B0,i        ; test low byte
                    M               .else
                    M                  BTFSS   BARG+B1,i-8      ; test high byte
                    M               .fi
06B4 C6B9           M                  GOTO    noadd2
06B5                M add2
06B5 6A1C           M                  MOVFP   AARG+B0,WREG
06B6 0F1A           M                  ADDWF   DPX+B2, F        ;add lsb
06B7 6A1D           M                  MOVFP   AARG+B1,WREG
06B8 111B           M                  ADDWFC  DPX+B3, F        ;add msb
                    M
06B9                M noadd2
                    M               .if i < 8
                    M
06B9 1A1D           M                  RLCF    AARG+B1,W        ; rotate sign into carry bit
06BA 191B           M                  RRCF    DPX+B3, F        ; for i < 8, no meaningful bits
06BB 191A           M                  RRCF    DPX+B2, F        ; are in DPX+B0
06BC 1919           M                  RRCF    DPX+B1, F
                    M
                    M               .else
                    M
                    M                  RLCF    AARG+B1,W        ; rotate sign into carry bit
                    M                  RRCF    DPX+B3, F
                    M                  RRCF    DPX+B2, F
                    M                  RRCF    DPX+B1, F
                    M                  RRCF    DPX+B0, F
                    M
                    M               .fi
                    M
  0003              M               variable i = i+1
                    M
                    M               .if i < 8
06BD 931E           M                  BTFSS   BARG+B0,i        ; test low byte
```

```
                        M              .else
                        M                 BTFSS   BARG+B1,i-8      ; test high byte
                        M              .fi
06BE C6C3               M                 GOTO    noadd3
06BF                    M add3
06BF 6A1C               M                 MOVFP   AARG+B0,WREG
06C0 0F1A               M                 ADDWF   DPX+B2, F         ;add lsb
06C1 6A1D               M                 MOVFP   AARG+B1,WREG
06C2 111B               M                 ADDWFC  DPX+B3, F         ;add msb
                        M
06C3                    M noadd3
                        M              .if i < 8
                        M
06C3 1A1D               M                 RLCF    AARG+B1,W         ; rotate sign into carry bit
06C4 191B               M                 RRCF    DPX+B3, F         ; for i < 8, no meaningful bits
06C5 191A               M                 RRCF    DPX+B2, F         ; are in DPX+B0
06C6 1919               M                 RRCF    DPX+B1, F
                        M
                        M              .else
                        M
                        M                 RLCF    AARG+B1,W         ; rotate sign into carry bit
                        M                 RRCF    DPX+B3, F
                        M                 RRCF    DPX+B2, F
                        M                 RRCF    DPX+B1, F
                        M                 RRCF    DPX+B0, F
                        M
                        M              .fi
                        M
  0004                  M              variable i = i+1
                        M
                        M              .if i < 8
06C7 941E               M                 BTFSS   BARG+B0,i         ; test low byte
                        M              .else
                        M                 BTFSS   BARG+B1,i-8       ; test high byte
                        M              .fi
06C8 C6CD               M                 GOTO    noadd4
06C9                    M add4
06C9 6A1C               M                 MOVFP   AARG+B0,WREG
06CA 0F1A               M                 ADDWF   DPX+B2, F         ;add lsb
06CB 6A1D               M                 MOVFP   AARG+B1,WREG
06CC 111B               M                 ADDWFC  DPX+B3, F         ;add msb
                        M
06CD                    M noadd4
                        M              .if i < 8
                        M
06CD 1A1D               M                 RLCF    AARG+B1,W        ; rotate sign into carry bit
06CE 191B               M                 RRCF    DPX+B3, F        ; for i < 8, no meaningful bits
```

```
06CF 191A               M                RRCF    DPX+B2, F       ; are in DPX+B0
06D0 1919               M                RRCF    DPX+B1, F
                        M
                        M             .else
                        M
                        M                RLCF    AARG+B1,W       ; rotate sign into carry bit
                        M                RRCF    DPX+B3, F
                        M                RRCF    DPX+B2, F
                        M                RRCF    DPX+B1, F
                        M                RRCF    DPX+B0, F
                        M
                        M             .fi
                        M
  0005                  M             variable i = i+1
                        M
                        M             .if i < 8
06D1 951E               M                BTFSS   BARG+B0,i        ; test low byte
                        M             .else
                        M                BTFSS   BARG+B1,i-8      ; test high byte
                        M             .fi
06D2 C6D7               M                GOTO    noadd5
06D3                    M add5
06D3 6A1C               M                MOVFP   AARG+B0,WREG
06D4 0F1A               M                ADDWF   DPX+B2, F        ;add lsb
06D5 6A1D               M                MOVFP   AARG+B1,WREG
06D6 111B               M                ADDWFC  DPX+B3, F        ;add msb
                        M
06D7                    M noadd5
                        M             .if i < 8
                        M
06D7 1A1D               M                RLCF    AARG+B1,W       ; rotate sign into carry bit
06D8 191B               M                RRCF    DPX+B3, F       ; for i < 8, no meaningful bits
06D9 191A               M                RRCF    DPX+B2, F       ; are in DPX+B0
06DA 1919               M                RRCF    DPX+B1, F
                        M
                        M             .else
                        M
                        M                RLCF    AARG+B1,W       ; rotate sign into carry bit
                        M                RRCF    DPX+B3, F
                        M                RRCF    DPX+B2, F
                        M                RRCF    DPX+B1, F
                        M                RRCF    DPX+B0, F
                        M
                        M             .fi
                        M
  0006                  M             variable i = i+1
                        M
```

```
                        M              .if i < 8
06DB 961E               M                 BTFSS   BARG+B0,i        ; test low byte
                        M              .else
                        M                 BTFSS   BARG+B1,i-8      ; test high byte
                        M              .fi
06DC C6E1               M                 GOTO    noadd6
06DD                    M add6
06DD 6A1C               M                 MOVFP   AARG+B0,WREG
06DE 0F1A               M                 ADDWF   DPX+B2, F        ;add lsb
06DF 6A1D               M                 MOVFP   AARG+B1,WREG
06E0 111B               M                 ADDWFC  DPX+B3, F        ;add msb
                        M
06E1                    M noadd6
                        M              .if i < 8
                        M
06E1 1A1D               M                 RLCF    AARG+B1,W       ; rotate sign into carry bit
06E2 191B               M                 RRCF    DPX+B3, F       ; for i < 8, no meaningful bits
06E3 191A               M                 RRCF    DPX+B2, F       ; are in DPX+B0
06E4 1919               M                 RRCF    DPX+B1, F
                        M
                        M              .else
                        M
                        M                 RLCF    AARG+B1,W       ; rotate sign into carry bit
                        M                 RRCF    DPX+B3, F
                        M                 RRCF    DPX+B2, F
                        M                 RRCF    DPX+B1, F
                        M                 RRCF    DPX+B0, F
                        M
                        M              .fi
                        M
  0007                  M              variable i = i+1
                        M
                        M              .if i < 8
06E5 971E               M                 BTFSS   BARG+B0,i        ; test low byte
                        M              .else
                        M                 BTFSS   BARG+B1,i-8      ; test high byte
                        M              .fi
06E6 C6EB               M                 GOTO    noadd7
06E7                    M add7
06E7 6A1C               M                 MOVFP   AARG+B0,WREG
06E8 0F1A               M                 ADDWF   DPX+B2, F        ;add lsb
06E9 6A1D               M                 MOVFP   AARG+B1,WREG
06EA 111B               M                 ADDWFC  DPX+B3, F        ;add msb
                        M
06EB                    M noadd7
                        M              .if i < 8
                        M
```

```
06EB 1A1D               M                RLCF    AARG+B1,W       ; rotate sign into carry bit
06EC 191B               M                RRCF    DPX+B3, F       ; for i < 8, no meaningful bits
06ED 191A               M                RRCF    DPX+B2, F       ; are in DPX+B0
06EE 1919               M                RRCF    DPX+B1, F
                        M
                        M             .else
                        M
                        M                RLCF    AARG+B1,W       ; rotate sign into carry bit
                        M                RRCF    DPX+B3, F
                        M                RRCF    DPX+B2, F
                        M                RRCF    DPX+B1, F
                        M                RRCF    DPX+B0, F
                        M
                        M             .fi
                        M
  0008                  M             variable i = i+1
                        M
                        M             .if i < 8
                        M                BTFSS   BARG+B0,i        ; test low byte
                        M             .else
06EF 901F               M                BTFSS   BARG+B1,i-8      ; test high byte
                        M             .fi
06F0 C6F5               M                GOTO    noadd8
06F1                    M add8
06F1 6A1C               M                MOVFP   AARG+B0,WREG
06F2 0F1A               M                ADDWF   DPX+B2, F        ;add lsb
06F3 6A1D               M                MOVFP   AARG+B1,WREG
06F4 111B               M                ADDWFC  DPX+B3, F        ;add msb
                        M
06F5                    M noadd8
                        M             .if i < 8
                        M
                        M                RLCF    AARG+B1,W       ; rotate sign into carry bit
                        M                RRCF    DPX+B3, F       ; for i < 8, no meaningful bits
                        M                RRCF    DPX+B2, F       ; are in DPX+B0
                        M                RRCF    DPX+B1, F
                        M
                        M             .else
                        M
06F5 1A1D               M                RLCF    AARG+B1,W       ; rotate sign into carry bit
06F6 191B               M                RRCF    DPX+B3, F
06F7 191A               M                RRCF    DPX+B2, F
06F8 1919               M                RRCF    DPX+B1, F
06F9 1918               M                RRCF    DPX+B0, F
                        M
                        M             .fi
                        M
```

```
  0009                  M               variable i = i+1
                        M
                        M               .if i < 8
                        M                  BTFSS   BARG+B0,i        ; test low byte
                        M               .else
06FA 911F               M                  BTFSS   BARG+B1,i-8      ; test high byte
                        M               .fi
06FB C700               M                  GOTO    noadd9
06FC                    M add9
06FC 6A1C               M                  MOVFP   AARG+B0,WREG
06FD 0F1A               M                  ADDWF   DPX+B2, F        ;add lsb
06FE 6A1D               M                  MOVFP   AARG+B1,WREG
06FF 111B               M                  ADDWFC  DPX+B3, F        ;add msb
                        M
0700                    M noadd9
                        M               .if i < 8
                        M
                        M                  RLCF    AARG+B1,W       ; rotate sign into carry bit
                        M                  RRCF    DPX+B3, F       ; for i < 8, no meaningful bits
                        M                  RRCF    DPX+B2, F       ; are in DPX+B0
                        M                  RRCF    DPX+B1, F
                        M
                        M               .else
                        M
0700 1A1D               M                  RLCF    AARG+B1,W       ; rotate sign into carry bit
0701 191B               M                  RRCF    DPX+B3, F
0702 191A               M                  RRCF    DPX+B2, F
0703 1919               M                  RRCF    DPX+B1, F
0704 1918               M                  RRCF    DPX+B0, F
                        M
                        M               .fi
                        M
  000A                  M               variable i = i+1
                        M
                        M               .if i < 8
                        M                  BTFSS      BARG+B0,i        ; test low byte
                        M               .else
0705 921F               M                  BTFSS      BARG+B1,i-8      ; test high byte
                        M               .fi
0706 C70B               M                  GOTO    noadd10
0707                    M add10
0707 6A1C               M                  MOVFP   AARG+B0,WREG
0708 0F1A               M                  ADDWF   DPX+B2, F        ;add lsb
0709 6A1D               M                  MOVFP   AARG+B1,WREG
070A 111B               M                  ADDWFC  DPX+B3, F        ;add msb
                        M
070B                    M noadd10
```

```
                        M               .if i < 8
                        M
                        M                  RLCF    AARG+B1,W       ; rotate sign into carry bit
                        M                  RRCF    DPX+B3, F       ; for i < 8, no meaningful bits
                        M                  RRCF    DPX+B2, F       ; are in DPX+B0
                        M                  RRCF    DPX+B1, F
                        M
                        M               .else
                        M
070B 1A1D               M                  RLCF    AARG+B1,W       ; rotate sign into carry bit
070C 191B               M                  RRCF    DPX+B3, F
070D 191A               M                  RRCF    DPX+B2, F
070E 1919               M                  RRCF    DPX+B1, F
070F 1918               M                  RRCF    DPX+B0, F
                        M
                        M               .fi
                        M
  000B                  M               variable i = i+1
                        M
                        M               .if i < 8
                        M                  BTFSS   BARG+B0,i        ; test low byte
                        M               .else
0710 931F               M                  BTFSS   BARG+B1,i-8      ; test high byte
                        M               .fi
0711 C716               M                  GOTO    noadd11
0712                    M add11
0712 6A1C               M                  MOVFP   AARG+B0,WREG
0713 0F1A               M                  ADDWF   DPX+B2, F        ; add lsb
0714 6A1D               M                  MOVFP   AARG+B1,WREG
0715 111B               M                  ADDWFC  DPX+B3, F        ; add msb
                        M
0716                    M noadd11
                        M               .if i < 8
                        M
                        M                  RLCF    AARG+B1,W        ; rotate sign into carry bit
                        M                  RRCF    DPX+B3, F        ; for i < 8, no meaningful bits
                        M                  RRCF    DPX+B2, F        ; are in DPX+B0
                        M                  RRCF    DPX+B1, F
                        M
                        M               .else
                        M
0716 1A1D               M                  RLCF    AARG+B1,W        ; rotate sign into carry bit
0717 191B               M                  RRCF    DPX+B3, F
0718 191A               M                  RRCF    DPX+B2, F
0719 1919               M                  RRCF    DPX+B1, F
071A 1918               M                  RRCF    DPX+B0, F
                        M
```

```
                    M               .fi
                    M
  000C              M               variable i = i+1
                    M
                    M               .if i < 8
                    M                  BTFSS   BARG+B0,i        ; test low byte
                    M               .else
071B 941F           M                  BTFSS   BARG+B1,i-8      ; test high byte
                    M               .fi
071C C721           M                  GOTO    noadd12
071D                M add12
071D 6A1C           M                  MOVFP   AARG+B0,WREG
071E 0F1A           M                  ADDWF   DPX+B2, F        ; add lsb
071F 6A1D           M                  MOVFP   AARG+B1,WREG
0720 111B           M                  ADDWFC  DPX+B3, F        ; add msb
                    M
0721                M noadd12
                    M               .if i < 8
                    M
                    M                  RLCF    AARG+B1,W       ; rotate sign into carry bit
                    M                  RRCF    DPX+B3, F       ; for i < 8, no meaningful bits
                    M                  RRCF    DPX+B2, F       ; are in DPX+B0
                    M                  RRCF    DPX+B1, F
                    M
                    M               .else
                    M
0721 1A1D           M                  RLCF    AARG+B1,W       ; rotate sign into carry bit
0722 191B           M                  RRCF    DPX+B3, F
0723 191A           M                  RRCF    DPX+B2, F
0724 1919           M                  RRCF    DPX+B1, F
0725 1918           M                  RRCF    DPX+B0, F
                    M
                    M               .fi
                    M
  000D              M               variable i = i+1
                    M
                    M               .if i < 8
                    M                  BTFSS   BARG+B0,i        ; test low byte
                    M               .else
0726 951F           M                  BTFSS   BARG+B1,i-8      ; test high byte
                    M               .fi
0727 C72C           M                  GOTO    noadd13
0728                M add13
0728 6A1C           M                  MOVFP   AARG+B0,WREG
0729 0F1A           M                  ADDWF   DPX+B2, F        ; add lsb
072A 6A1D           M                  MOVFP   AARG+B1,WREG
072B 111B           M                  ADDWFC  DPX+B3, F        ; add msb
```

```
                    M
072C                M noadd13
                    M               .if i < 8
                    M
                    M                  RLCF    AARG+B1,W       ; rotate sign into carry bit
                    M                  RRCF    DPX+B3, F       ; for i < 8, no meaningful bits
                    M                  RRCF    DPX+B2, F       ; are in DPX+B0
                    M                  RRCF    DPX+B1, F
                    M
                    M               .else
                    M
072C 1A1D           M                  RLCF    AARG+B1,W       ; rotate sign into carry bit
072D 191B           M                  RRCF    DPX+B3, F
072E 191A           M                  RRCF    DPX+B2, F
072F 1919           M                  RRCF    DPX+B1, F
0730 1918           M                  RRCF    DPX+B0, F
                    M
                    M               .fi
                    M
  000E              M               variable i = i+1
                    M
                    M               .if i < 8
                    M                  BTFSS   BARG+B0,i        ; test low byte
                    M               .else
0731 961F           M                  BTFSS   BARG+B1,i-8      ; test high byte
                    M               .fi
0732 C737           M                  GOTO    noadd14
0733                M add14
0733 6A1C           M                  MOVFP   AARG+B0,WREG
0734 0F1A           M                  ADDWF   DPX+B2, F        ;add lsb
0735 6A1D           M                  MOVFP   AARG+B1,WREG
0736 111B           M                  ADDWFC  DPX+B3, F        ;add msb
                    M
0737                M noadd14
                    M               .if i < 8
                    M
                    M                  RLCF    AARG+B1,W       ; rotate sign into carry bit
                    M                  RRCF    DPX+B3, F       ; for i < 8, no meaningful bits
                    M                  RRCF    DPX+B2, F       ; are in DPX+B0
                    M                  RRCF    DPX+B1, F
                    M
                    M               .else
                    M
0737 1A1D           M                  RLCF    AARG+B1,W       ; rotate sign into carry bit
0738 191B           M                  RRCF    DPX+B3, F
0739 191A           M                  RRCF    DPX+B2, F
073A 1919           M                  RRCF    DPX+B1, F
```

```
073B 1918                M                 RRCF    DPX+B0, F
                         M
                         M              .fi
                         M
  000F                   M              variable i = i+1
                         M              .endw
                         M
                         M           #if   SIGNED
                         M
073C 1A1D                M                 RLCF    AARG+B1,W       ; since BARG is always made positive,
073D 191B                M                 RRCF    DPX+B3, F       ; the last bit is known to be zero.
073E 191A                M                 RRCF    DPX+B2, F
073F 1919                M                 RRCF    DPX+B1, F
0740 1918                M                 RRCF    DPX+B0, F
                         M
                         M           #endif
                         M
                       02272
0741 0002              02273         RETURN
                       02274
                       02275 ;*****************************************************************************
                       02276 ;
                       02277
                       02278 ;*****************************************************************************
                       02279 ; NAME:         doCaptureRegs
                       02280 ;
                       02281 ; DESCRIPTION:  Captures Desired Register Values To PIC-MASTER Trace Buffer
                       02282 ;               Intended for PICMASTER Demo/debug/servo tuning Purposes Only
                       02283 ;               Capture The following registers to Trace Buffer by putting
                       02284 ;               A Trace point on a TABLW instruction. Trace only 2nd Cycle
                       02285 ;
                       02286 ;               (a) POSERROR (position error : 16 bits)
                       02287 ;               (b) VELERROR (velocity error : 16 bits)
                       02288 ;               (c) MPOSITION (measured position value : 24 bits)
                       02289 ;               (d) MVELOCITY (measured velocity value : 24 bits)
                       02290 ;               (e) POSITION (commanded position : 24 bits)
                       02291 ;               (f) VELOCITY (commanded velocity : 24 bits)
                       02292 ;               (g) Y (output of servo loop : 32 bits)
                       02293 ;               (h) YPWM (output value written to PWM : 10 bits)
                       02294 ;
                       02295 ;
                       02296 #define CaptureAddr     0x8000
                       02297 ;
0742                   02298 doCaptureRegs
                       02299                                         ; !end! hdr !skip start!
0742 B000              02300         movlw   (CaptureAddr & 0xff)
0743 010D              02301         movwf   TBLPTRL
```

```
0744 B080          02302         movlw   CaptureAddr/256
0745 010E          02303         movwf   TBLPTRH                ; setup table pointer address
                   02304
0746 AC7C          02305         tablwt  0,0,POSERROR+B0 ; dummy tablwt
0747 A67D          02306         tlwt    1,POSERROR+B1           ; now table latch = 16 bits contents of POSERROR
0748               02307 capPerr
0748 AC7C          02308         tablwt  0,0,POSERROR+B0 ; perform actual table write of POSERROR
                   02309
0749 AC7F          02310         tablwt  0,0,VELERROR+B0
074A A680          02311         tlwt    1,VELERROR+B1           ; capture Velocity error
074B               02312 capVerr
074B AC7F          02313         tablwt  0,0,VELERROR+B0
                   02314
074C AC75          02315         tablwt  0,0,MPOSITION+B0
074D A676          02316         tlwt    1,MPOSITION+B1          ; capture measured position
074E               02317 capMpos
074E AC75          02318         tablwt  0,0,MPOSITION+B0
                   02319
074F AC58          02320         tablwt  0,0,POSITION+B0
0750 A659          02321         tlwt    1,POSITION+B1           ; capture commanded position
0751               02322 capPos
0751 AC58          02323         tablwt  0,0,POSITION+B0
                   02324
0752 AC78          02325         tablwt  0,0,MVELOCITY+B0
0753 A679          02326         tlwt    1,MVELOCITY+B1          ; capture measured velocity
0754               02327 capMvel
0754 AC78          02328         tablwt  0,0,MVELOCITY+B0
                   02329
0755 AC5B          02330         tablwt  0,0,VELOCITY+B0
0756 A65C          02331         tlwt    1,VELOCITY+B1           ; capture commanded velocity
0757               02332 capVel
0757 AC5B          02333         tablwt  0,0,VELOCITY+B0
                   02334
                   02335         DEC16   CAPTMP
                       M
0758 290A              M         CLRF    WREG, F
0759 07C8              M         DECF    CAPTMP+B0, F
075A 03C9              M         SUBWFB  CAPTMP+B1, F
                       M
                   02336         TFSZ16  CAPTMP
                       M
075B 6AC8              M         MOVFP   CAPTMP+B0,WREG
075C 08C9              M         IORWF   CAPTMP+B1,W
075D 330A              M         TSTFSZ  WREG
075E 0002          02337         RETURN
075F 29C5          02338         CLRF    CAPFLAG, F
                   02339         MOV16   CAPCOUNT,CAPTMP
```

```
                        M
0760 6AC6               M         MOVFP   CAPCOUNT+B0,WREG        ; get byte of a into w
0761 4AC8               M         MOVPF   WREG,CAPTMP+B0          ; move to b(B0)
0762 6AC7               M         MOVFP   CAPCOUNT+B1,WREG        ; get byte of a into w
0763 4AC9               M         MOVPF   WREG,CAPTMP+B1          ; move to b(B1)
                        M
0764 0001           02340         HALT
0765 0002           02341         RETURN
                    02342
                    02343
                    02344 ;*************************************************************************
                    02345 ;
                    02346 ;
                    02347 ;       TABLES:
                    02348
                    02349
                    02350         CMD_START CMD_TABLE
                        M
0766                    M CMD_TABLE
                    02351         CMD_DEF do_null,DO_NULL
                        M
0766 000D               M         DATA    DO_NULL
0767 00A6               M         DATA    do_null
                    02352         CMD_DEF do_move,DO_MOVE
                        M
0768 004D               M         DATA    DO_MOVE
0769 00A8               M         DATA    do_move
                    02353         CMD_DEF do_mode,DO_MODE
                        M
076A 004F               M         DATA    DO_MODE
076B 00B6               M         DATA    do_mode
                    02354         CMD_DEF do_setparameter,DO_SETPARAMETER
                        M
076C 0053               M         DATA    DO_SETPARAMETER
076D 00D2               M         DATA    do_setparameter
                    02355         CMD_DEF do_readparameter,DO_READPARAMETER
                        M
076E 0052               M         DATA    DO_READPARAMETER
076F 00F9               M         DATA    do_readparameter
                    02356         CMD_DEF do_shutter,DO_SHUTTER
                        M
0770 0043               M         DATA    DO_SHUTTER
0771 0123               M         DATA    do_shutter
                    02357         CMD_DEF do_readcomposition,DO_READCOMPOSITION
                        M
0772 0050               M         DATA    DO_READCOMPOSITION
0773 0145               M         DATA    do_readcomposition
```

```
                    02358         CMD_DEF do_readcomvelocity,DO_READCOMVELOCITY
                        M
0774 0056               M         DATA    DO_READCOMVELOCITY
0775 014E               M         DATA    do_readcomvelocity
                    02359         CMD_DEF do_readactposition,DO_READACTPOSITION
                        M
0776 0070               M         DATA    DO_READACTPOSITION
0777 0157               M         DATA    do_readactposition
                    02360         CMD_DEF do_readactvelocity,DO_READACTVELOCITY
                        M
0778 0076               M         DATA    DO_READACTVELOCITY
0779 0160               M         DATA    do_readactvelocity
                    02361         CMD_DEF do_externalstatus,DO_EXTERNALSTATUS
                        M
077A 0058               M         DATA    DO_EXTERNALSTATUS
077B 0169               M         DATA    do_externalstatus
                    02362         CMD_DEF do_movestatus,DO_MOVESTATUS
                        M
077C 0059               M         DATA    DO_MOVESTATUS
077D 0170               M         DATA    do_movestatus
                    02363         CMD_DEF do_readindposition,DO_READINDPOSITION
                        M
077E 0049               M         DATA    DO_READINDPOSITION
077F 0174               M         DATA    do_readindposition
                    02364         CMD_DEF do_setposition,DO_SETPOSITION
                        M
0780 0048               M         DATA    DO_SETPOSITION
0781 017D               M         DATA    do_setposition
                    02365         CMD_DEF do_reset,DO_RESET
                        M
0782 005A               M         DATA    DO_RESET
0783 0190               M         DATA    do_reset
                    02366         CMD_DEF do_stop,DO_STOP
                        M
0784 0073               M         DATA    DO_STOP
0785 0193               M         DATA    do_stop
                    02367         CMD_DEF do_capture,DO_CAPTURE
                        M
0786 0063               M         DATA    DO_CAPTURE
0787 0196               M         DATA    do_capture
                    02368         CMD_END
                        M ;                                               ;
0788 0000               M         DATA    0x00
                    02369
                    02370
0789 0003           02371 PAR_TABLE       DATA    0x0003
078A 0020           02372                 DATA    VL
```

```
078B 0103           02373                DATA    0x0103
078C 0023           02374                DATA    AL
078D 0202           02375                DATA    0x0202
078E 0026           02376                DATA    KP
078F 0302           02377                DATA    0x0302
0790 0028           02378                DATA    KV
0791 0402           02379                DATA    0x0402
0792 002A           02380                DATA    KI
0793 0501           02381                DATA    0x0501
0794 002C           02382                DATA    IM
0795 0602           02383                DATA    0x0602
0796 002D           02384                DATA    FV
0797 0702           02385                DATA    0x0702
0798 002F           02386                DATA    FA
0799 0008           02387                DATA    NUMPAR
                    02388
                    02389          #if     DECIO
                    02390
079A 423F           02391 DEC_TABLE      DATA    0x423F
079B 000F           02392                DATA    0x000F
079C 869F           02393                DATA    0x869F
079D 0001           02394                DATA    0x0001
079E 270F           02395                DATA    0x270F
079F 0000           02396                DATA    0x0000
07A0 03E7           02397                DATA    0x03E7
07A1 0000           02398                DATA    0x0000
07A2 0063           02399                DATA    0x0063
07A3 0000           02400                DATA    0x0000
07A4 0009           02401                DATA    0x0009
07A5 0000           02402                DATA    0x0000
07A6 FFFF           02403                DATA    0xFFFF
                    02404          #endif
                    02405
                    02406
                    02407
                    02408
                    02409          END
MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : X--------------- ---------------- XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0080 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
00C0 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0100 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0140 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0180 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
01C0 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
```

```
0200 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0240 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0280 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
02C0 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0300 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0340 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0380 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
03C0 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0400 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0440 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0480 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
04C0 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0500 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0540 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0580 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
05C0 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0600 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0640 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0680 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
06C0 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0700 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0740 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0780 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXX--------- ----------------

All other memory blocks unused.

Program Memory Words Used:  1928

Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,     0 suppressed
```

Note the following details of the code protection feature on PICmicro® MCUs.

- The PICmicro family meets the specifications contained in the Microchip Data Sheet.
- Microchip believes that its family of PICmicro microcontrollers is one of the most secure products of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the PICmicro microcontroller in a manner outside the operating specifications contained in the data sheet. The person doing so may be engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as “unbreakable”.
- Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our product.

If you have any further questions about this matter, please contact the local sales office nearest to you.

DNV Certification, Inc. USA — ANSI-RAB QMS ACCREDITED — DNV MSC The Netherlands Accredited by the RvA — DNV — ISO 9001 / QS-9000 REGISTERED FIRM

Microchip received QS-9000 quality system certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona in July 1999. The Company’s quality system processes and procedures are QS-9000 compliant for its PICmicro® 8-bit MCUs, KEELOQ® code hopping devices, Serial EEPROMs and microperipheral products. In addition, Microchip’s quality system for the design and manufacture of development systems is ISO 9001 certified.

WORLDWIDE SALES AND SERVICE

AMERICAS

Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200 Fax: 480-792-7277
Technical Support: 480-792-7627
Web Address: http://www.microchip.com

Rocky Mountain
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7966 Fax: 480-792-7456

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

Boston
2 Lan Drive, Suite 120
Westford, MA 01886
Tel: 978-692-3848 Fax: 978-692-3821

Chicago
333 Pierce Road, Suite 180
Itasca, IL 60143
Tel: 630-285-0071 Fax: 630-285-0075

Dallas
4570 Westgrove Drive, Suite 160
Addison, TX 75001
Tel: 972-818-7423 Fax: 972-818-2924

Detroit
Tri-Atria Office Building
32255 Northwestern Highway, Suite 190
Farmington Hills, MI 48334
Tel: 248-538-2250 Fax: 248-538-2260

Kokomo
2767 S. Albright Road
Kokomo, Indiana 46902
Tel: 765-864-8360 Fax: 765-864-8387

Los Angeles
18201 Von Karman, Suite 1090
Irvine, CA 92612
Tel: 949-263-1888 Fax: 949-263-1338

New York
150 Motor Parkway, Suite 202
Hauppauge, NY 11788
Tel: 631-273-5305 Fax: 631-273-5335

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

Toronto
6285 Northam Drive, Suite 108
Mississauga, Ontario L4V 1X5, Canada
Tel: 905-673-0699 Fax: 905-673-6509

ASIA/PACIFIC

Australia
Microchip Technology Australia Pty Ltd
Suite 22, 41 Rawson Street
Epping 2121, NSW
Australia
Tel: 61-2-9868-6733 Fax: 61-2-9868-6755

China - Beijing
Microchip Technology Consulting (Shanghai) Co., Ltd., Beijing Liaison Office
Unit 915
Bei Hai Wan Tai Bldg.
No. 6 Chaoyangmen Beidajie
Beijing, 100027, No. China
Tel: 86-10-85282100 Fax: 86-10-85282104

China - Chengdu
Microchip Technology Consulting (Shanghai) Co., Ltd., Chengdu Liaison Office
Rm. 2401, 24th Floor,
Ming Xing Financial Tower
No. 88 TIDU Street
Chengdu 610016, China
Tel: 86-28-6766200 Fax: 86-28-6766599

China - Fuzhou
Microchip Technology Consulting (Shanghai) Co., Ltd., Fuzhou Liaison Office
Unit 28F, World Trade Plaza
No. 71 Wusi Road
Fuzhou 350001, China
Tel: 86-591-7503506 Fax: 86-591-7503521

China - Shanghai
Microchip Technology Consulting (Shanghai) Co., Ltd.
Room 701, Bldg. B
Far East International Plaza
No. 317 Xian Xia Road
Shanghai, 200051
Tel: 86-21-6275-5700 Fax: 86-21-6275-5060

China - Shenzhen
Microchip Technology Consulting (Shanghai) Co., Ltd., Shenzhen Liaison Office
Rm. 1315, 13/F, Shenzhen Kerry Centre,
Renminnan Lu
Shenzhen 518001, China
Tel: 86-755-2350361 Fax: 86-755-2366086

Hong Kong
Microchip Technology Hongkong Ltd.
Unit 901-6, Tower 2, Metroplaza
223 Hing Fong Road
Kwai Fong, N.T., Hong Kong
Tel: 852-2401-1200 Fax: 852-2401-3431

India
Microchip Technology Inc.
India Liaison Office
Divyasree Chambers
1 Floor, Wing A (A3/A4)
No. 11, O'Shaugnessey Road
Bangalore, 560 025, India
Tel: 91-80-2290061 Fax: 91-80-2290062

Japan
Microchip Technology Japan K.K.
Benex S-1 6F
3-18-20, Shinyokohama
Kohoku-Ku, Yokohama-shi
Kanagawa, 222-0033, Japan
Tel: 81-45-471- 6166 Fax: 81-45-471-6122

Korea
Microchip Technology Korea
168-1, Youngbo Bldg. 3 Floor
Samsung-Dong, Kangnam-Ku
Seoul, Korea 135-882
Tel: 82-2-554-7200 Fax: 82-2-558-5934

Singapore
Microchip Technology Singapore Pte Ltd.
200 Middle Road
#07-02 Prime Centre
Singapore, 188980
Tel: 65-6334-8870 Fax: 65-6334-8850

Taiwan
Microchip Technology Taiwan
11F-3, No. 207
Tung Hua North Road
Taipei, 105, Taiwan
Tel: 886-2-2717-7175 Fax: 886-2-2545-0139

EUROPE

Denmark
Microchip Technology Nordic ApS
Regus Business Centre
Lautrup hoj 1-3
Ballerup DK-2750 Denmark
Tel: 45 4420 9895 Fax: 45 4420 9910

France
Microchip Technology SARL
Parc d'Activite du Moulin de Massy
43 Rue du Saule Trapu
Batiment A - ler Etage
91300 Massy, France
Tel: 33-1-69-53-63-20 Fax: 33-1-69-30-90-79

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

Italy
Microchip Technology SRL
Centro Direzionale Colleoni
Palazzo Taurus 1 V. Le Colleoni 1
20041 Agrate Brianza
Milan, Italy
Tel: 39-039-65791-1 Fax: 39-039-6899883

United Kingdom
Arizona Microchip Technology Ltd.
505 Eskdale Road
Winnersh Triangle
Wokingham
Berkshire, England RG41 5TU
Tel: 44 118 921 5869 Fax: 44-118 921-5820

03/01/02