```
      TITLE       "16F684 mobile Decoder w/EMF feedback Program"
;
;
;
;-----------------------------------------------------------------------
; Change history (check also in sourcesafe)
;-----------------------------------------------------------------------

;
; 17/05/02  Added fixes to DCC->analog. Now tests and if speed is 0 will allow periodic
WDT to revise direction.
;           RP9.2.4B working correctly  (speed control in DC poor but runs OK as far as
direction goes)
;           If DC is 'marginal' i.e. very slow, it sometimes doesn't reverse.
; 18/05/02  This is the original int211 but with Gil's mods for the 'drop out' problem
;           Works correctly.
;           Page mode problem fixed
;           EMF_control and integral changed to latest version
;           Works fine. Added my speed table and CV defaults. Not to RP recommendations
but
;           optimised for can motors.
;           Save as dec132. This is the RP9.2.4 compliant version. Analog running in
pure DC only
;           Analog can be disabled with CV29.
; 03/09/02  Changes to hard reset and action on setting CV1.
;           Fixed jump table problem
;           Now version 132 rev b
; 05/11/02  Dec133. Modified version of Dec132 for analog switch
;           using CV54 bit 7. (Looks like Dec131 in this mode)
;           Also added Gil's change in integral. x16 below speed
;           .15
;           Tested and working.
; 24/11/02  Fix to CV29 default when changing CV1. Seems OK now.

;-----------------------------------------------------------------------

      list      p=16f684                  ; list directive to define processor
      #include <p16F684.inc>        ; processor specific variable definitions
      errorlevel  -302                    ; suppress message 302 from list file

 __config  _INTOSCIO & _WDT_OFF & _PWRTE_OFF & _MCLRE_OFF & _CP_OFF & _BOD_Off &
_IESO_OFF & _FCMEN_OFF

;-----------------------------------------
; constants and mem locations

; Controller conversion point from PDFF (PI+) to PI, to prevent overflow

KFR_LIMIT  equ    d'50'


; CV7 and CV8 implemented as constants to save EEPROM space
;
VERSION_VAL equ   d'2'
MAN_ID_VAL  equ   d'0'
Version           equ   h'F6' ; CV special mark (number -1)
```

```
ManId       equ   h'F7'

;
; FLG == |LV-7|VP-6|VB-5|LS-4|H-3|SC-0:2|
;
LV    equ   7           ; Last bit value
VP    equ   6           ; Valid Packet
VB    equ   5           ; Valid Byte
H     equ   3           ; Half-Bit
H_msk equ   b'00001000' ; mask of half-bit
LS    equ   4           ; Last sample
SC_msb      equ   2           ; sample count MSB
LS_msk      equ   B'00010000' ; mask for last sample
;

;-----------------------------------------------
; FLG2 = POL|AG|DG|DIRF|DIRC|SM2|SM|AM
AM          equ   0           ; Analog Mode
AM_msk            equ   b'00000001'
SM          equ   1           ; sample flag
SM2         equ   2           ; sample lock
DIRC        equ   3           ; direction change slow-down
DIRF        equ   4           ; current direction
DIRF_msk    equ   b'00010000'
DG          equ   5           ; digital mode
DG_msk            equ   b'00100000'
AG          equ   6           ; digital following analog (unidirectional)
AG_msk            equ   b'01000000'
POL         equ   7           ; DC polarity changed
POL_msk           equ   b'10000000'
;-----------------------------------------------
; pwmmode = |DCmode|0|HF|EMF|sw0-sw3
PWM_HF            equ   5     ; High freq. PWM
PWM_HF_msk  equ   b'00100000'
PWM_EMF           equ   4     ; enable feedback
PWM_EMF_msk equ   b'00010000'
                        ; sw0-sw3 speed step below of which to switch to low freq pwm
PWM_DCmode  equ   7
PWM_DCmode_msk    equ   b'10000000'
;-----------------------------------------------
; Effect bits
;
EF_SPD            equ   7     ; Speed related counter (by pwm)
EF_SPD_msk  equ   b'10000000'
EF_QA       equ   6     ; Qtr Sec phase A
EF_QA_msk   equ   b'01000000'
EF_QB       equ   5     ; Qtr Sec phase B
EF_QB_msk   equ   b'00100000'
EF_Fwd            equ   4     ; Fwd ON
EF_Fwd_msk  equ   b'00010000'
EF_Rev            equ   3
EF_Rev_msk  equ   b'00001000'
EF_MARs           equ   2     ; Off-Dim-On-Dim-
EF_MARs_msk equ   b'00000100'
EF_ST       equ   1     ; Strobe Light
```

```
EF_ST_msk   equ   b'00000010'
EF_Dim      equ   0
EF_Dim_msk  equ   b'00000001'

;-----------------------------------------------
; Logical Function control bits
;
#DEFINE           FN_NUM      5     ; set this to the total no. of functions (including
fl/rl)

FN_FL       equ   0
FN_RL       equ   1
FN_F1       equ   2
FN_F2       equ   3
FN_F3       equ   4
FN_F4       equ   5
FN_F5       equ   6
FN_RST      equ   7

OUT1        equ   0
OUT1_msk    equ   b'00000001'
OUT2        equ   1
OUT2_msk    equ   b'00000010'
OUT3        equ   2
OUT3_msk    equ   b'00000100'
OUT4        equ   3
OUT4_msk    equ   b'00001000'
OUT5        equ   4
OUT5_msk    equ   b'00010000'
OUT6        equ   5
OUT6_msk    equ   b'00100000'
OUT7        equ   6
OUT7_msk    equ   b'01000000'
OUT8        equ   7
OUT8_msk    equ   b'10000000'

;-----------------------------------------------
; PIO bits
;
;  The following definitions should be changed
;  according to pin assignments

;RA0 -13 - rail input pin/DAT
;RA1 -12 - CLK
;RA2 -11 - Function 2
;RA3 - 4 - VPP
;RA4 - 3 - Function 1
;RA5 - 2 - reverse light
;
;RC0 -10- feetback input
;RC1 - 9 - forward light
;RC2 - 8 - forward dir
;RC3 - 7 - backward dir
;RC4 - 6 - serial port
;RC5 - 5 - pwm signal
;
```

```
#DEFINE     OUT_fl_port       PORTC       ; *
OUT_fl                  equ   1                   ; forward light
OUT_fl_msk  equ b'00000010'               ; *
#DEFINE     OUT_rl_port       PORTA       ; *
OUT_rl                  equ   5                   ; reverse light
OUT_rl_msk  equ b'00100000'               ; *
#DEFINE     OUT_f1_port       PORTA       ; *
OUT_f1                  equ   4                   ; Function 1
OUT_f1_msk  equ b'00010000'         ; *
#DEFINE     OUT_f2_port       PORTA       ; *
OUT_f2            equ   2                 ; Function 2
OUT_f2_msk  equ b'00000100'         ; *
;#DEFINE    OUT_f3_port       PORTC       ; *
;OUT_f3           equ   2                 ; Function 3
;OUT_f3_msk equ b'00000100'         ; *

;
; Set the following mask as a combination of all the outputs controlled by logical Fn's
in PORTA
;
PORTC_FN_msk       equ         OUT_fl_msk                ;|OUT_f2_msk | OUT_f3_msk
PORTA_FN_msk       equ   OUT_f1_msk |OUT_f2_msk | OUT_rl_msk
;
;
;
;
#DEFINE     OUT_PWM_port      PORTC
OUT_PWM           equ   5                       ; pwm signal
OUT_PWM_msk equ   b'00100000'
#DEFINE     OUT_fwd_port            PORTC       ; *
OUT_fwd           equ   2                       ; forward dir
OUT_fwd_msk       equ   b'00000100'
#DEFINE     OUT_back_port           PORTC       ; *
OUT_back          equ   3                       ; backward dir
OUT_back_msk      equ   b'00001000'                   ; *
#DEFINE     IN_port                 PORTA       ; *
IN_bit                  equ   0                       ; rail input pin
IN_bit_msk        equ   b'00000001'                   ; *
#DEFINE     BEMF_port         PORTC
IN_BEMF           equ         0
IN_BEMF_msk equ         b'00000001' ; feetback input


PORTC_Input_msk               equ         IN_BEMF_msk
PORTA_Input_msk               equ         IN_bit_msk
PORTC_In_Analog_msk     equ         IN_BEMF_msk


Motor_Dmp_msk     equ   OUT_fwd_msk | OUT_back_msk
;
;
; Set the following mask with 1's in every position where an output is active low
;
```

```
Out_Active_Low    equ   0
;


;------------------------
; Timing constants
;
FREQ        equ   d'8'  ; MHz (16, 20, 24)
SAMPLE_PRD  equ   d'44' ; 44 usec
PWM_CNT_PRESET    equ   d'80' ; the number of 44usec cycles to increment/decrement PWM
threshold (256 increments)
                        ;    - based on formula t(incr) = (CV#3 + CV#23) * t = [(CV#3 +
CV#23) * .896] / 256
                        ;    - PWM_CNT_PRESET = t / 44us =  .896s / (256 * 44us)
CHAF_PRESET_CNT   equ   d'255'      ; 255 * 80 * 44 == .9 sec
SVC_TO_PRESET     equ   d'6'  ;  6 * 80 * 44 ==  20 ms
STEP_MULTIPLIER equ     d'151'  ; 255*16/151 = 27
PREAMBLE_CNT      equ   d'18' ; counter for min. preamble half-bits (for sync).

;------------------------
; Command bits
;
CMD_dir           equ   5
CMD_dir_msk equ   b'00100000'
;-------------------------------------
; volatile memory
;
      CBLOCK      0x0038
; PWM variables
            PWM_step    ; current step256                   20
            PWM_cnt           ; counter                           21
            PWM_step_tgt      ; for accel/decel effect                  22
            PWM_thr           ; current threshold                       23

            PWM_Accel_cnt     ; counts the 44us cycles between accel updates  24
            PWM_timer   ; contents of CV3 or CV4, decremented to 0      25
            TStep       ; time of speed step in usec              26
            PWM_t       ; accumulated time of step                27
            state       ; state machine state                     28
            SVC_timer   ; expiration of svc mode                  29
            Effect_cnt  ; rolls ~every sec                        2a
;-------------------------------------------------------------------------------
;     .5s | .25sA | .25sB | Fwd | Rev | Mars | Strb | Dim
;         00000000 - On/Off all the time.
            ON_Effect   ; mask for special effects                2b
;
; Byte variables
            Buf         ; byte buffer   addr                      2c
            B_cnt       ; counter for packet length               2d
            Err         ; error byte                              2e
; ----- Addr bytes ------
            Addr2_b           ; cover extended addressing               2f
            Addr_b            ; packet bytes - address                  30
; ----- Instruction bytes + error byte -------
            Cmd_b       ; " - command                             31
            Chk_b       ; " - check                         32
```

```
            Sup_1       ; long instructions                         33
            Sup_2       ;                                   34
            LastAddr2_b ;                                   35
            LastAddr_b  ;                                   36
            LastCmd_b   ;                                   37
            LastChk_b   ;                                   38
            LastSup1    ;                                   39
            LastSup2    ;                                   3a
;-------------------------------------------------
; Configuration Vars
;

            FLG         ; Flag word (byte)                          3b
            FLG2        ; second Flag                               3c
; volatile memory

            MEM_FILE_S:0      ; start of file
            Active_addr ; stores the active address of decoder          3d
            Decoder_addr      ; -"- decoder original addr               3e
            Arg1        ; general purpose args                      3f
            Arg2        ;                                   40
            Arg3        ;                                   41

            Arg4        ;                                   42
            Arg5        ;                                   43
            Arg6        ;                                   44
            DEC_STATUS  ; Decoder status word                       45
            Page_low    ; Register paging - 53,54                   46
            Page_high   ;                                   47
            B_rem       ; Bytes roll counter                        48
            Last_spd    ; Saved speed for half-steps                49
            Active_addr_x     ; extended addressing                     4a
            Decoder_addr_x    ; -"- decoder original addr               4b
            DEC_config  ; ram copy of Config_data_1 (CV29)          4c
            Fn_control  ; master on/off of functions                4d
            Sec_cnt           ; T/O counter (seconds)                   4e
            pwmmode           ; pwm mode byte                           4f
            magic       ; reset condition magic byte                50
            Chaf_cnt    ; for speed-dependent functions (sound)         51
            EEAddrBuf   ; buffer for EE address                     53
            EMFcutoff   ; RAM copy of CV10                          54
            PWM_val           ; calculated desired PWM value                  55
            PWM_thr_half
            Integ       ; integral accumulator of BEMF                  56
            PWM_step_tg1      ; temporary target speed for chg direction      57
            EMF_Ki            ; controller constants                    58
            EMF_Kp            ;                                 59
            EMF_Kfr           ;                                 5a
            EMF_val           ; last sampled value of BEMF              5b
            Vmin        ;                                   5c
            Vmax        ;                                   5d
            EEdata_temp ; temp. store for EE written data           5e
      ENDC
      CBLOCK      0x07A
            _w          ; save place for W                          70
            _status           ; " - status                              71
```

```
            _pclath           ; " - pclath                                72
      ENDC
;-------------------------------------
; Decoder status bits
; |SYNC|BP|DP|CR|NG|AC|CO|SV|
DEC_STAT_SV       equ   0     ; Service mode bit
DEC_STAT_SV_msk         equ   b'00000001'
DEC_STAT_CO       equ   1     ; Consist Control Active bit
DEC_STAT_CO_msk         equ   b'00000010' ; mask
DEC_STAT_AC       equ   2     ; sent to consist addr
DEC_STAT_AC_msk         equ   b'00000100'
DEC_STAT_NG       equ   3     ; negative number (for spline)
DEC_STAT_NG_msk         equ   b'00001000'
DEC_STAT_CR       equ   4     ; Consist reverse bit
DEC_STAT_CR_msk         equ   b'00010000'
DEC_STAT_DP       equ   5     ; Dup (repeat) packet
DEC_STAT_DP_msk         equ   b'00100000'
DEC_STAT_BP       equ   6     ; Begin of packet
DEC_STAT_BP_msk         equ   b'01000000'
DEC_STAT_SYNC           equ   7     ; SYNC mode
DEC_STAT_SYNC_msk equ   b'10000000'
;-------------------------------------
; Decoder Config bits
; |DT|0|AA|ST|AK|PW|FL|DR|
DEC_CFG_DR        equ   0     ; Direction. 0=Normal 1=Reversed
DEC_CFG_FL        equ   1     ; Lights Control Bit. 0=in SpeedDir instruction 1=bit 4
of Fn group 1
DEC_CFG_PW        equ   2     ; Analog Power conversion control
DEC_CFG_AK        equ   3     ; Advanced Ack
DEC_CFG_ST        equ   4     ; Speed table / polynomial
DEC_CFG_AA        equ   5     ; Extended Addressing
DEC_CFG_DT        equ   7     ; Decoder Type. 0=multifunction 1=accessory
;-------------------------------------
; EEPROM Data
;
;****** warning: locations in this list are critical.
;****** add vars at the end only
;
      CBLOCK      h'0'        ; EEPROM addresses

            PRIM_Addr   ;CV1  = 0
            V_Start           ;CV2
            ACC_Rate    ;CV3
            DEC_Rate    ;CV4
            V_High            ;CV5
            V_Mid       ;CV6
;           Version           ;CV7   **** implemented as constants
;           ManId       ;CV8
            PWMTot            ;CV9  = 6
            EMFCut            ;CV10
            Packet_TO   ;CV11 = 8
; CV14-16 reserved
            ExtAddr2    ;CV17 = 9 MSB
            ExtAddr1    ;CV18
            Consist_addr      ;CV19 = 11
; CV20 reserved
```

```
            CAct_F1_F8  ;CV21 = 12
            CAct_LMP    ;CV22
            Accel_Adj   ;CV23
            Decel_Adj   ;CV24
            CABSPD_Step ;CV25 = 16
; CV26-28 reserved
            Config_data_1     ;CV29 = 17
            Err_Info    ;CV30 = 18
;
            FL_loc            ;CV33 = 19
            RL_loc            ;CV34 = 20
            F1_loc            ;CV35 = 21
            F2_loc            ;CV36 = 22
            F3_loc            ;CV37 = 23
;
            FL_Effect   ;CV49 = 24
            RL_Effect   ;CV50 = 25
            F1_Effect   ;CV51 = 26
            F2_Effect   ;CV52 = 27
            F3_Effect   ;CV53 = 28
            PWM_Mode    ;CV54 = 29
            Ki          ;CV55 = 30
            Kp          ;CV56 =     31
            Kfr         ;CV57 = 32
            Reserv1           ;CV58 = 33
            Reserv2           ;CV59 = 34
            Reserv3           ;CV60 = 35

            SPD_Tbl:1C  ;CV67 - 94 = 36 - 63 (28 values)
            MEM_FILE_E  ; end of file
      ENDC


BANK1 MACRO
      BSF         STATUS, RP0             ;MACRO TO SELECT BANK 1
      ENDM

BANK0 MACRO
      BCF         STATUS, RP0             ;MACRO TO SELECT BANK 0
      ENDM




;-----------------------------------
; code starts here
;
            org 0
powerup           clrf  PCLATH
            clrf  Page_high
            clrf  Page_low
            goto  init_start
```

```
            org 4
;--------------------------------------
; Interrupt every 22us during sync,
; then drop rate to 1/44us
;

Int         movwf _w                            ;0
            swapf STATUS,     w                       ;1
            clrf  STATUS                              ;2
            movwf _status                             ;3
            movf  PCLATH, w                     ;4
            movwf _pclath                             ;5
            clrf  PCLATH                              ;6
            bcf   INTCON, T0IF                        ;
            movlw d'256' + d'14' - (SAMPLE_PRD * FREQ / 4);7
            btfsc DEC_STATUS, DEC_STAT_SYNC           ;8
            addlw SAMPLE_PRD * FREQ / 8                     ;9
            movwf TMR0              ; restart count   ;10
; pwm
            btfss DEC_STATUS, DEC_STAT_SYNC           ;11
            goto  l_Int_1                             ;12
            movlw H_msk                         ;13   in sync mode for pwm, divide
sample rate by 2
            xorwf FLG,        F                 ;14
            btfss FLG,        H                 ;15
            goto  l_Int_2
;
l_Int_1           incf  PWM_Accel_cnt,    F                 ;  This is handled in pwm
routine
            movlw SAMPLE_PRD                    ;
            addwf PWM_t,      F                       ;  add time in us
;
l_Int_2
            movf  FLG,        W                 ; * The following change made to allow
any pin/port as input
            btfsc IN_port,    IN_bit
            xorlw LS_msk
            andlw LS_msk
            btfss STATUS,           Z                 ; sample == last sample
            goto  l_Int_state
            btfss FLG,        SC_msb                  ; count up to 4 and stay at 4
            incf        FLG,        F
            call        Int_pwm
            goto  Int_end
;
; state machine dispatch
;
l_Int_state
            clrwdt                                    ; cleared only if signal is
arriving
            xorwf FLG,        F
            movf  state,            W
            addwf PCL,        F
            goto  l_state_sync0                       ; reset sync
            goto  l_state_sync1
            goto  l_state_sync2
            goto  l_state_sync3
            goto  l_state_hbit
;           goto  l_state_bit
```

```
;
l_state_bit call  Int_pwm
            bcf   FLG,        LV                              ;
            btfss FLG,        SC_msb      ; is samp no. >= 4          ;
            bsf   FLG,        LV
            call  byte
            addwf PCL,        F
            goto  l_state_sync0           ; ret w==0 for error
            decf  state,            F     ; next wait for hbit
            goto  l_clr_count


;-------------------------------------------
;
; calculate MARs
MARs        swapf   Effect_cnt ,    w
            andlw   0x0F
            addwf   PCL,      f

            retlw   0x0F
            retlw   0x07
            retlw   0x07
            retlw   0x03
            retlw   0x03
            retlw   0x01
            retlw   0x01
            retlw   0x00
            retlw   0x00
            retlw   0x01
            retlw   0x01
            retlw   0x03
            retlw   0x03
            retlw   0x07
            retlw   0x07
            retlw   0X0F

;
;----------------------------------------
;  Decoder control instructions
;
decod_ctl   btfsc Cmd_b,      4
            goto  consist_ctl
            movlw h'f'
            andwf Cmd_b,      W     ; get low nibble
            addwf PCL,F       ; jump table
            goto  start       ; 0000 decoder reset
            goto  hard_reset  ; 0001 hard reset
            goto  l_roll            ; 0010 reserved
            goto  l_roll            ; 0011  "
            goto  l_roll            ; 0100  "
            goto  l_roll            ; 0101  "
            goto  clr_adv_ack ; 0110  adv ack not supported
            goto  set_adv_ack ; 0111   "
            goto  l_roll            ; 1000  reserved
            goto  l_roll            ; 1001  "
```

```
            goto  clr_adv_addr      ; 1010  ext address
            goto  set_adv_addr      ; 1011  "
            goto  l_roll            ; 1100  reserved
            goto  l_roll            ; 1101  "
            goto  l_roll            ; 1110  "
            goto  ack         ; 1111 acknowledge (adv ack n/a)
;--------------------------------
; main dispatcher
;
l_disp_2    movf  Addr_b,           W
            xorwf Decoder_addr,     W     ; is it Decoder addr in consist mode
            btfss STATUS,           Z     ;
            goto  l_disp_svc
            movf  Addr2_b,    W
            xorwf Decoder_addr_x, W
            btfss STATUS,     Z
            goto  l_disp_svc
            movf  Cmd_b,            W
            andlw b'11000000'       ; is cmd == 10x (function group 1-2)
            xorlw b'10000000'
            btfsc STATUS,     Z
            goto  l_match
;
l_disp_svc  movf  Addr2_b,    W  ; test for ext. byte == 0
            btfss STATUS,           Z
            goto  p_delay_start
            movlw h'F0'
            andwf Addr_b,     W
            xorlw h'70'       ; test for service mode
            btfss STATUS,     Z     ; if Addr_b == 0x70 (C==1) go to service mode
            goto  p_delay_start     ; no match, release packet buffer
            goto  svc_mode
;-----------------------------------------

; dispatch- main entry
;
l_dispatch

;
l_disp_3    bcf   DEC_STATUS, DEC_STAT_AC
            movf  Addr_b,     W
            iorwf Addr2_b,    W
            btfsc STATUS,           Z     ; if addr == broadcast
            goto  l_match
            movf  Addr_b,           W
            xorwf Active_addr,      W     ; is it the active addr (decoder/consist)?
            btfss STATUS,           Z
            goto  l_disp_2
            movf  Addr2_b,    W
            xorwf Active_addr_x,    W     ; support ext. addr.it
            btfss STATUS,     Z
            goto  l_disp_2
            btfsc DEC_STATUS,       DEC_STAT_CO
            bsf   DEC_STATUS, DEC_STAT_AC
; address match
```

```
l_match           clrf  Sec_cnt           ; clear the Timeout counter
            movlw SVC_TO_PRESET     ; preset the SVC mode timer
            movwf SVC_timer
            call  isRepeatPacket    ; is it a dup packet ?
            movf  Cmd_b,      F     ; test if reset - if not, clear service mode bit
            btfss STATUS,     Z
            bcf   DEC_STATUS, DEC_STAT_SV
;
l_parse           clrf  B_rem
            incf  B_rem,      F     ; count first byte to roll
            swapf Cmd_b,      F     ; get major opcode
            rrf   Cmd_b,      W
            swapf Cmd_b,      F
            andlw h'07'
            addwf PCL,F       ; jump Table
            goto  decod_ctl   ; 000 decoder control / consist control
            goto  adv_op            ; 001 advanced op
            goto  speed_dir   ; 010 speed & direction
            goto  speed_dir   ; 011   "
            goto  func_1            ; 100 Function group 1
            goto  func_2            ; 101 Function group 2
            goto  p_sync_start      ; 110 reserved
;           goto  CV_access_long    ; 111 Config Var access, long form / short form
;---------------------------------------
; Direct access to CVs - long form
;
CV_access_long    incf  B_rem,      F
            btfsc Cmd_b,      4
            goto  CV_access_short
;
l_cvsl_start      incf  B_rem,      F
            btfsc DEC_STATUS, DEC_STAT_AC
            goto  l_roll                ; not active if sent to consist address
            movf  Chk_b,      w         ; get low CV# bits
            movwf Arg1
            movf  Cmd_b,      w
            andlw b'00000011'       ; get high CV# bits
            call  CV_to_addr
            movwf EEAddrBuf
            xorlw h'FF'             ; check illegal CV#
            btfsc STATUS, Z
            goto  l_roll
            movf  Sup_1,      w
            movwf Chk_b             ; make same as CV access short
            rrf   Cmd_b,      f
            rrf   Cmd_b,      w
            andlw b'00000011'       ; get cmd bits
            addwf PCL,  f
            goto  l_roll                ; reserved
            goto  l_cvsl_vf
            goto  l_cvsl_bit
            goto  l_cvsl_wr
;
l_cvsl_wr   btfss DEC_STATUS, DEC_STAT_DP ;test for repeat packet
            goto  l_roll                ;if got here, do nothing and wait for repeat
```

```
            movf  Chk_b,      w
l_cvsl_wr1  call  cvwr_ok
            addwf PCL,  f
            goto  l_roll
            goto  l_cvsl_wr2
            movf  EEdata_temp,      W     ; write CV1
            call  ee_write
            movlw Consist_addr            ; clear consist address
            movwf EEAddrBuf
            clrw
            call  ee_write
            movlw Config_data_1
            call  ee_read
            andlw   B'11011111'           ;clear extended addressing
            movwf DEC_config
            call  write_config
            goto  ack
l_cvsl_wr2  movf  EEdata_temp,      W
            call  ee_write
            goto  ack
;
l_cvsl_vf1  movwf EEAddrBuf
;
l_cvsl_vf   call  ee_read_1
            xorwf Chk_b,      w
            btfsc STATUS,     Z
            goto  ack
            goto  l_roll
;
l_cvsl_bit  movf  Chk_b,      w
            andlw b'00000111'
            movwf Arg4
            incf  Arg4, f
            clrf  Arg3
            bsf   STATUS,     C
;
l_cvsl_lp1  rlf   Arg3, f     ; get a 1 bit into position in Arg3
            decfsz      Arg4, f
            goto  l_cvsl_lp1
            call  ee_read_1
            movwf Arg4
            btfsc Chk_b,      4
            goto  l_cvsl_wbit
            movf  Arg3, w     ; load bit mask-> w
            andwf Arg4, w
            btfsc Chk_b,      3     ; if we test for 0, the value in w should be 0
            xorwf Arg3, w     ; if we test for 1, invert the bit
            btfsc STATUS,     Z
            goto  ack
            goto  l_roll
;


;---------------------------------------
; Direct access to CVs - short form
```

```
;
CV_access_short   movf  Cmd_b,      w
            andlw 0x7
            addwf PCL,  F
            goto  l_cvs_data  ; 000 data reg 0
            goto  l_cvs_data  ; 001 data reg 1
            goto  l_cvs_data  ; 010 data reg 2
            goto  l_cvs_data  ; 011 data reg 3
            goto  l_cvs_config      ; 100 Config CV29
            goto  l_cvs_Page  ; 101 page reg
            goto  l_cvsl_vf1  ;l_cvs_Version    ; 110 version # CV7
            goto  l_cvsl_vf1  ;l_cvs_ID   ; 111 manufacturer ID CV8
;
l_cvs_data  addwf Page_low, W
            btfss DEC_STATUS, DEC_STAT_SV       ; are we in service mode
            addlw d'20'                   ; make it CV#21 - 24 (std is 23,24)
            movwf Arg1
            movf  Page_high,W
            call  CV_to_addr
            goto  l_cvs_op
;
l_cvs_config      movlw Config_data_1     ; modify basic config CV29
            goto  l_cvs_op
;
l_cvs_Page  decf  Chk_b,      F     ; Page--
            clrf  Arg1        ; (Page# - 1) *4, high & low
            bcf   STATUS,     C
            rlf   Chk_b,      F
            rlf   Arg1, F
            rlf   Chk_b,      F
            rlf   Arg1, F
            btfss Cmd_b,      3     ; page is in mem, not EEPROM
            goto  l_cvs_pg_vf
            movf  Chk_b,      W     ; write value
            movwf Page_low
            movf  Arg1, W
            movwf Page_high
            goto  ack
;
l_cvs_pg_vf movf  Chk_b,      W
            xorwf Page_low, W
            btfss STATUS,     Z
            goto  l_roll
            movf  Arg1, W
            xorwf Page_high, W
            btfss STATUS,     Z
            goto  l_roll            ; not verified
            goto  ack

;
l_cvs_op    movwf EEAddrBuf
            xorlw h'FF'
            btfsc STATUS,     Z
            goto  l_roll
            btfss Cmd_b,      3     ; test op code
```

```
            goto  l_cvsl_vf
            goto  l_cvsl_wr

l_cvsl_wbit btfss DEC_STATUS, DEC_STAT_DP
            goto  l_roll            ;if got here, do nothing and wait for repeat
            movf  Arg3, W
            iorwf Arg4, W     ; get the orig value and set bit to 1
            btfss Chk_b,      3
            xorwf Arg3, W     ; clear the bit
            goto  l_cvsl_wr1
;-----------------------------------------
; Service Mode
;
svc_mode    movlw SVC_TO_PRESET     ; preset the SVC mode timer
            movwf SVC_timer
            bsf   DEC_STATUS, DEC_STAT_SV
            clrf  Sec_cnt           ; reset timeout counter
            call  isRepeatPacket    ; only act on repeat packets
            btfss DEC_STATUS, DEC_STAT_DP
            goto  p_delay_start
            movf  Chk_b,      W
            movwf Sup_1
            movf  Cmd_b,      W
            movwf Chk_b
            movf  Addr_b,     W
            movwf Cmd_b
            btfsc B_cnt,      1     ; is B_cnt <= 1
            goto  l_cvsl_start
            goto  CV_access_short
;-----------------------------------------


; validate CV write
cvwr_ok           movwf EEdata_temp
            movlw Version           ; check if addr == CV7 or CV8 (read only)
            xorwf EEAddrBuf, W
            andlw 0xFE        ; mask bit 0
            btfsc STATUS, Z
            retlw 0
            movf  EEAddrBuf, W
            sublw ExtAddr2    ;is it CV17 - ext address hi byte
            btfss STATUS,     Z
            goto  l_cvwr_1    ; continue tests
            movlw B'11000000'       ;0xC0 minimum
            subwf EEdata_temp,W
            btfss STATUS,     C
            retlw 0           ;not valid
            movlw B'11101000' ;< 0xE8 (0xE7 maximum)
            subwf EEdata_temp,W
            btfsc STATUS, C
            retlw 0           ;not valid
            retlw 1

l_cvwr_1    movf  EEAddrBuf, W
            sublw PRIM_Addr
```

```
            btfss STATUS, Z   ;is it CV1
            retlw 1
            btfss EEdata_temp,7     ;127 maximum
            retlw 2
            retlw 0
;-----------------------------------------
; bit recognizer state machine
;
l_state_sync0     clrf  state
            bsf   DEC_STATUS, DEC_STAT_SYNC
            movlw PREAMBLE_CNT            ; reset hbit count for sync
            movwf B_cnt
            clrf  Err
            movlw Addr2_b                 ; reset
            movwf FSR
;
l_next_state      incf  state,            F     ; state = sync1
;
l_clr_count movlw b'11111000'
            andwf FLG,        F     ; clear sample counter
            incf  FLG,        F     ; count 1st sample
;
Int_end           movf  _pclath,    W
            movwf PCLATH
            swapf _status,    W
            movwf STATUS
            swapf _w,         F
            swapf _w,         W
            retfie
;
Int_pwm           ;btfsc      FLG2, AM
            ;return
            btfss pwmmode,PWM_DCmode
            goto  int_pwm_3
            btfsc FLG2, AM
            return
int_pwm_3
            movf  TStep,      W
            subwf PWM_t,      W
            btfss STATUS,     C     ; it's time to count a step
            return

            movwf PWM_t       ; get residue
            incf        PWM_cnt, F
            incf        PWM_cnt, F  ; count 128 steps
            btfss pwmmode, PWM_EMF; is BEMF on?
            goto  l_ipwm_2

            movf  PWM_val, W
            subwf EMFcutoff, W      ; are we above EMF cutoff
            btfss STATUS, C
            goto  l_ipwm_2    ; yes. ignore feedback
;
            movlw .200        ; motor off period
            subwf PWM_cnt, W
```

```
            btfss STATUS,      C
            goto  l_ipwm_2
            movlw .232         ; sample time
            subwf PWM_cnt, W
            btfsc STATUS,      C
            btfsc FLG2,  SM2
            goto  l_Int_clr_pwm     ;
            bsf         ADCON0,  GO ; Start A/D conversion
            bsf         FLG2, SM    ; mark sample ready for EMF_control
            bsf         FLG2, SM2   ; prevent multiple samples
            goto  l_Int_clr_pwm
;
l_ipwm_2
            bcf         FLG2, SM2   ; clear for another sample
            btfss pwmmode, PWM_HF   ; is it HF PWM
            goto  l_ipwm_1    ; no. Goto LF pwm
            movf  pwmmode, W
            andlw h'0F'
            subwf PWM_val, W  ; is the speed > transition point to HF
            btfss STATUS,      C
            goto  l_ipwm_1
            movf  PWM_thr, W  ; HF pwm no need to handle here
;           movwf PWM_thr_half
;           bcf         STATUS,     C
;           rrf         PWM_thr_half,     W
            movwf CCPR1L
            return
;
l_ipwm_1    movf  PWM_thr, W
            subwf PWM_cnt, W
            btfss STATUS,          C
            goto  l_Int_set_pwm
;
l_Int_clr_pwm
            clrf  CCPR1L
            return
;
l_Int_set_pwm     movlw h'FF'       ; set to full
            movwf CCPR1L
            return
;-------------------------------------------------------------------------------
;
l_state_sync1     btfsc FLG,  SC_msb           ; 0 or 1
            goto  l_state_sync0          ; 0 hbit causes reset
            decfsz      B_cnt,           F
            goto  l_clr_count
            goto  l_next_state
;
l_state_sync2     btfsc FLG,  SC_msb
            incf  state,           F
            goto  l_clr_count
;
l_state_sync3     btfss FLG,  SC_msb
            goto  l_state_sync0          ; 1 after 0 hbit of preamble not accepted
            bcf   DEC_STATUS, DEC_STAT_SYNC
```

```
            movlw (SAMPLE_PRD * FREQ / 8) - 2
            subwf TMR0,       F     ; adjust sample time
            movlw h'FE'
            movwf Buf
            goto  l_next_state
;
l_state_hbit      incf  FLG,        F     ; add one count so that 3 samples are 4, over
the threshold
            incf  state,            F     ; do not reset the count on hbit. change
state to l_state_bit.
            call  Int_pwm
            goto  Int_end


;----------------------------------------
;  Consist control, activation/ deactivation
;
consist_ctl incf  B_rem,      F     ; 2 byte instruction
            movlw h'0E'
            andwf Cmd_b,      W
            xorlw 2           ; 001x == consist fwd/back
            btfss STATUS,     Z
            goto  l_roll
            bcf   Chk_b,      7     ; original bit 7 should be '0' ('1' is reserved)
            btfss Cmd_b,      0     ; is it normal or reversed ?
            bsf   Chk_b,      7
            movlw Consist_addr
            movwf EEAddrBuf
            movf  Chk_b,      W
            call  ee_write    ; inside, calls refresh_ram and updates all registers/
flags
            goto  l_roll

;-----------------------------------
; access routines - eeprom
;
;
ee_read
            movwf EEAddrBuf   ; address in w
;
ee_read_1
            movlw Version           ; check if addr == CV7 or CV8 (read only) which are
not implemented in EEPROM
            xorwf EEAddrBuf, W
            btfsc STATUS,     Z
            retlw VERSION_VAL
            movlw ManId
            xorwf EEAddrBuf, W
            btfsc STATUS,     Z
            retlw MAN_ID_VAL
;
            BANK0
            movf  EEAddrBuf, W
            BANK1
            movwf EEADR

            bsf   EECON1, RD
```

```
            movf  EEDAT, W
            BANK0
            return

; address in Arg1, value is in W reg.
ee_write    bcf   INTCON, GIE       ;disable int
            clrwdt
            BANK1
            movwf EEDAT
            BANK0
            movf  EEAddrBuf, W
            BANK1
            movwf EEADR

            bsf   EECON1, WREN      ;enable write

            movlw h'55'
            movwf EECON2            ; write 55
            movlw h'AA'
            movwf EECON2
            bsf   EECON1, WR  ; set wr bit, begin wr
; make sure write completed & verified
ee_wr_wait  btfsc EECON1, WR
            goto  ee_wr_wait
            bcf   EECON1,     WREN
;
ee_wr_fin
            BANK0
            bsf   INTCON, GIE ;enable int
            goto  refresh_ram ; put all time-critical values in ram copies



;-----------------------------------
; read saved CVs from eeprom
; Arg5 and Arg3 are changed
;
read_eeprom_file  movlw 1
            movwf PWM_timer
            movlw b'11000000'
            movwf DEC_STATUS
;
refresh_ram movlw PWMTot            ; (CV9 << 2) ^ 0x83
            call  ee_read
            movwf Arg5
            rlf   Arg5, F
            rlf   Arg5, F
            movlw 0x83
            iorwf Arg5, W
            movwf TStep
;
            movlw PWM_Mode
            call  ee_read
            movwf pwmmode
;
```

```
            movlw EMFCut            ; read EMF constants
            call  ee_read
            movwf EMFcutoff
            rlf   EMFcutoff, F      ; normalize to 0-254
            movlw Ki
            call  ee_read
            movwf EMF_Ki
            movlw Kp
            call  ee_read
            movwf EMF_Kp
            movlw Kfr
            call  ee_read
            movwf EMF_Kfr
;
            movlw Config_data_1     ; read config byte (CV29)
            call  ee_read
            movwf DEC_config  ; move to RAM copy
            btfss DEC_config, DEC_CFG_AA  ; is adv. addr on?
            goto  read_ef_1
            movlw ExtAddr1
            call  ee_read
            movwf Decoder_addr
            movwf Active_addr
            movlw ExtAddr2
            call  ee_read
            movwf Decoder_addr_x
            movwf Active_addr_x
            goto  read_ef_2
;
read_ef_1   clrf  Decoder_addr_x
            clrf  Active_addr_x
            movlw PRIM_Addr   ; read primary address
            call  ee_read
            movwf Decoder_addr
            movwf Active_addr
;
read_ef_2   bcf   DEC_STATUS, DEC_STAT_CO
            movlw Consist_addr      ; read consist address
            call  ee_read
            movwf Arg5
            bcf   DEC_STATUS, DEC_STAT_CR
            btfsc Arg5, 7
            bsf   DEC_STATUS, DEC_STAT_CR
            andlw b'01111111' ; is consist addr == 0 (inactive)
            btfsc STATUS,     Z
            return
            movwf Active_addr ; activate consist
            clrf  Active_addr_x     ; ext. byte is 0
            bsf   DEC_STATUS,       DEC_STAT_CO
            return
;---------------------------------------
; write contents of DEC_config into CV29
;
write_config      movlw Config_data_1     ; CV29
            movwf EEAddrBuf
```

```
            movf  DEC_config, W
            call  ee_write
            return
;-----------------------------------
; Multiply
; M1 = W; M2 = Arg1; Prod = Arg2:Arg1
; Arg3 is cleared
;
multiply          movwf Arg2
            movlw d'8'
            movwf Arg3        ; counter
            movf  Arg2,       W
            clrf  Arg2
;
l_mult_loop rrf   Arg1, F
            btfsc STATUS, C
            addwf Arg2,       F
            rrf   Arg2, F
            decfsz      Arg3,       F
            goto  l_mult_loop
            rrf   Arg1, F     ; one additional rotate
            return
;-----------------------------------
; spline calculation
; PWM_step = step256 [4 : 255]
; V(W) = {X1}  [W * (W - 128) * V_High] / (128 * 256) -
;      {X2}  [(W - 128)(256 - W)* V_Start] / (128 * 256) +
;      {X3}  [W * (256 - W) * V_Mid ] / (64  * 256)
;
splineIt    movf  PWM_step, W
            btfss STATUS, Z
            goto  l_spl_nz
            clrf  PWM_val
            return
;
l_spl_nz    btfss DEC_config, DEC_CFG_ST
            goto  l_spl_start ; calculate curve
            movwf Arg1
            movlw d'27'       ; get 28ss
            call  multiply    ; Arg2 is step28, Arg1 is fraction
            movlw SPD_Tbl + 1 ; get T[step28 +1]
            addwf Arg2, F
            movf  Arg2, W
            call  ee_read           ; read speed from table
            movwf Arg4
            decf  Arg2, W
            call  ee_read
            movwf Arg5
            subwf Arg4, W     ; W == T[step28 + 1] - T[step28]
            call  multiply    ; Arg2 = interpolation value
            movf  Arg2, W
            addwf Arg5,   W
;
l_spl_end   movwf PWM_val
            return
```

```
;
l_spl_start clrf  Arg6
            movlw V_High
            call  ee_read
            movwf Vmax
            andlw h'FE'
            btfsc STATUS, Z   ; test for default case
            movlw h'FF'
            movwf Arg1
            movwf Vmax
            bcf   STATUS, C
            rrf   Arg1, F     ; convert to 7 bit numbers
            movlw V_Start
            call  ee_read
            movwf Vmin
            movwf Arg4        ; save it for later
            bcf   STATUS,     C
            rrf   Arg4, F
            movf  Arg4, W
            addwf Arg1, F     ; guaranteed no overflow
            movf  PWM_step, W ; PWM_step is actually [W + 1] and curve starts at 1
            call  multiply    ; now Arg2== E1== W*(V_High + V_Start) / 256
            movf  Arg4, W
            subwf Arg2, W
            btfsc STATUS,     C
            goto  l_spl_c1
            incf  Arg6, F
            sublw 0
;
l_spl_c1    movwf Arg1        ; get ready to multiply by (W - 128)/128
            movlw d'128'
            subwf PWM_step, W
            btfsc STATUS, C   ; is result negative?
            goto  l_spl_c3    ; no.
            sublw 0           ; negate
            incf  Arg6, F     ; reverse sign
            bsf   Arg6, 7     ; remember graph area
;
l_spl_c3    call  multiply
            rlf   Arg1, F     ; to increase accuracy do not lose bits early
            rlf   Arg2, W     ; now w == E1 * |W - 128| / 128
            btfsc Arg6, 0     ; is result so far negative?
            sublw 0           ; negate, so we get the real comp' 2 number
            movwf Arg5
;
            movf  PWM_step, W ; PWM_step is [W + 1]
            movwf Arg1
            sublw 0
            call  multiply
            movlw V_Mid       ; calc E3 == (W * (256-W) * V_Mid) / (64 * 256)
            call  ee_read
            andlw h'FE'
            btfss STATUS,     Z     ; default case
            goto  l_spl_c4
            movf  Vmin, W     ;set mid value
```

```
            addwf Vmax, F
            rrf   Vmax, W     ;halve it

l_spl_c4    movwf Arg1
            bcf   STATUS,     C
            rrf   Arg1, F
            movf  Arg2, W
            call  multiply
            rlf   Arg1, F
            rlf   Arg2, F
            rlf   Arg1, F
            rlf   Arg2, W
            addwf Arg5,       F     ; that's it!
            btfsc Arg5,       7     ; do not want residue of overflow, so max out
            goto  l_spl_ovf
            rlf   Arg5,       W
            addlw 1
            goto  l_spl_end
;
l_spl_ovf   movlw 0     ; if underflow in left half of curve, fix to 0
            btfss Arg6, 7
            movlw h'FF' ; if overflow in right half of curve, fix to 255
            goto  l_spl_end

;-----------------------------------
;  EMF feedback control
;
;  PWM_val  - ▯▯▯▯▯▯▯▯▯▯▯▯ ▯▯▯▯▯▯▯  (C[n])
;  ADRESH  - ▯▯▯▯▯▯▯▯ ▯▯▯▯▯▯▯ EMF  (F[n])
;  PWM_thr  - ▯▯▯▯▯▯▯▯▯ ▯▯▯▯▯▯▯ PWM   (R[n])
;  Integ          -  integral accumulator of BEMF
;
;
;  Implemented a PDFF (PI+) controller
; R(z) = (Ki * Z) / (Z - 1)*[C(z) - F(z)]  + KP * [Kfr * C(z) - F(z)]
; R[n] = R[n-1] + Ki(C[n] - F[n]) + Kp * [Kfr * (C[n] - C[n-1]) - (F[n] - F[n-1])]
; if the recursive expression is expanded into a series, assuming initial constraints
0, the result is:
; R[n] = SUM(j=1,n)(Ki*(C[j] - F[j])) + Kp* (Kfr * C[n] - F[n])



EMF_control
            movf  PWM_val, W
            btfss STATUS,      Z                ; make sure a stop command is not
calculated.
            btfss pwmmode, PWM_EMF  ; is EMF mode on?
            goto  EMF_c_end
            subwf EMFcutoff, W                  ; are we above EMF cutoff speed
            movf  PWM_val, W
            btfss STATUS,     C
            goto  EMF_c_end
            btfss FLG2, SM                      ; is there a sample ready?
            return
            btfsc ADCON0, GO              ; conversion complete?
```

```
            return
;Integral
            clrf        Arg6                    ; sign buffer
            subwf ADRESH,     W             ; W = F[n] - C[n] = -(C[n] - F[n]) -
       ;
            btfsc STATUS,     C             ; if cleared error term is positive
            goto  EMF_c_8
            bsf         Arg6, 0                 ;  -  ;
            sublw 0                             ;  ;
       ;

EMF_c_8
            movwf Arg1                        ;  ;  ;
       ;
            incf        Arg1, F                 ; fix rounding errors (Ki is
255/256 ?)
            movf  EMF_Ki,     W             ;; EMF_Ki ;;  ;;,
            call        multiply                ;  ;
Arg2:Arg1
            movf  PWM_val,W
            sublw .15
            movf  Arg2, W
            btfss STATUS, C                   ;;  ;. ;  <15,
;;  ;
            goto  EMF_c_12                    ;;  ;,  ;;  ;;
            movlw h'F0'                       ; b'11110000'
            andwf Arg2, W                     ; is it going to overflow
            btfss STATUS,     Z
            goto  EMF_c_11
            movlw h'F0'                       ; the following code is mult. by 16
            andwf Arg1, F
            swapf Arg1, F
            swapf Arg2, F
            andwf Arg2, W
            iorwf Arg1, W
            goto  EMF_c_12

EMF_c_11
            movlw h'FF'

EMF_c_12
            btfsc Arg6, 0                     ; set if error term is positive
            goto  EMF_c_9
            subwf Integ,      W
            btfss STATUS, C
            clrw
            goto  EMF_c_10

EMF_c_9
            addwf Integ,      W
            btfsc STATUS, C
            movlw h'FF'
```

```
EMF_c_10
            movwf Integ                     ;
```

**********. EMF_Ki ** *******

```
;filtered proportional

            clrf        Arg6
            movf  PWM_val, W
            movwf Arg1
```

```
            movwf Arg2                  ; in case we have to skip the multiplication
            movf  EMF_Kfr, W            ; gradually fading Kfr effect (increase from
EMF_Kfr to h'ff') with demand speed
            subwf Arg2, F
            btfsc STATUS, C             ; if positive then we are under the threshold
speed
            movf  PWM_val, W
            call        multiply
;
EMF_c_7
            movf  ADRESH,     W
            subwf Arg2, W               ;W = Kfr*I[n] - F[n]
            btfsc STATUS, C             ;is it negative
            goto  EMF_c_3
            bsf         Arg6, 0
            sublw 0                           ; save sign and negate for mult. by Kp
;
EMF_c_3
            movwf Arg1
            movf  EMF_Kp, W
            call        multiply
            movlw h'F0'
            andwf Arg2, W               ; is it going to overflow
            btfss STATUS, Z
            goto  EMF_c_4
            movlw h'F0'                 ; the following code is mult. by 16
            andwf Arg1, F
            swapf Arg1, F
            swapf Arg2, F
            andwf Arg2, W
            iorwf Arg1, W
            goto  EMF_c_5
;
EMF_c_4
            movlw h'FF'
;
EMF_c_5
            btfsc Arg6, 0
            goto  EMF_c_6
            addwf Integ,      W
            btfsc STATUS,     C
            movlw h'FF'                 ;overflow
            goto  EMF_c_end
;
EMF_c_6
            subwf Integ,      W
            btfss STATUS,     C         ;underflow
            movlw 0
;
EMF_c_end
            movwf PWM_thr               ; now has the new control value
            bcf         FLG2,   SM            ; can get new sample now
            return
```

```
;-----------------------------------
; start point

hard_reset  movlw Consist_addr           ; zero in hard-reset
            movwf EEAddrBuf
            clrw
            call  ee_write
            movlw B'00000010'       ; default CV29
            movwf DEC_config
            call  write_config
            clrw
            incf  EEAddrBuf,  f          ; Clear error CV
            call  ee_write
            movlw PRIM_Addr
            movwf EEAddrBuf
            movlw 3                 ; default short address
            call  ee_write
;
init_start

                  BANK0
      CLRF  PORTA
      CLRF  PORTC
      MOVLW B'00010001'             ;A/D LEFT JUSTIFIED, VDD IS VOLTAGE REFERENCE
      MOVWF ADCON0                        ;
      CLRF  CCPR1L                        ;SET PWM DUTY CYCLE TO ZERO
      BCF         CCPR1H, 5               ;
      BCF         CCPR1H, 4               ;
      MOVLW B'00001100'             ;FULL BRIDGE FORWARD
      MOVWF CCP1CON                       ;PWM MODE P1A, P1B, P1C AND P1D ACTIVE HIGH
      CLRF  ECCPAS                        ;DISABLE THE AUTO-SHUTDOWN
      MOVLW B'00000100'             ;TMR2 POSTSCALER=1 TMR2 ON PRESCALER=1
      MOVWF T2CON
      MOVLW B'00000111'             ;COMPARATORS ARE OFF
      MOVWF CMCON0
      BANK1
      MOVLW B'00000001'             ;SET RA0  AS INPUTS
      MOVWF TRISA
      MOVLW B'00000001'             ;SET RC0  AS INPUTS
      MOVWF TRISC
      MOVLW B'11001011'             ;PORTA PULL-UP DISABLED, INT EDGE, TOCS INT, T0SE
LOW-HIGH
      MOVWF OPTION_REG              ;PSA TMR0, RATE 1:32
      MOVLW B'00000000'             ;ENABLE GLOBAL INTERRUPTS
      MOVWF INTCON                        ;ENABLE TMR0 INTERRPUT
;     CLRF  PIE1                          ;DISABLE ALL PERIPHERAL INTERRUPTS
;     CLRF  PCON                          ;DISABLE ULPWUE AND SBODEN
      MOVLW B'01110111'             ;SELECT 8MHz
      MOVWF OSCCON
      CLRF  IOCA                          ;DISABLE ALL INTERRUPT ON CHANGE FOR PORT A
      MOVLW B'00000000'             ;RC0 ANALOG INPUT
      MOVWF ANSEL
      MOVLW B'01010000'             ;SELECT A/D CONVERSION CLOCK FOSC/16
      MOVWF ADCON1                        ;CONVERSION TIME IS 2uSECS
      MOVLW B'1111011'              ;SETS THE PWM FREQ TO 16129Hz IF TRM2 PS=1
      MOVWF PR2
```

```
      BANK0
;     MOVLW .16
;     movwf CCPR1L



      movf  PCON, W
      BANK0
      bcf         INTCON, GIE
      movwf Arg4

            btfss Arg4, NOT_POR           ; Is it POR?
            goto  l_start_POR
            btfsc Arg4, NOT_BOD           ; is it BOD? -if not do a full reset.
            goto  start             ; some other reset. MCLR?
            movlw h'A5'             ; BOR. In addition check memory valid
            xorwf magic,      W
            btfsc STATUS, Z
            goto  l_start_pulse
            xorwf magic,      F
;
l_start_POR clrf  FLG2              ; DG is only cleared upon a real POR
            clrf  Fn_control
            bsf   Fn_control, FN_RST      ; signal 1st speed command is next, for
momentary reset
            clrf  PWM_step          ; initially stop.

l_start_P1  movlw b'10000000'
            movwf FLG
;
start
            bcf   INTCON, GIE       ; temp. disable timer int
            movlw b'00111111'
            movwf ON_Effect
            clrf  PWM_step_tgt
            movlw DIRF_msk | DG_msk
            andwf FLG2, F
            call  read_eeprom_file
            movlw PRIM_Addr
            call  ee_read
            btfsc STATUS, Z         ; is primary address 0?
            goto  l_analog_mode           ; yes. analog mode
            btfsc STATUS, NOT_TO          ; skip if a WDT reset
            goto  l_digital_mode

l_analog_mode
            clrwdt
;           btfss DEC_config, DEC_CFG_PW        ; get analog mode enable
            goto  l_digital_mode
;           bsf   FLG2, AM
;           btfsc pwmmode,PWM_DCmode
;           goto  l_start_pulse
;           bcf   Fn_control, FN_RST      ; no need for this if coming from analog to
DCC
;           movf  FLG2, W           ; are directions equal?
;           andlw DIRF_msk
;           xorwf Arg6, W           ; W = FLG2[DIRF] xor Arg6
```

```
;           btfss FLG2, DG
;           goto  l_start_c2
;           btfsc STATUS, Z         ; skip if no
;           goto  l_start_c3        ; set PWM = h'FF'
;           goto  l_start_pulse
;l_start_c2 xorwf FLG2, F           ; set direction to Arg7!
;l_start_c3 comf  PWM_step_tgt, F         ; This would set target speed to full scale
if direction matches
;           goto  l_start_pulse
;
l_digital_mode
            clrf  PWM_val
            clrf  PWM_thr
            clrf  Integ
            clrf  PWM_cnt
            clrf  PWM_Accel_cnt
            clrf  PORTA
            clrf  PORTC
            clrf  Sec_cnt
;
l_start_pulse
            BANK1
            bsf   PCON, 0           ; have to manually set PCON bits
            bsf   PCON, 1
            BANK0
            call  set_power
            call  pwm_setd_start          ; cause the actual direction to be set
            call  splineIt          ; set initial value for PWM_val
;
p_sync_start
      clrf  state       ; reset
;
p_delay_start
      clrf  B_cnt
;
p_delay
            movlw Addr2_b
            movwf FSR         ; reset pointer
            clrf  Err
            clrf  Sup_1
            clrf  Sup_2
            bcf   FLG,  VP    ; release packet buffers
            bsf   INTCON, T0IE
            bcf   INTCON, T0IF
            bsf   INTCON, GIE
;---------------------------------------
;  packet handler
;
packet            call  pwm
            btfsc FLG, VP           ; if no valid packet, check time-out & return or
sleep
            goto  l_dispatch
            movlw Packet_TO   ; read time-out value (sec)
            call  ee_read
            btfsc STATUS,     Z     ; is T/O active?
            goto  packet
```

```
            subwf Sec_cnt, W
            btfss STATUS, C
            goto  packet
            goto  l_spd_emr_stop



;-----------------------------------------
; update effects
;
effect_update     incf  Effect_cnt, F     ; rolls ~every 1sec
            btfsc STATUS,           Z
            incf  Sec_cnt,    F
            movlw b'00011000'
            andwf ON_Effect,  F
            movlw 0x40
            andwf Effect_cnt, W
            iorlw 0x83
            iorwf ON_Effect,  F
            bcf   ON_Effect,  EF_QB
            btfss ON_Effect,  EF_QA
            bsf   ON_Effect,  EF_QB
            call  MARs
            andwf Effect_cnt, W
            btfss STATUS,           Z
            bsf   ON_Effect,  EF_MARs
            movlw 0x06
            andwf Effect_cnt, W
            btfsc STATUS,           Z
            bcf   ON_Effect,  EF_Dim
            movlw 0x78
            andwf Effect_cnt, W
            btfsc STATUS,           Z
            bcf   ON_Effect,  EF_ST
            decfsz      Chaf_cnt,   F
            goto  l_ef_func
            bcf   ON_Effect,  EF_SPD
            rrf   PWM_thr,    W
            sublw CHAF_PRESET_CNT
            movwf Chaf_cnt

;
; handle functions
;
l_ef_func   clrf  Arg5              ; result output mask
            movlw FN_NUM
            movwf Arg6              ; Fn CV ( reversed ) index
            movlw 1
            movwf Arg3              ; used as bit mask
;
l_ef_loop   movf  Arg3,       W
            andwf Fn_control, W
            btfsc STATUS,           Z
            goto  l_ef_c1
```

```
            movf  Arg6,       W
            sublw FL_Effect + FN_NUM
            call  ee_read
            andwf ON_Effect,  W     ; if 0, turn on function outputs
            btfss STATUS,           Z
            goto  l_ef_c1
            movf  Arg6,       W
            sublw FL_loc + FN_NUM
            call  ee_read
            iorwf Arg5,       F
;
l_ef_c1           bcf   STATUS,           C
            rlf   Arg3,       F
            decfsz      Arg6,       F
            goto  l_ef_loop
;
;  when we finish this loop, Arg5 has the bits set for the output lines on/of status
;
            movlw Out_Active_Low
            xorwf Arg5,       F     ; set correct active polarity
;
;
      movf  Arg5,       W
      xorwf PORTA,            W
      andlw PORTA_FN_msk
      xorwf PORTA,            F

;no room for this bit!
;
      rlf         Arg5,       W     ; problem if want to comply w/ RP   *
;
      movf  Arg5,       W       ;                             *
      xorwf PORTC,            W     ;                               *
      andlw PORTC_FN_msk            ;                               *
      xorwf PORTC,            F     ;                               *
;
            return

;-----------------------------------------------
; Final direction change following a slow-down
;
pwm_set_dir
            btfss FLG2, DIRC
            return
pwm_setd_start
            bcf         FLG2, DIRC  ; unmark the slow-down (*)

            btfsc FLG2, DIRF
            goto  pwm_setd_fwd
;
            bcf         OUT_fwd_port,     OUT_fwd
            bsf         ON_Effect,  EF_Fwd
            bsf         OUT_back_port,    OUT_back
            bcf         ON_Effect,  EF_Rev
            goto  pwm_setd_spd
```

```
pwm_setd_fwd
            bcf         OUT_back_port,    OUT_back
            bsf         ON_Effect,  EF_Rev
            bsf         OUT_fwd_port,     OUT_fwd
            bcf         ON_Effect,  EF_Fwd
;
pwm_setd_spd
            movf  PWM_step_tgt,     W
            movwf PWM_step_tg1
            return


;-----------------------------------------------
; PWM routine - handles acceleration / deceleration
;
;
pwm         btfss FLG2, AM          ; next test for analog mode
            goto  l_pwm_3
            btfss FLG2, POL         ; make sure there is exactly one WDT following a
direction change.
            clrwdt                        ; if the flag is clr prevent WDT

l_pwm_3           movlw PWM_CNT_PRESET          ; compare to preset cycle value (for
256 increments)
            subwf PWM_Accel_cnt,    W
            btfss STATUS,           C
            goto  l_pwm_4                 ; no accel/decel
            movwf PWM_Accel_cnt           ; get the residue
;
;     effects are handled here
            call  effect_update

;
            decfsz      SVC_timer,  F     ; is svc mode expired
            goto  l_pwm_2
            bcf   DEC_STATUS,       DEC_STAT_SV ; clear svc mode
;
l_pwm_2           decfsz      PWM_timer,  F     ; count down based on CV#3 + CV#23 or
CV#4 + CV#24
            goto  l_pwm_4
            movf  PWM_step,   W
            subwf PWM_step_tg1,     W
            btfsc STATUS,           Z     ; if equal no acceleration needed
            goto  l_pwm_5

            btfss STATUS,     C     ; if positive, accel needed; otherwise decel
            goto  l_pwm_decel
            incf  PWM_step,   F     ; increment now
            movlw ACC_Rate
            call  ee_read
            movwf PWM_timer         ; preset to CV
            movlw Accel_Adj         ; add adjustment
            goto  l_pwm_1
;
l_pwm_decel decf  PWM_step,   F     ; decelerate
            movlw DEC_Rate
            call  ee_read
```

```
            movwf PWM_timer
            movlw Decel_Adj         ; add adjustment
;
l_pwm_1           call  ee_read
            movwf Arg1
            andlw h'7F'
            btfsc Arg1,             7     ; add or subtract?
            sublw 0                 ; W := 0 - W
            addwf PWM_timer,  F
            btfss STATUS,           Z
            goto  l_pwm_7
            movf  PWM_step_tg1,     W
            movwf PWM_step
;
l_pwm_7           call  splineIt
;
l_pwm_5           incf  PWM_timer,  F
            movf  PWM_step,   F
            btfsc STATUS,           Z     ; if stopped, adjust direction
            call  pwm_set_dir
;
l_pwm_4           call  EMF_control
            retlw 0

;---------------------------------------
;  byte constructor and error check
;
byte        rlf   FLG,        W           ; move LV -> C
            btfss FLG,        VB          ; skip if byte complete
            goto  l_rotate
            bcf   FLG,        VB
            btfss STATUS,     C           ; look for end-of-packet bit
            goto  l_rotate
            bsf   DEC_STATUS, DEC_STAT_BP ; mark beginning of packet
            movf  Err,        F
            btfss STATUS,           Z
            retlw 0                       ; error
            movlw 4                       ; verify min. packet length (treat as ext.
addr)
            subwf B_cnt,            F
            btfss STATUS,           C
            retlw 0
;
            incf  B_cnt,            F           ; B_cnt - 4 + 1, subtract addr bytes
and err byte from length
            bsf   FLG,        VP          ; packet is ok
;
l_rotate    rlf   Buf,        F
            btfsc STATUS,     C           ; skip if bit == 0, start bit (end of byte)
            retlw 1                       ; not a byte
            bsf   FLG,        VB
            btfsc FLG,        VP
            retlw 0                       ; if packet marked, drop byte on the floor
and error
            movf  Buf,        W
            xorwf Err,        F
            btfss DEC_STATUS, DEC_STAT_BP
            goto  l_byte_3
```

```
            bcf   DEC_STATUS, DEC_STAT_BP
            andlw b'11000000'             ; test if high address bits are '11',
extended address
            xorlw b'11000000'
            btfsc STATUS,           Z           ; if it is, start at Addr2_b
            goto  l_byte_2
            clrf  Addr2_b           ; ensures a comparison of short addr as ext. Addr
will succeed (MSB is 0)
            incf  FSR,        F           ; if not, skip the MSB of address, point to
Addr_b
            incf  B_cnt,            F           ;
;
l_byte_2    movf  Buf,        W; put original addr back
;
l_byte_3    movwf INDF
            incf  FSR,        F
            incf  B_cnt,            F
            movlw 7                       ; pkt size < 7 bytes
            subwf B_cnt,            W
            btfsc STATUS,           C
            retlw 0                       ; error
;
l_new_byte  clrf  Buf
            comf  Buf,        F
            retlw 1



;-----------------------------------------------------
; Is this a repeat packet
;  required for Service mode & CV access instructions
;
isRepeatPacket    bsf   DEC_STATUS, DEC_STAT_DP
            movf  Addr2_b,    W
            xorwf LastAddr2_b,      W
            btfss STATUS,           Z
            bcf   DEC_STATUS, DEC_STAT_DP
            xorwf LastAddr2_b,      F
            movf  Addr_b,           W
            xorwf LastAddr_b, W     ; should be zero
            btfss STATUS,           Z
            bcf   DEC_STATUS, DEC_STAT_DP
            xorwf LastAddr_b, F
            movf  Cmd_b,            W
            xorwf LastCmd_b,  W
            btfss STATUS,     Z
            bcf   DEC_STATUS, DEC_STAT_DP
            xorwf LastCmd_b,  F
            movf  Chk_b,            W
            xorwf LastChk_b,  W
            btfss STATUS,     Z
            bcf   DEC_STATUS, DEC_STAT_DP
            xorwf LastChk_b,  F
            movf  Sup_1,            W
            xorwf LastSup1,   W
            btfss STATUS,     Z
            bcf   DEC_STATUS, DEC_STAT_DP
```

```
            xorwf LastSup1,   F
            movf  Sup_2,            W
            xorwf LastSup2,   W
            btfss STATUS,     Z
            bcf   DEC_STATUS, DEC_STAT_DP
            xorwf LastSup2,   F
            return


;-------------------------------------
; set advanced acknowledgement
;
set_adv_ack
;           bsf   DEC_config, DEC_CFG_AK
;           goto  l_clr_ak_1

;-------------------------------------
; clr advanced acknowledgement
;
clr_adv_ack
;           bcf   DEC_config, DEC_CFG_AK
l_clr_ak_1  call  write_config
            goto  ack
;-------------------------------------
; set adv. addressing
;
set_adv_addr      bsf   DEC_config, DEC_CFG_AA
            goto  l_clr_ak_1
;-------------------------------------
; clr adv. addressing
;
clr_adv_addr      bcf   DEC_config, DEC_CFG_AA
l_clr_aa_1  goto  l_clr_ak_1
;-------------------------------------
;  acknowledge request response.
;  Note: this is implemented as a 5ms
;  motor-on, basic ack
;
ack
            btfss DEC_STATUS, DEC_STAT_SV ; if not in svc mode, do not ack
            goto  l_roll
            bcf   INTCON,     GIE         ; clear interrupts
            movlw FREQ * 2
            movwf Arg3              ; count 17 * 256 * 3 ~= 6.5 milisec
            ;movlw      OUT_back_msk           ; reverse direction with every ack
            ;xorwf      OUT_back_port,    F     ; this didn't work for some reason
            ;movlw      OUT_fwd_msk
            ;xorwf      OUT_fwd_port,     F
            bsf         OUT_fl_port,      OUT_fl                                ;*
            bsf         OUT_rl_port,      OUT_rl
            bcf         OUT_back_port,    OUT_back          ;*
            ;*
            bsf         OUT_fwd_port,     OUT_fwd           ;*

            movlw h'FF'
            movwf CCPR1L                ; turn on motor for >= 5msec
```

```
;
l_ack_delay clrwdt
            clrf  Arg4
;
l_ack_loop  decfsz      Arg4,       F
            goto  l_ack_loop
            decfsz      Arg3,             F
            goto  l_ack_delay
            clrf  CCPR1L
            bcf         OUT_fwd_port,     OUT_fwd
            bcf         OUT_fl_port,      OUT_fl                                  ;*
            bcf         OUT_rl_port,      OUT_rl                                  ;*
            goto  p_sync_start            ; no need to roll in svc mode

;---------------------------------------
; roll packet bytes
;     W : number of bytes
l_roll            movf  B_rem,      W
            subwf B_cnt,      F           ; decrease byte count
            btfsc STATUS,     Z
            goto  p_delay                 ; continue
            btfss STATUS, C
            goto  p_sync_start            ; error, reset
;
l_roll_1    movf  Chk_b,      W
            movwf Cmd_b
            movf  Sup_1,      W
            movwf Chk_b
            movf  Sup_2,      W
            movwf Sup_1
            decfsz      B_rem,      F
            goto  l_roll_1
            goto  l_parse


;---------------------------------------
; Set Direction
; Z-bit - direction
set_dir           bsf   FLG2,       DG
            bcf   FLG2,       DIRC        ; def. is direction not changed
            btfss DEC_STATUS, DEC_STAT_AC ; is this a consist command?
            goto  l_setd_1
            btfsc DEC_STATUS, DEC_STAT_CR
            xorwf Arg1,       W
            goto  l_setd_2
;
l_setd_1    btfsc DEC_config, DEC_CFG_DR  ; is direction reversed
            xorwf Arg1,       W           ; reverse bit
;
l_setd_2    andwf Arg1,       W
            btfss STATUS,     Z
            goto  l_setd_fwd
;                                         ; backwards direction
            bcf   FLG2,       DIRF        ; signal back direction
            btfsc ON_Effect,  EF_Rev
```

```
            goto  l_setd_3                ; (*)
            return
;
l_setd_fwd  bsf   FLG2,       DIRF        ; signal fwd direction
            btfss ON_Effect,  EF_Fwd            ; (*)

            return

l_setd_3    bsf   FLG2,       DIRC        ; direction has changed (*)
            btfsc Fn_control, FN_RST            ; (*)
            call  pwm_setd_start                ; (*)
            return
;---------------------------------------
; Set   lamp fl/rl
;
set_lmp           movlw CAct_LMP          ; read mask of lights that are controlled by
consist addr.
            call  ee_read
            btfss DEC_STATUS,       DEC_STAT_AC ; if not consist addr, then both lights
are controlled here
            movlw b'00000011'             ; both lamp Fn's
            xorlw h'FF'
            andwf Fn_control, F           ; first, mask controlled lights off
            xorlw h'FF'
            btfss Cmd_b,            4           ; this is the FL bit
            clrw
            iorwf Fn_control, F
            bcf   Cmd_b,            4           ; for speed/dir: set bit for
intermediate step to 0
            return

;---------------------------------------
; set power mode (DCC/Analog)
;
;                           ;On entry assumes Bank 1
set_power
            btfsc FLG2,       AM
            goto  l_sep_analog

            return
;
l_sep_analog
            return

;---------------------------------------
; Advanced op
;
adv_op            movf  Cmd_b,      w
            xorlw b'00111111' ; 128 speed step
            btfss STATUS,     Z
            goto  l_roll
            incf  B_rem,      F     ; 2 byte instruction
            movlw b'10000000' ; direction bit mask for 128ss
            movwf Arg1
            movf  Chk_b,      W
            call  set_dir
            rlf   Chk_b,      W
```

```
            andlw h'FE'       ; (0 - 127) -> (0 - 254)
            btfsc STATUS,     Z     ; is it stop (0)?
            goto  l_spd_stop
            addlw -2          ; (2 - 254) -> (0 - 252)
            btfsc STATUS,     Z     ; emergency stop?
            goto  l_spd_emr_stop
            addlw -2          ; adjust (2 - 252) to (0 - 250)
            goto  l_spd_calc

;---------------------------------------
; Speed and direction
;
speed_dir   bcf   DEC_STATUS, DEC_STAT_NG
            movlw b'00100000' ; direction bit mask for 14/28 ss
            movwf Arg1
            movf  Cmd_b,      W
            call  set_dir
            btfss DEC_config, DEC_CFG_FL
            call  set_lmp
;
l_speed           movlw b'00011111'
            andwf Cmd_b,      F
            movf  Cmd_b,      W
            andlw b'00001111'
            btfsc STATUS,     Z
            goto  l_spd_stop
            addlw -1
            btfsc STATUS,     Z
            goto  l_spd_emr_stop
            bcf   STATUS,     C
            rlf   Cmd_b,      F     ; make 28 spd steps out of 16
            btfsc Cmd_b,      5     ; bit 4 controls intermediate step
            bsf   Cmd_b,      0     ;
            bcf   Cmd_b,      5     ; erase bit 4
            movf  Last_spd, W
            subwf Cmd_b,      W
            btfsc STATUS,     C     ; treat negative values
            goto  l_spd_delta
            bsf   DEC_STATUS, DEC_STAT_NG
            addlw 2
;
l_spd_delta addlw -1          ; at this point, if |Cmd_b - Last_spd| == 1, then W == 0
            movwf Arg6        ; store it for later
            movf  Cmd_b,      W
            movwf Last_spd    ; keep last speed for half-steps
            addlw -4          ; convert (4-31) -> (0-27)
            movwf Arg1
            movlw STEP_MULTIPLIER ; (step-4) * 256 / 27 == (step-4) * 151 / 16
            call  multiply
            swapf Arg2, F     ; divide by 16
            swapf Arg1, W
            xorwf Arg2, W
            andlw h'0F'
            xorwf Arg2, W
            movf  Arg6, F     ; test if delta speed is 1
```

```
            btfss STATUS,     Z
            goto  l_spd_calc

; add or subtract 5 from step256 ~= (256/27) / 2

            btfss DEC_STATUS, DEC_STAT_NG ; if delta is positive, subtract half step
and vice versa
            addlw -d'10'
            addlw d'5'
;
l_spd_calc  btfsc FLG2, AG          ; if digital following analog mode and direction
changed ignore this packet.
            btfss FLG2, DIRC
            goto  l_spd_c1
            bcf   FLG2, DIRC
            movlw DIRF_msk
            xorwf FLG2, F           ; change direction back,
            goto  l_spd_stop1       ; and slow down to stop.

l_spd_c1    addlw 1                 ; speed sent to splineIt is not '0'
            movwf PWM_step_tgt
            btfsc Fn_control, FN_RST      ; this signals 1st speed packet after reset
            movwf PWM_step          ; if it is 1st speed packet, set immediately
            bcf   Fn_control, FN_RST
            goto  l_spd_end
;
l_spd_emr_stop    clrf  PWM_step
;
l_spd_stop  bcf   FLG2, AG
l_spd_stop1 clrf  PWM_step_tgt
;
l_spd_end   movf  PWM_step_tgt, W
            btfsc FLG2, DIRC
            clrw
            movwf PWM_step_tg1
            call  splineIt
            call  EMF_control
            goto  l_roll

;---------------------------------------
; enhanced special functions - group 1
;     100|FL|F4|F3|F2|F1
;  currently implemented FL, F1, F2
;
func_1            btfsc DEC_config,       DEC_CFG_FL  ; if bit is set, control FL here
            call  set_lmp
;
            movlw b'11000011'             ; complement the mask
            andwf Fn_control, F
            movlw CAct_F1_F8              ; read mask for Fn controlled by consist
            call  ee_read
            btfss DEC_STATUS, DEC_STAT_AC       ; if packet not to consist addr, mask
is h'FF'
            movlw h'FF'
            andwf Cmd_b,            F
            rlf   Cmd_b,            F
```

```
            rlf   Cmd_b,            W
            andlw b'00111100'
            iorwf Fn_control, F
;---------------------------------------
; enhanced special functions - group 1
;     1011|F8|F7|F6|F5
;  currently implemented none
;

func_2            goto  l_roll


;---------------------------------------
; translate CV# (0 based) to EEPROM address
; CV# bits 0-7 in Arg1
; CV# bits 8-9 in W (lsb)
;
CV_to_addr  andlw b'00000011'             ;check hi bits; have to be 0
            btfss STATUS,     Z
            retlw 0xFF
;
            movlw d'6'
            subwf Arg1, w           ; a-6
            btfsc STATUS,     C           ; CV <= 6 (a <= 5)
            goto  l_cva_8
            addlw d'6'              ; get 'a' back, it's the address
            return
;
l_cva_8           movlw d'8'
            subwf Arg1, w           ; a-8
            btfsc STATUS, C         ; CV <= 8 (a <= 7)
            goto  l_cva_1
            addlw h'F8'             ; mark as special reg. (read only)
            return
;
l_cva_1           movlw d'8'              ; CV9 - CV11 (8 <= a <= 10)
            subwf Arg1, w           ; a-8
            btfss STATUS,     C
            retlw 0xFF
            addlw -3                ; a-11
            btfsc STATUS, C
            goto  l_cva_2
            addlw d'9'              ; 9+(a-11) == a-2
            return
;
; CV17-19 (16 <= a <=18)
l_cva_2           movlw d'16'
            subwf Arg1, w           ; a-16
            btfss STATUS,     C
            retlw 0xFF              ; CV < 17
            addlw -3                ; a-19
            btfsc STATUS, C
            goto  l_cva_3
            addlw d'12'             ; 12+(a-19) == a-7
            return
```

```
;
; CV21-25
l_cva_3          movlw d'20'
            subwf Arg1, w           ; a-20
            btfss STATUS,     C
            retlw 0xFF              ; CV < 21
            addlw -5                ; a-25
            btfsc STATUS, C
            goto  l_cva_4
            addlw d'17'             ; 17+(a-25) == a-8
            return
;
; CV29-30
l_cva_4          movlw d'28'
            subwf Arg1, w           ; a-28
            btfss STATUS,     C
            retlw 0xFF              ; CV < 29
            addlw -2                ; a-30
            btfsc STATUS, C
            goto  l_cva_5
            addlw d'19'             ; 19+(a-30) == a-11
            return
;
; CV33-37 = 19-23
l_cva_5          movlw d'32'
            subwf Arg1, w           ; a-32
            btfss STATUS,     C
            retlw 0xFF              ; CV < 29
            addlw -5                ; a-37
            btfsc STATUS, C
            goto  l_cva_6
            addlw d'24'             ; 24+(a-37) == a-13
            return
; CV49 - 60 = 24-35
l_cva_6          movlw d'48'
            subwf Arg1, w           ; a - 48
            btfss STATUS, C
            retlw 0xFF              ; CV < 49
            addlw -d'12'                 ; a - 60
            btfsc STATUS, C
            goto  l_cva_7
            addlw d'36'             ; 36+(a-60) == a-24
            return
; CV67-94  == 36-63
l_cva_7          movlw d'66'
            subwf Arg1, w           ; a-66
            btfss STATUS,     C
            retlw 0xFF              ; CV < 67
            addlw -d'28'                 ; a-94
            btfsc STATUS, C
            retlw 0xFF              ; no more CVs supported
            addlw d'64'             ; 64+(a-94) == a-30
            return
```

```
;---------------------------------------
;     DATA EEPROM preset values
;
      org   0x2100
      de    3           ;PRIM_Addr        ;CV1
      de    1           ;V_Start    ;CV2  Use 1 here with BEMF
      de    2           ;ACC_Rate   ;CV3  A nice value. Set to 0 for no accel
      de    2           ;DEC_Rate   ;CV4  A nice value. Set to 0 for no decel
      de    1           ;V_High           ;CV5
      de    .75         ;V_Mid            ;CV6  A reasonable value if speed table off
                        ;                 Set to 0 for a linear response
; CV7-CV8 implemented as constants
      de    0           ;PWMTot           ;CV9  = 6  (not used in this version)
      de    h'FF'       ;EMFCut           ;CV10    Reduce if lower BEMF cutoff wanted
      de    0           ;Packet_TO  ;CV11 = 8  0 is off, else value in seconds
; CV14-16 reserved
      de    0           ;ExtAddr1   ;CV17 = 9
      de    0           ;ExtAddr2   ;CV18
      de    0           ;Consist_addr     ;CV19 = 11
; CV20 reserved
      de    0           ;CAct_F1_F8 ;CV21 = 12
      de    0           ;CAct_LGT   ;CV22
      de    0           ;Accel_Adj  ;CV23
      de    0           ;Decel_Adj  ;CV24
      de    1           ;CABSPD_Step      ;CV25 = 16
; CV26-28 reserved
      de    B'00010110' ;Config_data_1    ;CV29 = 17  Speed table on
      de    0           ;Err_Info   ;CV30 = 18
; CV31-32 n/a
      de    OUT_fl_msk  ;FL_loc           ;CV33 = 19  Output 1
      de    OUT_rl_msk  ;RL_loc           ;CV34     Output 2
      de    OUT_f1_msk  ;F1_loc           ;CV35     Output 3
      de    OUT_f2_msk  ;F2_loc           ;CV36     Output 4
;     de    OUT5_msk    ;F3_loc           ;CV37 = 23  Output 5
      de    0
;
      de    B'00010000' ;FL_Effect  ;CV49 = 24  Front light
      de    B'00001000' ;RL_Effect  ;CV50     Rear light
      de    0           ;F1_Effect  ;CV51
      de    0           ;F2_Effect  ;CV52
      de    0           ;F3_Effect  ;CV53
      de    h'30'       ;PWM_Mode   ;CV54  HF PWM, BEMF on, no low speed LF.
      de    h'20'       ;Ki         ;CV55  A conservative value, may be increased
      de    h'10'       ;Kp         ;CV56  Reduce for coreless motors
      de    d'166'           ;Kfr         ;CV57  Play around with this
      de    0           ;Reserv1    ;CV58
      de    0           ;Reserv2    ;CV59
      de    0           ;Reserv3    ;CV60  = 35
;
                        ;SPD_Tbl:1C       ;CV67 - 94 = 36 - 63 (28 values)
                        ;A roughly parabolic speed curve
                        ;Good control at low speeds
      de    d'1'        ;CV67
      de    d'2'        ;CV68
      de    d'3'        ;CV69
```

```
        de    d'5'        ;CV70
        de    d'8'        ;CV71
        de    d'12'       ;CV72
        de    d'16'       ;CV73
        de    d'21'       ;CV74
        de    d'26'       ;CV75
        de    d'33'       ;CV76
        de    d'39'       ;CV77
        de    d'47'       ;CV78
        de    d'55'       ;CV79
        de    d'64'       ;CV80
        de    d'73'       ;CV81
        de    d'83'       ;CV82
        de    d'94'       ;CV83
        de    d'105'            ;CV84
        de    d'117'            ;CV85
        de    d'129'            ;CV86
        de    d'143'            ;CV87
        de    d'156'            ;CV88
;       de    d'163'
        de    d'171'            ;CV89
;       de    d'178'
        de    d'186'            ;CV90
;       de    d'192'
        de    d'202'            ;CV91
;       de    d'210'
        de    d'218'            ;CV92
;       de    d'225'
        de    d'235'            ;CV93
;       de    d'245'
        de    d'255'            ;CV94


        END
```