$NOMOD51

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

;

; ESCNO EQU "ESC"

; MCU\_48MHZ EQU "N"

; FETON\_DELAY EQU "N"

;

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

; Minimum 8K Bytes of In-System Self-Programmable Flash

; Minimum 512 Bytes Internal SRAM

;

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

; Master clock is internal 24MHz oscillator (or 48MHz, for which the times below are halved)

; Although 24/48 are used in the code, the exact clock frequencies are 24.5MHz or 49.0 MHz

; Timer 0 (41.67ns counts) always counts up and is used for

; - RC pulse measurement

; Timer 1 (41.67ns counts) always counts up and is used for

; - DShot frame sync detection

; Timer 2 (500ns counts) always counts up and is used for

; - RC pulse timeout counts and commutation times

; Timer 3 (500ns counts) always counts up and is used for

; - Commutation timeouts

; PCA0 (41.67ns counts) always counts up and is used for

; - Hardware PWM generation

;

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

; Interrupt handling

; The C8051 does not disable interrupts when entering an interrupt routine.

; Also some interrupt flags need to be cleared by software

; The code disables interrupts in some interrupt routines

; - Interrupts are disabled during beeps, to avoid audible interference from interrupts

;

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

; Motor control:

; - Brushless motor control with 6 states for each electrical 360 degrees

; - An advance timing of 0deg has zero cross 30deg after one commutation and 30deg before the next

; - Timing advance in this implementation is set to 15deg nominally

; - Motor pwm is always damped light (aka complementary pwm, regenerative braking)

; Motor sequence starting from zero crossing:

; - Timer wait: Wt\_Comm 15deg ; Time to wait from zero cross to actual commutation

; - Timer wait: Wt\_Advance 15deg ; Time to wait for timing advance. Nominal commutation point is after this

; - Timer wait: Wt\_Zc\_Scan 7.5deg ; Time to wait before looking for zero cross

; - Scan for zero cross 22.5deg ; Nominal, with some motor variations

;

; Motor startup:

; There is a startup phase and an initial run phase, before normal bemf commutation run begins.

;

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

; List of enumerated supported ESCs

A\_ EQU 1 ; X X RC X MC MB MA CC X X Cc Cp Bc Bp Ac Ap

B\_ EQU 2 ; X X RC X MC MB MA CC X X Ap Ac Bp Bc Cp Cc

C\_ EQU 3 ; Ac Ap MC MB MA CC X RC X X X X Cc Cp Bc Bp

D\_ EQU 4 ; X X RC X CC MA MC MB X X Cc Cp Bc Bp Ac Ap Com fets inverted

E\_ EQU 5 ; L1 L0 RC X MC MB MA CC X L2 Cc Cp Bc Bp Ac Ap A with LEDs

F\_ EQU 6 ; X X RC X MA MB MC CC X X Cc Cp Bc Bp Ac Ap

G\_ EQU 7 ; X X RC X CC MA MC MB X X Cc Cp Bc Bp Ac Ap Like D, but noninverted com fets

H\_ EQU 8 ; RC X X X MA MB CC MC X Ap Bp Cp X Ac Bc Cc

I\_ EQU 9 ; X X RC X MC MB MA CC X X Ac Bc Cc Ap Bp Cp

J\_ EQU 10 ; L2 L1 L0 RC CC MB MC MA X X Cc Bc Ac Cp Bp Ap LEDs

K\_ EQU 11 ; X X MC X MB CC MA RC X X Ap Bp Cp Cc Bc Ac Com fets inverted

L\_ EQU 12 ; X X RC X CC MA MB MC X X Ac Bc Cc Ap Bp Cp

M\_ EQU 13 ; MA MC CC MB RC L0 X X X Cc Bc Ac Cp Bp Ap X LED

N\_ EQU 14 ; X X RC X MC MB MA CC X X Cp Cc Bp Bc Ap Ac

O\_ EQU 15 ; X X RC X CC MA MC MB X X Cc Cp Bc Bp Ac Ap Like D, but low side pwm

P\_ EQU 16 ; X X RC MA CC MB MC X X Cc Bc Ac Cp Bp Ap X

Q\_ EQU 17 ; Cp Bp Ap L1 L0 X RC X X MA MB MC CC Cc Bc Ac LEDs

R\_ EQU 18 ; X X RC X MC MB MA CC X X Ac Bc Cc Ap Bp Cp

S\_ EQU 19 ; X X RC X CC MA MC MB X X Cc Cp Bc Bp Ac Ap Like O, but com fets inverted

T\_ EQU 20 ; RC X MA X MB CC MC X X X Cp Bp Ap Ac Bc Cc

U\_ EQU 21 ; MA MC CC MB RC L0 L1 L2 X Cc Bc Ac Cp Bp Ap X Like M, but with 3 LEDs

V\_ EQU 22 ; Cc X RC X MC CC MB MA X Ap Ac Bp X X Bc Cp

W\_ EQU 23 ; RC MC MB X CC MA X X X Ap Bp Cp X X X X Tristate gate driver

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

; Select the port mapping to use (or unselect all for use with external batch compile file)

;ESCNO EQU A\_

;ESCNO EQU B\_

;ESCNO EQU C\_

;ESCNO EQU D\_

;ESCNO EQU E\_

;ESCNO EQU F\_

;ESCNO EQU G\_

;ESCNO EQU H\_

;ESCNO EQU I\_

;ESCNO EQU J\_

;ESCNO EQU K\_

;ESCNO EQU L\_

;ESCNO EQU M\_

;ESCNO EQU N\_

;ESCNO EQU O\_

;ESCNO EQU P\_

;ESCNO EQU Q\_

;ESCNO EQU R\_

;ESCNO EQU S\_

;ESCNO EQU T\_

;ESCNO EQU U\_

;ESCNO EQU V\_

;ESCNO EQU W\_

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

; Select the MCU type (or unselect for use with external batch compile file)

;MCU\_48MHZ EQU 0

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

; Select the fet deadtime (or unselect for use with external batch compile file)

;FETON\_DELAY EQU 15 ; 20.4ns per step

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

; ESC selection statements

IF ESCNO == A\_

$include (A.inc) ; Select pinout A

ENDIF

IF ESCNO == B\_

$include (B.inc) ; Select pinout B

ENDIF

IF ESCNO == C\_

$include (C.inc) ; Select pinout C

ENDIF

IF ESCNO == D\_

$include (D.inc) ; Select pinout D

ENDIF

IF ESCNO == E\_

$include (E.inc) ; Select pinout E

ENDIF

IF ESCNO == F\_

$include (F.inc) ; Select pinout F

ENDIF

IF ESCNO == G\_

$include (G.inc) ; Select pinout G

ENDIF

IF ESCNO == H\_

$include (H.inc) ; Select pinout H

ENDIF

IF ESCNO == I\_

$include (I.inc) ; Select pinout I

ENDIF

IF ESCNO == J\_

$include (J.inc) ; Select pinout J

ENDIF

IF ESCNO == K\_

$include (K.inc) ; Select pinout K

ENDIF

IF ESCNO == L\_

$include (L.inc) ; Select pinout L

ENDIF

IF ESCNO == M\_

$include (M.inc) ; Select pinout M

ENDIF

IF ESCNO == N\_

$include (N.inc) ; Select pinout N

ENDIF

IF ESCNO == O\_

$include (O.inc) ; Select pinout O

ENDIF

IF ESCNO == P\_

$include (P.inc) ; Select pinout P

ENDIF

IF ESCNO == Q\_

$include (Q.inc) ; Select pinout Q

ENDIF

IF ESCNO == R\_

$include (R.inc) ; Select pinout R

ENDIF

IF ESCNO == S\_

$include (S.inc) ; Select pinout S

ENDIF

IF ESCNO == T\_

$include (T.inc) ; Select pinout T

ENDIF

IF ESCNO == U\_

$include (U.inc) ; Select pinout U

ENDIF

IF ESCNO == V\_

$include (V.inc) ; Select pinout V

ENDIF

IF ESCNO == W\_

$include (W.inc) ; Select pinout W

ENDIF

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

; Programming defaults

;

DEFAULT\_PGM\_STARTUP\_PWR EQU 9 ; 1=0.031 2=0.047 3=0.063 4=0.094 5=0.125 6=0.188 7=0.25 8=0.38 9=0.50 10=0.75 11=1.00 12=1.25 13=1.50

DEFAULT\_PGM\_COMM\_TIMING EQU 3 ; 1=Low 2=MediumLow 3=Medium 4=MediumHigh 5=High

DEFAULT\_PGM\_DEMAG\_COMP EQU 2 ; 1=Disabled 2=Low 3=High

DEFAULT\_PGM\_DIRECTION EQU 1 ; 1=Normal 2=Reversed 3=Bidir 4=Bidir rev

DEFAULT\_PGM\_BEEP\_STRENGTH EQU 40 ; Beep strength

DEFAULT\_PGM\_BEACON\_STRENGTH EQU 80 ; Beacon strength

DEFAULT\_PGM\_BEACON\_DELAY EQU 4 ; 1=1m 2=2m 3=5m 4=10m 5=Infinite

; COMMON

DEFAULT\_PGM\_ENABLE\_TX\_PROGRAM EQU 1 ; 1=Enabled 0=Disabled

DEFAULT\_PGM\_MIN\_THROTTLE EQU 37 ; 4\*37+1000=1148

DEFAULT\_PGM\_MAX\_THROTTLE EQU 208 ; 4\*208+1000=1832

DEFAULT\_PGM\_CENTER\_THROTTLE EQU 122 ; 4\*122+1000=1488 (used in bidirectional mode)

DEFAULT\_PGM\_ENABLE\_TEMP\_PROT EQU 7 ; 0=Disabled 1=80C 2=90C 3=100C 4=110C 5=120C 6=130C 7=140C

DEFAULT\_PGM\_ENABLE\_POWER\_PROT EQU 1 ; 1=Enabled 0=Disabled

DEFAULT\_PGM\_BRAKE\_ON\_STOP EQU 0 ; 1=Enabled 0=Disabled

DEFAULT\_PGM\_LED\_CONTROL EQU 0 ; Byte for LED control. 2bits per LED, 0=Off, 1=On

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

; Temporary register definitions

Temp1 EQU R0

Temp2 EQU R1

Temp3 EQU R2

Temp4 EQU R3

Temp5 EQU R4

Temp6 EQU R5

Temp7 EQU R6

Temp8 EQU R7

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

; Register definitions

DSEG AT 20h ; Variables segment

Bit\_Access: DS 1 ; MUST BE AT THIS ADDRESS. Variable at bit accessible address (for non interrupt routines)

Bit\_Access\_Int: DS 1 ; Variable at bit accessible address (for interrupts)

Rcp\_Outside\_Range\_Cnt: DS 1 ; RC pulse outside range counter (incrementing)

Rcp\_Timeout\_Cntd: DS 1 ; RC pulse timeout counter (decrementing)

Flags0: DS 1 ; State flags. Reset upon init\_start

T3\_PENDING EQU 0 ; Timer 3 pending flag

DEMAG\_DETECTED EQU 1 ; Set when excessive demag time is detected

COMP\_TIMED\_OUT EQU 2 ; Set when comparator reading timed out

; EQU 3

; EQU 4

; EQU 5

; EQU 6

; EQU 7

Flags1: DS 1 ; State flags. Reset upon init\_start

STARTUP\_PHASE EQU 0 ; Set when in startup phase

INITIAL\_RUN\_PHASE EQU 1 ; Set when in initial run phase, before synchronized run is achieved

MOTOR\_STARTED EQU 2 ; Set when motor is started

DIR\_CHANGE\_BRAKE EQU 3 ; Set when braking before direction change

HIGH\_RPM EQU 4 ; Set when motor rpm is high (Comm\_Period4x\_H less than 2)

; EQU 5

; EQU 6

; EQU 7

Flags2: DS 1 ; State flags. NOT reset upon init\_start

RCP\_UPDATED EQU 0 ; New RC pulse length value available

RCP\_ONESHOT125 EQU 1 ; RC pulse input is OneShot125 (125-250us)

RCP\_ONESHOT42 EQU 2 ; RC pulse input is OneShot42 (41.67-83us)

RCP\_MULTISHOT EQU 3 ; RC pulse input is Multishot (5-25us)

RCP\_DSHOT EQU 4 ; RC pulse input is digital shot

RCP\_DIR\_REV EQU 5 ; RC pulse direction in bidirectional mode

RCP\_FULL\_RANGE EQU 6 ; When set full input signal range is used (1000-2000us) and stored calibration values are ignored

; EQU 7

Flags3: DS 1 ; State flags. NOT reset upon init\_start

PGM\_DIR\_REV EQU 0 ; Programmed direction. 0=normal, 1=reversed

PGM\_BIDIR\_REV EQU 1 ; Programmed bidirectional direction. 0=normal, 1=reversed

PGM\_BIDIR EQU 2 ; Programmed bidirectional operation. 0=normal, 1=bidirectional

; EQU 3

; EQU 4

; EQU 5

; EQU 6

; EQU 7

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

; RAM definitions

DSEG AT 30h ; Ram data segment, direct addressing

Initial\_Arm: DS 1 ; Variable that is set during the first arm sequence after power on

Min\_Throttle\_L: DS 1 ; Minimum throttle scaled (lo byte)

Min\_Throttle\_H: DS 1 ; Minimum throttle scaled (hi byte)

Center\_Throttle\_L: DS 1 ; Center throttle scaled (lo byte)

Center\_Throttle\_H: DS 1 ; Center throttle scaled (hi byte)

Max\_Throttle\_L: DS 1 ; Maximum throttle scaled (lo byte)

Max\_Throttle\_H: DS 1 ; Maximum throttle scaled (hi byte)

Power\_On\_Wait\_Cnt\_L: DS 1 ; Power on wait counter (lo byte)

Power\_On\_Wait\_Cnt\_H: DS 1 ; Power on wait counter (hi byte)

Startup\_Cnt: DS 1 ; Startup phase commutations counter (incrementing)

Startup\_Zc\_Timeout\_Cntd: DS 1 ; Startup zero cross timeout counter (decrementing)

Initial\_Run\_Rot\_Cntd: DS 1 ; Initial run rotations counter (decrementing)

Stall\_Cnt: DS 1 ; Counts start/run attempts that resulted in stall. Reset upon a proper stop

Demag\_Detected\_Metric: DS 1 ; Metric used to gauge demag event frequency

Demag\_Pwr\_Off\_Thresh: DS 1 ; Metric threshold above which power is cut

Low\_Rpm\_Pwr\_Slope: DS 1 ; Sets the slope of power increase for low rpms

Timer0\_X: DS 1 ; Timer 0 extended byte

Timer2\_X: DS 1 ; Timer 2 extended byte

Prev\_Comm\_L: DS 1 ; Previous commutation timer 3 timestamp (lo byte)

Prev\_Comm\_H: DS 1 ; Previous commutation timer 3 timestamp (hi byte)

Prev\_Comm\_X: DS 1 ; Previous commutation timer 3 timestamp (ext byte)

Prev\_Prev\_Comm\_L: DS 1 ; Pre-previous commutation timer 3 timestamp (lo byte)

Prev\_Prev\_Comm\_H: DS 1 ; Pre-previous commutation timer 3 timestamp (hi byte)

Comm\_Period4x\_L: DS 1 ; Timer 3 counts between the last 4 commutations (lo byte)

Comm\_Period4x\_H: DS 1 ; Timer 3 counts between the last 4 commutations (hi byte)

Comparator\_Read\_Cnt: DS 1 ; Number of comparator reads done

Wt\_Adv\_Start\_L: DS 1 ; Timer 3 start point for commutation advance timing (lo byte)

Wt\_Adv\_Start\_H: DS 1 ; Timer 3 start point for commutation advance timing (hi byte)

Wt\_Zc\_Scan\_Start\_L: DS 1 ; Timer 3 start point from commutation to zero cross scan (lo byte)

Wt\_Zc\_Scan\_Start\_H: DS 1 ; Timer 3 start point from commutation to zero cross scan (hi byte)

Wt\_Zc\_Tout\_Start\_L: DS 1 ; Timer 3 start point for zero cross scan timeout (lo byte)

Wt\_Zc\_Tout\_Start\_H: DS 1 ; Timer 3 start point for zero cross scan timeout (hi byte)

Wt\_Comm\_Start\_L: DS 1 ; Timer 3 start point from zero cross to commutation (lo byte)

Wt\_Comm\_Start\_H: DS 1 ; Timer 3 start point from zero cross to commutation (hi byte)

Dshot\_Cmd: DS 1 ; Dshot command

Dshot\_Cmd\_Cnt: DS 1 ; Dshot command count

New\_Rcp: DS 1 ; New RC pulse value in pca counts

Rcp\_Stop\_Cnt: DS 1 ; Counter for RC pulses below stop value

Power\_Pwm\_Reg\_L: DS 1 ; Power pwm register setting (lo byte)

Power\_Pwm\_Reg\_H: DS 1 ; Power pwm register setting (hi byte). 0x3F is minimum power

Damp\_Pwm\_Reg\_L: DS 1 ; Damping pwm register setting (lo byte)

Damp\_Pwm\_Reg\_H: DS 1 ; Damping pwm register setting (hi byte)

Current\_Power\_Pwm\_Reg\_H: DS 1 ; Current power pwm register setting that is loaded in the PCA register (hi byte)

Pwm\_Limit: DS 1 ; Maximum allowed pwm

Pwm\_Limit\_By\_Rpm: DS 1 ; Maximum allowed pwm for low or high rpms

Pwm\_Limit\_Beg: DS 1 ; Initial pwm limit

Adc\_Conversion\_Cnt: DS 1 ; Adc conversion counter

Current\_Average\_Temp: DS 1 ; Current average temperature (lo byte ADC reading, assuming hi byte is 1)

Throttle\_Gain: DS 1 ; Gain to be applied to RCP value

Throttle\_Gain\_M: DS 1 ; Gain to be applied to RCP value (multiplier 0=1x, 1=2x, 2=4x etc))

Throttle\_Gain\_BD\_Rev: DS 1 ; Gain to be applied to RCP value for reverse direction in bidirectional mode

Throttle\_Gain\_BD\_Rev\_M: DS 1 ; Gain to be applied to RCP value for reverse direction in bidirectional mode (multiplier 0=1x, 1=2x, 2=4x etc)

Beep\_Strength: DS 1 ; Strength of beeps

Skip\_T2\_Int: DS 1 ; Set for 48MHz MCUs when timer 2 interrupt shall be ignored

Clock\_Set\_At\_48MHz: DS 1 ; Variable set if 48MHz MCUs run at 48MHz

Flash\_Key\_1: DS 1 ; Flash key one

Flash\_Key\_2: DS 1 ; Flash key two

Temp\_Prot\_Limit: DS 1 ; Temperature protection limit

DShot\_Pwm\_Thr: DS 1 ; DShot pulse width threshold value

DShot\_Timer\_Preset: DS 1 ; DShot timer preset for frame sync detection

DShot\_Frame\_Start\_L: DS 1 ; DShot frame start timestamp (lo byte)

DShot\_Frame\_Start\_H: DS 1 ; DShot frame start timestamp (hi byte)

DShot\_Frame\_Length\_Thr: DS 1 ; DShot frame length criteria (in units of 4 timer 2 ticks)

; Indirect addressing data segment. The variables below must be in this sequence

ISEG AT 080h

\_Pgm\_Gov\_P\_Gain: DS 1 ; Programmed governor P gain

\_Pgm\_Gov\_I\_Gain: DS 1 ; Programmed governor I gain

\_Pgm\_Gov\_Mode: DS 1 ; Programmed governor mode

\_Pgm\_Low\_Voltage\_Lim: DS 1 ; Programmed low voltage limit

\_Pgm\_Motor\_Gain: DS 1 ; Programmed motor gain

\_Pgm\_Motor\_Idle: DS 1 ; Programmed motor idle speed

Pgm\_Startup\_Pwr: DS 1 ; Programmed startup power

\_Pgm\_Pwm\_Freq: DS 1 ; Programmed pwm frequency

Pgm\_Direction: DS 1 ; Programmed rotation direction

Pgm\_Input\_Pol: DS 1 ; Programmed input pwm polarity

Initialized\_L\_Dummy: DS 1 ; Place holder

Initialized\_H\_Dummy: DS 1 ; Place holder

Pgm\_Enable\_TX\_Program: DS 1 ; Programmed enable/disable value for TX programming

\_Pgm\_Main\_Rearm\_Start: DS 1 ; Programmed enable/disable re-arming main every start

\_Pgm\_Gov\_Setup\_Target: DS 1 ; Programmed main governor setup target

\_Pgm\_Startup\_Rpm: DS 1 ; Programmed startup rpm (unused - place holder)

\_Pgm\_Startup\_Accel: DS 1 ; Programmed startup acceleration (unused - place holder)

\_Pgm\_Volt\_Comp: DS 1 ; Place holder

Pgm\_Comm\_Timing: DS 1 ; Programmed commutation timing

\_Pgm\_Damping\_Force: DS 1 ; Programmed damping force (unused - place holder)

\_Pgm\_Gov\_Range: DS 1 ; Programmed governor range

\_Pgm\_Startup\_Method: DS 1 ; Programmed startup method (unused - place holder)

Pgm\_Min\_Throttle: DS 1 ; Programmed throttle minimum

Pgm\_Max\_Throttle: DS 1 ; Programmed throttle maximum

Pgm\_Beep\_Strength: DS 1 ; Programmed beep strength

Pgm\_Beacon\_Strength: DS 1 ; Programmed beacon strength

Pgm\_Beacon\_Delay: DS 1 ; Programmed beacon delay

\_Pgm\_Throttle\_Rate: DS 1 ; Programmed throttle rate (unused - place holder)

Pgm\_Demag\_Comp: DS 1 ; Programmed demag compensation

\_Pgm\_BEC\_Voltage\_High: DS 1 ; Programmed BEC voltage

Pgm\_Center\_Throttle: DS 1 ; Programmed throttle center (in bidirectional mode)

\_Pgm\_Main\_Spoolup\_Time: DS 1 ; Programmed main spoolup time

Pgm\_Enable\_Temp\_Prot: DS 1 ; Programmed temperature protection enable

Pgm\_Enable\_Power\_Prot: DS 1 ; Programmed low rpm power protection enable

\_Pgm\_Enable\_Pwm\_Input: DS 1 ; Programmed PWM input signal enable

\_Pgm\_Pwm\_Dither: DS 1 ; Programmed output PWM dither

Pgm\_Brake\_On\_Stop: DS 1 ; Programmed braking when throttle is zero

Pgm\_LED\_Control: DS 1 ; Programmed LED control

; The sequence of the variables below is no longer of importance

Pgm\_Startup\_Pwr\_Decoded: DS 1 ; Programmed startup power decoded

; Indirect addressing data segment

ISEG AT 0D0h

Temp\_Storage: DS 48 ; Temporary storage

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

CSEG AT 1A00h ; "Eeprom" segment

EEPROM\_FW\_MAIN\_REVISION EQU 16 ; Main revision of the firmware

EEPROM\_FW\_SUB\_REVISION EQU 7 ; Sub revision of the firmware

EEPROM\_LAYOUT\_REVISION EQU 33 ; Revision of the EEPROM layout

Eep\_FW\_Main\_Revision: DB EEPROM\_FW\_MAIN\_REVISION ; EEPROM firmware main revision number

Eep\_FW\_Sub\_Revision: DB EEPROM\_FW\_SUB\_REVISION ; EEPROM firmware sub revision number

Eep\_Layout\_Revision: DB EEPROM\_LAYOUT\_REVISION ; EEPROM layout revision number

\_Eep\_Pgm\_Gov\_P\_Gain: DB 0FFh

\_Eep\_Pgm\_Gov\_I\_Gain: DB 0FFh

\_Eep\_Pgm\_Gov\_Mode: DB 0FFh

\_Eep\_Pgm\_Low\_Voltage\_Lim: DB 0FFh

\_Eep\_Pgm\_Motor\_Gain: DB 0FFh

\_Eep\_Pgm\_Motor\_Idle: DB 0FFh

Eep\_Pgm\_Startup\_Pwr: DB DEFAULT\_PGM\_STARTUP\_PWR ; EEPROM copy of programmed startup power

\_Eep\_Pgm\_Pwm\_Freq: DB 0FFh

Eep\_Pgm\_Direction: DB DEFAULT\_PGM\_DIRECTION ; EEPROM copy of programmed rotation direction

\_Eep\_Pgm\_Input\_Pol: DB 0FFh

Eep\_Initialized\_L: DB 055h ; EEPROM initialized signature low byte

Eep\_Initialized\_H: DB 0AAh ; EEPROM initialized signature high byte

Eep\_Enable\_TX\_Program: DB DEFAULT\_PGM\_ENABLE\_TX\_PROGRAM ; EEPROM TX programming enable

\_Eep\_Main\_Rearm\_Start: DB 0FFh

\_Eep\_Pgm\_Gov\_Setup\_Target: DB 0FFh

\_Eep\_Pgm\_Startup\_Rpm: DB 0FFh

\_Eep\_Pgm\_Startup\_Accel: DB 0FFh

\_Eep\_Pgm\_Volt\_Comp: DB 0FFh

Eep\_Pgm\_Comm\_Timing: DB DEFAULT\_PGM\_COMM\_TIMING ; EEPROM copy of programmed commutation timing

\_Eep\_Pgm\_Damping\_Force: DB 0FFh

\_Eep\_Pgm\_Gov\_Range: DB 0FFh

\_Eep\_Pgm\_Startup\_Method: DB 0FFh

Eep\_Pgm\_Min\_Throttle: DB DEFAULT\_PGM\_MIN\_THROTTLE ; EEPROM copy of programmed minimum throttle

Eep\_Pgm\_Max\_Throttle: DB DEFAULT\_PGM\_MAX\_THROTTLE ; EEPROM copy of programmed minimum throttle

Eep\_Pgm\_Beep\_Strength: DB DEFAULT\_PGM\_BEEP\_STRENGTH ; EEPROM copy of programmed beep strength

Eep\_Pgm\_Beacon\_Strength: DB DEFAULT\_PGM\_BEACON\_STRENGTH ; EEPROM copy of programmed beacon strength

Eep\_Pgm\_Beacon\_Delay: DB DEFAULT\_PGM\_BEACON\_DELAY ; EEPROM copy of programmed beacon delay

\_Eep\_Pgm\_Throttle\_Rate: DB 0FFh

Eep\_Pgm\_Demag\_Comp: DB DEFAULT\_PGM\_DEMAG\_COMP ; EEPROM copy of programmed demag compensation

\_Eep\_Pgm\_BEC\_Voltage\_High: DB 0FFh

Eep\_Pgm\_Center\_Throttle: DB DEFAULT\_PGM\_CENTER\_THROTTLE ; EEPROM copy of programmed center throttle

\_Eep\_Pgm\_Main\_Spoolup\_Time: DB 0FFh

Eep\_Pgm\_Temp\_Prot\_Enable: DB DEFAULT\_PGM\_ENABLE\_TEMP\_PROT ; EEPROM copy of programmed temperature protection enable

Eep\_Pgm\_Enable\_Power\_Prot: DB DEFAULT\_PGM\_ENABLE\_POWER\_PROT ; EEPROM copy of programmed low rpm power protection enable

\_Eep\_Pgm\_Enable\_Pwm\_Input: DB 0FFh

\_Eep\_Pgm\_Pwm\_Dither: DB 0FFh

Eep\_Pgm\_Brake\_On\_Stop: DB DEFAULT\_PGM\_BRAKE\_ON\_STOP ; EEPROM copy of programmed braking when throttle is zero

Eep\_Pgm\_LED\_Control: DB DEFAULT\_PGM\_LED\_CONTROL ; EEPROM copy of programmed LED control

Eep\_Dummy: DB 0FFh ; EEPROM address for safety reason

CSEG AT 1A60h

Eep\_Name: DB " " ; Name tag (16 Bytes)

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

Interrupt\_Table\_Definition ; SiLabs interrupts

CSEG AT 80h ; Code segment after interrupt vectors

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

; Table definitions

STARTUP\_POWER\_TABLE: DB 04h, 06h, 08h, 0Ch, 10h, 18h, 20h, 30h, 40h, 60h, 80h, 0A0h, 0C0h

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

;

; Timer 0 interrupt routine

;

; No assumptions

;

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

IF MCU\_48MHZ == 1

t0\_int:

inc Timer0\_X

reti

ENDIF

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

;

; Timer 1 interrupt routine

;

; No assumptions

;

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

t1\_int:

clr IE\_EA

clr IE\_EX0 ; Disable int0 interrupts

anl EIE1, #0EFh ; Disable pca interrupts

clr TCON\_TR1 ; Stop timer 1

mov TL1, DShot\_Timer\_Preset ; Reset sync timer

push PSW

setb PSW.3 ; Select register bank 1 for this interrupt

push ACC

push B ; Will be pop'ed by int0 exit

clr TMR2CN0\_TR2 ; Timer 2 disabled

mov Temp1, TMR2L ; Read timer value

mov Temp2, TMR2H

setb TMR2CN0\_TR2 ; Timer 2 enabled

setb IE\_EA

; Reset timer 0

mov TL0, #0

; Check frame time length

clr C

mov A, Temp1

subb A, DShot\_Frame\_Start\_L

mov Temp1, A

mov A, Temp2

subb A, DShot\_Frame\_Start\_H

mov Temp2, A

; Divide by 2 (or 4 for 48MHz). Unit is then us

clr C

mov A, Temp2

rrc A

mov Temp2, A

mov A, Temp1

rrc A

mov Temp1, A

mov A, Clock\_Set\_At\_48MHz

jz t1\_int\_frame\_time\_scaled

clr C

mov A, Temp2

rrc A

mov Temp2, A

mov A, Temp1

rrc A

mov Temp1, A

t1\_int\_frame\_time\_scaled:

mov A, Temp2

jnz t1\_int\_msb\_fail ; Frame too long

mov A, Temp1

subb A, DShot\_Frame\_Length\_Thr

jc t1\_int\_msb\_fail ; Frame too short

subb A, DShot\_Frame\_Length\_Thr

jnc t1\_int\_msb\_fail ; Frame too long

; Check that correct number of pulses is received

mov A, DPL ; Read current pointer

cjne A, #16, t1\_int\_msb\_fail

; Decode transmitted data

mov Temp5, #0 ; Reset timestamp

mov Temp4, #0 ; High byte of receive buffer

mov Temp3, #0 ; Low byte of receive buffer

mov Temp2, #8 ; Number of bits per byte

mov DPTR, #0 ; Set pointer

mov Temp1, DShot\_Pwm\_Thr; DShot pulse width criteria

mov A, Clock\_Set\_At\_48MHz

jnz t1\_int\_decode

clr C

mov A, Temp1 ; Scale pulse width criteria

rrc A

mov Temp1, A

t1\_int\_decode:

ajmp t1\_int\_decode\_msb

t1\_int\_msb\_fail:

mov DPTR, #0 ; Set pointer to start

setb IE\_EX0 ; Enable int0 interrupts

setb IE\_EX1 ; Enable int1 interrupts

ajmp int0\_int\_outside\_range

t1\_int\_decode\_msb:

; Decode DShot data Msb. Use more code space to save time (by not using loop)

Decode\_DShot\_2Msb

Decode\_DShot\_2Msb

Decode\_DShot\_2Msb

Decode\_DShot\_2Msb

ajmp t1\_int\_decode\_lsb

t1\_int\_lsb\_fail:

mov DPTR, #0 ; Set pointer to start

setb IE\_EX0 ; Enable int0 interrupts

setb IE\_EX1 ; Enable int1 interrupts

ajmp int0\_int\_outside\_range

t1\_int\_decode\_lsb:

; Decode DShot data Lsb

Decode\_DShot\_2Lsb

Decode\_DShot\_2Lsb

Decode\_DShot\_2Lsb

Decode\_DShot\_2Lsb

; XOR check (in inverted data, which is ok)

mov A, Temp4

swap A

xrl A, Temp4

xrl A, Temp3

anl A, #0F0h

mov Temp2, A

mov A, Temp3

swap A

anl A, #0F0h

clr C

subb A, Temp2

jz t1\_int\_xor\_ok ; XOR check

mov DPTR, #0 ; Set pointer to start

setb IE\_EX0 ; Enable int0 interrupts

setb IE\_EX1 ; Enable int1 interrupts

ajmp int0\_int\_outside\_range

t1\_int\_xor\_ok:

; Swap to be LSB aligned to 12 bits (and invert)

mov A, Temp4

cpl A

swap A

anl A, #0F0h ; Low nibble of high byte

mov Temp2, A

mov A, Temp3

cpl A

swap A

anl A, #0Fh ; High nibble of low byte

orl A, Temp2

mov Temp3, A

mov A, Temp4 ; High nibble of high byte

cpl A

swap A

anl A, #0Fh

mov Temp4, A

; Subtract 96 (still 12 bits)

clr C

mov A, Temp3

mov Temp2, A

subb A, #96

mov Temp3, A

mov A, Temp4

subb A, #0

mov Temp4, A

jnc t1\_normal\_range

clr C

mov A, Temp2 ; Check for 0 or dshot command

mov Temp4, #0

mov Temp3, #0

mov Temp2, #0

jz t1\_normal\_range

clr C ; We are in the special dshot range

rrc A ; Divide by 2

jnc t1\_dshot\_set\_cmd ; Check for tlm bit set (if not telemetry, Temp2 will be zero and result in invalid command)

mov Temp2, A

clr C

subb A, Dshot\_Cmd

jz t1\_dshot\_inc\_cmd\_cnt

t1\_dshot\_set\_cmd:

mov A, Temp2

mov Dshot\_Cmd, A

mov Dshot\_Cmd\_Cnt, #0

mov Temp2, #0

jmp t1\_normal\_range

t1\_dshot\_inc\_cmd\_cnt:

inc Dshot\_Cmd\_Cnt

t1\_normal\_range:

; Check for bidirectional operation (0=stop, 96-2095->fwd, 2096-4095->rev)

jnb Flags3.PGM\_BIDIR, t1\_int\_not\_bidir ; If not bidirectional operation - branch

; Subtract 2000 (still 12 bits)

clr C

mov A, Temp3

subb A, #0D0h

mov Temp1, A

mov A, Temp4

subb A, #07h

mov Temp2, A

jc t1\_int\_bidir\_fwd ; If result is negative - branch

mov A, Temp1

mov Temp3, A

mov A, Temp2

mov Temp4, A

jb Flags2.RCP\_DIR\_REV, t1\_int\_bidir\_rev\_chk ; If same direction - branch

setb Flags2.RCP\_DIR\_REV

ajmp t1\_int\_bidir\_rev\_chk

t1\_int\_bidir\_fwd:

jnb Flags2.RCP\_DIR\_REV, t1\_int\_bidir\_rev\_chk ; If same direction - branch

clr Flags2.RCP\_DIR\_REV

t1\_int\_bidir\_rev\_chk:

jb Flags3.PGM\_BIDIR\_REV, ($+5)

cpl Flags2.RCP\_DIR\_REV

clr C ; Multiply throttle value by 2

mov A, Temp3

rlc A

mov Temp3, A

mov A, Temp4

rlc A

mov Temp4, A

t1\_int\_not\_bidir:

; Generate 4/256

mov A, Temp4

add A, Temp4

addc A, Temp4

addc A, Temp4

mov Temp2, A

; Align to 11 bits

clr C

mov A, Temp4

rrc A

mov Temp4, A

mov A, Temp3

rrc A

mov Temp3, A

; Scale from 2000 to 2048

mov A, Temp3

add A, Temp2 ; Holds 4/128

mov Temp3, A

mov A, Temp4

addc A, #0

mov Temp4, A

jnb ACC.3, ($+7)

mov Temp3, #0FFh

mov Temp4, #0FFh

; Boost pwm during direct start

mov A, Flags1

anl A, #((1 SHL STARTUP\_PHASE)+(1 SHL INITIAL\_RUN\_PHASE))

jz t1\_int\_startup\_boosted

jb Flags1.MOTOR\_STARTED, t1\_int\_startup\_boosted ; Do not boost when changing direction in bidirectional mode

mov A, Pwm\_Limit\_Beg ; Set 25% of max startup power as minimum power

rlc A

mov Temp2, A

mov A, Temp4

jnz t1\_int\_startup\_boost\_stall

clr C

mov A, Temp2

subb A, Temp3

jc t1\_int\_startup\_boost\_stall

mov A, Temp2

mov Temp3, A

t1\_int\_startup\_boost\_stall:

mov A, Stall\_Cnt ; Add an extra power boost during start

swap A

rlc A

add A, Temp3

mov Temp3, A

mov A, Temp4

addc A, #0

mov Temp4, A

t1\_int\_startup\_boosted:

; Set 8bit value

clr C

mov A, Temp3

rlc A

swap A

anl A, #0Fh

mov Temp1, A

mov A, Temp4

rlc A

swap A

anl A, #0F0h

orl A, Temp1

mov Temp1, A

jnz t1\_int\_zero\_rcp\_checked ; New\_Rcp (Temp1) is only zero if all 11 bits are zero

mov A, Temp3

jz t1\_int\_zero\_rcp\_checked

mov Temp1, #1

t1\_int\_zero\_rcp\_checked:

; Align to 10 bits for 24MHz MCU

IF MCU\_48MHZ == 0

clr C

mov A, Temp4

rrc A

mov Temp4, A

mov A, Temp3

rrc A

mov Temp3, A

ENDIF

mov DPTR, #0 ; Set pointer to start

setb IE\_EX0 ; Enable int0 interrupts

setb IE\_EX1 ; Enable int1 interrupts

; Decrement outside range counter

mov A, Rcp\_Outside\_Range\_Cnt

jz ($+4)

dec Rcp\_Outside\_Range\_Cnt

ajmp int0\_int\_pulse\_ready

t1\_int\_frame\_fail:

mov DPTR, #0 ; Set pointer to start

setb IE\_EX0 ; Enable int0 interrupts

setb IE\_EX1 ; Enable int1 interrupts

ajmp int0\_int\_outside\_range

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

;

; Timer 2 interrupt routine

;

; No assumptions

; Requirements: Temp variables can NOT be used since PSW.x is not set

;

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

t2\_int: ; Happens every 32ms

push PSW ; Preserve registers through interrupt

push ACC

clr TMR2CN0\_TF2H ; Clear interrupt flag

inc Timer2\_X

IF MCU\_48MHZ == 1

mov A, Clock\_Set\_At\_48MHz

jz t2\_int\_start

; Check skip variable

mov A, Skip\_T2\_Int

jz t2\_int\_start ; Execute this interrupt

mov Skip\_T2\_Int, #0

ajmp t2\_int\_exit

t2\_int\_start:

mov Skip\_T2\_Int, #1 ; Skip next interrupt

ENDIF

; Update RC pulse timeout counter

mov A, Rcp\_Timeout\_Cntd ; RC pulse timeout count zero?

jz ($+4) ; Yes - do not decrement

dec Rcp\_Timeout\_Cntd ; No decrement

; Check RC pulse against stop value

clr C

mov A, New\_Rcp ; Load new pulse value

jz t2\_int\_rcp\_stop ; Check if pulse is below stop value

; RC pulse higher than stop value, reset stop counter

mov Rcp\_Stop\_Cnt, #0 ; Reset rcp stop counter

ajmp t2\_int\_exit

t2\_int\_rcp\_stop:

; RC pulse less than stop value

mov A, Rcp\_Stop\_Cnt ; Increment stop counter

add A, #1

mov Rcp\_Stop\_Cnt, A

jnc ($+5) ; Branch if counter has not wrapped

mov Rcp\_Stop\_Cnt, #0FFh ; Set stop counter to max

t2\_int\_exit:

pop ACC ; Restore preserved registers

pop PSW

reti

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

;

; Timer 3 interrupt routine

;

; No assumptions

; Requirements: Temp variables can NOT be used since PSW.x is not set

; ACC can not be used, as it is not pushed to stack

;

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

t3\_int: ; Used for commutation timing

clr IE\_EA ; Disable all interrupts

anl EIE1, #7Fh ; Disable timer 3 interrupts

mov TMR3RLL, #0FAh ; Set a short delay before next interrupt

mov TMR3RLH, #0FFh

clr Flags0.T3\_PENDING ; Flag that timer has wrapped

anl TMR3CN0, #07Fh ; Timer 3 interrupt flag cleared

setb IE\_EA ; Enable all interrupts

reti

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

;

; Int0 interrupt routine

;

; No assumptions

;

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

int0\_int: ; Used for RC pulse timing

push ACC

mov A, TL0 ; Read pwm for DShot immediately

; Test for DShot

jnb Flags2.RCP\_DSHOT, int0\_int\_not\_dshot

mov TL1, DShot\_Timer\_Preset ; Reset sync timer

movx @DPTR, A ; Store pwm

inc DPTR

pop ACC

reti

; Not DShot

int0\_int\_not\_dshot:

pop ACC

clr IE\_EA

anl EIE1, #0EFh ; Disable pca interrupts

push PSW ; Preserve registers through interrupt

push ACC

push B

setb PSW.3 ; Select register bank 1 for this interrupt

setb IE\_EA

; Get the counter values

Get\_Rcp\_Capture\_Values

; Scale down to 10 bits (for 24MHz, and 11 bits for 48MHz)

jnb Flags2.RCP\_MULTISHOT, int0\_int\_fall\_not\_multishot

; Multishot - Multiply by 2 and add 1/16 and 1/32

mov A, Temp1 ; Divide by 16

swap A

anl A, #0Fh

mov Temp3, A

mov A, Temp2

swap A

anl A, #0F0h

orl A, Temp3

mov Temp3, A

clr C ; Make divided by 32

rrc A

add A, Temp3 ; Add 1/16 to 1/32

mov Temp3, A

clr C ; Multiply by 2

mov A, Temp1

rlc A

mov Temp1, A

mov A, Temp2

rlc A

mov Temp2, A

mov A, Temp1 ; Add 1/16 and 1/32

add A, Temp3

mov Temp3, A

mov A, Temp2

IF MCU\_48MHZ == 0

addc A, #03h ; Add to low end, to make signal look like 20-40us

ELSE

addc A, #06h

ENDIF

mov Temp4, A

ajmp int0\_int\_fall\_gain\_done

int0\_int\_fall\_not\_multishot:

jnb Flags2.RCP\_ONESHOT42, int0\_int\_fall\_not\_oneshot\_42

; Oneshot42 - Add 2/256

clr C

mov A, Temp1

rlc A

mov A, Temp2

rlc A

mov Temp3, A

mov A, Temp1

add A, Temp3

mov Temp3, A

mov A, Temp2

addc A, #0

mov Temp4, A

ajmp int0\_int\_fall\_gain\_done

int0\_int\_fall\_not\_oneshot\_42:

jnb Flags2.RCP\_ONESHOT125, int0\_int\_fall\_not\_oneshot\_125

; Oneshot125 - multiply by 86/256

mov A, Temp1 ; Multiply by 86 and divide by 256

mov B, #56h

mul AB

mov Temp3, B

mov A, Temp2

mov B, #56h

mul AB

add A, Temp3

mov Temp3, A

xch A, B

addc A, #0

mov Temp4, A

ajmp int0\_int\_fall\_gain\_done

int0\_int\_fall\_not\_oneshot\_125:

; Regular signal - multiply by 43/1024

IF MCU\_48MHZ == 1

clr C

mov A, Temp3 ; Divide by 2

rrc A

mov Temp3, A

mov A, Temp2

rrc A

mov Temp2, A

mov A, Temp1

rrc A

mov Temp1, A

ENDIF

mov A, Temp1 ; Multiply by 43 and divide by 1024

IF MCU\_48MHZ == 0

mov B, #2Bh

ELSE

mov B, #56h ; Multiply by 86

ENDIF

mul AB

mov Temp3, B

mov A, Temp2

IF MCU\_48MHZ == 0

mov B, #2Bh

ELSE

mov B, #56h ; Multiply by 86

ENDIF

mul AB

add A, Temp3

mov Temp3, A

xch A, B

addc A, #0

clr C

rrc A ; Divide by 2 for total 512

mov Temp4, A

mov A, Temp3

rrc A

mov Temp3, A

clr C

mov A, Temp4 ; Divide by 2 for total 1024

rrc A

mov Temp4, A

mov A, Temp3

rrc A

mov Temp3, A

int0\_int\_fall\_gain\_done:

; Check if 2235us or above (in order to ignore false pulses)

clr C

mov A, Temp4 ; Is pulse 2235us or higher?

IF MCU\_48MHZ == 0

subb A, #09h

ELSE

subb A, #12h

ENDIF

jnc int0\_int\_outside\_range ; Yes - ignore pulse

; Check if below 900us (in order to ignore false pulses)

clr C

mov A, Temp3

IF MCU\_48MHZ == 0

subb A, #9Ah

ELSE

subb A, #34h

ENDIF

mov A, Temp4

IF MCU\_48MHZ == 0

subb A, #03h

ELSE

subb A, #07h

ENDIF

jnc int0\_int\_check\_full\_range ; No - proceed

int0\_int\_outside\_range:

inc Rcp\_Outside\_Range\_Cnt

mov A, Rcp\_Outside\_Range\_Cnt

jnz ($+4)

dec Rcp\_Outside\_Range\_Cnt

clr C

mov A, Rcp\_Outside\_Range\_Cnt

subb A, #50 ; Allow a given number of outside pulses

jnc ($+4)

ajmp int0\_int\_set\_timeout ; If outside limits - ignore first pulses

mov New\_Rcp, #0 ; Set pulse length to zero

ajmp int0\_int\_exit ; Exit without reseting timeout

int0\_int\_check\_full\_range:

; Decrement outside range counter

mov A, Rcp\_Outside\_Range\_Cnt

jz ($+4)

dec Rcp\_Outside\_Range\_Cnt

; Calculate "1000us" plus throttle minimum

jnb Flags2.RCP\_FULL\_RANGE, int0\_int\_set\_min ; Check if full range is chosen

mov Temp5, #0 ; Set 1000us as default minimum

IF MCU\_48MHZ == 0

mov Temp6, #4

ELSE

mov Temp6, #8

ENDIF

ajmp int0\_int\_calculate

int0\_int\_set\_min:

mov Temp5, Min\_Throttle\_L ; Min throttle value scaled

mov Temp6, Min\_Throttle\_H

jnb Flags3.PGM\_BIDIR, ($+7)

mov Temp5, Center\_Throttle\_L ; Center throttle value scaled

mov Temp6, Center\_Throttle\_H

int0\_int\_calculate:

clr C

mov A, Temp3 ; Subtract minimum

subb A, Temp5

mov Temp3, A

mov A, Temp4

subb A, Temp6

mov Temp4, A

mov Bit\_Access\_Int.0, C

mov Temp7, Throttle\_Gain ; Load Temp7/Temp8 with throttle gain

mov Temp8, Throttle\_Gain\_M

jnb Flags3.PGM\_BIDIR, int0\_int\_not\_bidir ; If not bidirectional operation - branch

jnc int0\_int\_bidir\_fwd ; If result is positive - branch

jb Flags2.RCP\_DIR\_REV, int0\_int\_bidir\_rev\_chk ; If same direction - branch

setb Flags2.RCP\_DIR\_REV

ajmp int0\_int\_bidir\_rev\_chk

int0\_int\_bidir\_fwd:

jnb Flags2.RCP\_DIR\_REV, int0\_int\_bidir\_rev\_chk ; If same direction - branch

clr Flags2.RCP\_DIR\_REV

int0\_int\_bidir\_rev\_chk:

jnb Flags2.RCP\_DIR\_REV, ($+7)

mov Temp7, Throttle\_Gain\_BD\_Rev ; Load Temp7/Temp8 with throttle gain for bidirectional reverse

mov Temp8, Throttle\_Gain\_BD\_Rev\_M

jb Flags3.PGM\_BIDIR\_REV, ($+5)

cpl Flags2.RCP\_DIR\_REV

clr C ; Multiply throttle value by 2

mov A, Temp3

rlc A

mov Temp3, A

mov A, Temp4

rlc A

mov Temp4, A

mov C, Bit\_Access\_Int.0

jnc int0\_int\_bidir\_do\_deadband ; If result is positive - branch

mov A, Temp3 ; Change sign

cpl A

add A, #1

mov Temp3, A

mov A, Temp4

cpl A

addc A, #0

mov Temp4, A

int0\_int\_bidir\_do\_deadband:

clr C ; Subtract deadband

mov A, Temp3

IF MCU\_48MHZ == 0

subb A, #40

ELSE

subb A, #80

ENDIF

mov Temp3, A

mov A, Temp4

subb A, #0

mov Temp4, A

jnc int0\_int\_do\_throttle\_gain

mov Temp1, #0

mov Temp3, #0

mov Temp4, #0

ajmp int0\_int\_do\_throttle\_gain

int0\_int\_not\_bidir:

mov C, Bit\_Access\_Int.0

jnc int0\_int\_do\_throttle\_gain ; If result is positive - branch

int0\_int\_unidir\_neg:

mov Temp1, #0 ; Yes - set to minimum

mov Temp3, #0

mov Temp4, #0

ajmp int0\_int\_pulse\_ready

int0\_int\_do\_throttle\_gain:

; Boost pwm during direct start

mov A, Flags1

anl A, #((1 SHL STARTUP\_PHASE)+(1 SHL INITIAL\_RUN\_PHASE))

jz int0\_int\_startup\_boosted

jb Flags1.MOTOR\_STARTED, int0\_int\_startup\_boosted ; Do not boost when changing direction in bidirectional mode

mov A, Pwm\_Limit\_Beg ; Set 25% of max startup power as minimum power

IF MCU\_48MHZ == 1

rlc A

ENDIF

mov Temp2, A

mov A, Temp4

jnz int0\_int\_startup\_boost\_stall

clr C

mov A, Temp2

subb A, Temp3

jc int0\_int\_startup\_boost\_stall

mov A, Temp2

mov Temp3, A

int0\_int\_startup\_boost\_stall:

mov A, Stall\_Cnt ; Add an extra power boost during start

swap A

IF MCU\_48MHZ == 1

rlc A

ENDIF

add A, Temp3

mov Temp3, A

mov A, Temp4

addc A, #0

mov Temp4, A

int0\_int\_startup\_boosted:

mov A, Temp3 ; Multiply throttle value by throttle gain

mov B, Temp7 ; Temp7 has Throttle\_Gain

mul AB

mov Temp2, A

mov Temp3, B

mov A, Temp4

mov B, Temp7 ; Temp7 has Throttle\_Gain

mul AB

add A, Temp3

mov Temp3, A

xch A, B

addc A, #0

mov Temp4, A

clr C ; Generate 8bit number

mov A, Temp4

rrc A

mov Temp6, A

mov A, Temp3

rrc A

mov Temp1, A

IF MCU\_48MHZ == 1

clr C

mov A, Temp6

rrc A

mov Temp6, A

mov A, Temp1

rrc A

mov Temp1, A

ENDIF

inc Temp8 ; Temp8 has Throttle\_Gain\_M

int0\_int\_gain\_loop:

mov A, Temp8

dec A

jz int0\_int\_gain\_rcp\_done ; Skip one multiply by 2 of New\_Rcp

clr C

mov A, Temp1 ; Multiply New\_Rcp by 2

rlc A

mov Temp1, A

int0\_int\_gain\_rcp\_done:

clr C

mov A, Temp2 ; Multiply pwm by 2

rlc A

mov A, Temp3

rlc A

mov Temp3, A

mov A, Temp4

rlc A

mov Temp4, A

djnz Temp8, int0\_int\_gain\_loop

mov A, Temp4

IF MCU\_48MHZ == 0

jnb ACC.2, int0\_int\_pulse\_ready ; Check that RC pulse is within legal range

ELSE

jnb ACC.3, int0\_int\_pulse\_ready

ENDIF

mov Temp1, #0FFh

mov Temp3, #0FFh

IF MCU\_48MHZ == 0

mov Temp4, #3

ELSE

mov Temp4, #7

ENDIF

int0\_int\_pulse\_ready:

mov New\_Rcp, Temp1 ; Store new pulse length

setb Flags2.RCP\_UPDATED ; Set updated flag

; Check if zero

mov A, Temp1 ; Load new pulse value

jz ($+5) ; Check if pulse is zero

mov Rcp\_Stop\_Cnt, #0 ; Reset rcp stop counter

; Set pwm limit

clr C

mov A, Pwm\_Limit ; Limit to the smallest

mov Temp5, A ; Store limit in Temp5

subb A, Pwm\_Limit\_By\_Rpm

jc ($+4)

mov Temp5, Pwm\_Limit\_By\_Rpm

; Check against limit

clr C

mov A, Temp5

subb A, New\_Rcp

jnc int0\_int\_set\_pwm\_registers

mov A, Temp5 ; Multiply limit by 4 (8 for 48MHz MCUs)

IF MCU\_48MHZ == 0

mov B, #4

ELSE

mov B, #8

ENDIF

mul AB

mov Temp3, A

mov Temp4, B

int0\_int\_set\_pwm\_registers:

mov A, Temp3

cpl A

mov Temp1, A

mov A, Temp4

cpl A

IF MCU\_48MHZ == 0

anl A, #3

ELSE

anl A, #7

ENDIF

mov Temp2, A

IF FETON\_DELAY != 0

clr C

mov A, Temp1 ; Skew damping fet timing

IF MCU\_48MHZ == 0

subb A, #FETON\_DELAY

ELSE

subb A, #(FETON\_DELAY SHL 1)

ENDIF

mov Temp3, A

mov A, Temp2

subb A, #0

mov Temp4, A

jnc int0\_int\_set\_pwm\_damp\_set

mov Temp3, #0

mov Temp4, #0

int0\_int\_set\_pwm\_damp\_set:

ENDIF

mov Power\_Pwm\_Reg\_L, Temp1

mov Power\_Pwm\_Reg\_H, Temp2

IF FETON\_DELAY != 0

mov Damp\_Pwm\_Reg\_L, Temp3

mov Damp\_Pwm\_Reg\_H, Temp4

ENDIF

mov Rcp\_Timeout\_Cntd, #10 ; Set timeout count

IF FETON\_DELAY != 0

pop B ; Restore preserved registers

pop ACC

pop PSW

Clear\_COVF\_Interrupt

Enable\_COVF\_Interrupt ; Generate a pca interrupt

orl EIE1, #10h ; Enable pca interrupts

reti

ELSE

mov A, Current\_Power\_Pwm\_Reg\_H

IF MCU\_48MHZ == 0

jnb ACC.1, int0\_int\_set\_pca\_int\_hi\_pwm

ELSE

jnb ACC.2, int0\_int\_set\_pca\_int\_hi\_pwm

ENDIF

pop B ; Restore preserved registers

pop ACC

pop PSW

Clear\_COVF\_Interrupt

Enable\_COVF\_Interrupt ; Generate a pca interrupt

orl EIE1, #10h ; Enable pca interrupts

reti

int0\_int\_set\_pca\_int\_hi\_pwm:

pop B ; Restore preserved registers

pop ACC

pop PSW

Clear\_CCF\_Interrupt

Enable\_CCF\_Interrupt ; Generate pca interrupt

orl EIE1, #10h ; Enable pca interrupts

reti

ENDIF

int0\_int\_set\_timeout:

mov Rcp\_Timeout\_Cntd, #10 ; Set timeout count

int0\_int\_exit:

pop B ; Restore preserved registers

pop ACC

pop PSW

orl EIE1, #10h ; Enable pca interrupts

reti

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

;

; Int1 interrupt routine

;

; No assumptions

;

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

int1\_int: ; Used for RC pulse timing

clr IE\_EX1 ; Disable int1 interrupts

setb TCON\_TR1 ; Start timer 1

clr TMR2CN0\_TR2 ; Timer 2 disabled

mov DShot\_Frame\_Start\_L, TMR2L ; Read timer value

mov DShot\_Frame\_Start\_H, TMR2H

setb TMR2CN0\_TR2 ; Timer 2 enabled

reti

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

;

; PCA interrupt routine

;

; No assumptions

;

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

pca\_int: ; Used for setting pwm registers

clr IE\_EA

push PSW ; Preserve registers through interrupt

push ACC

setb PSW.3 ; Select register bank 1 for this interrupt

IF FETON\_DELAY != 0 ; HI/LO enable style drivers

mov Temp1, PCA0L ; Read low byte, to transfer high byte to holding register

mov A, Current\_Power\_Pwm\_Reg\_H

IF MCU\_48MHZ == 0

jnb ACC.1, pca\_int\_hi\_pwm

ELSE

jnb ACC.2, pca\_int\_hi\_pwm

ENDIF

mov A, PCA0H

IF MCU\_48MHZ == 0

jb ACC.1, pca\_int\_exit ; Power below 50%, update pca in the 0x00-0x0F range

jb ACC.0, pca\_int\_exit

ELSE

jb ACC.2, pca\_int\_exit

jb ACC.1, pca\_int\_exit

ENDIF

ajmp pca\_int\_set\_pwm

pca\_int\_hi\_pwm:

mov A, PCA0H

IF MCU\_48MHZ == 0

jnb ACC.1, pca\_int\_exit ; Power above 50%, update pca in the 0x20-0x2F range

jb ACC.0, pca\_int\_exit

ELSE

jnb ACC.2, pca\_int\_exit

jb ACC.1, pca\_int\_exit

ENDIF

pca\_int\_set\_pwm:

Set\_Power\_Pwm\_Regs

Set\_Damp\_Pwm\_Regs

mov Current\_Power\_Pwm\_Reg\_H, Power\_Pwm\_Reg\_H

Disable\_COVF\_Interrupt

ELSE ; EN/PWM style drivers

Set\_Power\_Pwm\_Regs

mov Current\_Power\_Pwm\_Reg\_H, Power\_Pwm\_Reg\_H

Disable\_COVF\_Interrupt

Disable\_CCF\_Interrupt

ENDIF

; Pwm updated, enable/disable interrupts

setb IE\_EX0 ; Enable int0 interrupts

jnb Flags2.RCP\_DSHOT, ($+5)

setb IE\_EX1 ; Enable int1 interrupts (DShot only)

anl EIE1, #0EFh ; Disable pca interrupts

pca\_int\_exit:

Clear\_COVF\_Interrupt

IF FETON\_DELAY == 0

Clear\_CCF\_Interrupt

ENDIF

pop ACC ; Restore preserved registers

pop PSW

setb IE\_EA

reti

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

;

; Wait xms ~(x\*4\*250) (Different entry points)

;

; No assumptions

;

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

wait1ms:

mov Temp2, #1

jmp waitxms\_o

wait3ms:

mov Temp2, #3

jmp waitxms\_o

wait10ms:

mov Temp2, #10

jmp waitxms\_o

wait30ms:

mov Temp2, #30

jmp waitxms\_o

wait100ms:

mov Temp2, #100

jmp waitxms\_o

wait200ms:

mov Temp2, #200

jmp waitxms\_o

waitxms\_o: ; Outer loop

mov Temp1, #23

waitxms\_m: ; Middle loop

clr A

djnz ACC, $ ; Inner loop (42.7us - 1024 cycles)

djnz Temp1, waitxms\_m

djnz Temp2, waitxms\_o

ret

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

;

; Set pwm limit low rpm

;

; No assumptions

;

; Sets power limit for low rpms and disables demag for low rpms

;

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

set\_pwm\_limit\_low\_rpm:

; Set pwm limit

mov Temp1, #0FFh ; Default full power

jb Flags1.STARTUP\_PHASE, set\_pwm\_limit\_low\_rpm\_exit ; Exit if startup phase set

mov Temp2, #Pgm\_Enable\_Power\_Prot ; Check if low RPM power protection is enabled

mov A, @Temp2

jz set\_pwm\_limit\_low\_rpm\_exit ; Exit if disabled

mov A, Comm\_Period4x\_H

jz set\_pwm\_limit\_low\_rpm\_exit ; Avoid divide by zero

mov A, #255 ; Divide 255 by Comm\_Period4x\_H

mov B, Comm\_Period4x\_H

div AB

mov B, Low\_Rpm\_Pwr\_Slope ; Multiply by slope

jnb Flags1.INITIAL\_RUN\_PHASE, ($+6) ; More protection for initial run phase

mov B, #5

mul AB

mov Temp1, A ; Set new limit

xch A, B

jz ($+4) ; Limit to max

mov Temp1, #0FFh

clr C

mov A, Temp1 ; Limit to min

subb A, Pwm\_Limit\_Beg

jnc set\_pwm\_limit\_low\_rpm\_exit

mov Temp1, Pwm\_Limit\_Beg

set\_pwm\_limit\_low\_rpm\_exit:

mov Pwm\_Limit\_By\_Rpm, Temp1

ret

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

;

; Set pwm limit high rpm

;

; No assumptions

;

; Sets power limit for high rpms

;

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

set\_pwm\_limit\_high\_rpm:

IF MCU\_48MHZ == 1

clr C

mov A, Comm\_Period4x\_L

subb A, #0A0h ; Limit Comm\_Period to 160, which is 500k erpm

mov A, Comm\_Period4x\_H

subb A, #00h

ELSE

clr C

mov A, Comm\_Period4x\_L

subb A, #0E4h ; Limit Comm\_Period to 228, which is 350k erpm

mov A, Comm\_Period4x\_H

subb A, #00h

ENDIF

mov A, Pwm\_Limit\_By\_Rpm

jnc set\_pwm\_limit\_high\_rpm\_inc\_limit

dec A

ajmp set\_pwm\_limit\_high\_rpm\_store

set\_pwm\_limit\_high\_rpm\_inc\_limit:

inc A

set\_pwm\_limit\_high\_rpm\_store:

jz ($+4)

mov Pwm\_Limit\_By\_Rpm, A

ret

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

;

; Start ADC conversion

;

; No assumptions

;

; Start conversion used for measuring power supply voltage

;

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

start\_adc\_conversion:

; Start adc

Start\_Adc

ret

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

;

; Check temperature, power supply voltage and limit power

;

; No assumptions

;

; Used to limit main motor power in order to maintain the required voltage

;

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

check\_temp\_voltage\_and\_limit\_power:

inc Adc\_Conversion\_Cnt ; Increment conversion counter

clr C

mov A, Adc\_Conversion\_Cnt ; Is conversion count equal to temp rate?

subb A, #8

jc check\_voltage\_start ; No - check voltage

; Wait for ADC conversion to complete

jnb ADC0CN0\_ADINT, check\_temp\_voltage\_and\_limit\_power

; Read ADC result

Read\_Adc\_Result

; Stop ADC

Stop\_Adc

mov Adc\_Conversion\_Cnt, #0 ; Yes - temperature check. Reset counter

mov A, Temp2 ; Move ADC MSB to Temp3

mov Temp3, A

mov Temp2, #Pgm\_Enable\_Temp\_Prot ; Is temp protection enabled?

mov A, @Temp2

jz temp\_check\_exit ; No - branch

mov A, Temp3 ; Is temperature reading below 256?

jnz temp\_average\_inc\_dec ; No - proceed

mov A, Current\_Average\_Temp ; Yes - decrement average

jz temp\_average\_updated ; Already zero - no change

jmp temp\_average\_dec ; Decrement

temp\_average\_inc\_dec:

clr C

mov A, Temp1 ; Check if current temperature is above or below average

subb A, Current\_Average\_Temp

jz temp\_average\_updated\_load\_acc ; Equal - no change

mov A, Current\_Average\_Temp ; Above - increment average

jnc temp\_average\_inc

jz temp\_average\_updated ; Below - decrement average if average is not already zero

temp\_average\_dec:

dec A ; Decrement average

jmp temp\_average\_updated

temp\_average\_inc:

inc A ; Increment average

jz temp\_average\_dec

jmp temp\_average\_updated

temp\_average\_updated\_load\_acc:

mov A, Current\_Average\_Temp

temp\_average\_updated:

mov Current\_Average\_Temp, A

clr C

subb A, Temp\_Prot\_Limit ; Is temperature below first limit?

jc temp\_check\_exit ; Yes - exit

mov Pwm\_Limit, #192 ; No - limit pwm

clr C

subb A, #(TEMP\_LIMIT\_STEP/2) ; Is temperature below second limit

jc temp\_check\_exit ; Yes - exit

mov Pwm\_Limit, #128 ; No - limit pwm

clr C

subb A, #(TEMP\_LIMIT\_STEP/2) ; Is temperature below third limit

jc temp\_check\_exit ; Yes - exit

mov Pwm\_Limit, #64 ; No - limit pwm

clr C

subb A, #(TEMP\_LIMIT\_STEP/2) ; Is temperature below final limit

jc temp\_check\_exit ; Yes - exit

mov Pwm\_Limit, #0 ; No - limit pwm

temp\_check\_exit:

ret

check\_voltage\_start:

; Increase pwm limit

mov A, Pwm\_Limit

add A, #16

jnc ($+4) ; If not max - branch

mov A, #255

mov Pwm\_Limit, A ; Increment limit

ret

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

;

; Set startup PWM routine

;

; Either the SETTLE\_PHASE or the STEPPER\_PHASE flag must be set

;

; Used for pwm control during startup

;

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

set\_startup\_pwm:

; Adjust startup power

mov A, #50 ; Set power

mov Temp2, #Pgm\_Startup\_Pwr\_Decoded

mov B, @Temp2

mul AB

xch A, B

mov C, B.7 ; Multiply result by 2 (unity gain is 128)

rlc A

mov Pwm\_Limit\_Beg, A ; Set initial pwm limit

ret

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

;

; Initialize timing routine

;

; No assumptions

;

; Part of initialization before motor start

;

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

initialize\_timing:

mov Comm\_Period4x\_L, #00h ; Set commutation period registers

mov Comm\_Period4x\_H, #0F0h

ret

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

;

; Calculate next commutation timing routine

;

; No assumptions

;

; Called immediately after each commutation

; Also sets up timer 3 to wait advance timing

; Two entry points are used

;

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

calc\_next\_comm\_timing: ; Entry point for run phase

; Read commutation time

clr IE\_EA

clr TMR2CN0\_TR2 ; Timer 2 disabled

mov Temp1, TMR2L ; Load timer value

mov Temp2, TMR2H

mov Temp3, Timer2\_X

jnb TMR2CN0\_TF2H, ($+4) ; Check if interrupt is pending

inc Temp3 ; If it is pending, then timer has already wrapped

setb TMR2CN0\_TR2 ; Timer 2 enabled

setb IE\_EA

IF MCU\_48MHZ == 1

clr C

mov A, Temp3

rrc A

mov Temp3, A

mov A, Temp2

rrc A

mov Temp2, A

mov A, Temp1

rrc A

mov Temp1, A

ENDIF

; Calculate this commutation time

mov Temp4, Prev\_Comm\_L

mov Temp5, Prev\_Comm\_H

mov Prev\_Comm\_L, Temp1 ; Store timestamp as previous commutation

mov Prev\_Comm\_H, Temp2

clr C

mov A, Temp1

subb A, Temp4 ; Calculate the new commutation time

mov Temp1, A

mov A, Temp2

subb A, Temp5

jb Flags1.STARTUP\_PHASE, calc\_next\_comm\_startup

IF MCU\_48MHZ == 1

anl A, #7Fh

ENDIF

mov Temp2, A

jnb Flags1.HIGH\_RPM, ($+5) ; Branch if high rpm

ajmp calc\_next\_comm\_timing\_fast

ajmp calc\_next\_comm\_normal

calc\_next\_comm\_startup:

mov Temp6, Prev\_Comm\_X

mov Prev\_Comm\_X, Temp3 ; Store extended timestamp as previous commutation

mov Temp2, A

mov A, Temp3

subb A, Temp6 ; Calculate the new extended commutation time

IF MCU\_48MHZ == 1

anl A, #7Fh

ENDIF

mov Temp3, A

jz calc\_next\_comm\_startup\_no\_X

mov Temp1, #0FFh

mov Temp2, #0FFh

ajmp calc\_next\_comm\_startup\_average

calc\_next\_comm\_startup\_no\_X:

mov Temp7, Prev\_Prev\_Comm\_L

mov Temp8, Prev\_Prev\_Comm\_H

mov Prev\_Prev\_Comm\_L, Temp4

mov Prev\_Prev\_Comm\_H, Temp5

mov Temp1, Prev\_Comm\_L ; Reload this commutation time

mov Temp2, Prev\_Comm\_H

clr C

mov A, Temp1

subb A, Temp7 ; Calculate the new commutation time based upon the two last commutations (to reduce sensitivity to offset)

mov Temp1, A

mov A, Temp2

subb A, Temp8

mov Temp2, A

calc\_next\_comm\_startup\_average:

clr C

mov A, Comm\_Period4x\_H ; Average with previous and save

rrc A

mov Temp4, A

mov A, Comm\_Period4x\_L

rrc A

mov Temp3, A

mov A, Temp1

add A, Temp3

mov Comm\_Period4x\_L, A

mov A, Temp2

addc A, Temp4

mov Comm\_Period4x\_H, A

jnc ($+8)

mov Comm\_Period4x\_L, #0FFh

mov Comm\_Period4x\_H, #0FFh

ajmp calc\_new\_wait\_times\_setup

calc\_next\_comm\_normal:

; Calculate new commutation time

mov Temp3, Comm\_Period4x\_L ; Comm\_Period4x(-l-h) holds the time of 4 commutations

mov Temp4, Comm\_Period4x\_H

mov Temp5, Comm\_Period4x\_L ; Copy variables

mov Temp6, Comm\_Period4x\_H

mov Temp7, #4 ; Divide Comm\_Period4x 4 times as default

mov Temp8, #2 ; Divide new commutation time 2 times as default

clr C

mov A, Temp4

subb A, #04h

jc calc\_next\_comm\_avg\_period\_div

dec Temp7 ; Reduce averaging time constant for low speeds

dec Temp8

clr C

mov A, Temp4

subb A, #08h

jc calc\_next\_comm\_avg\_period\_div

jb Flags1.INITIAL\_RUN\_PHASE, calc\_next\_comm\_avg\_period\_div ; Do not average very fast during initial run

dec Temp7 ; Reduce averaging time constant more for even lower speeds

dec Temp8

calc\_next\_comm\_avg\_period\_div:

clr C

mov A, Temp6

rrc A ; Divide by 2

mov Temp6, A

mov A, Temp5

rrc A

mov Temp5, A

djnz Temp7, calc\_next\_comm\_avg\_period\_div

clr C

mov A, Temp3

subb A, Temp5 ; Subtract a fraction

mov Temp3, A

mov A, Temp4

subb A, Temp6

mov Temp4, A

mov A, Temp8 ; Divide new time

jz calc\_next\_comm\_new\_period\_div\_done

calc\_next\_comm\_new\_period\_div:

clr C

mov A, Temp2

rrc A ; Divide by 2

mov Temp2, A

mov A, Temp1

rrc A

mov Temp1, A

djnz Temp8, calc\_next\_comm\_new\_period\_div

calc\_next\_comm\_new\_period\_div\_done:

mov A, Temp3

add A, Temp1 ; Add the divided new time

mov Temp3, A

mov A, Temp4

addc A, Temp2

mov Temp4, A

mov Comm\_Period4x\_L, Temp3 ; Store Comm\_Period4x\_X

mov Comm\_Period4x\_H, Temp4

jnc calc\_new\_wait\_times\_setup; If period larger than 0xffff - go to slow case

mov Temp4, #0FFh

mov Comm\_Period4x\_L, Temp4 ; Set commutation period registers to very slow timing (0xffff)

mov Comm\_Period4x\_H, Temp4

calc\_new\_wait\_times\_setup:

; Set high rpm bit (if above 156k erpm)

clr C

mov A, Temp4

subb A, #2

jnc ($+4)

setb Flags1.HIGH\_RPM ; Set high rpm bit

; Load programmed commutation timing

jnb Flags1.STARTUP\_PHASE, calc\_new\_wait\_per\_startup\_done ; Set dedicated timing during startup

mov Temp8, #3

ajmp calc\_new\_wait\_per\_demag\_done

calc\_new\_wait\_per\_startup\_done:

mov Temp1, #Pgm\_Comm\_Timing ; Load timing setting

mov A, @Temp1

mov Temp8, A ; Store in Temp8

clr C

mov A, Demag\_Detected\_Metric ; Check demag metric

subb A, #130

jc calc\_new\_wait\_per\_demag\_done

inc Temp8 ; Increase timing

clr C

mov A, Demag\_Detected\_Metric

subb A, #160

jc ($+3)

inc Temp8 ; Increase timing again

clr C

mov A, Temp8 ; Limit timing to max

subb A, #6

jc ($+4)

mov Temp8, #5 ; Set timing to max

calc\_new\_wait\_per\_demag\_done:

; Set timing reduction

mov Temp7, #2

; Load current commutation timing

mov A, Comm\_Period4x\_H ; Divide 4 times

swap A

anl A, #00Fh

mov Temp2, A

mov A, Comm\_Period4x\_H

swap A

anl A, #0F0h

mov Temp1, A

mov A, Comm\_Period4x\_L

swap A

anl A, #00Fh

add A, Temp1

mov Temp1, A

clr C

mov A, Temp1

subb A, Temp7

mov Temp3, A

mov A, Temp2

subb A, #0

mov Temp4, A

jc load\_min\_time ; Check that result is still positive

clr C

mov A, Temp3

subb A, #1

mov A, Temp4

subb A, #0

jnc calc\_new\_wait\_times\_exit ; Check that result is still above minumum

load\_min\_time:

mov Temp3, #1

clr A

mov Temp4, A

calc\_new\_wait\_times\_exit:

ljmp wait\_advance\_timing

; Fast calculation (Comm\_Period4x\_H less than 2)

calc\_next\_comm\_timing\_fast:

; Calculate new commutation time

mov Temp3, Comm\_Period4x\_L ; Comm\_Period4x(-l-h) holds the time of 4 commutations

mov Temp4, Comm\_Period4x\_H

mov A, Temp4 ; Divide by 2 4 times

swap A

mov Temp7, A

mov A, Temp3

swap A

anl A, #0Fh

orl A, Temp7

mov Temp5, A

clr C

mov A, Temp3 ; Subtract a fraction

subb A, Temp5

mov Temp3, A

mov A, Temp4

subb A, #0

mov Temp4, A

clr C

mov A, Temp1

rrc A ; Divide by 2 2 times

clr C

rrc A

mov Temp1, A

mov A, Temp3 ; Add the divided new time

add A, Temp1

mov Temp3, A

mov A, Temp4

addc A, #0

mov Temp4, A

mov Comm\_Period4x\_L, Temp3 ; Store Comm\_Period4x\_X

mov Comm\_Period4x\_H, Temp4

clr C

mov A, Temp4 ; If erpm below 156k - go to normal case

subb A, #2

jc ($+4)

clr Flags1.HIGH\_RPM ; Clear high rpm bit

; Set timing reduction

mov Temp1, #2

mov A, Temp4 ; Divide by 2 4 times

swap A

mov Temp7, A

mov Temp4, #0

mov A, Temp3

swap A

anl A, #0Fh

orl A, Temp7

mov Temp3, A

clr C

mov A, Temp3

subb A, Temp1

mov Temp3, A

jc load\_min\_time\_fast ; Check that result is still positive

clr C

subb A, #1

jnc calc\_new\_wait\_times\_fast\_done ; Check that result is still above minumum

load\_min\_time\_fast:

mov Temp3, #1

calc\_new\_wait\_times\_fast\_done:

mov Temp1, #Pgm\_Comm\_Timing ; Load timing setting

mov A, @Temp1

mov Temp8, A ; Store in Temp8

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

;

; Wait advance timing routine

;

; No assumptions

; NOTE: Be VERY careful if using temp registers. They are passed over this routine

;

; Waits for the advance timing to elapse and sets up the next zero cross wait

;

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

wait\_advance\_timing:

jnb Flags0.T3\_PENDING, ($+5)

ajmp wait\_advance\_timing

; Setup next wait time

mov TMR3RLL, Wt\_ZC\_Tout\_Start\_L

mov TMR3RLH, Wt\_ZC\_Tout\_Start\_H

setb Flags0.T3\_PENDING

orl EIE1, #80h ; Enable timer 3 interrupts

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

;

; Calculate new wait times routine

;

; No assumptions

;

; Calculates new wait times

;

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

calc\_new\_wait\_times:

clr C

clr A

subb A, Temp3 ; Negate

mov Temp1, A

clr A

subb A, Temp4

mov Temp2, A

IF MCU\_48MHZ == 1

clr C

mov A, Temp1 ; Multiply by 2

rlc A

mov Temp1, A

mov A, Temp2

rlc A

mov Temp2, A

ENDIF

jnb Flags1.HIGH\_RPM, ($+6) ; Branch if high rpm

ljmp calc\_new\_wait\_times\_fast

mov A, Temp1 ; Copy values

mov Temp3, A

mov A, Temp2

mov Temp4, A

setb C ; Negative numbers - set carry

mov A, Temp2

rrc A ; Divide by 2

mov Temp6, A

mov A, Temp1

rrc A

mov Temp5, A

mov Wt\_Zc\_Tout\_Start\_L, Temp1; Set 15deg time for zero cross scan timeout

mov Wt\_Zc\_Tout\_Start\_H, Temp2

clr C

mov A, Temp8 ; (Temp8 has Pgm\_Comm\_Timing)

subb A, #3 ; Is timing normal?

jz store\_times\_decrease ; Yes - branch

mov A, Temp8

jb ACC.0, adjust\_timing\_two\_steps ; If an odd number - branch

mov A, Temp1 ; Add 7.5deg and store in Temp1/2

add A, Temp5

mov Temp1, A

mov A, Temp2

addc A, Temp6

mov Temp2, A

mov A, Temp5 ; Store 7.5deg in Temp3/4

mov Temp3, A

mov A, Temp6

mov Temp4, A

jmp store\_times\_up\_or\_down

adjust\_timing\_two\_steps:

mov A, Temp1 ; Add 15deg and store in Temp1/2

add A, Temp1

mov Temp1, A

mov A, Temp2

addc A, Temp2

mov Temp2, A

clr C

mov A, Temp1

add A, #1

mov Temp1, A

mov A, Temp2

addc A, #0

mov Temp2, A

mov Temp3, #-1 ; Store minimum time in Temp3/4

mov Temp4, #0FFh

store\_times\_up\_or\_down:

clr C

mov A, Temp8

subb A, #3 ; Is timing higher than normal?

jc store\_times\_decrease ; No - branch

store\_times\_increase:

mov Wt\_Comm\_Start\_L, Temp3 ; Now commutation time (~60deg) divided by 4 (~15deg nominal)

mov Wt\_Comm\_Start\_H, Temp4

mov Wt\_Adv\_Start\_L, Temp1 ; New commutation advance time (~15deg nominal)

mov Wt\_Adv\_Start\_H, Temp2

mov Wt\_Zc\_Scan\_Start\_L, Temp5 ; Use this value for zero cross scan delay (7.5deg)

mov Wt\_Zc\_Scan\_Start\_H, Temp6

ljmp wait\_before\_zc\_scan

store\_times\_decrease:

mov Wt\_Comm\_Start\_L, Temp1 ; Now commutation time (~60deg) divided by 4 (~15deg nominal)

mov Wt\_Comm\_Start\_H, Temp2

mov Wt\_Adv\_Start\_L, Temp3 ; New commutation advance time (~15deg nominal)

mov Wt\_Adv\_Start\_H, Temp4

mov Wt\_Zc\_Scan\_Start\_L, Temp5 ; Use this value for zero cross scan delay (7.5deg)

mov Wt\_Zc\_Scan\_Start\_H, Temp6

jnb Flags1.STARTUP\_PHASE, store\_times\_exit

mov Wt\_Comm\_Start\_L, #0F0h ; Set very short delays for all but advance time during startup, in order to widen zero cross capture range

mov Wt\_Comm\_Start\_H, #0FFh

mov Wt\_Zc\_Scan\_Start\_L, #0F0h

mov Wt\_Zc\_Scan\_Start\_H, #0FFh

mov Wt\_Zc\_Tout\_Start\_L, #0F0h

mov Wt\_Zc\_Tout\_Start\_H, #0FFh

store\_times\_exit:

ljmp wait\_before\_zc\_scan

calc\_new\_wait\_times\_fast:

mov A, Temp1 ; Copy values

mov Temp3, A

setb C ; Negative numbers - set carry

mov A, Temp1 ; Divide by 2

rrc A

mov Temp5, A

mov Wt\_Zc\_Tout\_Start\_L, Temp1; Set 15deg time for zero cross scan timeout

clr C

mov A, Temp8 ; (Temp8 has Pgm\_Comm\_Timing)

subb A, #3 ; Is timing normal?

jz store\_times\_decrease\_fast; Yes - branch

mov A, Temp8

jb ACC.0, adjust\_timing\_two\_steps\_fast ; If an odd number - branch

mov A, Temp1 ; Add 7.5deg and store in Temp1

add A, Temp5

mov Temp1, A

mov A, Temp5 ; Store 7.5deg in Temp3

mov Temp3, A

ajmp store\_times\_up\_or\_down\_fast

adjust\_timing\_two\_steps\_fast:

mov A, Temp1 ; Add 15deg and store in Temp1

add A, Temp1

add A, #1

mov Temp1, A

mov Temp3, #-1 ; Store minimum time in Temp3

store\_times\_up\_or\_down\_fast:

clr C

mov A, Temp8

subb A, #3 ; Is timing higher than normal?

jc store\_times\_decrease\_fast; No - branch

store\_times\_increase\_fast:

mov Wt\_Comm\_Start\_L, Temp3 ; Now commutation time (~60deg) divided by 4 (~15deg nominal)

mov Wt\_Adv\_Start\_L, Temp1 ; New commutation advance time (~15deg nominal)

mov Wt\_Zc\_Scan\_Start\_L, Temp5 ; Use this value for zero cross scan delay (7.5deg)

ljmp wait\_before\_zc\_scan

store\_times\_decrease\_fast:

mov Wt\_Comm\_Start\_L, Temp1 ; Now commutation time (~60deg) divided by 4 (~15deg nominal)

mov Wt\_Adv\_Start\_L, Temp3 ; New commutation advance time (~15deg nominal)

mov Wt\_Zc\_Scan\_Start\_L, Temp5 ; Use this value for zero cross scan delay (7.5deg)

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

;

; Wait before zero cross scan routine

;

; No assumptions

;

; Waits for the zero cross scan wait time to elapse

; Also sets up timer 3 for the zero cross scan timeout time

;

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

wait\_before\_zc\_scan:

jnb Flags0.T3\_PENDING, ($+5)

ajmp wait\_before\_zc\_scan

mov Startup\_Zc\_Timeout\_Cntd, #2

setup\_zc\_scan\_timeout:

setb Flags0.T3\_PENDING

orl EIE1, #80h ; Enable timer 3 interrupts

mov A, Flags1

anl A, #((1 SHL STARTUP\_PHASE)+(1 SHL INITIAL\_RUN\_PHASE))

jz wait\_before\_zc\_scan\_exit

mov Temp1, Comm\_Period4x\_L ; Set long timeout when starting

mov Temp2, Comm\_Period4x\_H

clr C

mov A, Temp2

rrc A

mov Temp2, A

mov A, Temp1

rrc A

mov Temp1, A

IF MCU\_48MHZ == 0

clr C

mov A, Temp2

rrc A

mov Temp2, A

mov A, Temp1

rrc A

mov Temp1, A

ENDIF

jnb Flags1.STARTUP\_PHASE, setup\_zc\_scan\_timeout\_startup\_done

mov A, Temp2

add A, #40h ; Increase timeout somewhat to avoid false wind up

mov Temp2, A

setup\_zc\_scan\_timeout\_startup\_done:

clr IE\_EA

anl EIE1, #7Fh ; Disable timer 3 interrupts

mov TMR3CN0, #00h ; Timer 3 disabled and interrupt flag cleared

clr C

clr A

subb A, Temp1 ; Set timeout

mov TMR3L, A

clr A

subb A, Temp2

mov TMR3H, A

mov TMR3CN0, #04h ; Timer 3 enabled and interrupt flag cleared

setb Flags0.T3\_PENDING

orl EIE1, #80h ; Enable timer 3 interrupts

setb IE\_EA

wait\_before\_zc\_scan\_exit:

ret

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

;

; Wait for comparator to go low/high routines

;

; No assumptions

;

; Waits for the zero cross scan wait time to elapse

; Then scans for comparator going low/high

;

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

wait\_for\_comp\_out\_low:

setb Flags0.DEMAG\_DETECTED ; Set demag detected flag as default

mov Comparator\_Read\_Cnt, #0 ; Reset number of comparator reads

mov Bit\_Access, #00h ; Desired comparator output

jnb Flags1.DIR\_CHANGE\_BRAKE, ($+6)

mov Bit\_Access, #40h

ajmp wait\_for\_comp\_out\_start

wait\_for\_comp\_out\_high:

setb Flags0.DEMAG\_DETECTED ; Set demag detected flag as default

mov Comparator\_Read\_Cnt, #0 ; Reset number of comparator reads

mov Bit\_Access, #40h ; Desired comparator output

jnb Flags1.DIR\_CHANGE\_BRAKE, ($+6)

mov Bit\_Access, #00h

wait\_for\_comp\_out\_start:

; Set number of comparator readings

mov Temp1, #1 ; Number of OK readings required

mov Temp2, #1 ; Max number of readings required

jb Flags1.HIGH\_RPM, comp\_scale\_samples ; Branch if high rpm

mov A, Flags1 ; Clear demag detected flag if start phases

anl A, #((1 SHL STARTUP\_PHASE)+(1 SHL INITIAL\_RUN\_PHASE))

jz ($+4)

clr Flags0.DEMAG\_DETECTED

mov Temp2, #20 ; Too low value (~<15) causes rough running at pwm harmonics. Too high a value (~>35) causes the RCT4215 630 to run rough on full throttle

mov A, Comm\_Period4x\_H ; Set number of readings higher for lower speeds

clr C

rrc A

jnz ($+3)

inc A

mov Temp1, A

clr C

subb A, #20

jc ($+4)

mov Temp1, #20

jnb Flags1.STARTUP\_PHASE, comp\_scale\_samples

mov Temp1, #27 ; Set many samples during startup, approximately one pwm period

mov Temp2, #27

comp\_scale\_samples:

IF MCU\_48MHZ == 1

clr C

mov A, Temp1

rlc A

mov Temp1, A

clr C

mov A, Temp2

rlc A

mov Temp2, A

ENDIF

comp\_check\_timeout:

jb Flags0.T3\_PENDING, comp\_check\_timeout\_not\_timed\_out ; Has zero cross scan timeout elapsed?

mov A, Comparator\_Read\_Cnt ; Check that comparator has been read

jz comp\_check\_timeout\_not\_timed\_out ; If not read - branch

jnb Flags1.STARTUP\_PHASE, comp\_check\_timeout\_timeout\_extended ; Extend timeout during startup

djnz Startup\_Zc\_Timeout\_Cntd, comp\_check\_timeout\_extend\_timeout

comp\_check\_timeout\_timeout\_extended:

setb Flags0.COMP\_TIMED\_OUT

ajmp setup\_comm\_wait

comp\_check\_timeout\_extend\_timeout:

call setup\_zc\_scan\_timeout

comp\_check\_timeout\_not\_timed\_out:

inc Comparator\_Read\_Cnt ; Increment comparator read count

Read\_Comp\_Out ; Read comparator output

anl A, #40h

cjne A, Bit\_Access, comp\_read\_wrong

ajmp comp\_read\_ok

comp\_read\_wrong:

jnb Flags1.STARTUP\_PHASE, comp\_read\_wrong\_not\_startup

inc Temp1 ; Increment number of OK readings required

clr C

mov A, Temp1

subb A, Temp2 ; If above initial requirement - do not increment further

jc ($+3)

dec Temp1

ajmp comp\_check\_timeout ; Continue to look for good ones

comp\_read\_wrong\_not\_startup:

jb Flags0.DEMAG\_DETECTED, comp\_read\_wrong\_extend\_timeout

inc Temp1 ; Increment number of OK readings required

clr C

mov A, Temp1

subb A, Temp2

jc ($+4)

ajmp wait\_for\_comp\_out\_start ; If above initial requirement - go back and restart

ajmp comp\_check\_timeout ; Otherwise - take another reading

comp\_read\_wrong\_extend\_timeout:

clr Flags0.DEMAG\_DETECTED ; Clear demag detected flag

anl EIE1, #7Fh ; Disable timer 3 interrupts

mov TMR3CN0, #00h ; Timer 3 disabled and interrupt flag cleared

jnb Flags1.HIGH\_RPM, comp\_read\_wrong\_low\_rpm ; Branch if not high rpm

mov TMR3L, #00h ; Set timeout to ~1ms

IF MCU\_48MHZ == 1

mov TMR3H, #0F0h

ELSE

mov TMR3H, #0F8h

ENDIF

comp\_read\_wrong\_timeout\_set:

mov TMR3CN0, #04h ; Timer 3 enabled and interrupt flag cleared

setb Flags0.T3\_PENDING

orl EIE1, #80h ; Enable timer 3 interrupts

ljmp wait\_for\_comp\_out\_start ; If comparator output is not correct - go back and restart

comp\_read\_wrong\_low\_rpm:

mov A, Comm\_Period4x\_H ; Set timeout to ~4x comm period 4x value

mov Temp7, #0FFh ; Default to long

IF MCU\_48MHZ == 1

clr C

rlc A

jc comp\_read\_wrong\_load\_timeout

ENDIF

clr C

rlc A

jc comp\_read\_wrong\_load\_timeout

clr C

rlc A

jc comp\_read\_wrong\_load\_timeout

mov Temp7, A

comp\_read\_wrong\_load\_timeout:

clr C

clr A

subb A, Temp7

mov TMR3L, #0

mov TMR3H, A

ajmp comp\_read\_wrong\_timeout\_set

comp\_read\_ok:

clr C

mov A, Startup\_Cnt ; Force a timeout for the first commutation

subb A, #1

jnc ($+4)

ajmp wait\_for\_comp\_out\_start

jnb Flags0.DEMAG\_DETECTED, ($+5) ; Do not accept correct comparator output if it is demag

ajmp wait\_for\_comp\_out\_start

djnz Temp1, comp\_read\_ok\_jmp ; Decrement readings counter - repeat comparator reading if not zero

ajmp ($+4)

comp\_read\_ok\_jmp:

ajmp comp\_check\_timeout

clr Flags0.COMP\_TIMED\_OUT

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

;

; Setup commutation timing routine

;

; No assumptions

;

; Sets up and starts wait from commutation to zero cross

;

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

setup\_comm\_wait:

clr IE\_EA

anl EIE1, #7Fh ; Disable timer 3 interrupts

mov TMR3CN0, #00h ; Timer 3 disabled and interrupt flag cleared

mov TMR3L, Wt\_Comm\_Start\_L

mov TMR3H, Wt\_Comm\_Start\_H

mov TMR3CN0, #04h ; Timer 3 enabled and interrupt flag cleared

; Setup next wait time

mov TMR3RLL, Wt\_Adv\_Start\_L

mov TMR3RLH, Wt\_Adv\_Start\_H

setb Flags0.T3\_PENDING

orl EIE1, #80h ; Enable timer 3 interrupts

setb IE\_EA ; Enable interrupts again

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

;

; Evaluate comparator integrity

;

; No assumptions

;

; Checks comparator signal behaviour versus expected behaviour

;

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

evaluate\_comparator\_integrity:

mov A, Flags1

anl A, #((1 SHL STARTUP\_PHASE)+(1 SHL INITIAL\_RUN\_PHASE))

jz eval\_comp\_check\_timeout

jb Flags1.INITIAL\_RUN\_PHASE, ($+5) ; Do not increment beyond startup phase

inc Startup\_Cnt ; Increment counter

jmp eval\_comp\_exit

eval\_comp\_check\_timeout:

jnb Flags0.COMP\_TIMED\_OUT, eval\_comp\_exit ; Has timeout elapsed?

jb Flags1.DIR\_CHANGE\_BRAKE, eval\_comp\_exit ; Do not exit run mode if it is braking

jb Flags0.DEMAG\_DETECTED, eval\_comp\_exit ; Do not exit run mode if it is a demag situation

dec SP ; Routine exit without "ret" command

dec SP

ljmp run\_to\_wait\_for\_power\_on\_fail ; Yes - exit run mode

eval\_comp\_exit:

ret

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

;

; Wait for commutation routine

;

; No assumptions

;

; Waits from zero cross to commutation

;

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

wait\_for\_comm:

; Update demag metric

mov Temp1, #0

jnb Flags0.DEMAG\_DETECTED, ($+5)

mov Temp1, #1

mov A, Demag\_Detected\_Metric ; Sliding average of 8, 256 when demag and 0 when not. Limited to minimum 120

mov B, #7

mul AB ; Multiply by 7

mov Temp2, A

mov A, B ; Add new value for current demag status

add A, Temp1

mov B, A

mov A, Temp2

mov C, B.0 ; Divide by 8

rrc A

mov C, B.1

rrc A

mov C, B.2

rrc A

mov Demag\_Detected\_Metric, A

clr C

subb A, #120 ; Limit to minimum 120

jnc ($+5)

mov Demag\_Detected\_Metric, #120

clr C

mov A, Demag\_Detected\_Metric ; Check demag metric

subb A, Demag\_Pwr\_Off\_Thresh

jc wait\_for\_comm\_wait ; Cut power if many consecutive demags. This will help retain sync during hard accelerations

All\_pwmFETs\_off

Set\_Pwms\_Off

wait\_for\_comm\_wait:

jnb Flags0.T3\_PENDING, ($+5)

ajmp wait\_for\_comm\_wait

; Setup next wait time

mov TMR3RLL, Wt\_Zc\_Scan\_Start\_L

mov TMR3RLH, Wt\_Zc\_Scan\_Start\_H

setb Flags0.T3\_PENDING

orl EIE1, #80h ; Enable timer 3 interrupts

ret

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

;

; Commutation routines

;

; No assumptions

;

; Performs commutation switching

;

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

; Comm phase 1 to comm phase 2

comm1comm2:

Set\_RPM\_Out

jb Flags3.PGM\_DIR\_REV, comm12\_rev

clr IE\_EA ; Disable all interrupts

BcomFET\_off ; Turn off comfet

AcomFET\_on ; Turn on comfet

Set\_Pwm\_C ; To reapply power after a demag cut

setb IE\_EA

Set\_Comp\_Phase\_B ; Set comparator phase

jmp comm\_exit

comm12\_rev:

clr IE\_EA ; Disable all interrupts

BcomFET\_off ; Turn off comfet

CcomFET\_on ; Turn on comfet (reverse)

Set\_Pwm\_A ; To reapply power after a demag cut

setb IE\_EA

Set\_Comp\_Phase\_B ; Set comparator phase

jmp comm\_exit

; Comm phase 2 to comm phase 3

comm2comm3:

Clear\_RPM\_Out

jb Flags3.PGM\_DIR\_REV, comm23\_rev

clr IE\_EA ; Disable all interrupts

CpwmFET\_off ; Turn off pwmfet

Set\_Pwm\_B ; To reapply power after a demag cut

AcomFET\_on

setb IE\_EA

Set\_Comp\_Phase\_C ; Set comparator phase

ajmp comm\_exit

comm23\_rev:

clr IE\_EA ; Disable all interrupts

ApwmFET\_off ; Turn off pwmfet (reverse)

Set\_Pwm\_B ; To reapply power after a demag cut

CcomFET\_on

setb IE\_EA

Set\_Comp\_Phase\_A ; Set comparator phase (reverse)

ajmp comm\_exit

; Comm phase 3 to comm phase 4

comm3comm4:

Set\_RPM\_Out

jb Flags3.PGM\_DIR\_REV, comm34\_rev

clr IE\_EA ; Disable all interrupts

AcomFET\_off ; Turn off comfet

CcomFET\_on ; Turn on comfet

Set\_Pwm\_B ; To reapply power after a demag cut

setb IE\_EA

Set\_Comp\_Phase\_A ; Set comparator phase

jmp comm\_exit

comm34\_rev:

clr IE\_EA ; Disable all interrupts

CcomFET\_off ; Turn off comfet (reverse)

AcomFET\_on ; Turn on comfet (reverse)

Set\_Pwm\_B ; To reapply power after a demag cut

setb IE\_EA

Set\_Comp\_Phase\_C ; Set comparator phase (reverse)

jmp comm\_exit

; Comm phase 4 to comm phase 5

comm4comm5:

Clear\_RPM\_Out

jb Flags3.PGM\_DIR\_REV, comm45\_rev

clr IE\_EA ; Disable all interrupts

BpwmFET\_off ; Turn off pwmfet

Set\_Pwm\_A ; To reapply power after a demag cut

CcomFET\_on

setb IE\_EA

Set\_Comp\_Phase\_B ; Set comparator phase

jmp comm\_exit

comm45\_rev:

clr IE\_EA ; Disable all interrupts

BpwmFET\_off ; Turn off pwmfet

Set\_Pwm\_C

AcomFET\_on ; To reapply power after a demag cut

setb IE\_EA

Set\_Comp\_Phase\_B ; Set comparator phase

jmp comm\_exit

; Comm phase 5 to comm phase 6

comm5comm6:

Set\_RPM\_Out

jb Flags3.PGM\_DIR\_REV, comm56\_rev

clr IE\_EA ; Disable all interrupts

CcomFET\_off ; Turn off comfet

BcomFET\_on ; Turn on comfet

Set\_Pwm\_A ; To reapply power after a demag cut

setb IE\_EA

Set\_Comp\_Phase\_C ; Set comparator phase

jmp comm\_exit

comm56\_rev:

clr IE\_EA ; Disable all interrupts

AcomFET\_off ; Turn off comfet (reverse)

BcomFET\_on ; Turn on comfet

Set\_Pwm\_C ; To reapply power after a demag cut

setb IE\_EA

Set\_Comp\_Phase\_A ; Set comparator phase (reverse)

jmp comm\_exit

; Comm phase 6 to comm phase 1

comm6comm1:

Clear\_RPM\_Out

jb Flags3.PGM\_DIR\_REV, comm61\_rev

clr IE\_EA ; Disable all interrupts

ApwmFET\_off ; Turn off pwmfet

Set\_Pwm\_C

BcomFET\_on ; To reapply power after a demag cut

setb IE\_EA

Set\_Comp\_Phase\_A ; Set comparator phase

jmp comm\_exit

comm61\_rev:

clr IE\_EA ; Disable all interrupts

CpwmFET\_off ; Turn off pwmfet (reverse)

Set\_Pwm\_A

BcomFET\_on ; To reapply power after a demag cut

setb IE\_EA

Set\_Comp\_Phase\_C ; Set comparator phase (reverse)

comm\_exit:

ret

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

;

; Beeper routines (4 different entry points)

;

; No assumptions

;

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

beep\_f1: ; Entry point 1, load beeper frequency 1 settings

mov Temp3, #20 ; Off wait loop length

mov Temp4, #120 ; Number of beep pulses

jmp beep

beep\_f2: ; Entry point 2, load beeper frequency 2 settings

mov Temp3, #16

mov Temp4, #140

jmp beep

beep\_f3: ; Entry point 3, load beeper frequency 3 settings

mov Temp3, #13

mov Temp4, #180

jmp beep

beep\_f4: ; Entry point 4, load beeper frequency 4 settings

mov Temp3, #11

mov Temp4, #200

jmp beep

beep: ; Beep loop start

mov A, Beep\_Strength

djnz ACC, beep\_start

ret

beep\_start:

mov Temp2, #2

beep\_onoff:

clr A

BcomFET\_off ; BcomFET off

djnz ACC, $ ; Allow some time after comfet is turned off

BpwmFET\_on ; BpwmFET on (in order to charge the driver of the BcomFET)

djnz ACC, $ ; Let the pwmfet be turned on a while

BpwmFET\_off ; BpwmFET off again

djnz ACC, $ ; Allow some time after pwmfet is turned off

BcomFET\_on ; BcomFET on

djnz ACC, $ ; Allow some time after comfet is turned on

; Turn on pwmfet

mov A, Temp2

jb ACC.0, beep\_apwmfet\_on

ApwmFET\_on ; ApwmFET on

beep\_apwmfet\_on:

jnb ACC.0, beep\_cpwmfet\_on

CpwmFET\_on ; CpwmFET on

beep\_cpwmfet\_on:

mov A, Beep\_Strength

djnz ACC, $

; Turn off pwmfet

mov A, Temp2

jb ACC.0, beep\_apwmfet\_off

ApwmFET\_off ; ApwmFET off

beep\_apwmfet\_off:

jnb ACC.0, beep\_cpwmfet\_off

CpwmFET\_off ; CpwmFET off

beep\_cpwmfet\_off:

mov A, #150 ; 25祍 off

djnz ACC, $

djnz Temp2, beep\_onoff

; Copy variable

mov A, Temp3

mov Temp1, A

beep\_off: ; Fets off loop

djnz ACC, $

djnz Temp1, beep\_off

djnz Temp4, beep

BcomFET\_off ; BcomFET off

ret

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

;

; Switch power off routine

;

; No assumptions

;

; Switches all fets off

;

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

switch\_power\_off:

All\_pwmFETs\_Off ; Turn off all pwm fets

All\_comFETs\_Off ; Turn off all commutation fets

Set\_Pwms\_Off

ret

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

;

; Set default parameters

;

; No assumptions

;

; Sets default programming parameters

;

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

set\_default\_parameters:

mov Temp1, #\_Pgm\_Gov\_P\_Gain

mov @Temp1, #0FFh ; Governor P gain

inc Temp1

mov @Temp1, #0FFh ; Governor I gain

inc Temp1

mov @Temp1, #0FFh ; Governor mode

inc Temp1

mov @Temp1, #0FFh ; Low voltage limit

inc Temp1

mov @Temp1, #0FFh ; Multi gain

inc Temp1

mov @Temp1, #0FFh

inc Temp1

mov @Temp1, #DEFAULT\_PGM\_STARTUP\_PWR

inc Temp1

mov @Temp1, #0FFh ; Pwm freq

inc Temp1

mov @Temp1, #DEFAULT\_PGM\_DIRECTION

mov Temp1, #Pgm\_Enable\_TX\_Program

mov @Temp1, #DEFAULT\_PGM\_ENABLE\_TX\_PROGRAM

inc Temp1

mov @Temp1, #0FFh ; Main rearm start

inc Temp1

mov @Temp1, #0FFh ; Governor setup target

inc Temp1

mov @Temp1, #0FFh ; Startup rpm

inc Temp1

mov @Temp1, #0FFh ; Startup accel

inc Temp1

mov @Temp1, #0FFh ; Voltage comp

inc Temp1

mov @Temp1, #DEFAULT\_PGM\_COMM\_TIMING

inc Temp1

mov @Temp1, #0FFh ; Damping force

inc Temp1

mov @Temp1, #0FFh ; Governor range

inc Temp1

mov @Temp1, #0FFh ; Startup method

inc Temp1

mov @Temp1, #DEFAULT\_PGM\_MIN\_THROTTLE

inc Temp1

mov @Temp1, #DEFAULT\_PGM\_MAX\_THROTTLE

inc Temp1

mov @Temp1, #DEFAULT\_PGM\_BEEP\_STRENGTH

inc Temp1

mov @Temp1, #DEFAULT\_PGM\_BEACON\_STRENGTH

inc Temp1

mov @Temp1, #DEFAULT\_PGM\_BEACON\_DELAY

inc Temp1

mov @Temp1, #0FFh ; Throttle rate

inc Temp1

mov @Temp1, #DEFAULT\_PGM\_DEMAG\_COMP

inc Temp1

mov @Temp1, #0FFh ; Bec voltage high

inc Temp1

mov @Temp1, #DEFAULT\_PGM\_CENTER\_THROTTLE

inc Temp1

mov @Temp1, #0FFh

inc Temp1

mov @Temp1, #DEFAULT\_PGM\_ENABLE\_TEMP\_PROT

inc Temp1

mov @Temp1, #DEFAULT\_PGM\_ENABLE\_POWER\_PROT

inc Temp1

mov @Temp1, #0FFh ; Enable pwm input

inc Temp1

mov @Temp1, #0FFh ; Pwm dither

inc Temp1

mov @Temp1, #DEFAULT\_PGM\_BRAKE\_ON\_STOP

inc Temp1

mov @Temp1, #DEFAULT\_PGM\_LED\_CONTROL

ret

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

;

; Scale throttle cal

;

; No assumptions

;

; Scales a throttle cal value

; Input is ACC, output is Temp2/Temp1

;

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

scale\_throttle\_cal:

mov Temp3, A

mov B, #0Ch ; Calculate "3%" (for going from 1000us to numerical 1024)

mul AB

mov Temp4, B

mov A, Temp3

clr C ; Shift to 9 bits

rlc A

mov Temp1, A

mov A, #1

rlc A

mov Temp2, A

mov A, Temp1 ; Shift to 10 bits

clr C

rlc A

mov Temp1, A

mov A, Temp2

rlc A

mov Temp2, A

mov A, Temp1 ; Add "3%"

clr C

add A, Temp4

mov Temp1, A

mov A, Temp2

addc A, #0

mov Temp2, A

IF MCU\_48MHZ == 1

mov A, Temp1 ; Shift to 11 bits

clr C

rlc A

mov Temp1, A

mov A, Temp2

rlc A

mov Temp2, A

ENDIF

ret

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

;

; Decode settings

;

; No assumptions

;

; Decodes various settings

;

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

decode\_settings:

; Load programmed direction

mov Temp1, #Pgm\_Direction

mov A, @Temp1

clr C

subb A, #3

setb Flags3.PGM\_BIDIR

jnc ($+4)

clr Flags3.PGM\_BIDIR

clr Flags3.PGM\_DIR\_REV

mov A, @Temp1

jnb ACC.1, ($+5)

setb Flags3.PGM\_DIR\_REV

mov C, Flags3.PGM\_DIR\_REV

mov Flags3.PGM\_BIDIR\_REV, C

; Decode startup power

mov Temp1, #Pgm\_Startup\_Pwr

mov A, @Temp1

dec A

mov DPTR, #STARTUP\_POWER\_TABLE

movc A, @A+DPTR

mov Temp1, #Pgm\_Startup\_Pwr\_Decoded

mov @Temp1, A

; Decode low rpm power slope

mov Temp1, #Pgm\_Startup\_Pwr

mov A, @Temp1

mov Low\_Rpm\_Pwr\_Slope, A

clr C

subb A, #2

jnc ($+5)

mov Low\_Rpm\_Pwr\_Slope, #2

; Decode demag compensation

mov Temp1, #Pgm\_Demag\_Comp

mov A, @Temp1

mov Demag\_Pwr\_Off\_Thresh, #255 ; Set default

cjne A, #2, decode\_demag\_high

mov Demag\_Pwr\_Off\_Thresh, #160 ; Settings for demag comp low

decode\_demag\_high:

cjne A, #3, decode\_demag\_done

mov Demag\_Pwr\_Off\_Thresh, #130 ; Settings for demag comp high

decode\_demag\_done:

; Decode temperature protection limit

mov Temp1, #Pgm\_Enable\_Temp\_Prot

mov A, @Temp1

mov Temp1, A

jz decode\_temp\_done

mov A, #(TEMP\_LIMIT-TEMP\_LIMIT\_STEP)

decode\_temp\_step:

add A, #TEMP\_LIMIT\_STEP

djnz Temp1, decode\_temp\_step

decode\_temp\_done:

mov Temp\_Prot\_Limit, A

; Decode throttle cal

mov Temp1, #Pgm\_Min\_Throttle ; Throttle cal is in 4us units

mov A, @Temp1

call scale\_throttle\_cal

mov Min\_Throttle\_L, Temp1

mov Min\_Throttle\_H, Temp2

mov Temp1, #Pgm\_Center\_Throttle ; Throttle cal is in 4us units

mov A, @Temp1

call scale\_throttle\_cal

mov Center\_Throttle\_L, Temp1

mov Center\_Throttle\_H, Temp2

mov Temp1, #Pgm\_Max\_Throttle ; Throttle cal is in 4us units

mov A, @Temp1

call scale\_throttle\_cal

mov Max\_Throttle\_L, Temp1

mov Max\_Throttle\_H, Temp2

call switch\_power\_off

ret

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

;

; Find throttle gains

;

; No assumptions

;

; Finds throttle gains for both directions in bidirectional mode

;

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

find\_throttle\_gains:

; Check if full range is chosen

jnb Flags2.RCP\_FULL\_RANGE, find\_throttle\_gains\_normal

mov Temp3, #0 ; Min throttle

mov Temp4, #0

mov Temp5, #255 ; Max throttle

mov Temp6, #0

mov Temp7, #0 ; Deadband

call find\_throttle\_gain

mov Throttle\_Gain\_M, Temp4

mov Throttle\_Gain, Temp3

ret

find\_throttle\_gains\_normal:

; Check if bidirectional operation

jnb Flags3.PGM\_BIDIR, find\_throttle\_gains\_bidir\_done

mov Temp1, #Pgm\_Min\_Throttle

mov A, @Temp1

mov Temp3, A

mov Temp4, #0

mov Temp1, #Pgm\_Center\_Throttle

mov A, @Temp1

mov Temp5, A

mov Temp6, #0

clr C

mov A, Temp3 ; Scale gains in bidirectional

rlc A

mov Temp3, A

mov A, Temp4

rlc A

mov Temp4, A

clr C

mov A, Temp5

rlc A

mov Temp5, A

mov A, Temp6

rlc A

mov Temp6, A

mov Temp7, #10 ; Compensate for deadband in bidirectional

call find\_throttle\_gain

mov Throttle\_Gain\_BD\_Rev\_M, Temp4

mov Throttle\_Gain\_BD\_Rev, Temp3

find\_throttle\_gains\_bidir\_done:

mov Temp1, #Pgm\_Min\_Throttle

jnb Flags3.PGM\_BIDIR, ($+5)

mov Temp1, #Pgm\_Center\_Throttle

mov A, @Temp1

mov Temp3, A

mov Temp4, #0

mov Temp1, #Pgm\_Max\_Throttle

mov A, @Temp1

mov Temp5, A

mov Temp6, #0

mov Temp7, #0 ; No deadband

jnb Flags3.PGM\_BIDIR, find\_throttle\_gain\_fwd

clr C

mov A, Temp3 ; Scale gains in bidirectional

rlc A

mov Temp3, A

mov A, Temp4

rlc A

mov Temp4, A

clr C

mov A, Temp5

rlc A

mov Temp5, A

mov A, Temp6

rlc A

mov Temp6, A

mov Temp7, #10 ; Compensate for deadband in bidirectional

find\_throttle\_gain\_fwd:

call find\_throttle\_gain

mov Throttle\_Gain\_M, Temp4

mov Throttle\_Gain, Temp3

ret

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

;

; Find throttle gain

;

; The difference between max and min throttle must be more than 140us (a Pgm\_xxx\_Throttle difference of 35)

; Temp4/3 holds min throttle, Temp6/5 holds max throttle, Temp7 holds deadband, Temp4/Temp3 gives resulting gain

;

; Finds throttle gain from throttle calibration values

;

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

find\_throttle\_gain:

; Subtract deadband from max

clr C

mov A, Temp5

subb A, Temp7

mov Temp5, A

mov A, Temp6

subb A, #0

mov Temp6, A

; Calculate difference

clr C

mov A, Temp5

subb A, Temp3

mov Temp5, A

mov A, Temp6

subb A, Temp4

mov Temp6, A

; Check that difference is minimum 35

clr C

mov A, Temp5

subb A, #35

mov A, Temp6

subb A, #0

jnc ($+6)

mov Temp5, #35

mov Temp6, #0

; Check that difference is maximum 511

clr C

mov A, Temp5

subb A, #255

mov A, Temp6

subb A, #1

jc ($+6)

mov Temp5, #255

mov Temp6, #1

; Find gain

mov Temp4, #0FFh

find\_throttle\_gain\_loop:

inc Temp4

mov Temp3, #0

test\_throttle\_gain:

inc Temp3

mov A, Temp3

jnz test\_throttle\_gain\_mult

clr C

mov A, Temp5 ; Set multiplier x2 and range /2

rlc A

mov Temp5, A

mov A, Temp6

rlc A

mov Temp6, A

ajmp find\_throttle\_gain\_loop

test\_throttle\_gain\_mult:

mov A, Temp5 ; A has difference, B has gain

mov B, Temp3

mul AB

mov Temp7, B

mov A, Temp6

mov B, Temp3

mul AB

add A, Temp7

subb A, #124

jc test\_throttle\_gain

mov A, Temp3

cpl A

jz find\_throttle\_gain\_loop

ret

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

;

; Average throttle

;

; Outputs result in Temp8

;

; Averages throttle calibration readings

;

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

average\_throttle:

setb Flags2.RCP\_FULL\_RANGE ; Set range to 1000-2020us

call find\_throttle\_gains ; Set throttle gains

call wait30ms

call wait30ms

mov Temp3, #0

mov Temp4, #0

mov Temp5, #16 ; Average 16 measurments

average\_throttle\_meas:

call wait3ms ; Wait for new RC pulse value

mov A, New\_Rcp ; Get new RC pulse value

add A, Temp3

mov Temp3, A

mov A, #0

addc A, Temp4

mov Temp4, A

djnz Temp5, average\_throttle\_meas

mov Temp5, #4 ; Shift 4 times

average\_throttle\_div:

clr C

mov A, Temp4 ; Shift right

rrc A

mov Temp4, A

mov A, Temp3

rrc A

mov Temp3, A

djnz Temp5, average\_throttle\_div

mov Temp8, A ; Copy to Temp8

mov A, Temp4

jz ($+4)

mov Temp8, #0FFh

clr Flags2.RCP\_FULL\_RANGE

call find\_throttle\_gains ; Set throttle gains

ret

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

;

; LED control

;

; No assumptions

;

; Controls LEDs

;

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

led\_control:

mov Temp1, #Pgm\_LED\_Control

mov A, @Temp1

mov Temp2, A

anl A, #03h

Set\_LED\_0

jnz led\_0\_done

Clear\_LED\_0

led\_0\_done:

mov A, Temp2

anl A, #0Ch

Set\_LED\_1

jnz led\_1\_done

Clear\_LED\_1

led\_1\_done:

mov A, Temp2

anl A, #030h

Set\_LED\_2

jnz led\_2\_done

Clear\_LED\_2

led\_2\_done:

mov A, Temp2

anl A, #0C0h

Set\_LED\_3

jnz led\_3\_done

Clear\_LED\_3

led\_3\_done:

ret

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

;

; Main program start

;

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

pgm\_start:

; Initialize flash keys to invalid values

mov Flash\_Key\_1, #0

mov Flash\_Key\_2, #0

; Disable the WDT.

mov WDTCN, #0DEh ; Disable watchdog

mov WDTCN, #0ADh

; Initialize stack

mov SP, #0c0h ; Stack = 64 upper bytes of RAM

; Initialize VDD monitor

orl VDM0CN, #080h ; Enable the VDD monitor

mov RSTSRC, #06h ; Set missing clock and VDD monitor as a reset source if not 1S capable

; Set clock frequency

mov CLKSEL, #00h ; Set clock divider to 1

; Switch power off

call switch\_power\_off

; Ports initialization

mov P0, #P0\_INIT

mov P0MDIN, #P0\_DIGITAL

mov P0MDOUT, #P0\_PUSHPULL

mov P0, #P0\_INIT

mov P0SKIP, #P0\_SKIP

mov P1, #P1\_INIT

mov P1MDIN, #P1\_DIGITAL

mov P1MDOUT, #P1\_PUSHPULL

mov P1, #P1\_INIT

mov P1SKIP, #P1\_SKIP

mov P2MDOUT, #P2\_PUSHPULL

; Initialize the XBAR and related functionality

Initialize\_Xbar

; Switch power off again, after initializing ports

call switch\_power\_off

; Clear RAM

clr A ; Clear accumulator

mov Temp1, A ; Clear Temp1

clear\_ram:

mov @Temp1, A ; Clear RAM

djnz Temp1, clear\_ram ; Is A not zero? - jump

; Set default programmed parameters

call set\_default\_parameters

; Read all programmed parameters

call read\_all\_eeprom\_parameters

; Set beep strength

mov Temp1, #Pgm\_Beep\_Strength

mov Beep\_Strength, @Temp1

; Set initial arm variable

mov Initial\_Arm, #1

; Initializing beep

clr IE\_EA ; Disable interrupts explicitly

call wait200ms

call beep\_f1

call wait30ms

call beep\_f2

call wait30ms

call beep\_f3

call wait30ms

call led\_control

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

;

; No signal entry point

;

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

init\_no\_signal:

; Disable interrupts explicitly

clr IE\_EA

; Initialize flash keys to invalid values

mov Flash\_Key\_1, #0

mov Flash\_Key\_2, #0

; Check if input signal is high for more than 15ms

mov Temp1, #250

input\_high\_check\_1:

mov Temp2, #250

input\_high\_check\_2:

jnb RTX\_PORT.RTX\_PIN, bootloader\_done ; Look for low

djnz Temp2, input\_high\_check\_2

djnz Temp1, input\_high\_check\_1

ljmp 1C00h ; Jump to bootloader

bootloader\_done:

; Decode settings

call decode\_settings

; Find throttle gain from stored min and max settings

call find\_throttle\_gains

; Set beep strength

mov Temp1, #Pgm\_Beep\_Strength

mov Beep\_Strength, @Temp1

; Switch power off

call switch\_power\_off

; Set clock frequency

IF MCU\_48MHZ == 1

Set\_MCU\_Clk\_24MHz

ENDIF

; Setup timers for pwm input

mov IT01CF, #RTX\_PIN ; Route RCP input to INT0

mov TCON, #11h ; Timer 0 run and INT0 edge triggered

mov CKCON0, #04h ; Timer 0 clock is system clock

mov TMOD, #09h ; Timer 0 set to 16bits and gated by INT0

mov TMR2CN0, #04h ; Timer 2 enabled

mov TMR3CN0, #04h ; Timer 3 enabled

Initialize\_PCA ; Initialize PCA

Set\_Pwm\_Polarity ; Set pwm polarity

Enable\_Power\_Pwm\_Module ; Enable power pwm module

Enable\_Damp\_Pwm\_Module ; Enable damping pwm module

; Enable interrupts

IF MCU\_48MHZ == 0

mov IE, #21h ; Enable timer 2 interrupts and INT0 interrupts

ELSE

mov IE, #23h ; Enable timer 0, timer 2 interrupts and INT0 interrupts

ENDIF

mov EIE1, #90h ; Enable timer 3 and PCA0 interrupts

mov IP, #01h ; High priority to INT0 interrupts

; Initialize comparator

Initialize\_Comparator ; Initialize comparator

; Initialize ADC

Initialize\_Adc ; Initialize ADC operation

call wait1ms

setb IE\_EA ; Enable all interrupts

; Reset stall count

mov Stall\_Cnt, #0

; Initialize RC pulse

clr Flags2.RCP\_UPDATED ; Clear updated flag

call wait200ms

; Clear all shot flags

clr Flags2.RCP\_ONESHOT125 ; Clear OneShot125 flag

clr Flags2.RCP\_ONESHOT42 ; Clear OneShot42 flag

clr Flags2.RCP\_MULTISHOT ; Clear Multishot flag

clr Flags2.RCP\_DSHOT ; Clear DShot flag

mov Dshot\_Cmd, #0 ; Clear Dshot command

mov Dshot\_Cmd\_Cnt, #0 ; Clear Dshot command count

; Test whether signal is regular pwm

mov Rcp\_Outside\_Range\_Cnt, #0 ; Reset out of range counter

call wait100ms ; Wait for new RC pulse

clr C

mov A, Rcp\_Outside\_Range\_Cnt ; Check how many pulses were outside normal range ("900-2235us")

subb A, #10

jnc ($+4)

ajmp validate\_rcp\_start

; Test whether signal is OneShot125

setb Flags2.RCP\_ONESHOT125 ; Set OneShot125 flag

mov Rcp\_Outside\_Range\_Cnt, #0 ; Reset out of range counter

call wait100ms ; Wait for new RC pulse

clr C

mov A, Rcp\_Outside\_Range\_Cnt ; Check how many pulses were outside normal range ("900-2235us")

subb A, #10

jnc ($+4)

ajmp validate\_rcp\_start

; Test whether signal is OneShot42

clr Flags2.RCP\_ONESHOT125

setb Flags2.RCP\_ONESHOT42 ; Set OneShot42 flag

mov Rcp\_Outside\_Range\_Cnt, #0 ; Reset out of range counter

call wait100ms ; Wait for new RC pulse

clr C

mov A, Rcp\_Outside\_Range\_Cnt ; Check how many pulses were outside normal range ("900-2235us")

subb A, #10

jnc ($+4)

ajmp validate\_rcp\_start

; Setup timers for DShot

mov IT01CF, #(80h+(RTX\_PIN SHL 4)+(RTX\_PIN)) ; Route RCP input to INT0/1, with INT1 inverted

mov TCON, #51h ; Timer 0/1 run and INT0 edge triggered

mov CKCON0, #01h ; Timer 0/1 clock is system clock divided by 4 (for DShot150)

mov TMOD, #0AAh ; Timer 0/1 set to 8bits auto reload and gated by INT0

mov TH0, #0 ; Auto reload value zero

mov TH1, #0

; Setup interrupts for DShot

clr IE\_ET0 ; Disable timer 0 interrupts

setb IE\_ET1 ; Enable timer 1 interrupts

setb IE\_EX1 ; Enable int1 interrupts

; Setup variables for DSshot150

IF MCU\_48MHZ == 1

mov DShot\_Timer\_Preset, #128 ; Load DShot sync timer preset (for DShot150)

ELSE

mov DShot\_Timer\_Preset, #192

ENDIF

mov DShot\_Pwm\_Thr, #20 ; Load DShot qualification pwm threshold (for DShot150)

mov DShot\_Frame\_Length\_Thr, #80 ; Load DShot frame length criteria

; Test whether signal is DShot150

clr Flags2.RCP\_ONESHOT42

setb Flags2.RCP\_DSHOT

mov Rcp\_Outside\_Range\_Cnt, #10 ; Set out of range counter

call wait100ms ; Wait for new RC pulse

mov DShot\_Pwm\_Thr, #16 ; Load DShot regular pwm threshold

clr C

mov A, Rcp\_Outside\_Range\_Cnt ; Check if pulses were accepted

subb A, #10

mov Dshot\_Cmd, #0

mov Dshot\_Cmd\_Cnt, #0

jc validate\_rcp\_start

; Setup variables for DShot300

mov CKCON0, #0Ch ; Timer 0/1 clock is system clock (for DShot300)

IF MCU\_48MHZ == 1

mov DShot\_Timer\_Preset, #0 ; Load DShot sync timer preset (for DShot300)

ELSE

mov DShot\_Timer\_Preset, #128

ENDIF

mov DShot\_Pwm\_Thr, #40 ; Load DShot qualification pwm threshold (for DShot300)

mov DShot\_Frame\_Length\_Thr, #40 ; Load DShot frame length criteria

; Test whether signal is DShot300

mov Rcp\_Outside\_Range\_Cnt, #10 ; Set out of range counter

call wait100ms ; Wait for new RC pulse

mov DShot\_Pwm\_Thr, #32 ; Load DShot regular pwm threshold

clr C

mov A, Rcp\_Outside\_Range\_Cnt ; Check if pulses were accepted

subb A, #10

mov Dshot\_Cmd, #0

mov Dshot\_Cmd\_Cnt, #0

jc validate\_rcp\_start

; Setup variables for DShot600

mov CKCON0, #0Ch ; Timer 0/1 clock is system clock (for DShot600)

IF MCU\_48MHZ == 1

mov DShot\_Timer\_Preset, #128 ; Load DShot sync timer preset (for DShot600)

ELSE

mov DShot\_Timer\_Preset, #192

ENDIF

mov DShot\_Pwm\_Thr, #20 ; Load DShot qualification pwm threshold (for DShot600)

mov DShot\_Frame\_Length\_Thr, #20 ; Load DShot frame length criteria

; Test whether signal is DShot600

mov Rcp\_Outside\_Range\_Cnt, #10 ; Set out of range counter

call wait100ms ; Wait for new RC pulse

mov DShot\_Pwm\_Thr, #16 ; Load DShot regular pwm threshold

clr C

mov A, Rcp\_Outside\_Range\_Cnt ; Check if pulses were accepted

subb A, #10

mov Dshot\_Cmd, #0

mov Dshot\_Cmd\_Cnt, #0

jc validate\_rcp\_start

; Setup timers for Multishot

mov IT01CF, #RTX\_PIN ; Route RCP input to INT0

mov TCON, #11h ; Timer 0 run and INT0 edge triggered

mov CKCON0, #04h ; Timer 0 clock is system clock

mov TMOD, #09h ; Timer 0 set to 16bits and gated by INT0

; Setup interrupts for Multishot

setb IE\_ET0 ; Enable timer 0 interrupts

clr IE\_ET1 ; Disable timer 1 interrupts

clr IE\_EX1 ; Disable int1 interrupts

; Test whether signal is Multishot

clr Flags2.RCP\_DSHOT

setb Flags2.RCP\_MULTISHOT ; Set Multishot flag

mov Rcp\_Outside\_Range\_Cnt, #0 ; Reset out of range counter

call wait100ms ; Wait for new RC pulse

clr C

mov A, Rcp\_Outside\_Range\_Cnt ; Check how many pulses were outside normal range ("900-2235us")

subb A, #10

jc validate\_rcp\_start

ajmp init\_no\_signal

validate\_rcp\_start:

; Validate RC pulse

call wait3ms ; Wait for new RC pulse

jb Flags2.RCP\_UPDATED, ($+6) ; Is there an updated RC pulse available - proceed

ljmp init\_no\_signal ; Go back to detect input signal

; Beep arm sequence start signal

clr IE\_EA ; Disable all interrupts

call beep\_f1 ; Signal that RC pulse is ready

call beep\_f1

call beep\_f1

setb IE\_EA ; Enable all interrupts

call wait200ms

; Arming sequence start

arming\_start:

jb Flags2.RCP\_DSHOT, ($+6) ; Disable tx programming for DShot

jnb Flags3.PGM\_BIDIR, ($+6)

ljmp program\_by\_tx\_checked ; Disable tx programming if bidirectional operation

call wait3ms

mov Temp1, #Pgm\_Enable\_TX\_Program; Start programming mode entry if enabled

mov A, @Temp1

clr C

subb A, #1 ; Is TX programming enabled?

jnc arming\_initial\_arm\_check ; Yes - proceed

jmp program\_by\_tx\_checked ; No - branch

arming\_initial\_arm\_check:

mov A, Initial\_Arm ; Yes - check if it is initial arm sequence

clr C

subb A, #1 ; Is it the initial arm sequence?

jnc arming\_check ; Yes - proceed

jmp program\_by\_tx\_checked ; No - branch

arming\_check:

; Initialize flash keys to valid values

mov Flash\_Key\_1, #0A5h

mov Flash\_Key\_2, #0F1h

; Throttle calibration and tx program entry

mov Temp8, #2 ; Set 1 seconds wait time

throttle\_high\_cal:

setb Flags2.RCP\_FULL\_RANGE ; Set range to 1000-2020us

call find\_throttle\_gains ; Set throttle gains

call wait100ms ; Wait for new throttle value

clr IE\_EA ; Disable interrupts (freeze New\_Rcp value)

clr Flags2.RCP\_FULL\_RANGE ; Set programmed range

call find\_throttle\_gains ; Set throttle gains

clr C

mov A, New\_Rcp ; Load new RC pulse value

subb A, #(255/2) ; Is RC pulse above midstick?

setb IE\_EA ; Enable interrupts

jc program\_by\_tx\_checked ; No - branch

call wait1ms

clr IE\_EA ; Disable all interrupts

call beep\_f4

setb IE\_EA ; Enable all interrupts

djnz Temp8, throttle\_high\_cal ; Continue to wait

call average\_throttle

clr C

mov A, Temp8

mov Temp1, #Pgm\_Max\_Throttle ; Store

mov @Temp1, A

call wait200ms

call success\_beep

throttle\_low\_cal\_start:

mov Temp8, #10 ; Set 3 seconds wait time

throttle\_low\_cal:

setb Flags2.RCP\_FULL\_RANGE ; Set range to 1000-2020us

call find\_throttle\_gains ; Set throttle gains

call wait100ms

clr IE\_EA ; Disable interrupts (freeze New\_Rcp value)

clr Flags2.RCP\_FULL\_RANGE ; Set programmed range

call find\_throttle\_gains ; Set throttle gains

clr C

mov A, New\_Rcp ; Load new RC pulse value

subb A, #(255/2) ; Below midstick?

setb IE\_EA ; Enable interrupts

jnc throttle\_low\_cal\_start ; No - start over

call wait1ms

clr IE\_EA ; Disable all interrupts

call beep\_f1

call wait10ms

call beep\_f1

setb IE\_EA ; Enable all interrupts

djnz Temp8, throttle\_low\_cal ; Continue to wait

call average\_throttle

mov A, Temp8

add A, #3 ; Add about 1%

mov Temp1, #Pgm\_Min\_Throttle ; Store

mov @Temp1, A

mov Temp1, A ; Min throttle in Temp1

mov Temp2, #Pgm\_Max\_Throttle

mov A, @Temp2

clr C

subb A, #35 ; Subtract 35 (140us) from max throttle

jc program\_by\_tx\_entry\_limit

subb A, Temp1 ; Subtract min from max

jnc program\_by\_tx\_entry\_store

program\_by\_tx\_entry\_limit:

mov A, Temp1 ; Load min

add A, #35 ; Make max 140us higher than min

mov Temp1, #Pgm\_Max\_Throttle ; Store new max

mov @Temp1, A

program\_by\_tx\_entry\_store:

call wait200ms

call erase\_and\_store\_all\_in\_eeprom

call success\_beep\_inverted

program\_by\_tx\_entry\_wait:

call wait100ms

call find\_throttle\_gains ; Set throttle gains

ljmp init\_no\_signal ; Go back

program\_by\_tx\_checked:

; Initialize flash keys to invalid values

mov Flash\_Key\_1, #0

mov Flash\_Key\_2, #0

call wait100ms ; Wait for new throttle value

clr C

mov A, New\_Rcp ; Load new RC pulse value

subb A, #1 ; Below stop?

jc arm\_end\_beep ; Yes - proceed

jmp arming\_start ; No - start over

arm\_end\_beep:

; Beep arm sequence end signal

clr IE\_EA ; Disable all interrupts

call beep\_f4 ; Signal that rcpulse is ready

call beep\_f4

call beep\_f4

setb IE\_EA ; Enable all interrupts

call wait200ms

; Clear initial arm variable

mov Initial\_Arm, #0

; Armed and waiting for power on

wait\_for\_power\_on:

clr A

mov Power\_On\_Wait\_Cnt\_L, A ; Clear wait counter

mov Power\_On\_Wait\_Cnt\_H, A

wait\_for\_power\_on\_loop:

inc Power\_On\_Wait\_Cnt\_L ; Increment low wait counter

mov A, Power\_On\_Wait\_Cnt\_L

cpl A

jnz wait\_for\_power\_on\_no\_beep; Counter wrapping (about 3 sec)

inc Power\_On\_Wait\_Cnt\_H ; Increment high wait counter

mov Temp1, #Pgm\_Beacon\_Delay

mov A, @Temp1

mov Temp1, #25 ; Approximately 1 min

dec A

jz beep\_delay\_set

mov Temp1, #50 ; Approximately 2 min

dec A

jz beep\_delay\_set

mov Temp1, #125 ; Approximately 5 min

dec A

jz beep\_delay\_set

mov Temp1, #250 ; Approximately 10 min

dec A

jz beep\_delay\_set

mov Power\_On\_Wait\_Cnt\_H, #0 ; Reset counter for infinite delay

beep\_delay\_set:

clr C

mov A, Power\_On\_Wait\_Cnt\_H

subb A, Temp1 ; Check against chosen delay

jc wait\_for\_power\_on\_no\_beep; Has delay elapsed?

call switch\_power\_off ; Switch power off in case braking is set

call wait1ms

dec Power\_On\_Wait\_Cnt\_H ; Decrement high wait counter

mov Power\_On\_Wait\_Cnt\_L, #0 ; Set low wait counter

mov Temp1, #Pgm\_Beacon\_Strength

mov Beep\_Strength, @Temp1

clr IE\_EA ; Disable all interrupts

call beep\_f4 ; Signal that there is no signal

setb IE\_EA ; Enable all interrupts

mov Temp1, #Pgm\_Beep\_Strength

mov Beep\_Strength, @Temp1

call wait100ms ; Wait for new RC pulse to be measured

wait\_for\_power\_on\_no\_beep:

call wait10ms

mov A, Rcp\_Timeout\_Cntd ; Load RC pulse timeout counter value

jnz wait\_for\_power\_on\_not\_missing ; If it is not zero - proceed

jmp init\_no\_signal ; If pulses missing - go back to detect input signal

wait\_for\_power\_on\_not\_missing:

clr C

mov A, New\_Rcp ; Load new RC pulse value

subb A, #1 ; Higher than stop

jnc wait\_for\_power\_on\_nonzero ; Yes - proceed

clr C

mov A, Dshot\_Cmd

subb A, #1 ; 1 or higher

jnc check\_dshot\_cmd ; Check Dshot command

ljmp wait\_for\_power\_on\_loop ; If not Dshot command - start over

wait\_for\_power\_on\_nonzero:

lcall wait100ms ; Wait to see if start pulse was only a glitch

mov A, Rcp\_Timeout\_Cntd ; Load RC pulse timeout counter value

jnz ($+5) ; If it is not zero - proceed

ljmp init\_no\_signal ; If it is zero (pulses missing) - go back to detect input signal

mov Dshot\_Cmd, #0

mov Dshot\_Cmd\_Cnt, #0

ljmp init\_start

check\_dshot\_cmd:

clr C

mov A, Dshot\_Cmd

subb A, #1

jnz dshot\_beep\_2

clr IE\_EA

call switch\_power\_off ; Switch power off in case braking is set

mov Temp1, #Pgm\_Beacon\_Strength

mov Beep\_Strength, @Temp1

call beep\_f1

mov Temp1, #Pgm\_Beep\_Strength

mov Beep\_Strength, @Temp1

setb IE\_EA

call wait100ms

jmp clear\_dshot\_cmd

dshot\_beep\_2:

clr C

mov A, Dshot\_Cmd

subb A, #2

jnz dshot\_beep\_3

clr IE\_EA

call switch\_power\_off ; Switch power off in case braking is set

mov Temp1, #Pgm\_Beacon\_Strength

mov Beep\_Strength, @Temp1

call beep\_f2

mov Temp1, #Pgm\_Beep\_Strength

mov Beep\_Strength, @Temp1

setb IE\_EA

call wait100ms

jmp clear\_dshot\_cmd

dshot\_beep\_3:

clr C

mov A, Dshot\_Cmd

subb A, #3

jnz dshot\_beep\_4

clr IE\_EA

call switch\_power\_off ; Switch power off in case braking is set

mov Temp1, #Pgm\_Beacon\_Strength

mov Beep\_Strength, @Temp1

call beep\_f3

mov Temp1, #Pgm\_Beep\_Strength

mov Beep\_Strength, @Temp1

setb IE\_EA

call wait100ms

jmp clear\_dshot\_cmd

dshot\_beep\_4:

clr C

mov A, Dshot\_Cmd

subb A, #4

jnz dshot\_beep\_5

clr IE\_EA

call switch\_power\_off ; Switch power off in case braking is set

mov Temp1, #Pgm\_Beacon\_Strength

mov Beep\_Strength, @Temp1

call beep\_f4

mov Temp1, #Pgm\_Beep\_Strength

mov Beep\_Strength, @Temp1

setb IE\_EA

call wait100ms

jmp clear\_dshot\_cmd

dshot\_beep\_5:

clr C

mov A, Dshot\_Cmd

subb A, #5

jnz dshot\_direction\_1

clr IE\_EA

call switch\_power\_off ; Switch power off in case braking is set

mov Temp1, #Pgm\_Beacon\_Strength

mov Beep\_Strength, @Temp1

call beep\_f4

mov Temp1, #Pgm\_Beep\_Strength

mov Beep\_Strength, @Temp1

setb IE\_EA

call wait100ms

jmp clear\_dshot\_cmd

dshot\_direction\_1:

clr C

mov A, Dshot\_Cmd

subb A, #7

jnz dshot\_direction\_2

clr C

mov A, Dshot\_Cmd\_Cnt

subb A, #6 ; Needs to receive it 6 times in a row

jnc ($+4) ; Same as "jc dont\_clear\_dshot\_cmd"

ajmp wait\_for\_power\_on\_not\_missing

mov A, #1

jnb Flags3.PGM\_BIDIR, ($+5)

mov A, #3

mov Temp1, #Pgm\_Direction

mov @Temp1, A

clr Flags3.PGM\_DIR\_REV

clr Flags3.PGM\_BIDIR\_REV

jmp clear\_dshot\_cmd

dshot\_direction\_2:

clr C

mov A, Dshot\_Cmd

subb A, #8

jnz dshot\_direction\_bidir\_off

clr C

mov A, Dshot\_Cmd\_Cnt

subb A, #6 ; Needs to receive it 6 times in a row

jnc ($+4) ; Same as "jc dont\_clear\_dshot\_cmd"

ajmp wait\_for\_power\_on\_not\_missing

mov A, #2

jnb Flags3.PGM\_BIDIR, ($+5)

mov A, #4

mov Temp1, #Pgm\_Direction

mov @Temp1, A

setb Flags3.PGM\_DIR\_REV

setb Flags3.PGM\_BIDIR\_REV

jmp clear\_dshot\_cmd

dshot\_direction\_bidir\_off:

clr C

mov A, Dshot\_Cmd

subb A, #9

jnz dshot\_direction\_bidir\_on

clr C

mov A, Dshot\_Cmd\_Cnt

subb A, #6 ; Needs to receive it 6 times in a row

jnc ($+4) ; Same as "jc dont\_clear\_dshot\_cmd"

ajmp wait\_for\_power\_on\_not\_missing

jnb Flags3.PGM\_BIDIR, dshot\_direction\_bidir\_on

clr C

mov Temp1, #Pgm\_Direction

mov A, @Temp1

subb A, #2

mov @Temp1, A

clr Flags3.PGM\_BIDIR

jmp clear\_dshot\_cmd

dshot\_direction\_bidir\_on:

clr C

mov A, Dshot\_Cmd

subb A, #10

jnz dshot\_direction\_normal

clr C

mov A, Dshot\_Cmd\_Cnt

subb A, #6 ; Needs to receive it 6 times in a row

jnc ($+4) ; Same as "jc dont\_clear\_dshot\_cmd"

ajmp wait\_for\_power\_on\_not\_missing

jb Flags3.PGM\_BIDIR, dshot\_direction\_normal

mov Temp1, #Pgm\_Direction

mov A, @Temp1

add A, #2

mov @Temp1, A

setb Flags3.PGM\_BIDIR

jmp clear\_dshot\_cmd

dshot\_direction\_normal:

clr C

mov A, Dshot\_Cmd

subb A, #20

jnz dshot\_direction\_reverse

clr C

mov A, Dshot\_Cmd\_Cnt

subb A, #6 ; Needs to receive it 6 times in a row

jnc ($+4) ; Same as "jc dont\_clear\_dshot\_cmd"

ajmp wait\_for\_power\_on\_not\_missing

clr IE\_EA ; DPTR used in interrupts

mov DPTR, #Eep\_Pgm\_Direction ; Read from flash

mov A, #0

movc A, @A+DPTR

setb IE\_EA

mov Temp1, #Pgm\_Direction

mov @Temp1, A

rrc A ; Lsb to carry

clr Flags3.PGM\_DIR\_REV

clr Flags3.PGM\_BIDIR\_REV

jc ($+4)

setb Flags3.PGM\_DIR\_REV

jc ($+4)

setb Flags3.PGM\_BIDIR\_REV

jmp clear\_dshot\_cmd

dshot\_direction\_reverse: ; Temporary reverse

clr C

mov A, Dshot\_Cmd

subb A, #21

jnz dshot\_save\_settings

clr C

mov A, Dshot\_Cmd\_Cnt

subb A, #6 ; Needs to receive it 6 times in a row

jc dont\_clear\_dshot\_cmd

clr IE\_EA ; DPTR used in interrupts

mov DPTR, #Eep\_Pgm\_Direction ; Read from flash

mov A, #0

movc A, @A+DPTR

setb IE\_EA

mov Temp1, A

cjne Temp1, #1, ($+5)

mov A, #2

cjne Temp1, #2, ($+5)

mov A, #1

cjne Temp1, #3, ($+5)

mov A, #4

cjne Temp1, #4, ($+5)

mov A, #3

mov Temp1, #Pgm\_Direction

mov @Temp1, A

rrc A ; Lsb to carry

clr Flags3.PGM\_DIR\_REV

clr Flags3.PGM\_BIDIR\_REV

jc ($+4)

setb Flags3.PGM\_DIR\_REV

jc ($+4)

setb Flags3.PGM\_BIDIR\_REV

jmp clear\_dshot\_cmd

dshot\_save\_settings:

clr C

mov A, Dshot\_Cmd

subb A, #12

jnz clear\_dshot\_cmd

mov Flash\_Key\_1, #0A5h ; Initialize flash keys to valid values

mov Flash\_Key\_2, #0F1h

clr C

mov A, Dshot\_Cmd\_Cnt

subb A, #6 ; Needs to receive it 6 times in a row

jc dont\_clear\_dshot\_cmd

call erase\_and\_store\_all\_in\_eeprom

setb IE\_EA

clear\_dshot\_cmd:

mov Dshot\_Cmd, #0

mov Dshot\_Cmd\_Cnt, #0

dont\_clear\_dshot\_cmd:

mov Flash\_Key\_1, #0 ; Initialize flash keys to invalid values

mov Flash\_Key\_2, #0

jmp wait\_for\_power\_on\_not\_missing

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

;

; Start entry point

;

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

init\_start:

clr IE\_EA

call switch\_power\_off

clr A

setb IE\_EA

clr A

mov Adc\_Conversion\_Cnt, A

mov Flags0, A ; Clear flags0

mov Flags1, A ; Clear flags1

mov Demag\_Detected\_Metric, A ; Clear demag metric

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

; Motor start beginning

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

mov Adc\_Conversion\_Cnt, #8 ; Make sure a temp reading is done

call wait1ms

call start\_adc\_conversion

read\_initial\_temp:

jnb ADC0CN0\_ADINT, read\_initial\_temp

Read\_Adc\_Result ; Read initial temperature

mov A, Temp2

jnz ($+3) ; Is reading below 256?

mov Temp1, A ; Yes - set average temperature value to zero

mov Current\_Average\_Temp, Temp1 ; Set initial average temperature

call check\_temp\_voltage\_and\_limit\_power

mov Adc\_Conversion\_Cnt, #8 ; Make sure a temp reading is done next time

; Set up start operating conditions

clr IE\_EA ; Disable interrupts

call set\_startup\_pwm

mov Pwm\_Limit, Pwm\_Limit\_Beg

mov Pwm\_Limit\_By\_Rpm, Pwm\_Limit\_Beg

setb IE\_EA

; Begin startup sequence

IF MCU\_48MHZ == 1

Set\_MCU\_Clk\_48MHz

ENDIF

jnb Flags3.PGM\_BIDIR, init\_start\_bidir\_done ; Check if bidirectional operation

clr Flags3.PGM\_DIR\_REV ; Set spinning direction. Default fwd

jnb Flags2.RCP\_DIR\_REV, ($+5) ; Check force direction

setb Flags3.PGM\_DIR\_REV ; Set spinning direction

init\_start\_bidir\_done:

setb Flags1.STARTUP\_PHASE ; Set startup phase flag

mov Startup\_Cnt, #0 ; Reset counter

call comm5comm6 ; Initialize commutation

call comm6comm1

call initialize\_timing ; Initialize timing

call calc\_next\_comm\_timing ; Set virtual commutation point

call initialize\_timing ; Initialize timing

call calc\_next\_comm\_timing

call initialize\_timing ; Initialize timing

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

;

; Run entry point

;

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

; Run 1 = B(p-on) + C(n-pwm) - comparator A evaluated

; Out\_cA changes from low to high

run1:

call wait\_for\_comp\_out\_high ; Wait for high

; setup\_comm\_wait ; Setup wait time from zero cross to commutation

; evaluate\_comparator\_integrity ; Check whether comparator reading has been normal

call wait\_for\_comm ; Wait from zero cross to commutation

call comm1comm2 ; Commutate

call calc\_next\_comm\_timing ; Calculate next timing and wait advance timing wait

; wait\_advance\_timing ; Wait advance timing and start zero cross wait

; calc\_new\_wait\_times

; wait\_before\_zc\_scan ; Wait zero cross wait and start zero cross timeout

; Run 2 = A(p-on) + C(n-pwm) - comparator B evaluated

; Out\_cB changes from high to low

run2:

call wait\_for\_comp\_out\_low

; setup\_comm\_wait

; evaluate\_comparator\_integrity

jb Flags1.HIGH\_RPM, ($+6) ; Skip if high rpm

lcall set\_pwm\_limit\_low\_rpm

jnb Flags1.HIGH\_RPM, ($+6) ; Do if high rpm

lcall set\_pwm\_limit\_high\_rpm

call wait\_for\_comm

call comm2comm3

call calc\_next\_comm\_timing

; wait\_advance\_timing

; calc\_new\_wait\_times

; wait\_before\_zc\_scan

; Run 3 = A(p-on) + B(n-pwm) - comparator C evaluated

; Out\_cC changes from low to high

run3:

call wait\_for\_comp\_out\_high

; setup\_comm\_wait

; evaluate\_comparator\_integrity

call wait\_for\_comm

call comm3comm4

call calc\_next\_comm\_timing

; wait\_advance\_timing

; calc\_new\_wait\_times

; wait\_before\_zc\_scan

; Run 4 = C(p-on) + B(n-pwm) - comparator A evaluated

; Out\_cA changes from high to low

run4:

call wait\_for\_comp\_out\_low

; setup\_comm\_wait

; evaluate\_comparator\_integrity

call wait\_for\_comm

call comm4comm5

call calc\_next\_comm\_timing

; wait\_advance\_timing

; calc\_new\_wait\_times

; wait\_before\_zc\_scan

; Run 5 = C(p-on) + A(n-pwm) - comparator B evaluated

; Out\_cB changes from low to high

run5:

call wait\_for\_comp\_out\_high

; setup\_comm\_wait

; evaluate\_comparator\_integrity

call wait\_for\_comm

call comm5comm6

call calc\_next\_comm\_timing

; wait\_advance\_timing

; calc\_new\_wait\_times

; wait\_before\_zc\_scan

; Run 6 = B(p-on) + A(n-pwm) - comparator C evaluated

; Out\_cC changes from high to low

run6:

call start\_adc\_conversion

call wait\_for\_comp\_out\_low

; setup\_comm\_wait

; evaluate\_comparator\_integrity

call wait\_for\_comm

call comm6comm1

call check\_temp\_voltage\_and\_limit\_power

call calc\_next\_comm\_timing

; wait\_advance\_timing

; calc\_new\_wait\_times

; wait\_before\_zc\_scan

; Check if it is direct startup

jnb Flags1.STARTUP\_PHASE, normal\_run\_checks

; Set spoolup power variables

mov Pwm\_Limit, Pwm\_Limit\_Beg ; Set initial max power

; Check startup counter

mov Temp2, #24 ; Set nominal startup parameters

mov Temp3, #12

clr C

mov A, Startup\_Cnt ; Load counter

subb A, Temp2 ; Is counter above requirement?

jc direct\_start\_check\_rcp ; No - proceed

clr Flags1.STARTUP\_PHASE ; Clear startup phase flag

setb Flags1.INITIAL\_RUN\_PHASE ; Set initial run phase flag

mov Initial\_Run\_Rot\_Cntd, Temp3 ; Set initial run rotation count

mov Pwm\_Limit, Pwm\_Limit\_Beg

mov Pwm\_Limit\_By\_Rpm, Pwm\_Limit\_Beg

jmp normal\_run\_checks

direct\_start\_check\_rcp:

clr C

mov A, New\_Rcp ; Load new pulse value

subb A, #1 ; Check if pulse is below stop value

jc ($+5)

ljmp run1 ; Continue to run

jmp run\_to\_wait\_for\_power\_on

normal\_run\_checks:

; Check if it is initial run phase

jnb Flags1.INITIAL\_RUN\_PHASE, initial\_run\_phase\_done ; If not initial run phase - branch

jb Flags1.DIR\_CHANGE\_BRAKE, initial\_run\_phase\_done ; If a direction change - branch

; Decrement startup rotaton count

mov A, Initial\_Run\_Rot\_Cntd

dec A

; Check number of initial rotations

jnz initial\_run\_check\_startup\_rot ; Branch if counter is not zero

clr Flags1.INITIAL\_RUN\_PHASE ; Clear initial run phase flag

setb Flags1.MOTOR\_STARTED ; Set motor started

jmp run1 ; Continue with normal run

initial\_run\_check\_startup\_rot:

mov Initial\_Run\_Rot\_Cntd, A ; Not zero - store counter

jb Flags3.PGM\_BIDIR, initial\_run\_continue\_run ; Check if bidirectional operation

clr C

mov A, New\_Rcp ; Load new pulse value

subb A, #1 ; Check if pulse is below stop value

jc ($+5)

initial\_run\_continue\_run:

ljmp run1 ; Continue to run

jmp run\_to\_wait\_for\_power\_on

initial\_run\_phase\_done:

; Reset stall count

mov Stall\_Cnt, #0

; Exit run loop after a given time

jb Flags3.PGM\_BIDIR, run6\_check\_timeout ; Check if bidirectional operation

mov Temp1, #250

mov Temp2, #Pgm\_Brake\_On\_Stop

mov A, @Temp2

jz ($+4)

mov Temp1, #3 ; About 100ms before stopping when brake is set

clr C

mov A, Rcp\_Stop\_Cnt ; Load stop RC pulse counter low byte value

subb A, Temp1 ; Is number of stop RC pulses above limit?

jnc run\_to\_wait\_for\_power\_on ; Yes, go back to wait for poweron

run6\_check\_timeout:

mov A, Rcp\_Timeout\_Cntd ; Load RC pulse timeout counter value

jz run\_to\_wait\_for\_power\_on ; If it is zero - go back to wait for poweron

run6\_check\_dir:

jnb Flags3.PGM\_BIDIR, run6\_check\_speed ; Check if bidirectional operation

jb Flags3.PGM\_DIR\_REV, run6\_check\_dir\_rev ; Check if actual rotation direction

jb Flags2.RCP\_DIR\_REV, run6\_check\_dir\_change ; Matches force direction

jmp run6\_check\_speed

run6\_check\_dir\_rev:

jnb Flags2.RCP\_DIR\_REV, run6\_check\_dir\_change

jmp run6\_check\_speed

run6\_check\_dir\_change:

jb Flags1.DIR\_CHANGE\_BRAKE, run6\_check\_speed

setb Flags1.DIR\_CHANGE\_BRAKE ; Set brake flag

mov Pwm\_Limit, Pwm\_Limit\_Beg ; Set max power while braking

jmp run4 ; Go back to run 4, thereby changing force direction

run6\_check\_speed:

mov Temp1, #0F0h ; Default minimum speed

jnb Flags1.DIR\_CHANGE\_BRAKE, run6\_brake\_done; Is it a direction change?

mov Pwm\_Limit, Pwm\_Limit\_Beg ; Set max power while braking

mov Temp1, #20h ; Bidirectional braking termination speed

run6\_brake\_done:

clr C

mov A, Comm\_Period4x\_H ; Is Comm\_Period4x more than 32ms (~1220 eRPM)?

subb A, Temp1

jnc ($+5) ; Yes - stop or turn direction

ljmp run1 ; No - go back to run 1

jnb Flags1.DIR\_CHANGE\_BRAKE, run\_to\_wait\_for\_power\_on ; If it is not a direction change - stop

clr Flags1.DIR\_CHANGE\_BRAKE ; Clear brake flag

clr Flags3.PGM\_DIR\_REV ; Set spinning direction. Default fwd

jnb Flags2.RCP\_DIR\_REV, ($+5) ; Check force direction

setb Flags3.PGM\_DIR\_REV ; Set spinning direction

setb Flags1.INITIAL\_RUN\_PHASE

mov Initial\_Run\_Rot\_Cntd, #18

mov Pwm\_Limit, Pwm\_Limit\_Beg ; Set initial max power

jmp run1 ; Go back to run 1

run\_to\_wait\_for\_power\_on\_fail:

inc Stall\_Cnt ; Increment stall count

mov A, New\_Rcp ; Check if RCP is zero, then it is a normal stop

jz run\_to\_wait\_for\_power\_on

ajmp run\_to\_wait\_for\_power\_on\_stall\_done

run\_to\_wait\_for\_power\_on:

mov Stall\_Cnt, #0

run\_to\_wait\_for\_power\_on\_stall\_done:

clr IE\_EA

call switch\_power\_off

mov Flags0, #0 ; Clear flags0

mov Flags1, #0 ; Clear flags1

IF MCU\_48MHZ == 1

Set\_MCU\_Clk\_24MHz

ENDIF

setb IE\_EA

call wait100ms ; Wait for pwm to be stopped

call switch\_power\_off

mov Temp1, #Pgm\_Brake\_On\_Stop

mov A, @Temp1

jz run\_to\_wait\_for\_power\_on\_brake\_done

AcomFET\_on

BcomFET\_on

CcomFET\_on

run\_to\_wait\_for\_power\_on\_brake\_done:

clr C

mov A, Stall\_Cnt

subb A, #4

jc jmp\_wait\_for\_power\_on

jmp init\_no\_signal

jmp\_wait\_for\_power\_on:

jmp wait\_for\_power\_on ; Go back to wait for power on

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

$include (BLHeliPgm.inc) ; Include source code for programming the ESC

$include (BLHeliBootLoad.inc) ; Include source code for bootloader

;\*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\* \*\*\*\*

CSEG AT 19FDh

reset:

ljmp pgm\_start

END