```
$NOMOD51
;**** **** **** **** ****
;
; BLHeli program for controlling brushless motors in multirotors
;
; Copyright 2011, 2012 Steffen Skaug
; This program is distributed under the terms of the GNU General Public License
;
; This file is part of BLHeli.
;
; BLHeli is free software: you can redistribute it and/or modify
; it under the terms of the GNU General Public License as published by
; the Free Software Foundation, either version 3 of the License, or
; (at your option) any later version.
;
; BLHeli is distributed in the hope that it will be useful,
; but WITHOUT ANY WARRANTY; without even the implied warranty of
; MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
; GNU General Public License for more details.
;
; You should have received a copy of the GNU General Public License
; along with BLHeli.  If not, see <http://www.gnu.org/licenses/>.
;
;**** **** **** **** ****
;
; This software was initially designed for use with Eflite mCP X, but is now adapted to
copters/planes in general
;
; The software was inspired by and started from from Bernard Konze's BLMC:
http://home.versanet.de/~bkonze/blc_6a/blc_6a.htm
; And also Simon Kirby's TGY: https://github.com/sim-/tgy
;
; This file is best viewed with tab width set to 5
;
; The code is designed for multirotor applications, running damped light mode
;
; The input signal can be Normal (1-2ms), OneShot125 (125-250us), OneShot42 (41.7-83.3us) or
Multishot (5-25us) at rates as high as allowed by the format.
; Three Dshot signal rates are also supported, Dshot150, Dshot300 and Dshot600. A 48MHz MCU is
required for Dshot600.
; The code autodetects normal, OneShot125, Oneshot42, Multishot or Dshot.
;
; The first lines of the software must be modified according to the chosen environment:
; ESCNO EQU "ESC"
; MCU_48MHZ EQU "N"
; FETON_DELAY EQU "N"
;
;**** **** **** **** ****
; Revision history:
; - Rev16.0 Started. Built upon rev 14.5 of base code
;           Using hardware pwm for very smooth throttle response, silent running and support of very high rpms
;           Implemented reverse bidirectional mode
;           Implemented separate throttle gains fwd and rev in bidirectional mode
;           Implemented support for Oneshot42 and Multishot
; - Rev16.1 Made low rpm power limiting programmable through the startup power parameter
; - Rev16.2 Fixed bug that prevented temperature protection
;           Improved robustness to very high input signal rates
;           Beeps can be turned off by programming beep strength to 1
;           Throttle cal difference is checked to be above required minimum before storing. Throttle cal max is not stored until successful min throttle cal
; - Rev16.3 Implemented programmable temperature protection
;           Improved protection of bootloader and generally reduced risk of flash corruption
;           Some small changes for improved sync hold
; - Rev16.4 Fixed bug where bootloader operation could be blocked by a defective "eeprom" signature
; - Rev16.5 Added support for DShot150, DShot300 and DShot600
; - Rev16.6 Fixed signal detection issue of multishot at 32kHz
;           Improved bidirectional mode for high input signal rates
; - Rev16.7 Addition of Dshot commands for beeps and temporary reverse direction (largely by brycedjohnson)
;
;
;**** **** **** **** ****
; 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
```