```
;**************************************************************
;* This is the assembly language driver code for the mouse.   *
;* The driver modules are :                                   *
;*   initialize the ports for motor drive, sensor read and    *
;*     interrupt read                                         *
;*   drive the motors both forward                            *
;*   read the sensors                                         *
;*   perform a left or right turn                             *
;*   perform a left or right direction adjust                 *
;*   stop the motion                                          *
;* Create KCB - Aug 10, 2011, mod - Feb 27, 2012              *
;**************************************************************

; export symbols, add to main_asm.h, add to linker .prm
            XDEF initial
            XDEF sensors
            XDEF forward
            XDEF turn_left
            XDEF turn_right
            XDEF adj_left
            XDEF adj_right
            XDEF reverse
            XDEF stop_dead
            XDEF asm_main
            XDEF Delay
            XDEF printPC

; Include derivative-specific definitions
                INCLUDE 'derivative.inc'

; equates section
ADdelay:   equ $600 ;Delay value for A/D setup

SCFflag:   equ $80
admask2:   equ $80  ;ADPU = 1 Enable A/D
admask3:   equ $28  ;do 5 conversions - 5 sensors
admask4:   equ $E5  ;8 bit conver,16 A/D conver cycles, prescale 00101
admask5:   equ $B0  ;DJM=1(right),unsigned,continue convt,multi @ 0

printf     equ $EE88 ; printf function
out2hex    equ $EE9C ; output 2 chars to screen; in B
out4hex    equ $EEA0 ; output 4 chars to screen; in D

lcd_banner equ $0FEE ; LCD banner
lcd_clear  equ $0FE4 ; clear the display
lcd_cmd    equ $0FEA ; send a command
lcd_init   equ $0FEC ; initialize the LCD
lcd_putc   equ $0FE8 ; send a single char
lcd_puts:  equ $0FE6 ; display my message

; constant section
MY_EXTENDED_CON: SECTION
MsgSens:    DC.B "Sensor Data" , $00
MsgDelay:   DC.B "Delay Msg"   , $00
LcdInit:    DC.B "Inital"      , $00
MsgFrwd:    DC.B "Forward"     , $00
MsgRevs:    DC.B "Reverse"     , $00
MsgLeft:    DC.B "Left"        , $00
MsgRite:    DC.B "Rite"        , $00
MsgStop:    DC.B "Stop"        , $00
MsgDisp:    DC.B "Display"     , $00

; variable/data section
MY_EXTENDED_RAM: SECTION
```

```
temp_byte:  DS.B   1
counter:    DS.B   1
leftcnt:    DS.B   1
ritecnt:    DS.B   1
lflvnow:    DS.B   1
rtlvnow:    DS.B   1
lflvb4:     DS.B   1
rtlvb4:     DS.B   1
results:    DS.B   8

; code section
MyCode:     SECTION
; this assembly routine is called by the C/C++ application
; This code initializes LCD, sensors, motor direction
;    interrupts, and pulse width modulation
initial:
; Initialize the LCD display
;      LDX   #0
;      JSR   [lcd_init,x]   ; Initialize the LCD
;      LDD   #LcdInit
;      LDX   #0
;      JSR   [lcd_puts,x]
; Initialize the analog sensors to digital ports - port AD0
      MOVB   #$00,ATD0DIEN     ;disable digital inputs
      MOVB   #admask2,ATD0CTL2 ;config ATD0CTL2 - power up
      LDX    #ADdelay          ;Delay 100 usec - A/D powers up
loop: DEX
      BNE    loop
      MOVB   #admask3,ATD0CTL3 ;configure ATD0CTL3
      MOVB   #admask4,ATD0CTL4 ;configure ATD0CTL4
; Initialize the motor direction and IRQ I/P - port T
; Port T, bits 0,1,2,3 - used for motor direction (O/P)
; Port T, bits 4,5,6,7 - used for IRQ input determination (I/P)
      LDAA  #$0F       ; bits0-3=output, bits4-7=input
      STAA  DDRT       ; Port T Data Dir Reg
      LDAA  #$00
      STAA  RDRT       ; Reduced drive disabled
      LDAA  #$F0
      STAA  PERT       ; pull-up selected for I/P
      LDAA  #$00
      STAA  PPST       ; enable pull-up
; Initialize up the PWM ports - port P, bits 4-7
; The crystal is 24MHz - -  Fbus freq = 12Mhz
      LDAA  #$22
      STAA  PWMPRCLK   ; ClockA&B=FBus/4=3000K
      LDAA  #$F0       ; 1111 0000
      STAA  PWMCLK     ; ClockSB - ch.7,6 ; ClockSA - ch.5,4
      LDAA  #25
      STAA  PWMSCLA
      STAA  PWMSCLB    ; ClockSA&B=ClockA&B/2*15=100KHz
      LDAA  #$F0       ; 1111 0000
      STAA  PWMPOL     ; high, then low for polarity
      LDAA  #$00
      STAA  PWMCAE     ; left aligned
      LDAA  #$00
      STAA  PWMCTL     ; 8-bit chan, PWM during freeze and wait
      LDAA  #200
;      STAA  PWMPER4    ; PWM_Freq=ClockSA&B/200 = 0.5KHz
;      STAA  PWMPER5
      STAA  PWMPER6     ; motor 2
      STAA  PWMPER7     ; motor 1
;      JSR   stop_dead  ; initialize motor to stop!
      RTS               ; return to caller

; Initial sensor code
; Read all of the sensors and return one to caller
sensors:
; See initial for pre-code
      MOVB  #admask5,ATD0CTL5 ;start conversion
      BRCLR ATD0STAT0,SCFflag,* ;wait for conversion to complete
; process here is to read results and save
      LDAA  ATD0DR0L    ;get 1st result    LEFT   - bl/wh
      STAA  results     ;store in memory
      LDAA  ATD0DR1L    ;get 2nd result    CENTRE - grn
      STAA  results+1   ;store in memory
      LDAA  ATD0DR2L    ;get 3rd result    RIGHT  - yel
      STAA  results+2   ;store in memory
      LDAA  ATD0DR3L    ;get 4th result    LEFT WHEEL  - red
      STAA  results+3   ;store in memory
      LDAA  ATD0DR4L    ;get 5th result    RIGHT WHEEL - blk
      STAA  results+4   ;store in memory
;      LDAA  ATD0DR5L    ;get 6th result
;      STAA  results+5   ;store in memory
;      LDAA  ATD0DR6L    ;get 7th result
;      STAA  results+6   ;store in memory
;      LDAA  ATD0DR7L    ;get 8th result
;      STAA  results+7   ;store in memory
;      LDD  #MsgSens     ;load LCD title "Sensor Data"
;      LDX  #0
;      JSR  [lcd_puts,x]  ; message to lcd
;      LDD  #MsgDisp      ; load pc message  "Disp"
;      LDX  #0
;      JSR  [printf,x]
;      LDD  results+8
;      LDD  #1234
;      LDX  #0
;      CALL [out4hex,x]   ;message to screen
;      LDD    #$0001      ; Load sample sensor data
      RTS                ; return with sensor value in D

; Start the forward motion of the motors
forward:
;      LDD  #MsgFrwd      ; load pc message  "Forward"
;      LDX  #0
;      JSR  [printf,x]
; start both motors to forward
      LDAA  #$0F       ; FWD=0F  BAK=00
      ;(bit/mtr : 0/1FrontLeft 1/2FRight 2/3RearR 3/4RL)
      STAA  PTT
; How many steps forward
      LDAA  #$36       ; set counters to 07
      STAA  leftcnt    ; 7-0 and check for sub to get carry
      STAA  ritecnt
      JMP   adj_entr
; start the motors (second half of 'initial')
mtr_go:
;      STAA  PTT        ; set direction forward,forward
      LDAA  #120        ; duty cycle = PWMDTY/PWMPER (x/200)
;      STAA  PWMDTY4    ; min value = 50 on blocks
      STAA  PWMDTY6    ; motor 2 - right
      LDAA  #120        ;
;      STAA  PWMDTY5
      STAA  PWMDTY7    ; motor 1 - left
stop_go:
      LDAA  #$00
;      STAA  PWMCNT4    ; clear the channel
;      STAA  PWMCNT5    ; clear the channel
      STAA  PWMCNT6    ; clear the channel
      STAA  PWMCNT7    ; clear the channel
      BSET  PWME,%11110000 ; turn on ch. 4,5,6,7 - start PWMCNT
      RTS                   ; return to caller

; start the reverse motion of the motors
reverse:
;      LDD  #MsgRevs      ; load pc message  "Reverse"
;      LDX  #0
;      JSR  [printf,x]
; start both motors to backward
      LDAA  #$00       ; FWD=0F  BAK=00
       ;(bit/mtr : 0/1Fleft 1/2Fright 2/3RR 3/4RL)
      STAA  PTT
; How many steps backward
      LDAA  #$02       ; set counters to 07
      STAA  leftcnt    ; 7-0 and check for sub to get carry
      STAA  ritecnt
      JMP   adj_entr
;      JMP   mtr_go     ; go to common code in forward

; STOP both motors
stop_dead:
       JSR   stop_left
       JSR   stop_rite
       RTS
; start the STOP motion of the left motor
stop_left:
;      LDD  #MsgStop      ; load pc message  "Stop"
;      LDX  #0
;      JSR  [printf,x]
; reverse left motor direction
      LDAA  PTT
      EORA  #$0A
      STAA  PTT
; set the PWM to 30 ; motor will stall stopped!
      LDAA  #30        ; duty cycle = PWMDTY/PWMPER (x/200)
      STAA  PWMDTY7    ; motor 1 - left
      JMP   stop_go
 ; start the STOP motion of the right motor
stop_rite:
;      LDD  #MsgStop      ; load pc message  "Stop"
;      LDX  #0
;      JSR  [printf,x]
; reverse both motors
      LDAA  PTT
      EORA  #$05
      STAA  PTT
; set the PWM to 30 ; motor will stall stopped!
      LDAA  #30        ; duty cycle = PWMDTY/PWMPER (x/200)
      STAA  PWMDTY6    ; motor 2 - right
      JMP   stop_go

; start the turn left motion of the motors
turn_left:
;      LDD  #MsgLeft      ; load pc message  "Left"
;      LDX  #0
;      JSR  [printf,x]
; start left motor forward and right motor backward
      LDAA  #$0A       ; FWD=0F  BAK=00
       ;(bit/mtr : 0/1Fleft 1/2Fright 2/3RR 3/4RL)
turn_cnt:
      STAA  PTT
; Count the turn around
      LDAA  #$07       ; set counters to 07
      STAA  leftcnt    ; 7-0 and check for sub to get carry
      STAA  ritecnt
adj_entr:
      JSR   mtr_go     ; start the motors
      JSR   get_lvl    ; get wheel sensor level
      LDAA  lflvnow    ; save initial levels
      STAA  lflvb4
      LDAA  lflvnow
      STAA  lflvb4
;      STAB  level      ; save level as current level
look_agn:
      JSR   get_lvl    ; get wheel sensor level
; check the left wheel
chklf:
      LDAA  lflvnow
      CMPA  #$00       ; check for no level
      BEQ   chkrt      ; no level yet
           ; we have a level
      CMPA  lflvb4     ; chk for same as level before?
      BEQ   chkrt      ; same as before, no change
      STAA  lflvb4     ; set new before level
      DEC   leftcnt    ; dec the count
      BCC   chkrt      ; skip next if not zero yet
      JSR   stop_left  ; done left, stop left motor
; now check the right wheel
chkrt:
      LDAA  rtlvnow
      CMPA  #$00       ; check for no level
      BEQ   chkcnt      ; no level yet
           ; we have a level
      CMPA  rtlvb4     ; chk for same as level before?
      BEQ   chkcnt     ; same as before, no change
      STAA  rtlvb4     ; set new before level
      DEC   ritecnt    ; dec the count
      BCC   chkcnt     ; skip next if not zero yet
      JSR   stop_rite  ; done rite, stop rite motor
; now check the counts
chkcnt:
      LDAA  leftcnt
      ADDA  #$00
      BPL   look_agn   ; left is not zero, do again
      LDAA  ritecnt
      ADDA  #$00
      BPL   look_agn   ; rite is not zero, do again
      RTS

;      LDAA  results+4 ;level    ; left byte displays level
;      LDAB  counter    ; counter = right byte
;      LDX   #0
;      CALL  [out4hex,x]   ;o/p counter to screen
;      DEC   counter
;      BNE   look_agn   ; counter not 0 yet
;      LDD   #testMsg
;      JSR   [lcd_puts-*-4,pc]
;      BRA   *
;      JMP stop_dead
;      RTS                   ; return to caller

; start the turn left motion of the motors
turn_right:
;      LDD   #MsgRite      ; load pc message  "Right"
;      LDX   #0
;      JSR   [printf,x]
; start left motor backward and right motor forward
      LDAA  #$05       ; FWD=0F  BAK=00
       ;(bit/mtr : 0/1Fleft 1/2Fright 2/3RR 3/4RL)
      JMP   turn_cnt  ; use turn left for rest of counts
```

```
; Read the current level at the wheel (high/low)
; Start conversion and get result from wheel
get_lvl:
      JSR   sensors     ; get sensor values
; Determine level value -1=lo; 0=non; 1=hi
      LDAA  #00         ; assume both none
      STAA  lflvnow
      STAA  rtlvnow
;  ldaa results+3
;  ldaa results+4
      LDAA  #$60        ; less than is low
      CMPA  results+4   ; right
      BCS   X1          ; neg, Rt>$60
      DEC   rtlvnow     ; pos, Rt<$60, set rt to -1
X1:
      CMPA  results+3   ; left
      BCS   X2          ; neg, Lt>$60
      DEC   lflvnow     ; pos, Lt<$60, set lt to -1
X2:
      LDAA  #$A0        ; greater than is high
      CMPA  results+4   ; right
      BCC   X3          ; pos, Rt<$A0
      INC   rtlvnow     ; neg, Rt>$A0, set rt to 1
X3:
      CMPA  results+3
      BCC   X4          ; pos, Lt<$A0
      INC   lflvnow     ; pos, Lt>$A0, set lt to 1
X4:
      RTS


adj_left:
      LDAA  #$00       ; FWD=0F  BAK=00
       ;(bit/mtr : 0/1Fleft 1/2Fright 2/3RR 3/4RL)
      STAA  PTT
; Count the turn around
      LDAA  #$02       ; set counter to 02
      STAA  leftcnt
      LDAA  #$01       ; set couter to 01
      STAA  ritecnt
      JMP   adj_entr


;            MOVB   #1,temp_byte   ; just some demonstration code
;            NOP                   ; Insert here your own code
;            RTS                   ; return to caller

adj_right:
      LDAA  #$00       ; FWD=0F  BAK=00
       ;(bit/mtr : 0/1Fleft 1/2Fright 2/3RR 3/4RL)
      STAA  PTT
; Count the turn around
      LDAA  #$01       ; set counter to 02
      STAA  leftcnt
      LDAA  #$02       ; set couter to 01
      STAA  ritecnt
      JMP   adj_entr

;      MOVB   #1,temp_byte   ; just some demonstration code
;      NOP                   ; Insert here your own code
;      RTS                   ; return to caller


; Simple test assembly function
asm_main:
```

```
            MOVB   #1,temp_byte   ; just some demonstration code
            NOP                   ; Insert here your own code
            RTS                   ; return to caller

; Subroutine to convert a sensor reading byte (00-FF)
; to an ASCII code (30-39,41-45)
; input:  reg A
; output: reg D (A-hi,B-lo)
byte_asc: TFR   A,B    ; copy A to B
          LSRA         ; shift hi nibble to right
          LSRA
          LSRA
          LSRA
          ANDA  #$0F   ; clear upper lo nibble
          ANDB  #$0F   ; clear upper hi nibble
          ; convert A nibble to ASCII
          CMPA  #$0A
          BMI   dig_asc  ; A-10
let_asc:  ADDA  #$27
dig_asc:  ADDA  #$30
          ; convert B nibble to ASCII
          CMPB  #$0A
          BMI   dig_bsc  ; A-10
let_bsc:  ADDB  #$27
dig_bsc:  ADDB  #$30
          RTS

; Subroutine to display info on the PC terminal
; Data is in reg. D
printPC:  LDX  #0
          JSR  [printf,x]
          RTS

; Subroutine to delay for one-tenth of a second
;   times the value in reg A
; Input: reg A (number of tenth's of a second)
; Output: nothing
Delay:    TBA        ; value comes in D
          PSHX
          LSLA         ; shift left twice (x4)
          LSLA
          ABA          ; A=A+B   (4x + 1x = 5X)
Delay1:   LDX   #$FFFF  ;delay=19.1ms@24MHz
Dly_20:   NOP
          NOP
          NOP
          DEX
          BNE   Dly_20
          DECA
          BNE   Delay1  ; repeat inner
;          LDD  #MsgDelay   ;load LCD title
;          LDX  #0
;          JSR  [lcd_puts,x]
          PULX
          RTS

; End of code
```