COE538 Microprocessor Systems
The *eebot* Guider[1]

Peter Hiscocks
Department of Electrical and Computer Engineering
Ryerson University
phiscock@ee.ryerson.ca

Contents

List of Figures

[1]This exercise was adapted to be used with the HCS12 microcontroller by V. Geurkov.

1 The *Guider* Concept

The *eebot* robot is intended to find and follow a line on the floor surface. The line is a 3/4 inch wide red electrical tape on a background of black floor tiles[2]. For the purpose of following this line, eebot has a *guider* section mounted at the underside of the bow area.

The sensors are 6 CdS (Cadmium Sulphide) photoresistors that are high resistance in darkness and low resistance when illuminated. The sensors are labeled **A** through **F**, as shown in figure 1.

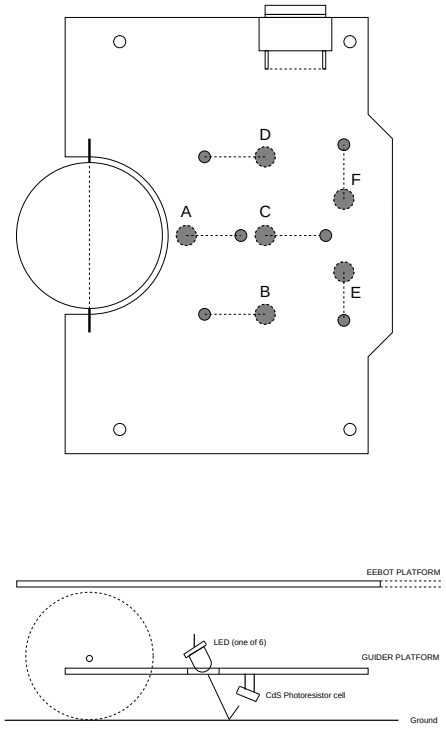

Figure 1: Guider Sensor Locations, Top View

The small circles are high-intensity red LEDs, which illuminate the floor surface under each sensor, so there is one LED per sensor. The large circles are the sensors themselves. Lines show LED-sensor pairs.

The relationship between a typical sensor and LED is shown in the bottom half of the diagram. The LEDs are mounted on the top side of the guider circuit board and illuminate the floor surface via a hole in the guider circuit board. This arrangement allows the CdS cells to be very close to the floor surface, which improves the sensitivity and reduces the effect of ambient light. As well, the LEDs are protected from being knocked out of alignment.

Each CdS photoresistor is driven by a constant current source so that it generates a voltage proportional to its resistance. Since the resistance decreases with increasing light level, so does the sensor voltage.

A simplified schematic of the Guider Sensors is shown in figure 2 on page 3.

The LEDs and CdS cells are selected by the same signals, the three PORTA general purpose digital outputs PA2, PA3 and PA4. The 74HC138 decoder that drives the LEDs is enabled and disabled by the general purpose digital output PA5.

One of the voltages developed across the sensors is selected by a 74HC4051 analogue multiplexer and directed to the HCS12 A/D input channel AN1. The same three general purpose digital outputs PA2, PA3 and PA4 selecting the LED also select the corresponding photosensor. Notice that 3 of the 8 possible sensor inputs are unused and connected to ground, so they should read zero.

[2]Black tape on a light background works equally well

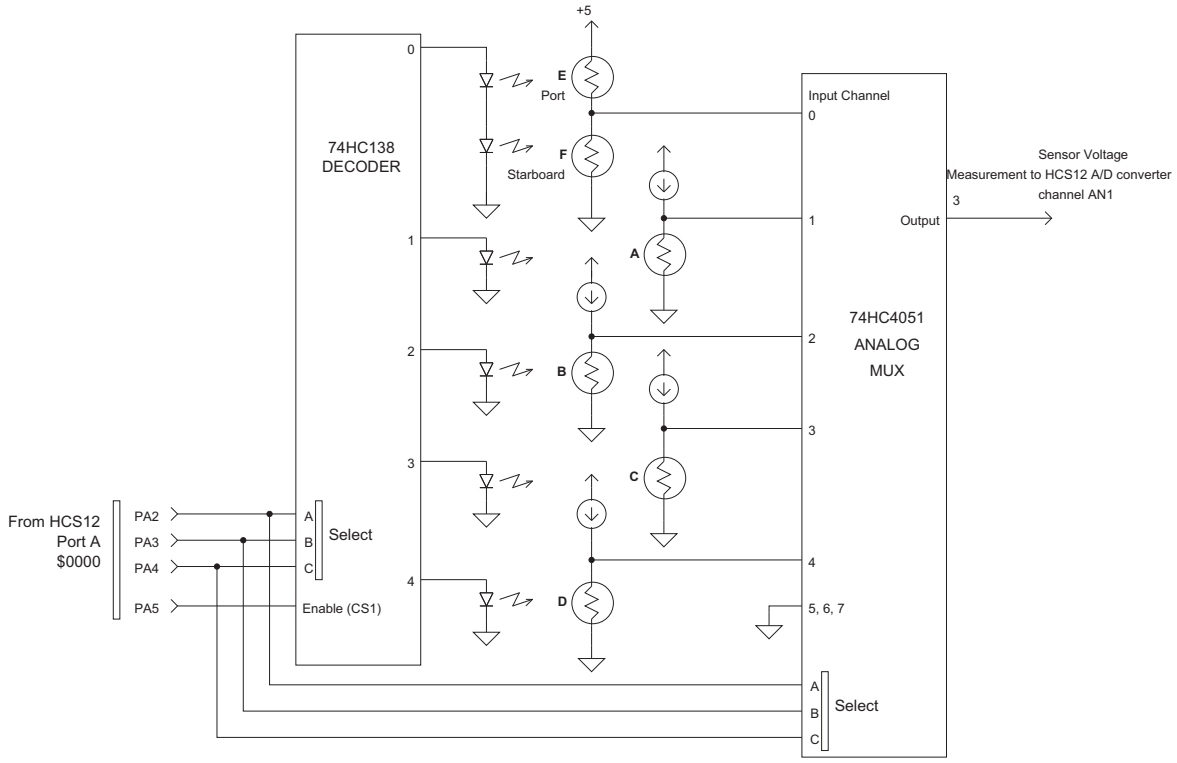

Figure 2: Guider Sensors

In general, the LEDs will be enabled so PA5 should be set to a high (logic 1, +5V) level. The PA2, PA3 and PA4 signals will have some value between 000 and 100 (msb-nsb-lsb) to illuminate one of the LEDs and simultaneously select one of the guider signals. The guider software could scan the 5 sensor signals or repeatedly read one sensor, whatever is required.

To detect the background illumination, the LEDs can be disabled by lowering the PA5 signal. The sensor readings would then indicate read the lighting without any LED illumination. This could be used as a baseline or *noise* value and subtracted from the value obtained when the LED is activated.

(The original guider design illuminated all LEDs at once, but that caused problems in light leaking from one LED into several sensors. Furthermore, there was no way to determine background illumination level.)

1.0.1 Guider Transient Response

The CdS photoresistors respond quickly to an increase in illumination (decreasing resistance) but rather slowly to decreasing illumination (increasing resistance). Consequently, for accurate measurements the photoresistor should be allowed time to darken between each measurement. The darkening time constant is in the order of 50 milliseconds[3]. Consequently, the most rapid allowable scan rate to scan one sensor is 20 times per second. If all five sensors are being scanned, the fastest allowable scan rate increases because a given sensor can be darkening while others are being read. Then the minimum scan time per sensor is 10 milliseconds, or about 100 times per second.

The differential line tracker pair of cells **E-F** actually recovers faster than a single cell, and the effective time constant is about 25 milliseconds. (This is because one of the pair is brightening while the other darkens.) So the

[3]This was determined by aiming an LED directly at a photoresistor and pulsing it from a function generator while measuring the voltage across the photoresistor with an oscilloscope.

line tracker signal could be scanned as fast as 40 times per second, which may be important for rapid response in steering.

1.0.2 Line Tracker

Sensors **E** and **F** used for *line tracking*. They are spaced 0.75 inches apart, so that they are centered over the two edges of electrical tape line when the robot is centered over the line. The two sensors are a voltage divider, so that the voltage at the center tap is determined by the ratio of the two sensor resistances. The guidance software of the robot should position the robot so that the voltage at the centre tap from voltage divider sensors **E** and **F** is exactly midway between 5 volts and zero, ie, 2.5 volts. Then the two sensor resistances are equal, the light level on the two cells is equal, and the two cells are equally over the black line and light background.

Notice that the absolute value of the resistance of sensors **E** and **F** is unimportant: it is their ratio that determines the output voltage. Consequently, they should largely ignore changes in ambient light level and this does indeed prove to be the case.

However, if both sensors are over a completely black background or a completely white background, they will also generate 2.5 volts. Consequently, some other method is required for ensuring that the robot is over a valid line, and that is the function of the pattern detector.

1.0.3 Pattern Detector

Sensors **A**, **B**, **C** and **D** form a *pattern detector* for various line configurations. For example, if sensors **A** and **C** detect a dark line while sensors **B** and **D** detect a light background, then the robot is probably over a valid line and the information from the line tracker can be used for guidance.

If the pattern detector sensors all detect a light background, then the robot is off line, and should begin to search for a line. The line tracker cannot be used.

If the pattern detector senses a black line at right angles to the current line (for example, sensors **A** reads light, sensor **B** reads dark, sensor **C** reads dark, sensor **D** reads light, so the line angles off to port), then the robot should advance until the drive motors are over the bend in the line and then pivot counter-clockwise until the spur line is under sensors **A** and **C**. Similar interpretation strategies should enable the pattern detector to sense a **T** junction or a line that simply comes to an end.

The sensors of the pattern detector each detect absolute light level. A constant current is passed through each sensor and the voltage generated across the sensor is then proportional to its resistance. The sensor currents are each adjusted so that the sensor generates about 1.8 volts over a light surface and 3 volts over a dark surface[4]. One of the voltages developed across the sensors is selected by an analogue multiplexer and directed to the HCS12 A/D input channel AN1.

The selection of one of the 5 sensor voltages is determined from the three general purpose digital outputs PA2, PA3 and PA4. Notice that 3 of the 8 possible analog inputs are unused and connected to ground so they should read zero.

2 Setting Up the Guider

In this section, we describe how to test the operation and calibration of the *eebot* line sensor system.

2.1 Checking the LED Pattern

A template for checking the guider LED illumination pattern is shown in figure 3 on page 5.

1. Set up the *eebot* so that it is powered up and connected to a HCS12 microcontroller board.

2. Load and run the program `read-guider` using CodeWarrior.

[4] As part of the operating procedure, the operator should check the A/D reading from each of the pattern detector sensors over light and dark surfaces, and calibrate the software thresholds accordingly.

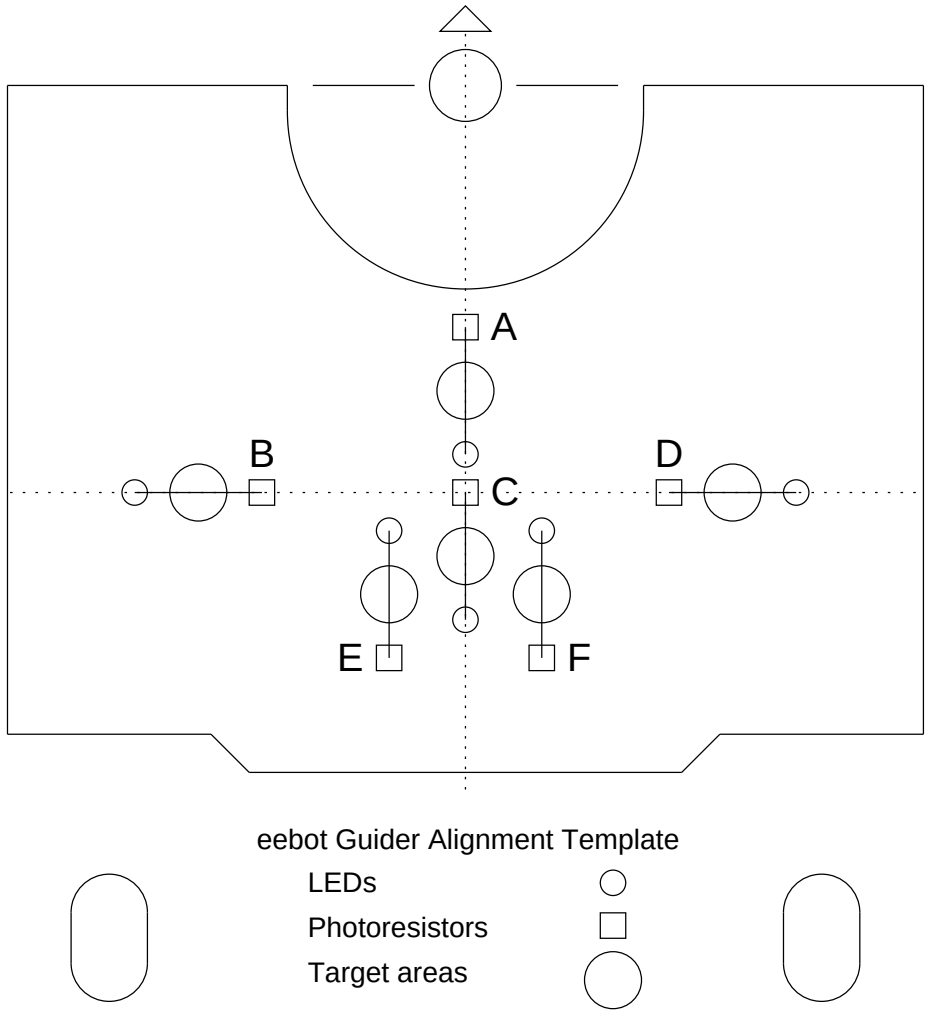

Figure 3: Guider Sensor Template

3. Make a full-sized copy of the template of figure 3. Place it under the eebot guider, lined up with the front sphere and rear driving tires as marked on the template.

4. Check that the 6 LED illumination circles match up with the large circles on the template.

5. If an LED is not aligned with its illumination circle, advise the lab supervisor and he/she will adjust the aiming of the LED.

2.2 The Background and Line

A number of line and background colors and textures were tested to determine the combination for line detection by the guider. The surface to be provided for the line following project consists of black floor tiles, each one foot square, each with a line of red electrician's tape, 3/4 inch wide. The tiles may be assembled in different arrangements to create different robot courses.

2.3 Checking Guider Operation

Here is a quick procedure for checking that the guider is working, at least to some extent.

1. Connect the *eebot* chassis to the HCS12 board development system and ensure that both are powered up correctly. Turn on the `LOGIC` switch on the robot chassis.
2. Load and run the program `read-guider` using CodeWarrior. You should see a display of the five sensor readings **A**, **B**, **C**, **D** and **E-F** on the microcomputer LCD.

   The guider sensor readings are shown in hexadecimal format, so we will use hexadecimal numbers (unless otherwise noted) throughout this section on calibration.
3. Apply the *finger test* to the guider. Make sure that the robot guider is over a light coloured surface. With an extended finger, carefully cover up each of the four different absolute sensors **A**, **B**, **C** and **D** in turn. As you do so, the reading for that sensor should increase noticeably, at least 50 counts (one volt). A variation of one or two counts is not significant - it may be due to noise in the A/D signals. Be careful not to disturb the aiming of the sensors or LEDs.
4. For the differential steering sensor pair **E-F**, covering up sensor **E** should cause the fifth reading to decrease. Covering up sensor F should cause the fifth reading to increase. Be careful not to dislodge the shades on sensors **E** and **F**: the differential sensor will not work correctly without them.

If this test fails on any or all of the sensors, get help: the guider is not functioning for some reason.

2.4 Guider Calibration: The Line Follower

Once the robot is over the line and following it approximately (which can be determined from the absolute sensors **A** and **C**), the line-follower sensor **E-F** can be used to generate a correction signal to steer the robot.

As the robot strays from the line, one of the line-follower **E-F** sensors will get darker and the other one lighter, changing the **E-F** sensor reading.

Suppose the **E-F** sensor reading varies between 28 with the robot pointing toward the PORT side to a value of A8 with the robot pointing toward the STARBOARD side.

Consequently, the robot should try to steer so that the line sensor has a value halfway between the extremes of 28 and A8, ie, about 68. Values above this should cause the robot to steer in one direction, and below this it should steer in the other direction. If you use this feature to steer the robot, you would build this value into your computer program.

If the line sensor is to be used as part of a negative feedback system, it will be necessary to know the *sensor gain*. This is *the amount by which the sensor reading changes* divided by *the distance over which this change occurs*. For example, if the sensor reading changes from 28 to A8 when the line sensor moves sideways by 0.5cm, then the sensor gain (in decimal units) is 256 counts per centimeter.

3 Guider Software

In this section, we describe the `read-guider` routine. You will need to use this routine (possibly with modifications) in your robot guidance project.

3.1 Guider API

At its most abstract level, we can picture that there is some routine that reads the guider sensors and puts them in a specific location in memory. The way in which this routine is accessed by other software routines is known as the *Application Program Interface* (API) for that routine. (The name derives from the common situation where a particular routine is part of the operating system of the computer. When an application must access some feature of the hardware, then it does so through the API of that particular routine.)

To improve documentation, we can relabel the names to something more informative than **A**, **B** and so as shown in the following table:

Designation	Name	Function
E-F	SENSOR_LINE	Line sensor
A	SENSOR_BOW	Front sensor
B	SENSOR_PORT	Port sensor
C	SENSOR_MID	Middle sensor
D	SENSOR_STAR	Starboard sensor

Then the header for the routine, which defines the interface for this routine (the API), might look like the code of figure 4.

```
*-------------------------------------------------------
*                    Read Sensors
*
* This routine reads the eebot guider sensors and puts the results in RAM registers
* Passed:       None
* Returns:      Sensor readings in:
*               SENSOR_LINE
*               SENSOR_BOW
*               SENSOR_PORT
*               SENSOR_MID
*               SENSOR_STAR
```

Figure 4: The API for *Read Sensors*

The control software main loop would periodically call READ_SENSORS to update the sensor readings. Routines that use the sensor signals can do so simply by reading these RAM registers.

3.2 The A/D Converter Hardware

The HCS12 A/D converter was described and applied previously in Lab 3. In this application, we will use the A/D in a slightly different mode.

Referring to figure 2 on page 3, all the guider signals are brought into Channel AN1 of the A/D converter via a multiplexer on the guider board. Consequently, the READ_SENSORS routine should do three things:

- enable the guider LEDs by setting pin PA5 to 1
- select the appropriate sensor (by setting the appropriate bit pattern in PORTA, which selects the appropriate multiplexer channel on the robot guider) and
- then read Channel AN1 of the A/D converter.

The bit assignments in the PORTA register are shown in figure 5. Bits 2, 3 and 4 control which of the sensor voltage is routed to channel 1 of the HCS12 A/D converter. When changing these bits, it is important not to affect the motor direction bits, also in PORTA.

Now we need to consider how to set up the A/D converter. The four registers controlling the A/D converter are ATDCTL2, ATDCTL3, ATDCTL4 and ATDCTL5.

Only one bit in the ATDCTL2 register is relevant to our application, as shown in figure 6. This bit must be set to power up the A/D converter. Therefore, the ATDCTL2 will be loaded with 80.

We are going to perform 4 consecutive conversions on the channel AN1 (in case we need to average the result), therefore the contents of the ATDCTL3 register will be 20.

We will select 8-bit resolution and set prescaler bits to 10111. This value will make the ADC clock rate equal to 24MHz : 48 = 500KHz. The contents of the ATDCTL4 register will be 97.

The ATDCTL2, ATDCTL3 and ATDCTL4 registers need only be loaded once. The ATDCTL5 register must be loaded every time an A/D conversion is to take place[5].The bit assignments of the ATDCTL5 register are shown in figure 9 on page 9.

The appropriate value to store in ATDCTL5, working from MSBit to LSBit, can be determined as follows:

[5]Assuming that we are in *single conversion* scan mode, which is the most appropriate mode for this application.

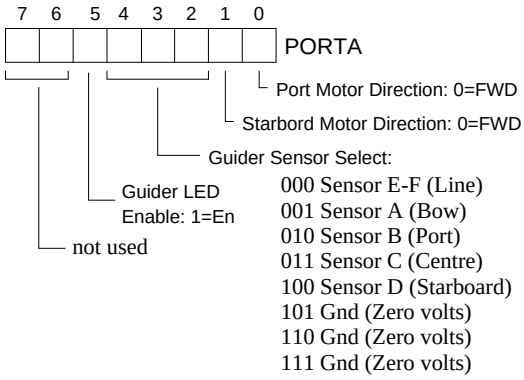

Figure 5: PORTA Register

ADPU							
res: 0	0	0	0	0	0	0	0

ADPU: ATD power down bit:
0 = power down ATD; 1 = normal ATD operation

Figure 6: ATDCTL2 Register

- Bit 7 selects result data justification. Let's choose right justification, so bit 7 is 1.
- Bit 5: We want conversions to occur *on request* by writing to `ATDCTL5` and then waiting for the *Sequence Complete Flag* to go high, so bit 5 should be 0.
- Bit 4: We want conversions on one channel, so bit 4 is 0.
- Bits 2 through 0: These bits should select channel `AN1`, so they contain 001.

Consequently, the conversation with the A/D hardware will consist of writing `81` to `ATDCTL5` and then looping while waiting for the *Sequence Complete Flag* (SCF) to go high. The SCF is the MSBit of the `ARDSTAT0` register shown in figure 10 on page 9.

In *single conversion* mode, the A/D takes 12 cycles per conversion on the selected channel. We configured the A/D to performs 4 conversions, so a complete conversion will take $4 \times 12 = 48$ A/D clock cycles to complete. Since the conversion clock rate equals to 500 KHz, the total conversion time will be $48 \times 1/(500 \times 10^3) = 96\mu s$.

Now we can write pseudocode for `READ_SENSORS`, as shown in figure 11.

The complete listing for `read-guider` is shown in section 5.

0	S8C	S4C	S2C	S1C			
res: 0	0	0	0	0	0	0	0

S8C,S4C,S2C,S1C: Conversion sequence limit
0001 = 1 conversion
0010 = 2 conversions
...
0111 = 7 conversions
1xxx = 8 conversions

Figure 7: ATDCTL3 Register

SRES8			PRS4	PRS3	PRS2	PRS1	PRS0
res: 0	0	0	0	0	0	0	0

SRES8: Select Resolution: 0 = 10-bits; 1 = 8-bits
PRS4--PRS0: ATD clock prescaler bits
00101 :12
01011 :24
10111 :48

Figure 8: ATDCTL4 Register

DJM		SCAN	MULT		CC	CB	CA
res: 0	0	0	0	0	0	0	0

DJM: 0 or 1 - left or right justified data in the rslt reg.
SCAN: 0 or 1 - single or continuous convers. sequences
MULT: 0 or 1 - sample one or several channels
CC, CB, CA: Channel select code
0 0 0 AN0
0 0 1 AN1
…
1 1 1 AN7

Figure 9: ATDCTL5 Register

4 Assignment: Read Guider

Demonstrate operation of the provided program `read-guider.asm`. Be prepared to answer technical questions on the hardware and programming of the guider.

SCF							
res: 0	0	0	0	0	0	0	0

SCF: Sequence complete flag
0 / 1 = conversion sequence not / has completed

Figure 10: ATDSTAT0 Register

```
        Initialize a pointer PTR into the RAM at the start of the Sensor Array storage
Loop    Store %10000001 to the ATDCTL5
        Repeat:
          Read ATDSTAT0
        Until Bit SCF of ATDSTAT0 == 1
        Store the contents of ATDDR0L at the pointer PTR
        If the pointer is at the last entry in Sensor Array, then exit
        Else increment the pointer and loop again.
```

Figure 11: *Read Sensors* Pseudocode

5 The *read-guider* Code

```
;*****************************************************************************
;*                 lab 7                                                     *
;*****************************************************************************

; export symbols
            XDEF Entry, _Startup            ; export 'Entry' symbol
            ABSENTRY Entry        ; for absolute assembly: mark this as application entry point



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


;-----------------------------------------------------------------------
;               'Read Guider' Demo Routine
;
; Reads the eebot guider sensors and displays the values
;  on the Liquid Crystal Display.

; Peter Hiscocks
; Version 2

; Modified from version 1 to support selection of the individual LED
;  associated with a sensor, to reduce crosstalk from unselected sensor
;  LEDs.
; The guider hardware was modified with the addition of a 74HC138 decoder that
;  drives the individual LEDs, so that only the LED associated with a given
;  sensor is ON when that sensor is being read.
; This requires that the software be modified to enable the decoder with bit PA5
;  in PORTA.
; The CdS cells are very slow in responding to changes in light, so a 20
;  millisecond delay is inserted between selecting a particular sensor and
;  reading its value.
; Substantial improvement:
;       Draws less battery current for longer life
;       Creates less heat in the 5V logic regulator
;       Much greater contrast between dark and light readings

; Overview:
; --------
; This program is intended as a test routine for the guider sensors of the
;  eebot robot and contains routines that will be useful in the robot
;  guidance project.
; The guider consists of four absolute brightness sensors and one
;  differential brightness pair of sensors. They are arranged at the nose of
;  the robot in the following pattern (viewed from above):

;                        A
;                      B  C  D
;                        E-F

; The sensors are cadmium sulphide (CdS) photoresistive cells, for which the
;  resistance increases with decreasing light level. The absolute cells
;  A,B,C and D are driven from a constant current source, and the voltage
;  across the cell measured via the HCS12 A/D converter channel AN1. Thus
;  the sensor reading increases as the sensor becomes darker (over a black
;  line, for example).

; The differential sensor E-F is a voltage divider with the two CdS cells E
;  and F separated 0.75 inches, which is the width of electrical tape. It is
;  intended to be used to track the edges of the electrical tape 'line' once
;  the absolute cells have 'found' a black line. Cell E is at the top of the
;  divider, so as the reading from this sensor increases, cell E is becoming
;  lighter, ie, cell E is straying onto the white background.
;  Simultaneously, cell F is becoming darker as it moves over the black
;  tape, and its resistance is increasing, aiding the same effect. The
;  differential action should ignore ambient light.

; The program reads the sensor values, hopefully without disturbing any
;  other settings on the robot. The values are displayed in hexadecimal on
;  the LCD. On the LCD display, the pattern is as described in the routine
;  'DISPLAY_SENSORS'.
```

```
; The 4 absolute sensors should show readings equivalent to approximately 2
;  volts when over a light surface and 4 volts when covered by a finger. The
;  range from light background to black tape background is typically 1.5 volts
;  over a light background to 2.4 volts over black tape.
; We have yet to quantify the readings from the differential sensor E-F.

; Using the program:
; -----------------
; Connect the eebot chassis to an HCS12 computer board as usual. Load
;  'read-guider' profram into the microcomputer. Run the routine 'MAIN'. The
;  display should show the five sensor readings. Placing a finger over
;  one of the sensors to block its illumination should cause the reading to
;  increase significantly. Be extremely careful not to bend the sensors or
;  LED illuminators when doing this.

; equates section

; A/D Converter Equates (all these are done in the 9S12C32.inc file):
;----------------------
; ATDCTL2 EQU $0082 ; A/D Control Register 2
;       7   6   5   4   3   2   1   0
;      --- --- --- --- --- --- --- ---
;     |   |   |   |   |   |   |   |   |
;      --- --- --- --- --- --- --- ---
;       ^   ^   ^   ^   ^   ^   ^   ^
;       |   |   |   |   |   |   |   |
;       |
;       +--------ADPU: 0 = A/D powered down
;                      1 = A/D powered up
;
; ATDCTL3 EQU $0083 ; A/D Control Register 3
;       7   6   5   4   3   2   1   0
;      --- --- --- --- --- --- --- ---
;     |   |   |   |   |   |   |   |   |
;      --- --- --- --- --- --- --- ---
;       ^   ^   ^   ^   ^   ^   ^   ^
;       |   |   |   |   |   |   |   |
;           |   |   |   |
;           +---+---+---+--- Conversion Sequence Limit: 0001 = 1 conversion
;                                                       ...
;                                                       0111 = 7 conversions
;                                                       1xxx = 8 conversions
;
; ATDCTL4 EQU $0084 ; A/D Control Register 4
;       7   6   5   4   3   2   1   0
;      --- --- --- --- --- --- --- ---
;     |   |   |   |   |   |   |   |   |
;      --- --- --- --- --- --- --- ---
;       ^   ^   ^   ^   ^   ^   ^   ^
;       |   |   |   |   |   |   |   |
;       |           +---+---+---+---+-- ATD Clock Prescaler Bits: 00101 = :12
;       |                                                         01011 = :24
;       |                                                         10111 = :48
;       |
;       +--- SRES8: 0 = 10 bits
;                   1 = 8  bits
;
; ATDCTL5 EQU $0085 ; A/D Control Register 5
;       7   6   5   4   3   2   1   0
;      --- --- --- --- --- --- --- ---
;     |   |   |   |   |   |   |   |   |
;      --- --- --- --- --- --- --- ---
;       ^   ^   ^   ^   ^   ^   ^   ^
;       |   |   |   |   |   |   |   |
;       |   |   |   |       +---+---+--- Channel Select
;       |   |   |   |
;       |   |   |   +-- MULT: 0 = Sample one channel
;       |   |   |             1 = Sample several channels starting
;       |   |   |                     with selected channel
;       |   |   |
;       |   |   +-- SCAN: 0 = Single conversion sequence per write to ADTCTL5
;       |   |             1 = Continuous conversion sequences
;       |   |
;       |   +-- not used
;       |
;       +--- DJM: 0 = Left justified data in the result register
;                 1 = Right justified data in the result register
;
```

```
; ATDSTAT0 EQU $0086 ; A/D Status Register 0
;       7   6   5   4   3   2   1   0
;      --- --- --- --- --- --- --- ---
;     |   |   |   |   |   |   |   |   |
;      --- --- --- --- --- --- --- ---
;       ^   ^   ^   ^   ^   ^   ^   ^
;       |   |   |   |   |   |   |   |
;       |
;       +--------SCF: 0 = Conversion sequence not completed
;                     1 = Conversion sequence has completed
;
; The A/D converter automatically puts the 4 results in these registers.
; ATDDR0L  EQU $0091   ; A/D Result Register 0
; ATDDR1L  EQU $0093   ; A/D Result Register 1
; ATDDR2L  EQU $0095   ; A/D Result Register 2
; ATDDR3L  EQU $0097   ; A/D Result Register 3


; PORTA Register
;--------------------------------
; This register selects which sensor is routed to AN1 of the A/D converter
;
; PORTA EQU $0000 ; PORTA Register
;       7   6   5   4   3   2   1   0
;      --- --- --- --- --- --- --- ---
;     |   |   |   |   |   |   |   |   |
;      --- --- --- --- --- --- --- ---
;       ^   ^   ^   ^   ^   ^   ^   ^
;       |   |   |   |   |   |   |   |
;       |   |   |   |   |   |   |   +--- Port Motor Direction (0 = FWD)
;       |   |   |   |   |   |   |
;       |   |   |   |   |   |   +------- Starboard Motor Direction (0 = FWD)
;       |   |   |   |   |   |
;       |   |   |   +---+---+-- Sensor Select
;       |   |   | 000  Sensor Line
;       |   |   | 001  Sensor Bow
;       |   |   | 010  Sensor Port
;       |   |   | 011  Sensor Mid
;       |   |   | 100  Sensor Starboard
;       |   |   |
;       |   |   +-- Sensor LED enable (1 = ON)
;       |   |
;       +---+- not used


; Liquid Crystal Display Equates
;--------------------------------
CLEAR_HOME      EQU $01             ; Clear the display and home the cursor
INTERFACE       EQU $38             ; 8 bit interface, two line display
CURSOR_OFF      EQU $0C             ; Display on, cursor off
SHIFT_OFF       EQU $06             ; Address increments, no character shift
LCD_SEC_LINE    EQU 64              ; Starting addr. of 2nd line of LCD (note decimal value!)

; LCD Addresses
LCD_CNTR        EQU PTJ             ; LCD Control Register: E = PJ7, RS = PJ6
LCD_DAT         EQU PORTB           ; LCD Data Register: D7 = PB7, ... , D0 = PB0
LCD_E           EQU $80             ; LCD E-signal pin
LCD_RS          EQU $40             ; LCD RS-signal pin

; Other codes
NULL            EQU 00              ; The string 'null terminator'
CR              EQU $0D             ; 'Carriage Return' character
SPACE           EQU ' '             ; The 'space' character


; variable/data section

                ORG $3800
;---------------------------------------------------------------------------
; Storage Registers (9S12C32 RAM space: $3800 ... $3FFF)

SENSOR_LINE     FCB $01             ; Storage for guider sensor readings
SENSOR_BOW      FCB $23             ; Initialized to test values
SENSOR_PORT     FCB $45
SENSOR_MID      FCB $67
SENSOR_STBD     FCB $89

SENSOR_NUM      RMB 1               ; The currently selected sensor
```

```
TOP_LINE        RMB 20              ; Top line of display
                FCB NULL            ;  terminated by null

BOT_LINE        RMB 20              ; Bottom line of display
                FCB NULL            ;  terminated by null

CLEAR_LINE      FCC '                   '
                FCB NULL            ;  terminated by null

TEMP            RMB 1               ; Temporary location

; code section

                ORG $4000           ; Start of program text (FLASH memory)
;-------------------------------------------------------------------------
;               Initialization

Entry:
_Startup:
                LDS #$4000          ; Initialize the stack pointer
                CLI                 ; Enable interrupts

                JSR INIT            ; Initialize ports
                JSR openADC         ; Initialize the ATD
                JSR openLCD         ; Initialize the LCD
                JSR CLR_LCD_BUF     ; Write 'space' characters to the LCD buffer

;-------------------------------------------------------------------------
;               Display Sensors

MAIN            JSR G_LEDS_ON       ; Enable the guider LEDs
                JSR READ_SENSORS    ; Read the 5 guider sensors
                JSR G_LEDS_OFF      ; Disable the guider LEDs
                JSR DISPLAY_SENSORS ; and write them to the LCD
                LDY #6000           ; 300 ms delay to avoid
                JSR del_50us        ;  display artifacts
                BRA MAIN            ; Loop forever

; subrotine section

;-------------------------------------------------------------------------
;               Initialize ports

INIT            BCLR DDRAD,$FF      ; Make PORTAD an input (DDRAD @ $0272)
                BSET DDRA,$FF       ; Make PORTA an output (DDRA @ $0002)
                BSET DDRB,$FF       ; Make PORTB an output (DDRB @ $0003)
                BSET DDRJ,$C0       ; Make pins 7,6 of PTJ outputs (DDRJ @ $026A)
                RTS

;-------------------------------------------------------------------------
;               Initialize the ADC

openADC         MOVB #$80,ATDCTL2   ; Turn on ADC (ATDCTL2 @ $0082)
                LDY  #1             ; Wait for 50 us for ADC to be ready
                JSR  del_50us       ;    - " -
                MOVB #$20,ATDCTL3   ; 4 conversions on channel AN1 (ATDCTL3 @ $0083)
                MOVB #$97,ATDCTL4   ; 8-bit resolution, prescaler=48 (ATDCTL4 @ $0084)
                RTS

;-------------------------------------------------------------------------
;               Clear LCD Buffer

; This routine writes 'space' characters (ascii 20) into the LCD display
;  buffer in order to prepare it for the building of a new display buffer.
; This needs only to be done once at the start of the program. Thereafter the
;  display routine should maintain the buffer properly.

CLR_LCD_BUF     LDX #CLEAR_LINE
                LDY #TOP_LINE
                JSR STRCPY

CLB_SECOND      LDX #CLEAR_LINE
                LDY #BOT_LINE
                JSR STRCPY

CLB_EXIT        RTS
```

```
;-------------------------------------------------------------------------
;               String Copy

; Copies a null-terminated string (including the null) from one location to
; another

; Passed: X contains starting address of null-terminated string
;   Y contains first address of destination

STRCPY          PSHX                ; Protect the registers used
                PSHY
                PSHA
STRCPY_LOOP     LDAA 0,X            ; Get a source character
                STAA 0,Y            ; Copy it to the destination
                BEQ  STRCPY_EXIT    ; If it was the null, then exit
                INX                 ; Else increment the pointers
                INY
                BRA  STRCPY_LOOP    ; and do it again
STRCPY_EXIT     PULA                ; Restore the registers
                PULY
                PULX
                RTS

;-------------------------------------------------------------------------
;               Guider LEDs ON

; This routine enables the guider LEDs so that readings of the sensor
;  correspond to the 'illuminated' situation.

; Passed:  Nothing
; Returns: Nothing
; Side:    PORTA bit 5 is changed

G_LEDS_ON       BSET PORTA,%00100000 ; Set bit 5
                RTS

;
;               Guider LEDs OFF

; This routine disables the guider LEDs. Readings of the sensor
;  correspond to the 'ambient lighting' situation.

; Passed:  Nothing
; Returns: Nothing
; Side:    PORTA bit 5 is changed

G_LEDS_OFF      BCLR PORTA,%00100000 ; Clear bit 5
                RTS

;-------------------------------------------------------------------------
;               Read Sensors
;
; This routine reads the eebot guider sensors and puts the results in RAM
;  registers.

; Note: Do not confuse the analog multiplexer on the Guider board with the
;  multiplexer in the HCS12. The guider board mux must be set to the
;  appropriate channel using the SELECT_SENSOR routine. The HCS12 always
;  reads the selected sensor on the HCS12 A/D channel AN1.

; The A/D conversion mode used in this routine is to read the A/D channel
;  AN1 four times into HCS12 data registers ATDDR0,1,2,3. The only result
;  used in this routine is the value from AN1, read from ATDDR0. However,
;  other routines may wish to use the results in ATDDR1, 2 and 3.
; Consequently, Scan=0, Mult=0 and Channel=001 for the ATDCTL5 control word.

; Passed:       None
; Returns:      Sensor readings in:
;               SENSOR_LINE (0) (Sensor E/F)
;               SENSOR_BOW  (1) (Sensor A)
;               SENSOR_PORT (2) (Sensor B)
;               SENSOR_MID  (3) (Sensor C)
;               SENSOR_STBD (4) (Sensor D)
; Note:
;   The sensor number is shown in brackets
;
; Algorithm:
;        Initialize the sensor number to 0
```

```
;        Initialize a pointer into the RAM at the start of the Sensor Array storage
; Loop   Store %10000001 to the ATDCTL5 (to select AN1 and start a conversion)
;        Repeat
;           Read ATDSTAT0
;        Until Bit SCF of ATDSTAT0 == 1 (at which time the conversion is complete)
;        Store the contents of ATDDR0L at the pointer
;        If the pointer is at the last entry in Sensor Array, then
;           Exit
;        Else
;           Increment the sensor number
;           Increment the pointer
;        Loop again.

READ_SENSORS    CLR  SENSOR_NUM      ; Select sensor number 0
                LDX  #SENSOR_LINE    ; Point at the start of the sensor array

RS_MAIN_LOOP    LDAA SENSOR_NUM      ; Select the correct sensor input
                JSR  SELECT_SENSOR   ;  on the hardware
                LDY  #400            ; 20 ms delay to allow the
                JSR  del_50us        ;  sensor to stabilize

                LDAA #%10000001      ; Start A/D conversion on AN1
                STAA ATDCTL5
                BRCLR ATDSTAT0,$80,* ; Repeat until A/D signals done

                LDAA ATDDR0L         ; A/D conversion is complete in ATDDR0L
                STAA 0,X             ;  so copy it to the sensor register
                CPX  #SENSOR_STBD    ; If this is the last reading
                BEQ  RS_EXIT         ; Then exit

                INC  SENSOR_NUM      ; Else, increment the sensor number
                INX                  ;  and the pointer into the sensor array
                BRA  RS_MAIN_LOOP    ;  and do it again

RS_EXIT         RTS

;---------------------------------------------------------------------------
;               Select Sensor

; This routine selects the sensor number passed in ACCA. The motor direction
;  bits 0, 1, the guider sensor select bit 5 and the unused bits 6,7 in the
;  same machine register PORTA are not affected.
; Bits PA2,PA3,PA4 are connected to a 74HC4051 analog mux on the guider board,
;  which selects the guider sensor to be connected to AN1.

; Passed: Sensor Number in ACCA
; Returns: Nothing
; Side Effects: ACCA is changed

; Algorithm:
; First, copy the contents of PORTA into a temporary location TEMP and clear
;        the sensor bits 2,3,4 in the TEMP to zeros by ANDing it with the mask
;        11100011. The zeros in the mask clear the corresponding bits in the
;        TEMP. The 1's have no effect.
; Next, move the sensor selection number left two positions to align it
;        with the correct bit positions for sensor selection.
; Clear all the bits around the (shifted) sensor number by ANDing it with
;  the mask 00011100. The zeros in the mask clear everything except
;        the sensor number.
; Now we can combine the sensor number with the TEMP using logical OR.
;  The effect is that only bits 2,3,4 are changed in the TEMP, and these
;  bits now correspond to the sensor number.
; Finally, save the TEMP to the hardware.

SELECT_SENSOR   PSHA                 ; Save the sensor number for the moment

                LDAA PORTA           ; Clear the sensor selection bits to zeros
                ANDA #%11100011      ;
                STAA TEMP            ; and save it into TEMP

                PULA                 ; Get the sensor number
                ASLA                 ; Shift the selection number left, twice
                ASLA                 ;
                ANDA #%00011100      ; Clear irrelevant bit positions

                ORAA TEMP            ; OR it into the sensor bit positions
                STAA PORTA           ; Update the hardware
                RTS
```

```
;-------------------------------------------------------------------------
;               Display Sensor Readings

; Passed: Sensor values in RAM locations SENSOR_LINE through SENSOR_STBD.
; Returns: Nothing
; Side: Everything

; This routine writes the sensor values to the LCD. It uses the 'shadow buffer' approach.
;  The display buffer is built by the display controller routine and then copied in its
;  entirety to the actual LCD display. Although simpler approaches will work in this
;  application, we take that approach to make the code more re-useable.
; It's important that the display controller not write over other information on the
;  LCD, so writing the LCD has to be centralized with a controller routine like this one.
; In a more complex program with additional things to display on the LCD, this routine
;  would be extended to read other variables and place them on the LCD. It might even
;  read some 'display select' variable to determine what should be on the LCD.

; For the purposes of this routine, we'll put the sensor values on the LCD
;  in such a way that they (sort of) mimic the position of the sensors, so
;  the display looks like this:
;   0123456789012345678
;   ___FF______________
;   PP_MM_SS_LL_________

; Where FF is the front sensor, PP is port, MM is mid, SS is starboard and
;  LL is the line sensor.

; The corresponding addresses in the LCD buffer are defined in the following
;  equates (In all cases, the display position is the MSDigit).

DP_FRONT_SENSOR EQU TOP_LINE+3
DP_PORT_SENSOR  EQU BOT_LINE+0
DP_MID_SENSOR   EQU BOT_LINE+3
DP_STBD_SENSOR  EQU BOT_LINE+6
DP_LINE_SENSOR  EQU BOT_LINE+9

DISPLAY_SENSORS LDAA SENSOR_BOW       ; Get the FRONT sensor value
                JSR  BIN2ASC          ; Convert to ascii string in D
                LDX  #DP_FRONT_SENSOR ; Point to the LCD buffer position
                STD  0,X              ;  and write the 2 ascii digits there

                LDAA SENSOR_PORT      ; Repeat for the PORT value
                JSR  BIN2ASC
                LDX  #DP_PORT_SENSOR
                STD  0,X

                LDAA SENSOR_MID       ; Repeat for the MID value
                JSR  BIN2ASC
                LDX  #DP_MID_SENSOR
                STD  0,X

                LDAA SENSOR_STBD      ; Repeat for the STARBOARD value
                JSR  BIN2ASC
                LDX  #DP_STBD_SENSOR
                STD  0,X

                LDAA SENSOR_LINE      ; Repeat for the LINE value
                JSR  BIN2ASC
                LDX  #DP_LINE_SENSOR
                STD  0,X

                LDAA #CLEAR_HOME      ; Clear the display and home the cursor
                JSR  cmd2LCD          ;       "

                LDY  #40              ; Wait 2 ms until "clear display" command is complete
                JSR  del_50us

                LDX  #TOP_LINE        ; Now copy the buffer top line to the LCD
                JSR  putsLCD

                LDAA #LCD_SEC_LINE    ; Position the LCD cursor on the second line
                JSR  LCD_POS_CRSR

                LDX  #BOT_LINE        ; Copy the buffer bottom line to the LCD
                JSR  putsLCD
                RTS
```

```
;-------------------------------------------------------------------------
;               Binary to ASCII

; Converts an 8 bit binary value in ACCA to the equivalent ASCII character 2
;  character string in accumulator D
; Uses a table-driven method rather than various tricks.

; Passed: Binary value in ACCA
; Returns: ASCII Character string in D
; Side Fx: ACCB is destroyed

HEX_TABLE       FCC '0123456789ABCDEF' ; Table for converting values

BIN2ASC         PSHA                   ; Save a copy of the input number on the stack
                TAB                    ;  and copy it into ACCB
                ANDB #%00001111        ; Strip off the upper nibble of ACCB
                CLRA                   ; D now contains 000n where n is the LSnibble
                ADDD #HEX_TABLE        ; Set up for indexed load
                XGDX
                LDAA 0,X               ; Get the LSnibble character

                PULB                   ; Retrieve the input number into ACCB
                PSHA                   ; and push the LSnibble character in its place
                RORB                   ; Move the upper nibble of the input number
                RORB                   ;  into the lower nibble position.
                RORB
                RORB
                ANDB #%00001111        ; Strip off the upper nibble
                CLRA                   ; D now contains 000n where n is the MSnibble
                ADDD #HEX_TABLE        ; Set up for indexed load
                XGDX
                LDAA 0,X               ; Get the MSnibble character into ACCA
                PULB                   ; Retrieve the LSnibble character into ACCB
                RTS

;-------------------------------------------------------------------------
;     Routines to control the Liquid Crystal Display

;-------------------------------------------------------------------------
;               Initialize the LCD

openLCD         LDY  #2000             ; Wait 100 ms for LCD to be ready
                JSR  del_50us          ;       "
                LDAA #INTERFACE        ; Set 8-bit data, 2-line display, 5x8 font
                JSR  cmd2LCD           ;       "
                LDAA #CURSOR_OFF       ; Display on, cursor off, blinking off
                JSR  cmd2LCD           ;       "
                LDAA #SHIFT_OFF        ; Move cursor right (address increments, no char. shift)
                JSR  cmd2LCD           ;       "
                LDAA #CLEAR_HOME       ; Clear the display and home the cursor
                JSR  cmd2LCD           ;       "
                LDY  #40               ; Wait 2 ms until "clear display" command is complete
                JSR  del_50us          ;       "
                RTS

;-------------------------------------------------------------------------
;               Send a command in accumulator A to the LCD

cmd2LCD         BCLR LCD_CNTR,LCD_RS   ; Select the LCD Instruction register
                JSR  dataMov           ; Send data to IR or DR of the LCD
                RTS

;-------------------------------------------------------------------------
;               Send a character in accumulator in A to LCD

putcLCD         BSET LCD_CNTR,LCD_RS   ; select the LCD Data register
                JSR  dataMov           ; send data to IR or DR of the LCD
                RTS

;-------------------------------------------------------------------------
;               Send a NULL-terminated string pointed to by X

putsLCD         LDAA 1,X+              ; get one character from the string
                BEQ  donePS            ; reach NULL character?
                JSR  putcLCD
                BRA  putsLCD
donePS          RTS
```

```
;-------------------------------------------------------------------------
;               Send data to the LCD IR or DR depending on the RS signal

dataMov         BSET LCD_CNTR,LCD_E    ; pull the LCD E-sigal high
                STAA LCD_DAT           ; send the 8 bits of data to LCD
                NOP
                NOP
                NOP
                BCLR LCD_CNTR,LCD_E    ; pull the E signal low to complete the write operation

                LDY  #1                ; adding this delay will complete the internal
                JSR  del_50us          ; operation for most instructions
                RTS

;-------------------------------------------------------------------------
;               Position the Cursor

; This routine positions the display cursor in preparation for the writing
;  of a character or string.
; For a 20x2 display:
; The first line of the display runs from 0 .. 19.
; The second line runs from 64 .. 83.

; The control instruction to position the cursor has the format
;            1aaaaaaa
; where aaaaaaa is a 7 bit address.

; Passed:   7 bit cursor Address in ACCA
; Returns:  Nothing
; Side Effects: None

LCD_POS_CRSR    ORAA #%10000000        ; Set the high bit of the control word
                JSR  cmd2LCD           ;  and set the cursor address
                RTS

;-------------------------------------------------------------------------
;               50 Microsecond Delay

del_50us        PSHX                   ; (2 E-clk) Protect the X register
eloop           LDX #300               ; (2 E-clk) Initialize the inner loop counter
iloop           NOP                    ; (1 E-clk) No operation
                DBNE  X,iloop          ; (3 E-clk) If the inner cntr not 0, loop again
                DBNE  Y,eloop          ; (3 E-clk) If the outer cntr not 0, loop again
                PULX                   ; (3 E-clk) Restore the X register
                RTS                    ; (5 E-clk) Else return

;-------------------------------------------------------------------------
;               Interrupt Vectors

                ORG   $FFFE
                DC.W  Entry            ; Reset Vector
```

6 Appendix: Signal Averaging

If a particular signal is corrupted by random noise, the variance of the noise may be reduced by collecting a series of samples and then averaging them together. Each time the number of samples is doubled, the random component is reduced by 3db (a factor of 0.707). Consequently, 4 samples decrease the noise by a factor of 6db, or 0.5. This strategy may be applied to the values read by an A/D channel in order to reduce the noise on the reading.

For this to apply, the samples must be *uncorrelated*, that is, random with respect to each other. To see why this is necessary , consider that a (desired) DC signal is applied to an A/D input. To this is added a very slow-changing (undesired) noise signal. The A/D clock is 1μ second and according to the data book each conversion takes 32 cycles so each conversion takes 32μ seconds. If the noise signal does not change significantly in this 32μ second interval between samples it will not average to zero.

As well, a periodic signal which is synchronous with the samples will also not average to zero. (Think of an AC noise signal where the peak of each cycle coincides with the sampling of the waveform: this will always return the same value, so the average will not go to zero.) In essence, this is a problem in digital signal processing, in which the sampling interval affects the frequencies filtered. Without knowing the properties of a noise signal, we can't predict how effective it will be to average 4 sensor readings. However, it's easy to do in software, it will not worsen the noise level and might improve it.

When the HCS12 A/D converter is set up to read one channel, we can configure it to do e.g. four successive reads and conversions. The results will be put in the four HCS12 registers `ATDDR0` through `ATDDR3`. It is then a simple matter of summing the four readings and dividing the result by 4 to obtain an average. (You may wish to add this feature to the `READ_SENSORS` routine.)

Before using software averaging to reduce noise, electronic methods should be used to ensure that the signal into the A/D is as noise-free as possible. For example, it may be necessary to shield a signal from electromagnetic coupling, move the analog ground connection, or use electronic filtering on the signal to remove noise. Software averaging is not a good substitute for these measures.