```
 1
 2 /*
 3  * File:   adc_driver.c
 4  * Author: M.Ricchieri
 5  *
 6  * Created on 31. mai 2023, 08:55
 7  *
 8  * Inspired by the "Mc32DriverAdc.c" file
 9  */
10
11
12 //----------------------------------------------------------------------------// Includes
13 #include "voltageADC_driver.h"
14
15
16 //----------------------------------------------------------------------------// Constants
17 #define ADC_RESOLUTION  1024
18 #define ADC_VREF        3.3
19
20 //-------------------------------------------------//
21 // Help to select the SCAN_MASK value
22 //
23 // AN10   AN9    AN8    AN7    AN6    AN5    AN4    AN3    AN2    AN1    AN0
24 //  |      |      |      |      |      |      |      |      |      |      |
25 //  1      1      0      0      0      0      0      0      0      0      0
26 //
27 // Value in CONFIGSCAN = 0b0011 0000 0000 = 0x600
28 //
29 #define SCAN_MASK  0x0600
30 //-------------------------------------------------//
31
32
33 //----------------------------------------------------------------------------// initAdc
34 void initAdc(void){
35
36     // Mask configuration
37     PLIB_ADC_InputScanMaskAdd(ADC_ID_1, SCAN_MASK);
38     // Data return configuration
39     PLIB_ADC_ResultFormatSelect(ADC_ID_1, ADC_RESULT_FORMAT_INTEGER_16BIT);
40     // Alternate buffer selection
41     PLIB_ADC_ResultBufferModeSelect(ADC_ID_1, ADC_BUFFER_MODE_TWO_8WORD_BUFFERS);
42     // Multiplex mode selection
43     PLIB_ADC_SamplingModeSelect(ADC_ID_1, ADC_SAMPLING_MODE_MUXA);
44     PLIB_ADC_ConversionTriggerSourceSelect(ADC_ID_1,
45             ADC_CONVERSION_TRIGGER_INTERNAL_COUNT);
46     // Reference selection
47     PLIB_ADC_VoltageReferenceSelect(ADC_ID_1, ADC_REFERENCE_VDD_TO_AVSS );
48     PLIB_ADC_SampleAcquisitionTimeSet(ADC_ID_1, 0x1F);
49     PLIB_ADC_ConversionClockSet(ADC_ID_1, SYS_CLK_FREQ, 32);
50     // Configuration of number of readings (depends on the number of inputs)
51     PLIB_ADC_SamplesPerInterruptSelect(ADC_ID_1, ADC_2SAMPLES_PER_INTERRUPT);
52     PLIB_ADC_MuxAInputScanEnable(ADC_ID_1);
53     // Enable the ADC module
54     PLIB_ADC_Enable(ADC_ID_1);
55 }
56
57
58 //----------------------------------------------------------------------------// readRawAdc
59 void readRawAdc(RAW_ADC *pRawAdc){
60
61     ADC_RESULT_BUF_STATUS BufStatus;
62
63     // Stop sample/convert
64     PLIB_ADC_SampleAutoStartDisable(ADC_ID_1);
65
66     // Treatment with alternating buffer
67     BufStatus = PLIB_ADC_ResultBufferStatusGet(ADC_ID_1);
68
69     if (BufStatus == ADC_FILLING_BUF_0TO7){
70
71         pRawAdc->AN9_V_GEN  = PLIB_ADC_ResultGetByIndex(ADC_ID_1, 0);
72         pRawAdc->AN10_V_BAT = PLIB_ADC_ResultGetByIndex(ADC_ID_1, 1);
73     }else{
74
```

```
75          pRawAdc->AN9_V_GEN  = PLIB_ADC_ResultGetByIndex(ADC_ID_1, 8);
76          pRawAdc->AN10_V_BAT = PLIB_ADC_ResultGetByIndex(ADC_ID_1, 9);
77      }
78
79      // Auto start sampling
80      PLIB_ADC_SampleAutoStartEnable(ADC_ID_1);
81  }
82
83
84  //------------------------------------------------------------------------------// convertRawToVoltage
85  void convertRawToVoltage(RAW_ADC *pRawAdc, SENS_DATA *pSensData){
86
87      // Converts RAW data of the battery voltage into a real decimal value
88      // The number 2 is present because of the hardware bridge divider
89      pSensData->batVoltage = (2 *(pRawAdc->AN10_V_BAT *
90              (ADC_VREF / ADC_RESOLUTION)));
91
92      // Converts RAW data of the generator voltage into a real decimal value
93      // The number 5 is present because of the hardware bridge divider
94      pSensData->genVoltage = (5 *(pRawAdc->AN9_V_GEN *
95              (ADC_VREF / ADC_RESOLUTION)));
96  }
```