

ADC - Analog to Digital Converter

June 3rd, 2014



This is a graded exercise. For further information regarding the evaluation criteria read the document *gradedExercises.pdf* carefully. The deadline will be 22.06.2014 23:59 (CEST). The interviews will be held on Tuesday / Friday after submission.

Send your solution as ZIP file to the address florian.meier@tuhh.de using as subject *SES TASK 6, Name1 (Matriculation number), Name2 (Matriculation number)*. The ZIP file **shall** contain the following directories:

- `ses` (contains your current ses library source/header files, including the new ADC driver)
- `task_6_2to3` (containing source/header files for task 6.2 and optionally challenge task 6.3)

Ignoring this directory structure will lead to point deductions. Make sure that the ZIP file only contains **header**, **source** and **text** files (object-files in ZIP archives may not pass the spam filter). Additionally, make sure the included files can be compiled and linked.

The Analog-to-Digital Converter converts an analog input voltage to a digital value. The ATmega128RFA1 features an ADC with 10 bit resolution, yielding 1024 distinct digital values which correspond to analog voltages¹. The ADC is connected via a multiplexer to one of 8 external channels and a few internal channels. In the SES board, two channels are used for reading a temperature sensor and a light sensor. Two channels are used as differential channel for the microphone². One channel is used for the joystick. All components output an analog voltage between 0 V and 1.6 V.

More information is provided in the Atmel ATmega 128RFA1 datasheet in chapter 27. *ADC - Analog to Digital Converter*. Template files `ses_adc.c` and `ses_adc.h` are available in the exercise zip file in StudIP.

Task 6.1 : ADC (Initialization and reading)

Write an abstraction layer for the ADC of the board according to the following description!

The initialization of the ADC should be done in

```
void adc_init(void)
```

- Configure the data direction registers (temperature, light, microphone, joystick) and deactivate their internal pull-up resistors.
- Disable power reduction mode for the ADC module. This is done by clearing the `PRADC` bit in register `PRR0`.
- To define the reference voltage used, set the macro `ADC_VREF_SRC` in the corresponding header file. The reference voltage should be set to 1.6 V.

¹In this exercise, a reference voltage of 1.6 V is used, thus the ADC output corresponds to analog voltages from the interval $[0 \text{ V}, 1.6 \text{ V} - \frac{1.6 \text{ V}}{1024}]$

²The microphone is not used in this exercise, however

- Use this `ADC_VREF_SRC` macro to configure the voltage reference in register `ADMUX`.
- Configure the `ADLAR` bit, which should set the ADC result right adjusted.
- The ADC operates on its own clock, which is derived from the CPU clock via a prescaler. It must be 330 kHz or less for full 10 bit resolution. Find an appropriate prescaler setting, which sets the operation within this range at fastest possible operation! Set the prescaler in the macro `ADC_PRESCALE` in the corresponding header file.
- Use the macro `ADC_PRESCALE` to configure the prescaler in register `ADCSRA`.
- Do not select auto triggering (`ADATE`).
- Enable the ADC interrupt (`ADIE`). The flag is stored in `ADIF`.
- Enable the ADC (`ADEN`).

The ADC can be deactivated for power saving:

```
void adc_disable(void)
```

Write this function to allow deactivation of the ADC and entering of the power reduction mode for the ADC module. For simplicity reasons, you do not have to implement an explicit enable function. `adc_init` can be used to re-enable the ADC.

The ADC should be ready for conversion now. The following function triggers the conversion:

```
uint16_t adc_read(uint8_t adc_channel)
```

`adc_channel` should contain the sensor type (`enum ADChannels`). By now the ADC is configured in a run once mode, which will be started for the configured channel in `ADMUX`. If an invalid channel is provided as parameter, `ADC_INVALID_CHANNEL` should be returned.

A conversion can be started by setting `ADSC` in `ADCSRA`. The conversion will start in parallel to the program execution. When the conversion is finished, the microcontroller hardware will reset the `ADSC` bit to zero. With ADC interrupts enabled, additionally the `ISR(ADC_vect)` will be called upon completion of a conversion.

The result is readable from the 16 bit register `ADC`, which is automatically combined from the 8 bit registers `ADCL` and `ADCH`.

Use the noise cancellation mode of the ADC to improve the accuracy of the conversion. For this, a special sleep mode can be used, which automatically starts a conversion as soon as the CPU sleeps. The CPU is wakened again by the ADC interrupt. As, theoretically, also some other interrupts can wake the CPU from this sleep mode, make sure to check if the conversion is really finished (and if not, go to sleep again), before returning the measured value.

For using sleep functionality these functions will be helpful:

```
#include <avr/sleep.h>

set_sleep_mode(SLEEP_MODE_ADC); // use this sleep mode to reduce ADC noise
sleep_enable();                 // enable sleep capability
sleep_disable();                 // disable sleep capability
sleep_cpu();                     // go to sleep (and trigger ADC conversion)
```

- For testing purposes you can stop program flow and use polling to check the `ADSC` bit.
- The `adc_read` function can be implemented as a blocking function (the CPU sleeps anyway).
- When the ADC noise reduction sleep mode is set, `sleep_cpu()` will also trigger a conversion.
- If you use `ADCL` and `ADCH` to read out the ADC, make sure that you **always** read from `ADCH` after you have read from `ADCL`, otherwise the data register will be blocked and conversion results are lost.

Task 6.2 : Using the ADC

Display the current values of the light sensor, the direction of the joystick and the current temperature (in °C) on the LCD (e.g., each in a separate row). Concerning the light sensor, you may directly use the raw ADC values.

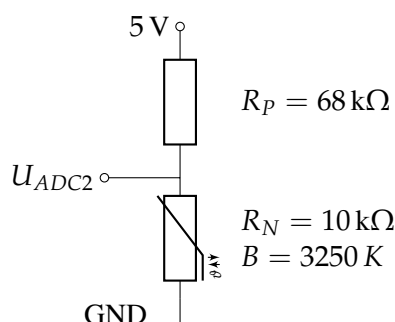
The joystick on the SES Board is a 4-axis joystick with a button. The button press is evaluated like in the previous task using the pin change interrupt. The four directions are not connected to any interrupt channel, but to a single ADC channel (`ADC_JOYSTICK_CH`).

The RAW value of the ADC corresponds to the following directions:

- 200 ⇒ Right
- 400 ⇒ Up
- 600 ⇒ Left
- 800 ⇒ Down
- 1000 ⇒ No direction

The RAW values are never exact, therefore choose an error tolerant mapping. Set a task in the scheduler, which polls the joystick every 100 ms and outputs the current direction as text string on the LCD! You can extend the file `adc_test.c` from the previous task.

For displaying the current temperature on the LCD, the ADC values have to be converted to °C. The component used for temperature measurement is an NTC thermistor with a temperature dependent resistance R_T . It is connected to the ADC as given in the circuit diagram below.



According to the voltage divider rule, the voltage U_{ADC2} at the ADC pin 2 is


$$U_{ADC2} = 5 \text{ V} \cdot \frac{R_T}{R_T + R_P}.$$

Thereby, the current resistance of the thermistor can be calculated and used to get the current temperature in Kelvin as given by

$$T = \frac{B \cdot 298.15 \text{ K}}{B + \ln\left(\frac{R_T}{R_N}\right) \cdot 298.15 \text{ K}},$$

with B and R_N being characteristics of the thermistor as given in the schematic. Your task is to provide a function that converts a raw ADC value to the corresponding temperature:

```
int16_t adc_convertTemp(uint16_t val)
```

 Note that the unit of the return value `int16_t` should be $\frac{1}{10}^\circ\text{C}$, i.e., a value of 10 represents 1.0°C .

However, since the computational power of the ATmega128RFA1 is limited and floating point operations are quite expensive, a lookup table with linear interpolation is to be implemented instead of using the above formulas directly. At first, write a small program in a programming language of your choice for building up a table that maps from raw ADC values to temperatures in $^\circ\text{C}$. Use this table to implement the `adc_convertTemp` function. Find a good compromise between memory consumption and accuracy. Higher accuracy with a smaller number of elements in the table can be achieved by doing a linear interpolation when `val` is in between two values in the table.

Task 6.3 : Datalogger with Wear-Leveling (Challenge)

For datalogging the measurements must be stored persistently. In this exercise you will use the external EEPROM, which is integrated in the microcontroller package. The EEPROM is connected to the microcontroller with an TWI (I²C) interface. Fortunately, the library using TWI to access the EEPROM is already present (*libses_eeprom.a*). Add the library to the linked libraries of your project as described in the previous exercises and read the file *ses_eeprom.h* for more information about the interface (after having added it to your existing library).

The task is to store the values for temperature and light in the EEPROM in a ringbuffer structure. The size and memory position are given (*datalogger.h*). Optimize your storage structure to save as many measurement values as possible. The measurement frequency is 1 measurement per second, but only every 5 seconds the median (**not** average) of the last measurement values is stored in the EEPROM. The recent median values and the current values for temperature and light have to be displayed on the LCD. It also has to be possible to read the last n values from the storage and, e.g., print them to UART.

The challenge is to make the datalogger fail-proof. In case of a sudden reset, the datalogger must be able to read the already stored values (and display the last median on the LCD before five more seconds have passed). Comment your code and explain how your solution works. No predefined C functions are given for this task, therefore try to declare a meaningful interface for using your datalogger!

 **To reduce EEPROM wear-out all write accesses must be shared equally among all cells! E.g. storing a structure pointer in a single EEPROM cell is not allowed!**