

# Buttons and the ADC

April 26<sup>th</sup>, 2016



This exercise is not graded and has to be finished within two weeks! Even though this exercise is not graded, the created library is part of the later submission and the grading, so coding style is important! During the last exercise we learned to use the LEDs, the UART and the LCD for output. In this exercise you will learn about input, that is reading digital and analog signals. First, create a new project that has to end up at

[https://svn.ti5.tu-harburg.de/courses/ses/2016/teamX/task\\_3](https://svn.ti5.tu-harburg.de/courses/ses/2016/teamX/task_3) where X is your team's number.

The new project has to know the location of the header files of your library and also needs to link to the compiled library as described in the previous exercise. Furthermore, download and extract the *sheet3\_templates.zip* archive from the Stud.IP page to the project of your ses library.

## Task 3.1 : Buttons

The states of the two buttons on the SES board can be read as digital signals. The button of the rotary encoder is located on the lower right of the module and is connected to pin 6 of port B. The joystick button is located on the lower left and is connected to pin 7 of port B.

- In the library project, create a *ses\_button.c* and provide the implementation of the functions declared in the *ses\_button.h*:
- Create macros for the button wiring analogous to the ones in the *ses\_led.c*.

```
void button_init(void)
```

- Configure the pin of each button as an input (write a 0 to the corresponding bit of the data direction register (DDRB))
- Activate the internal pull-up resistor for each of the buttons<sup>1</sup> (write a 1 to the corresponding bits of the buttons' port PORTB)

```
bool button_isJoystickPressed(void)
bool button_isRotaryPressed(void)
```

- When the button is pressed, the corresponding pin is grounded and the input can be read as logic low level via PINB
- In the task\_3 project, create a main.c with a loop that polls the state of the two buttons. While the rotary button is pressed, light up the red LED. While the joystick button is pressed, light up the green LED. Reuse your LED driver from the previous exercise!



When setting or clearing bits in any register, be sure to leave all other bits unchanged!



You can use the `PIN_REGISTER(x)` and `DDR_REGISTER(x)` macro from *ses\_common.h* to calculate the pin and data direction register for a given port x. These macros help to avoid redundancy.

<sup>1</sup>A pull-up resistor ensures a logic high level in the electric circuit, when a high-impedance device like an open switch or button is connected.

## Analog to Digital Converter

The Analog-to-Digital Converter converts an analog input voltage to a digital value. The ATmega128RFA1 features an ADC with 10 bit resolution, yielding  $2^{10}$  distinct digital values which correspond to analog voltages<sup>2</sup>. The ADC is connected via a multiplexer to one of 8 external channels and a few internal channels. On the SES board, the following peripherals are connected to the ADC:

- A temperature sensor at pin 2 of port F
- A light sensor at pin 4 of port F
- The joystick at pin 5 of port F
- A microphone connected differentially to the pins 0 and 1 of port F<sup>3</sup>

All components output an analog voltage between 0 V and 1.6 V. More information is provided in the ATmega128RFA1 datasheet in chapter 27. *ADC - Analog to Digital Converter*.

### Task 3.2 : 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.
- 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 (ADEN).

<sup>2</sup>In this exercise, a reference voltage of 1.6 V is used, thus the ADC output corresponds to analog voltages from the interval  $[0 \text{ V}, 1.6 \text{ V} - \frac{1.6 \text{ V}}{2^{10}}]$

<sup>3</sup>The microphone is not used in this exercise, however

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` shall 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. So you can stop the program flow and use polling to check the `ADSC` bit.<sup>4</sup> The result is readable from the 16 bit register `ADC`, which is automatically combined from the 8 bit registers `ADCL` and `ADCH`.



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 3.3 : ADC Peripheral Abstractions

Usually the raw ADC value is not very useful, instead provide abstractions to calculate the corresponding physical measure for the joystick and the temperature sensor. For the light sensor, use the raw value. For testing purposes, write the results to the UART or the LCD.

```
uint8_t adc_getJoystickDirection(void)
```

The joystick on the SES Board is a 4-axis joystick with a button. The button press is evaluated as done before. The four directions are not connected to any digital 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

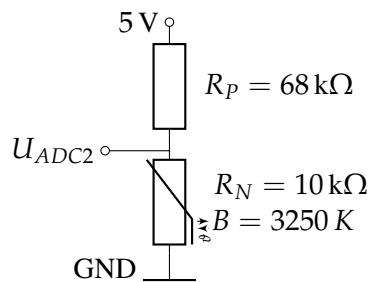


The RAW values are never exact, therefore choose an error tolerant mapping.

```
int16_t adc_getTemperature(void)
```

For the temperature sensor, 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 following circuit diagram.

<sup>4</sup>A better possibility would be to use interrupts to avoid stopping the program flow or to put the controller in noise reduction mode. The concept of interrupts will be introduced in the next exercise.



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 the raw ADC value to the corresponding temperature**. The unit of the return value `int16_t` shall be  $\frac{1}{10}^\circ\text{C}$ , i.e., a value of 10 represents  $1.0^\circ\text{C}$ .

However, since the computational power of the ATmega128RFA1 is limited and floating point operations are quite expensive, linear interpolation should be used instead of applying above formulas directly. For this, calculate the raw ADC values for two temperature values (e.g.  $40^\circ\text{C}$  and  $20^\circ\text{C}$ ) and use the following function to calculate interim values. **A suitable `ADC_TEMP_FACTOR` is necessary to avoid rounding the slope to zero.**

```
int16_t adc = adc_read(ADC_TEMP_CH);
int16_t slope = (ADC_TEMP_MAX - ADC_TEMP_MIN) / (ADC_TEMP_RAW_MAX - ADC_TEMP_RAW_MIN);
int16_t offset = ADC_TEMP_MAX - (ADC_TEMP_RAW_MAX * slope);
return (adc * slope + offset) / ADC_TEMP_FACTOR;
```

### Task 3.4 : Superloop

Extend the `main.c` from the `task_3` project so that it conducts the following tasks:


- While the rotary button is pressed, light up the red LED
- While the joystick button is pressed, light up the green LED
- While the joystick is in the left position, light up the yellow LED
- Show the seconds since reset on the LCD
- Every 2500 milliseconds, read the values of the temperature and the light and show it on the LCD




This subtask is mainly meant to illustrate the pain of superloops. Unless you provide a very clever implementation, the buttons will show bad responsiveness and/or the timings are not accurate<sup>5</sup>. Do not bother yourself about this too much, you will learn about a much better approach in the next exercise.

<sup>5</sup>For example, the ADC conversion takes a certain, even variable, time that is difficult to account for.

### Task 3.5 : Lookup Table (optional)

-  If you have no time left to finish this subtask during the current two week, please skip it to catch up in the following week.

As you might guess, the above temperature conversion is not very accurate and is only useful for a narrow temperature range. In this task you should increase the number of supporting points, store them in a lookup table (LUT) and interpolate between the points. Write a small program in a programming language of your choice to create a LUT that maps from raw ADC values to temperatures in °C and use the result in your program code.

-  The supported range of temperatures shall be  $[0, 80]^{\circ}\text{C}$  and you should use the minimum possible number of supporting points that achieves an accuracy of  $0.1^{\circ}\text{C}$ .