Université Paul Sabatier TOULOUSE III

# Lab Work

***Architecture, System, Network***

---

# Lab Work 3 : Using the ADC

Sources for this labwork are in directory ASN/lab3.

## *1 ADC by Polling*

The ADC, Analog-Digital Converter, provides a conversion from an input voltage, between 0V and 5V, to a digital number on 10-bit, 0 for 0V, 1023 for 5V.

The ADC of the AT91SAM7S has 8 input channels and implements the successive approximation method. This means that it has to be powered by the MCK and configured (a) to select which channel to convert and (b) at which rate the conversion has to be done.

On our board, ADC0, channel 0 of the ADC, is connected to a manual sensor that may be configured by hand to represent a temperature between 0°C and 100°C using a UI slider component (if required, scroll down the board view to the bottom to see it). According to the position of the slider handle, the temperature is converted to a voltage by the manual sensor. From this voltage, the program can find back the corresponding temperature.

First, the ADC has to be configured:

```
ADC_MR = ADC_PRESCAL(0x3f);
ADC_CHER = 1 << THM;
```

The main configuration is contained in `ADC_MR`: `ADC_PRESCAL` represents the rate of the conversion. A too fast rate can lead to increase the error of the measure. As the temperature changes very slowly, we select the slower conversion time that is MCK / (0x3f + 1) / 2 = MCK / 128. Then we select with `ADC_CHER` which channels to sample: there is one bit for each input channel from 0 to 7. In our program, the constant `THM` is equal to 0 and corresponds to `ADC0`.

```
ADC_CR = ADC_START;
```

This command starts the conversion of each enabled channel of `ADC_CHER`. In our case, this only concerns the channel 0.

```
while(!(ADC_SR & ADC_EOC(THM)));
```

Then we have to wait until the conversion is ended, that is, when bit `ADC_EOC(THM)` of register `ADC_SR`, pass to 1.

```
v = ADC_CDR0;
```

This final command read the value obtained from channel 0 stored in register `ADC_CDR0`. This value is in the range [0, 1023] and has to be converted to a temperature in the range [0°C, 100°C]. As our hardware is not very powerful, I advise you to use integer operations and multiply the value by 100 to mimic a fractional part with a precision of 1/100 (that is

0,01°C). So we will get a temperature in the range [0, 10 000]. In the same way, we cannot use standard input/output operations from the standard C library. Therefore, you have to implement your own conversion to string of the temperature to display on the terminal.

**TO DO** Write a program which main loop ask a conversion to the ADC on channel 0, wait for it, convert it the number to a temperature in 1/100°C and display it using the USART on the terminal.

## *2 ADC by Interrupt with Timer*

As most of the peripheral controllers of the AT91SAM7S, the ADC can be driven using interrupts. As usual, you have to program:

- the ADC itself by selecting in ADC_IER which channels will cause an interrupt (one bit for each channel),
- the AIC to support interrupts coming from the ADC (interrupt number ID_ADC),
- the microcontroller to unmask the interrupts.

In the interrupt handler, as usual, you have to:

- read the status register of ADC, ADC_SR, to acknowledge for the interrupt;
- read the register EOICR to acknowledge interrupt on the microcontroller.

To get the ADC value on more regular basis, that is, with a more regular time between each sample, one can use the timer to start regularly the ADC.

**TO DO** Write a program that display the temperature coming from ADC0 using interrupts from TC0, 4 times per second, and from ADC to use the sampled value.

## *3 PIO + ADC + Timer (optional)*

Now, we want to simulate a small application implementing an electrical heater. It works as described below:

- it uses a reference temperature *r* (initialized to 20°C),
- 4 times per second, it compares the current temperature *y* with the reference temperature *r*: if $r - 1 > y$, it switches on the heater; if $r + 1 < y$, it switches off the header.[1]
- a click on PUSH1 increases the reference temperature by 1°C,
- a click on PUSH2 decreases the reference temperature by 1°C.

As we do not have a heater in our simulation, you will only realize the heater work by switching on and off the YELLOW LED.

1 The small gap [-1°C, +1°C] around the reference temperature prevent from switching on/off too often the heater and from causing damages to the heater.