POLITECNICO
DI MILANO

www.polimi.it

# ADC

Federica Villa

# ADC features (1/4)

- 12 bit ADC (4096 levels) converted data are stored in the 16-bit ADC_DR register

  Configurable: 12, 10, 8, 6 bit

  converted data are stored in the 16-bit ADC_DR / ADC_JDRx register

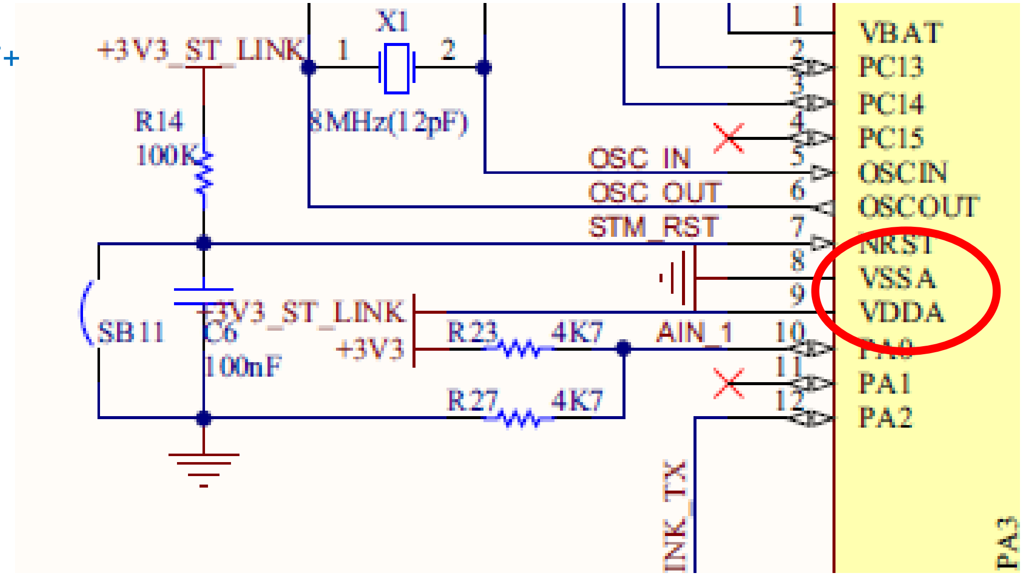
- One S&H and one SAR ADC

- ADC input range: $V_{REF-} \leq V_{IN} \leq V_{REF+}$

  On NUCLEO board:

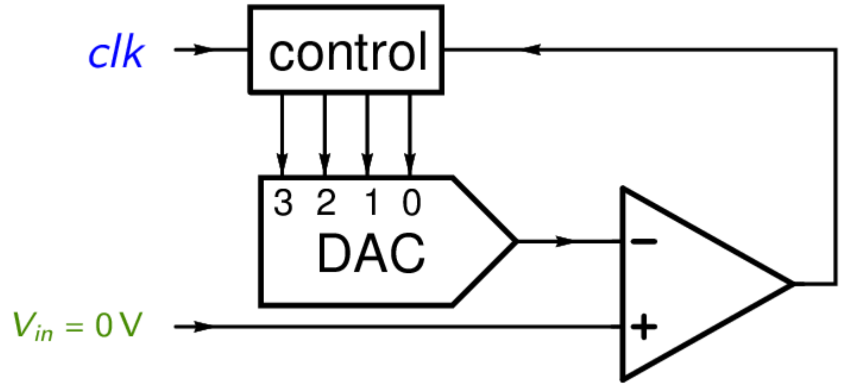
  $V_{REF-} = V_{SSA} = GND$

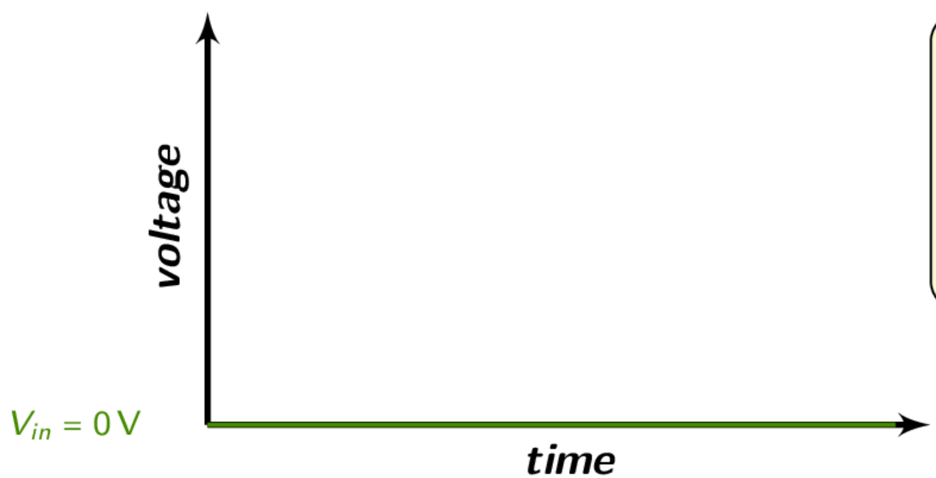
  $V_{REF+} = V_{DDA} = 3.3\ V$



# SAR ADC

## Successive Approximation – example of a 4-bit ADC




Resolution:

| | |
|---|---|
| $5\,\mathrm{V} \times 1/2$ | $2.5000\,\mathrm{V}$ |
| $5\,\mathrm{V} \times 1/4$ | $1.2500\,\mathrm{V}$ |
| $5\,\mathrm{V} \times 1/8$ | $0.6250\,\mathrm{V}$ |
| $5\,\mathrm{V} \times 1/16$ | $0.3125\,\mathrm{V}$ |
| ... | |
| $5\,\mathrm{V} \times 1/1024$ | $0.0049\,\mathrm{V}$ |

# ADC features (2/4)

- 16 external input channels (all routed on the NUCLEO board)

  2 internal channels:

  - $V_{REFINT}$ (generic reference voltage $V_{REFINT}$=1.21 V)
  - temperature sensor *or* battery charge monitor (check VBAT pin)

- Conversion can be started by:
  - software
  - timer (TRGO, Capture and Compare)
  - external trigger (EXTI_11 or EXTI_15)

- Execution conversion modes:
  - polling (checking the EoC flag)
  - interrupt (ADC generates the interrupts at the EoC)
  - DMA (converted values are directly sent to the memory)

# ADC features (3/4)

- Continuous / single acquisition:
  - continuous: the SoC is automatically generated at the maximum sampling rate (i.e., ADC starts a new conversion as soon as it finishes one)
  - single acquisition: each SoC is provided by software / TIM / external trigger

- Single channel / scanning mode (with a specific sequence):
  - up to 16 regular channels
  - up to 4 injected channels (higher priority and dedicated triggers)

  Autoinjection to scan 20 channels

  Normally all the channels are converted in sequence continuously. The EoC can be generated for each channel or at the end of the sequence.

  In discontinuous mode a shorter sequence can be converted at each trigger.

- Right / left alignment

| | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | D11 | D10 | D9 | D8 | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 | right |
| D11 | D10 | D9 | D8 | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 | 0 | 0 | 0 | 0 | left |

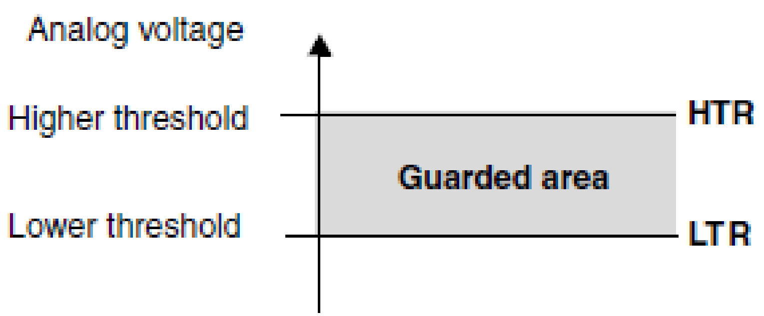
- Channel-wise programmable sampling time

  Maximum clock frequency: 36 MHz (APB2 and clock prescaler)

  Sampling time (ST) expressed in clock cycles

$$f_{sampling} = \frac{f_{ADCclock}}{(ST+12)}$$ (maximum sampling frequency = 2.4 MHz, 12-bit)

- Analog watchdog



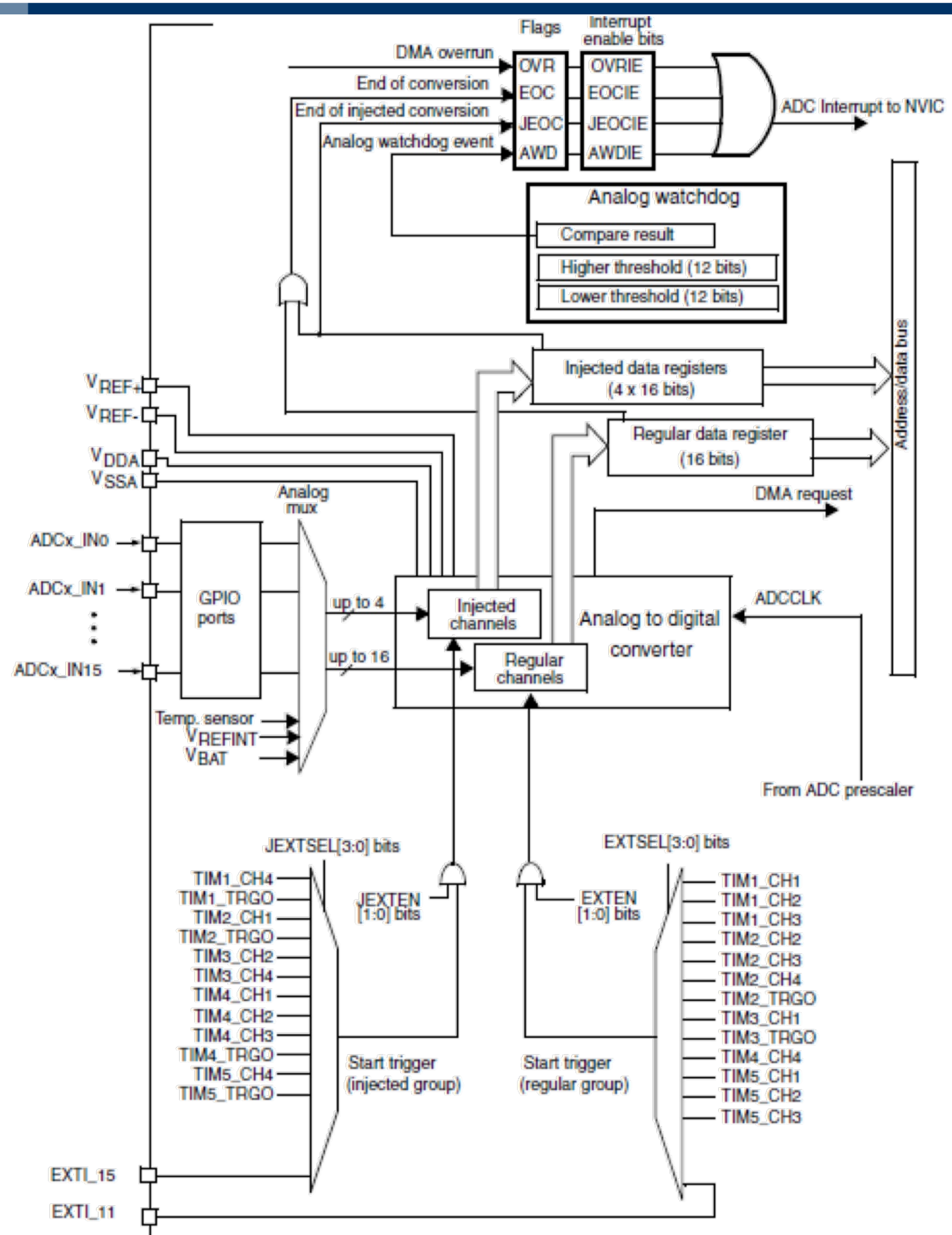
# ADC and DMA

- DMA request can be enabled: values converted by ADC are directly sent to the SRAM without occupying the CPU resources

- destination in the SRAM and number of conversions are configured by software

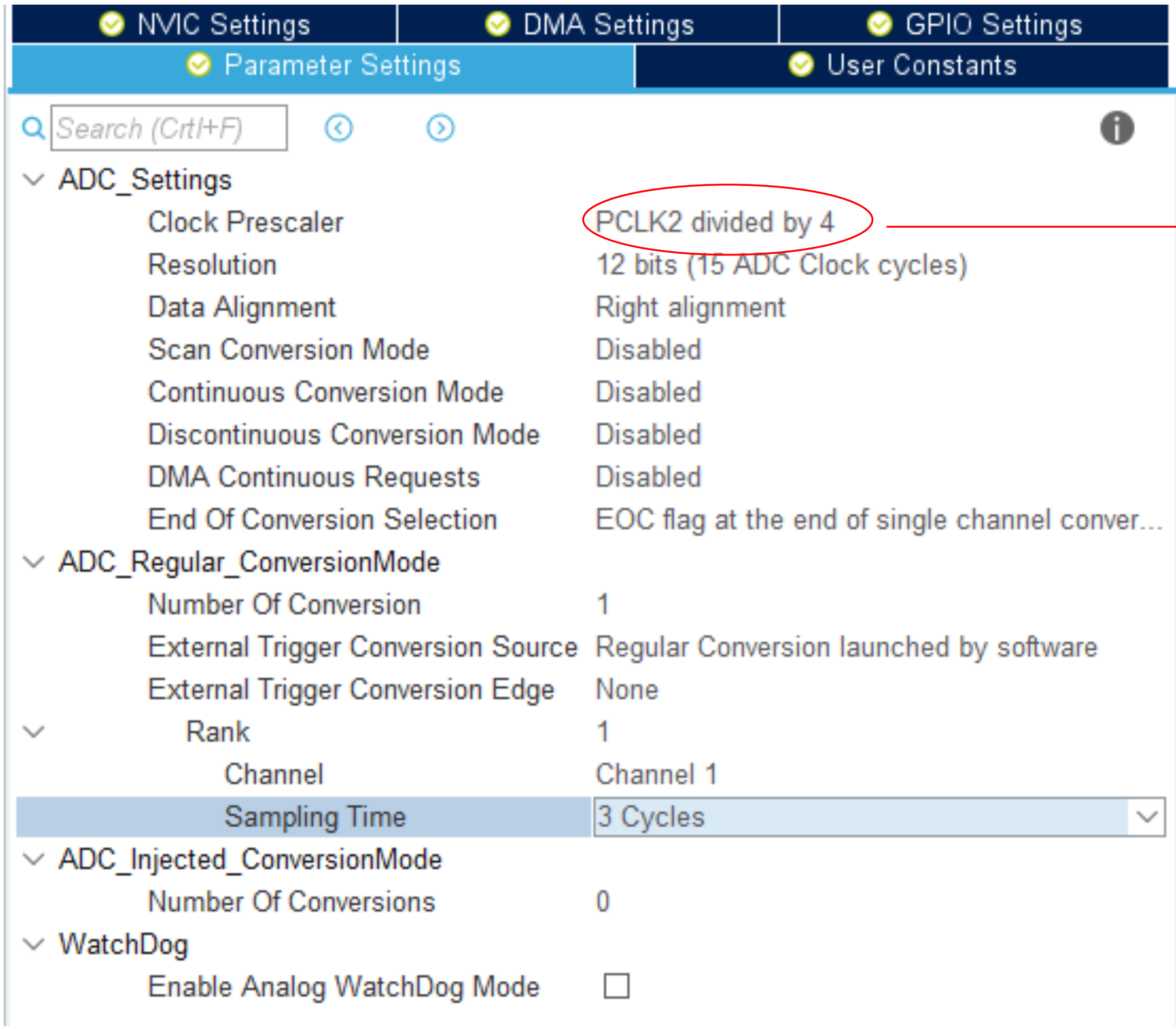
- Interrupts are generated when either half or all the buffer is filled.

# ADC block diagram




RM0368 pag. 213/841

# CUBE configurations




$f_{ADC}$= 84 MHz /4 = 21 MHz

# ADC HAL functions

There are many HAL functions for ADC.

In the next projects we will use the following functions:

HAL_StatusTypeDef **HAL_ADC_Start**(ADC_HandleTypeDef* hadc)

HAL_StatusTypeDef **HAL_ADC_Start_IT**(ADC_HandleTypeDef* hadc) (needed at the beginning of the code also just to setup the peripheral)

HAL_StatusTypeDef **HAL_ADC_Start_DMA**(ADC_HandleTypeDef* hadc, uint32_t* pData, uint32_t Length)

HAL_StatusTypeDef **HAL_ADC_Stop**(ADC_HandleTypeDef* hadc) (simply aborts the convertion)

HAL_StatusTypeDef **HAL_ADC_Stop_IT**(ADC_HandleTypeDef* hadc)

HAL_StatusTypeDef **HAL_ADC_Stop_DMA**(ADC_HandleTypeDef* hadc)

HAL_StatusTypeDef **HAL_ADC_PollForConversion**(ADC_HandleTypeDef* hadc, uint32_t Timeout)

uint32_t **HAL_ADC_GetValue**(ADC_HandleTypeDef* hadc)

__weak void **HAL_ADC_ConvCpltCallback**(ADC_HandleTypeDef* hadc)

__weak void **HAL_ADC_ConvHalfCpltCallback**(ADC_HandleTypeDef* hadc)

# Project 1: ADC started by software

The objective of the project is to acquire the voltage of the potentiometer on the POLIMI board, starting the conversion by software and then sending the value to the PC on a remote terminal.

- We will start with polling mode, single acquisition (Project 1)
- Finally, we will use the interrupt mode (Project 2a)

The first step is determining which microcontroller pin is connected to the potentiometer.

NOTE: Project 1 is useful for understanding the ADC working mode, but we are not using efficiently the microcontroller resources.

# Project 1: ADC single acquisition - polling

Objective of this project is
to **acquire the voltage of the potentiometer every 1 second**
and send this value to a remote terminal.

The ADC will be used in polling mode.

# Project hints - 1

1. Set the GPIO connected to the potentiometer as an analog input and configure the ADC to acquire one value from that channel, triggered by software. Set the sampling time to 480 clock cycles.

2. Generate the c code.

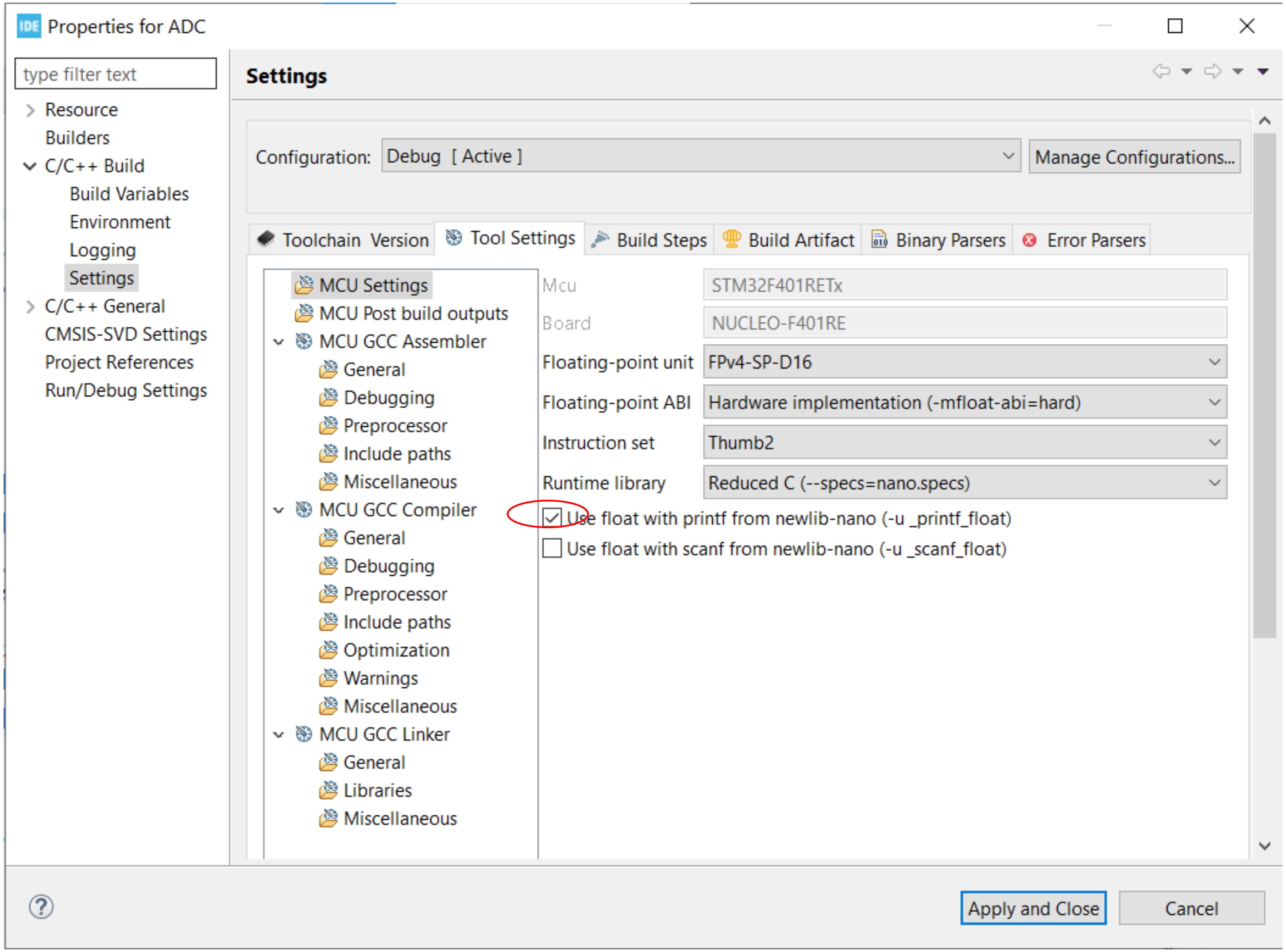
3. Modify the code to acquire a value every 1 second, and debug using the watch variable.

4. Convert the ADC value into a voltage.

5. Send the value to the remote terminal.

6. Debug the project.

# Project hints - 2




1. Float formatting is not enabled by default to save resources. Tick the box in Project -> Settings to enable float in printf.

# Project 2a: ADC single acq. - interrupt

Objective of this project is
to **acquire the voltage of the potentiometer** and send this value to a remote terminal every 1 s.

The ADC will be used in interrupt mode.

# Project hints

1. Modify the previous project: in CUBE enable ADC interrupt.
2. Generate the c code.
3. Modify the code to get the value from the ADC data register, and debug using the watch variable. Remember to use the interrupt routine.
4. Convert the ADC value into a voltage
5. Send the value to the remote terminal.
6. Debug the project.

# Project 2b: ADC triggered by TIM

Objective of the project is to

**acquire the potentiometer voltage**

using a timer to trigger a conversion at a

**regular conversion rate of 1 Hz**

and sending the value to a remote terminal

# Project hints

1. In CUBE enable the potentiometer analog input channel and configure ADC in interrupt mode. Configure ADC to be triggered by a timer.

2. Configure the TIM to generate a trigger at the update event every 1 s.

3. Generate the c code.

4. Modify the code to get a value from the ADC data register every 1 s using the timer, and debug using the watch variable. Remember to use the ADC interrupt routine.

5. Convert the ADC value into a voltage send the value to the remote terminal.

6. Debug the project.

# Project 2c: ADC triggered by TIM to LCD

Objective of the project is to

**acquire the potentiometer voltage**

using a timer to trigger a conversion at a

**regular conversion rate of 1 Hz**

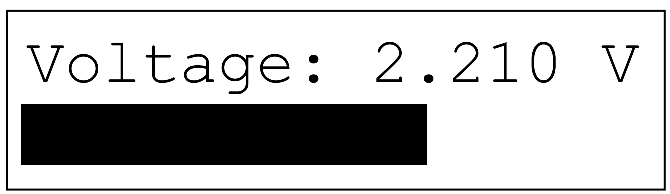
and showing the value on the LCD

# Project hints

1. Start from project 3b, but increase conversion rate to 5 Hz..

2. Enable and initialize the LCD (see slides M 07).

3. Show the voltage on the LCD top row (three decimal digits), and a bar graph in the bottom row (empty row when the voltage is zero, full row when the voltage is 3.3 V), as in the example below.
   Update the LCD every time a new conversion of the ADC is performed.



# Project 3a: ADC scan using DMA

Objective of the project is to acquire
**3 voltages (potentiometer, temperature sensor, Vref)**
every 1 s and to send them to a remote terminal.

The acquisition are started by software
and data are saved in the microcontroller memory
using DMA.

# Internal temperature sensor

- Supported temperature range: –40 to 125 °C
- Precision: ±1.5 °C
- Low accuracy (the internal temperature sensor is more suited for applications that detect temperature variations instead of absolute temperatures).

$$Temperature(in\,°C) = \frac{V_{sense} - V_{25}}{Avg\_Slope} + 25$$

**Table 72. Temperature sensor characteristics**

| Symbol | Parameter | Min | Typ | Max | Unit |
|---|---|---|---|---|---|
| $T_L$ (1) | $V_{SENSE}$ linearity with temperature | - | ±1 | ±2 | °C |
| Avg_Slope(1) | Average slope | - | 2.5 | - | mV/°C |
| $V_{25}$ (1) | Voltage at 25 °C | - | 0.76 | - | V |
| $t_{START}$ (2) | Startup time | - | 6 | 10 | µs |
| $T_{S\_temp}$ (2) | ADC sampling time when reading the temperature (1 °C accuracy) | 10 | - | - | µs |

1. Guaranteed by characterization, not tested in production.
2. Guaranteed by design, not tested in production.

# Project hints

1. In CUBE enable the 3 correct ADC channels and configure the ADC enabling the DMA in circular mode with continuous DMA requests.
2. Generate the c code.
3. Modify the code to acquire the 3 values every 1 s, and debug using the watch variable.
4. Convert the ADC value into either a voltage or a temperature.
5. Send the value to the remote terminal.
6. Debug the project.

# Project 3b: Light Dependent Resistor

Objective of the project is to acquire
**LDR resistance value**
every ms and to send its average value to a remote terminal every 1s.

Step 2: convert the resistance value to a lux level and send that to the remote terminal.

# Light Dependent Resistor (LDR)

In the dark → R very high (up to 1MΩ)

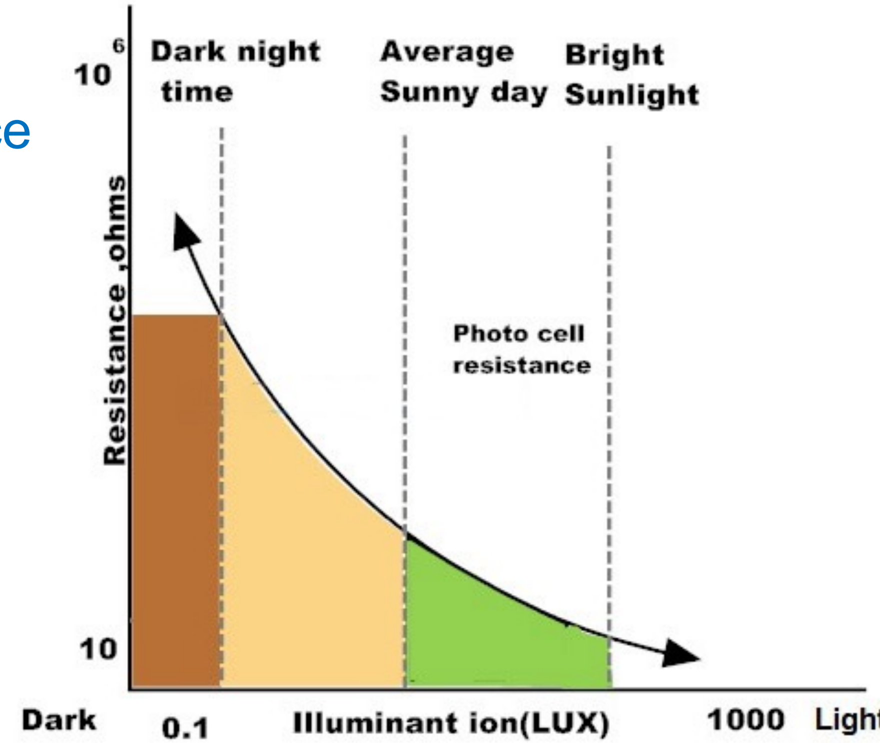
Exposed to light → R drops dramatically (few ohms)

## Working principle:

Photons excite electrons from the valence band to the conduction band

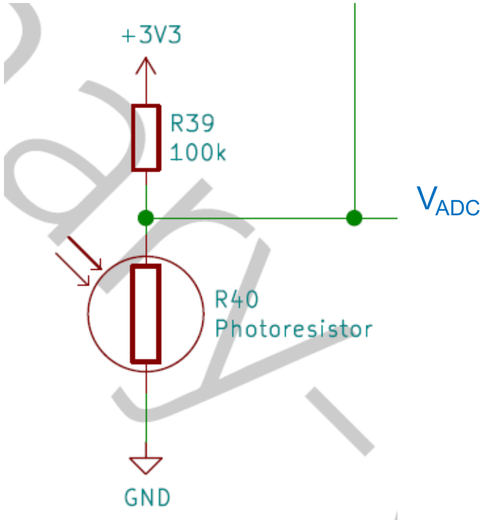
→ more free electrons in the material

→ lower resistance




But…
- low sensitivity
- non-linear characteristic
- sensitive to temperature changes

# Volt to kΩ to lux conversion

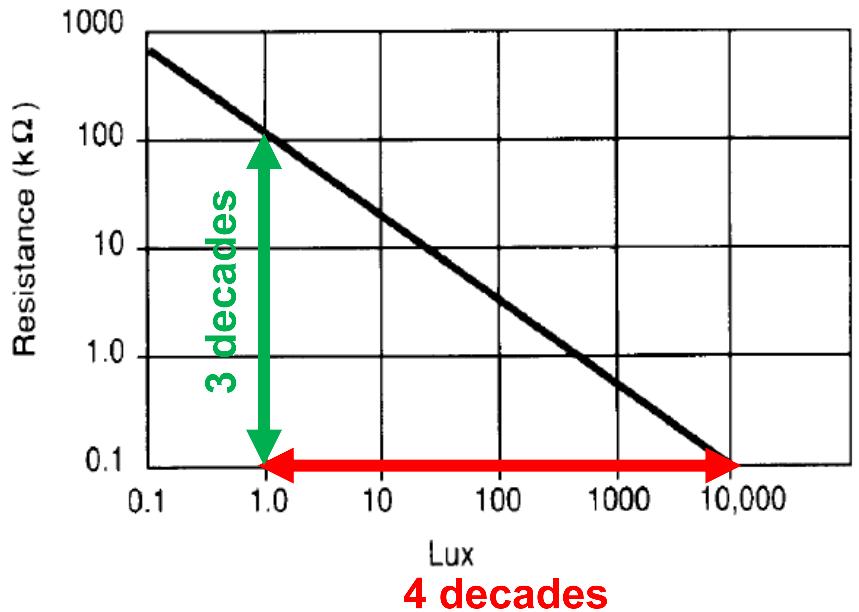



$$V_{ADC} = 3.3\ \text{V} \times (LDR)/(LDR + 100\ \text{k}\Omega)$$

$$LDR = (V_{ADC} \times 100\ \text{k}\Omega)/(3.3\ \text{V} - V_{ADC})$$

$$LDR \simeq 100\ \text{k}\Omega/((LUX/10)^{0.8})$$

$$LUX \simeq 10\times(100\ \text{k}\Omega/LDR)^{1.25}$$



# Project hints

1. In CUBE enable the correct ADC channel and configure the ADC enabling the DMA in circular mode with continuous DMA requests.

2. Choose a timer trigger out as start of conversion of the ADC. Configure that timer.

3. Enable the required interrupts in the NVIC.

4. Generate the c code.

5. Debug the project.