LAB 05
USART & ADC

UNIVERSITY
OF TRENTO

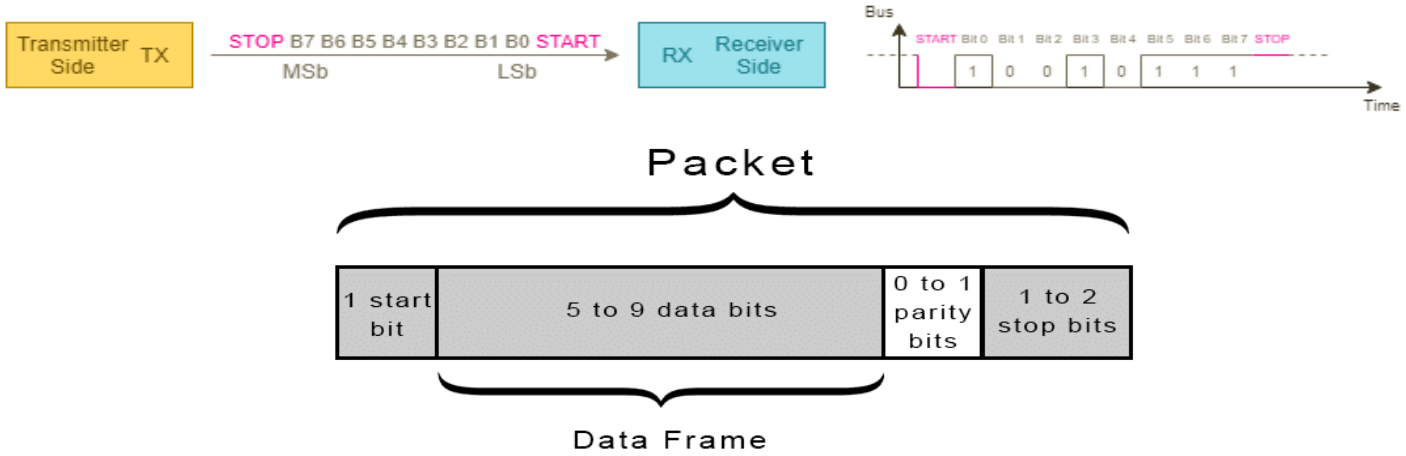
Prof. Davide Brunelli
Dept. of Industrial Engineering – DII
University of Trento, Italy
davide.brunelli@unitn.it

USART

STM32 USART

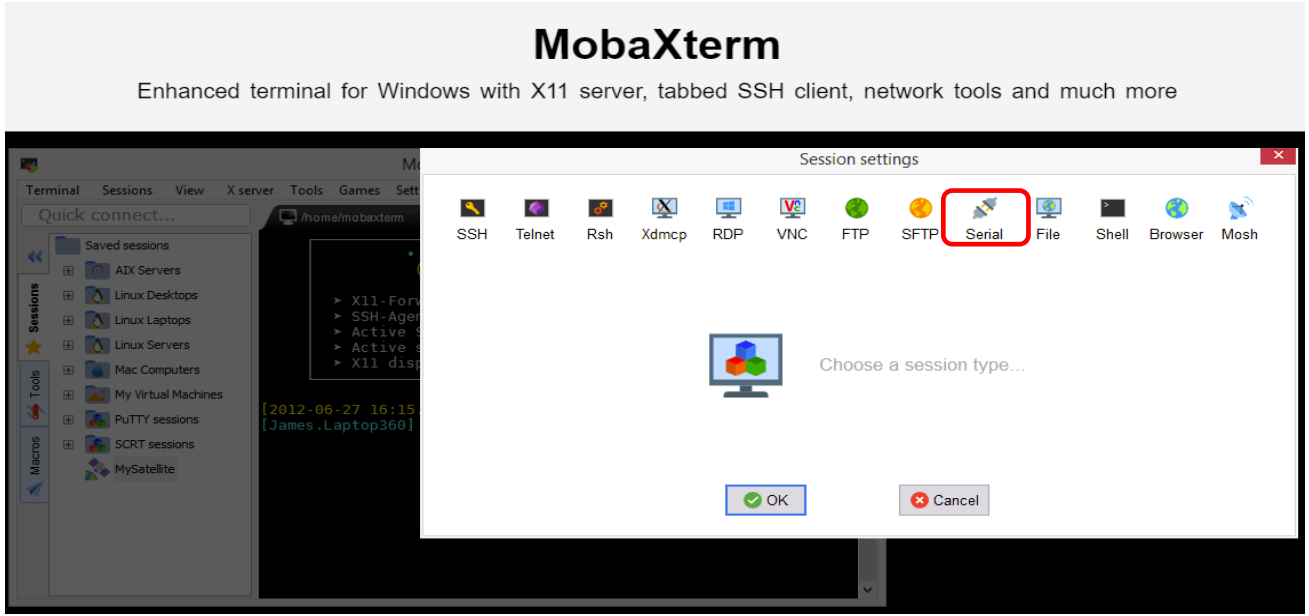
Universal synchronous and asynchronous receiver-transmitter (**USART**) is a type of serial interface device that can be programmed to communicate asynchronously or synchronously.

The **serial communication in asynchronous mode** is one of the simplest and most used methods to exchange data between a microcontroller and other devices.



STM32 USART

STM32 MCUs provide USART communication. We can communicate with the Nucleo board by activating USART peripheral. First you need a serial terminal, like MobaXterm. If necessary, download the software from -> https://mobaxterm.mobatek.net/download.html

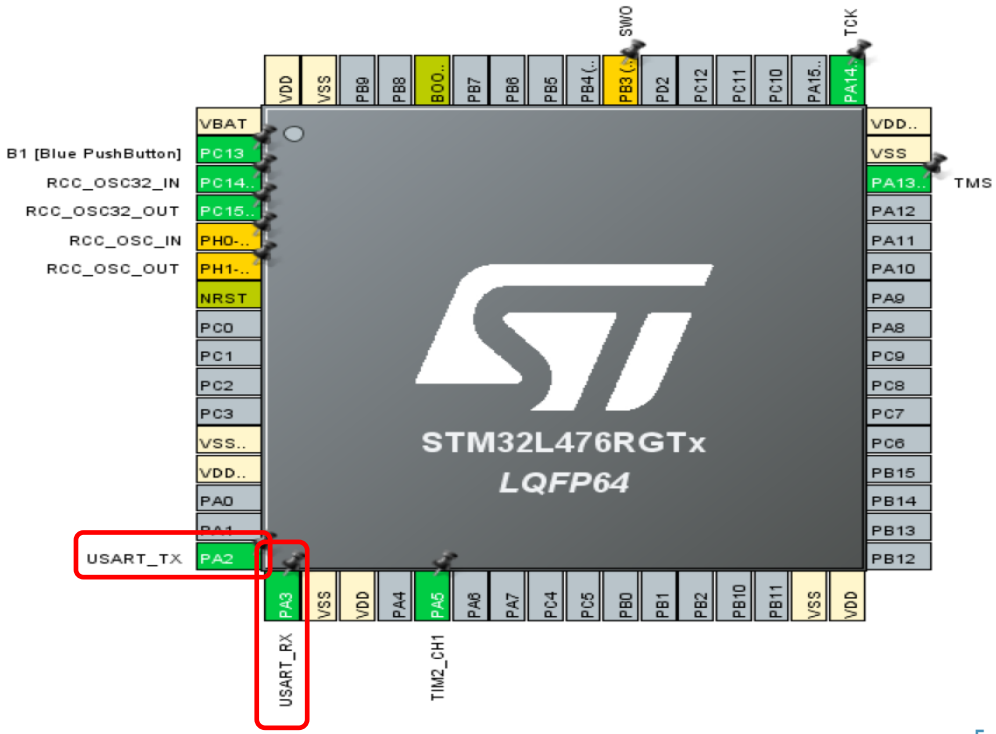


STM32 USART

Nucleo boards directly connect USART RX/TX pins to the mini USB port integrated in the STLink

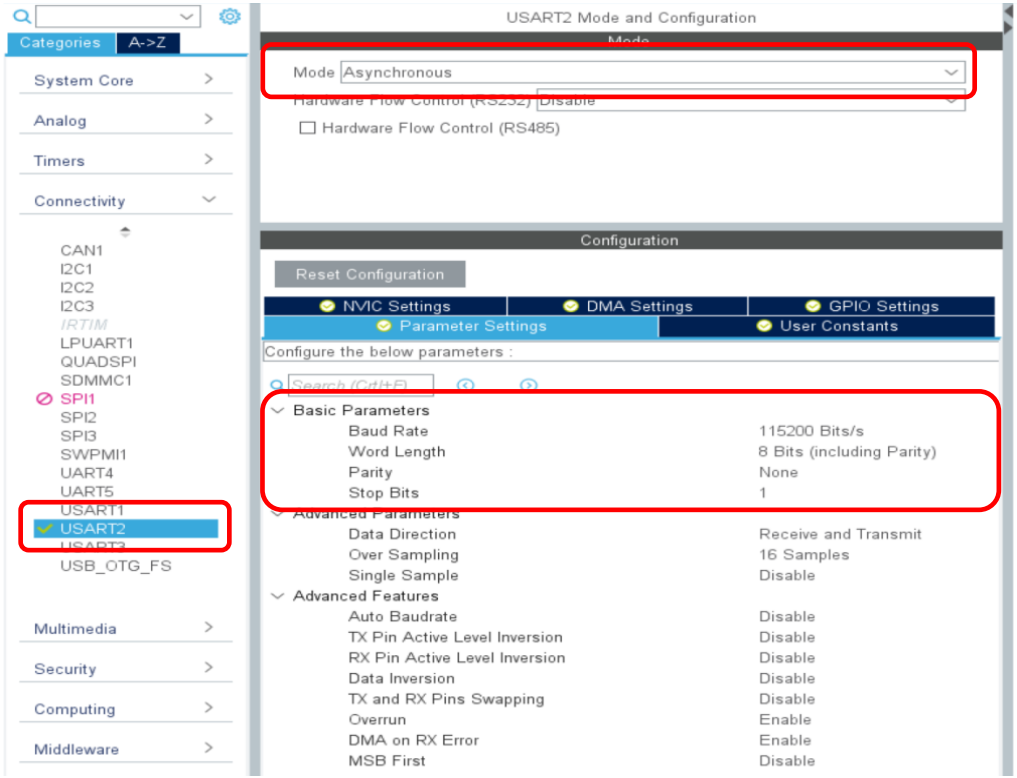
PA2 = TX

PA3 = RX



STM32 USART

USART parameters can be configured as always using CubeMX. In our case, we can stick with the default parameters.



STM32 USART

Once everything is configure, we can generate the code. CubeMX will create the new configuration methods with all the parameters set using CubeMX

```
UART_HandleTypeDef huart2;

/* USART2 init function */

void MX_USART2_UART_Init(void)
{

  huart2.Instance = USART2;
  huart2.Init.BaudRate = 115200;
  huart2.Init.WordLength = UART_WORDLENGTH_8B;
  huart2.Init.StopBits = UART_STOPBITS_1;
  huart2.Init.Parity = UART_PARITY_NONE;
  huart2.Init.Mode = UART_MODE_TX_RX;
  huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart2.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart2) != HAL_OK)
  {
    _Error_Handler(__FILE__, __LINE__);
  }

}
```

We can then transmit messages using the "HAL_UART_Transmit" function

```
/**
  * @brief  Sends an amount of data in blocking mode.
  * @param  huart pointer to a UART_HandleTypeDef structure that contains
  *                the configuration information for the specified UART module.
  * @param  pData Pointer to data buffer
  * @param  Size Amount of data to be sent
  * @param  Timeout Timeout duration
  * @retval HAL status
  */
HAL_StatusTypeDef HAL_UART_Transmit(UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size, uint32_t Timeout)
```

SEND A MESSAGE WHEN BUTTON IS PRESSED

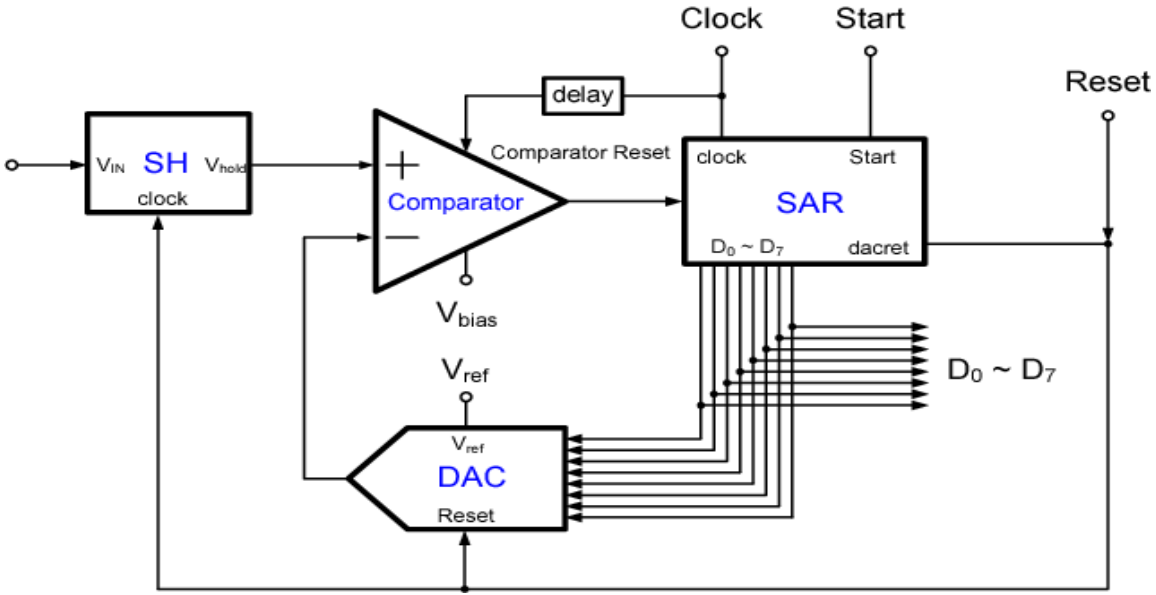
SEND A MESSAGE WHEN BUTTON IS PRESSED

Use the previously configured USART to send a message each time the pushbutton is pressed.

1. Use CubeMX to **configure the pushbutton interrupt**
2. Send "Hello from STM32" **when such interrupt is triggered**

ADC

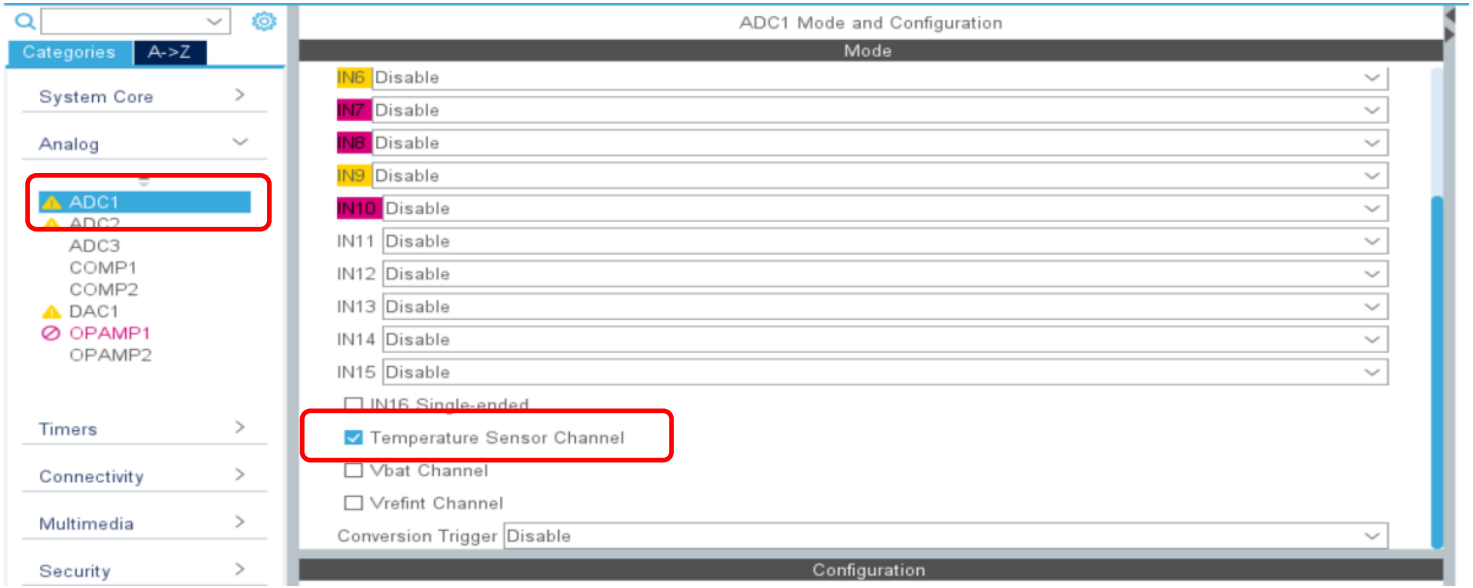
SAR ADC


Clock
Start
Reset
delay
V_{IN}
SH
V_{hold}
clock
Comparator
Comparator Reset
clock
Start
SAR
$D_0 \sim D_7$
dacret
V_{bias}
V_{ref}
$D_0 \sim D_7$
V_{ref}
DAC
Reset

STM32 ADC

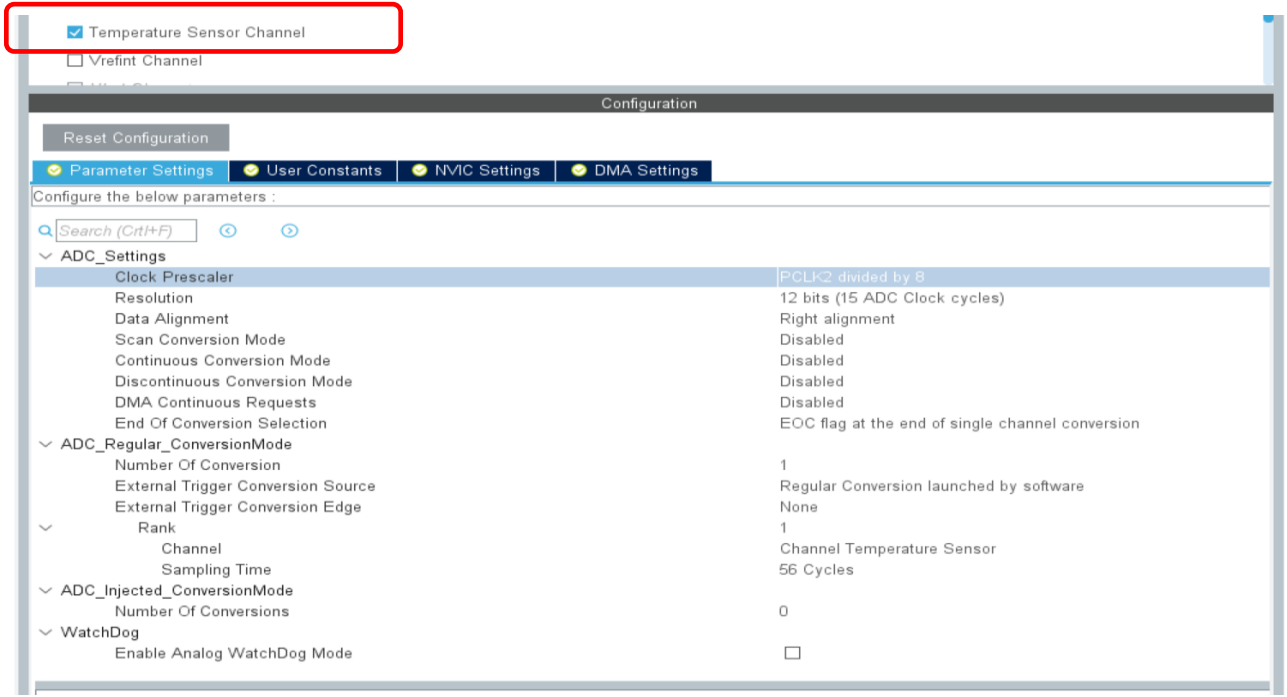
STM32 boards provide one, or multiples, ADCs connected to different channels, each one consisting of a 12-bit successive approximation A/D converter.

Among all, ADC1 provides two channels internally connected to a temperature sensor and to Vref.



STM32 ADC

As for the other peripherals, CubeMX let us configure the ADC in a graphical way.



STM32 ADC

Once generated the code, the ADC configuration can be found inside the **MX_ADC1_Init function**

```
49 /* ADC1 init function */
50 void MX_ADC1_Init(void)
51 {
52   ADC_ChannelConfTypeDef sConfig;
53
54     /**Configure the global features of the ADC (Clock, Resolution, Data Alignment and number of conversion)
55     */
56   hadc1.Instance = ADC1;
57   hadc1.Init.ClockPrescaler = ADC_CLOCK_SYNC_PCLK_DIV6;
58   hadc1.Init.Resolution = ADC_RESOLUTION_12B;
59   hadc1.Init.ScanConvMode = DISABLE;
60   hadc1.Init.ContinuousConvMode = DISABLE;
61   hadc1.Init.DiscontinuousConvMode = DISABLE;
62   hadc1.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_NONE;
63   hadc1.Init.ExternalTrigConv = ADC_SOFTWARE_START;
64   hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
65   hadc1.Init.NbrOfConversion = 1;
66   hadc1.Init.DMAContinuousRequests = DISABLE;
67   hadc1.Init.EOCSelection = ADC_EOC_SINGLE_CONV;
68   if (HAL_ADC_Init(&hadc1) != HAL_OK)
69   {
70     _Error_Handler(__FILE__, __LINE__);
71   }
72
73     /**Configure for the selected ADC regular channel its corresponding rank in the sequencer and its sample time.
74     */
75   sConfig.Channel = ADC_CHANNEL_TEMPSENSOR;
76   sConfig.Rank = 1;
77   sConfig.SamplingTime = ADC_SAMPLETIME_480CYCLES;
78   if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
79   {
80     _Error_Handler(__FILE__, __LINE__);
81   }
82
83 }
```

STM32 ADC

The ADC routine provides us the binary representation of the voltage sensed. Before having human readable numbers, we have to convert the raw data coming from the ADC.

We can create a simple method named **get_ADC(),** that takes care of starting the ADC, retrieve the value and then stops the ADC. The value returned can later be converted to a voltage value using the following formula(referenced to 3.3 V)

```
244 uint16_t get_ADC(){
245
246     HAL_ADC_Start(&hadc1);
247     HAL_ADC_PollForConversion(&hadc1, 1000);
248     uint16_t value = HAL_ADC_GetValue(&hadc1);
249
250     HAL_ADC_Stop(&hadc1);
251
252     return value;
253 }
```

$$V_{ADC}[mV] = \frac{ADC_{RAW} * V_{ref}}{2^{ADC_{bit}} - 1}$$

STM32 ADC

Once we have converted the ADC raw value, we have to convert this voltage to a Celsius value. MCU datasheet provides the parameters and the formula for conversion.

6.3.21 Temperature sensor characteristics

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.

$$T\,[°C] = \frac{V_{ADC} - V_{25}}{AVG_{Slope}} + 25$$

```
236 float ConvertTemp(uint16_t D_ADC){
237
238     float V_ADC = D_ADC * ( Vref / 4095.0 );          // Convert bit to voltage
239     float temp =  ((V_ADC - V25) / Avg_Slope ) + 25;  // Convert voltage to Temp
240
241     return temp;
242 }
```

Vref and Avg_Slope must be defined accordingly !

SEND TEMPERATURE READINGS THROUGH USART

SEND TEMPERATURE READINGS THROUGH USART

Use the previously configured USART and ADC as temperature sensor channel, send temperature readings through USART.

1. Use CubeMX to **configure the USART**
2. Use CubeMX to **configure the ADC as temperature sensor channel**
3. Send temperature readings through USART every 5 seconds.

VISUAL THERMOMETER

VISUAL THERMOMETER

Starting from the last exercise, adjust the brightness of the led proportionally to the temperature readings.

1. Use CubeMX to **configure the USART**
2. Use CubeMX to **configure the ADC as temperature sensor channel**
3. Send temperature readings through USART every 5 seconds.
4. **PLUS: adjust the led brightness proportionally to the temperature readings.** [E.g. 25% if T < 20, 50% if 20 <= T <= 22 and 100% if T > 22]