Mark	

Team name:	*A5*		
Homework number:	*HOMEWORK 5*		
Due date:	29/10/2023		
Contribution	NO	Partial	Full
Luca Daidone			*x*
Leonardo Ritter			*x*
Luca Cordaro			*x*
Lorenzo Strechelli			*x*
Ignazio Neto Dell'Acqua			*x*
Notes:			

Project name	ADC Scan mode and LDR measurement		
Not done	Partially done (major problems)	Partially done (minor problems)	Completed
			x

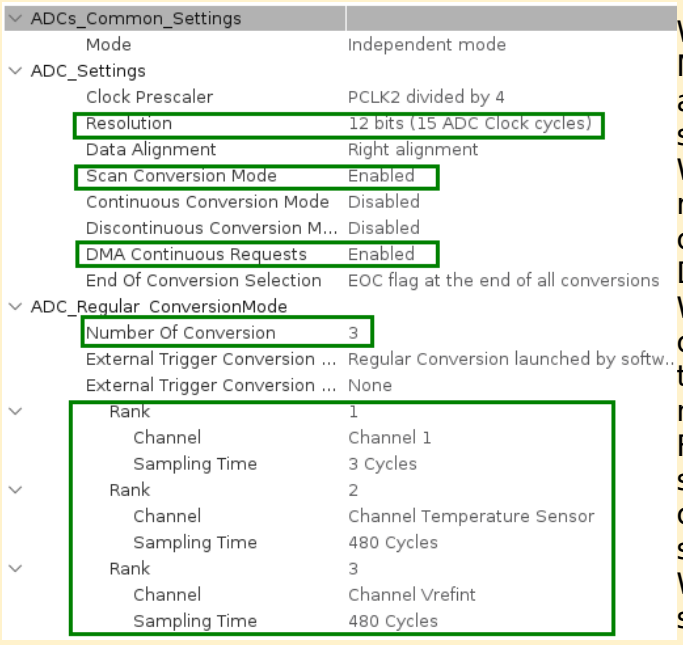
Explanation:
We successfully completed the homework.

Excercise 3a - ADC scan using DMA:

CubeIDE Settings:

- We begin by configuring the potentiometer's output as an input for our STM32, specifically setting it as the first input to ADC1, with PA1 designated as ADC1_IN.

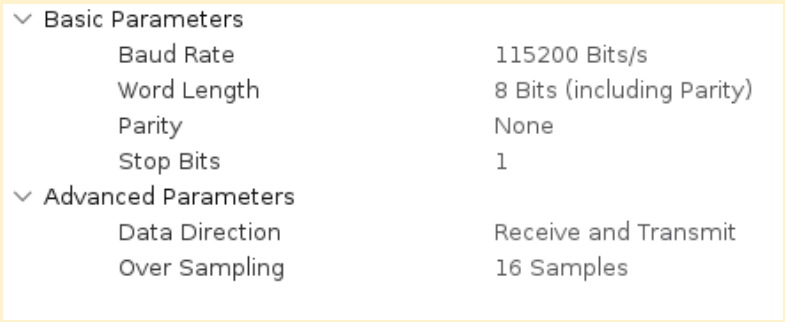
- In the “Analog” section we activate the IN1 in the “Mode” panel. The configuration must be as follow:



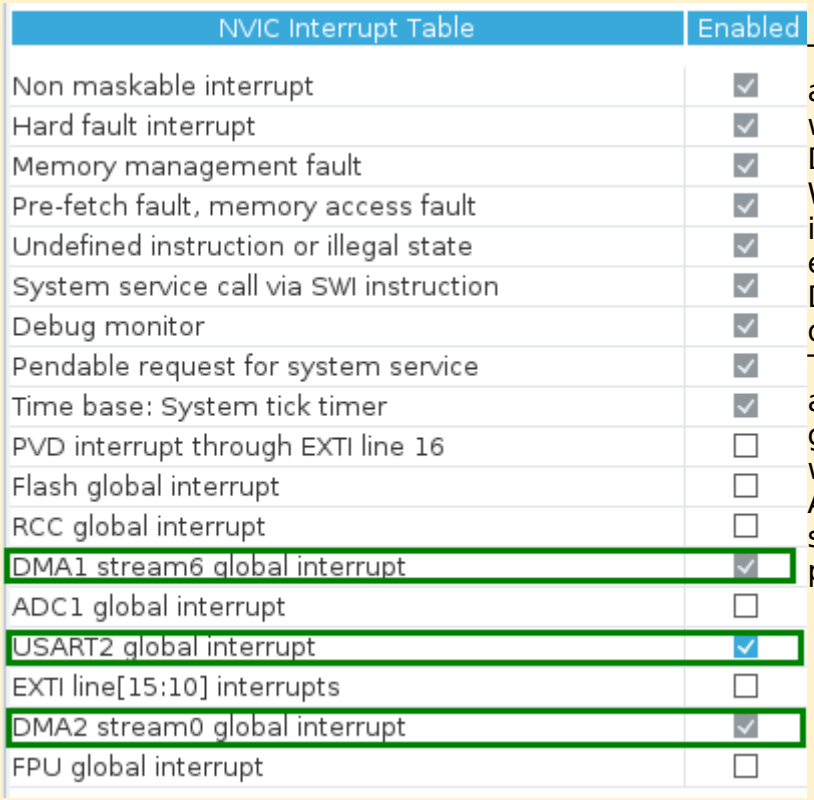
We enable "Scan Conversion Mode" to allow the ADC to scan and convert multiple channels sequentially.
We enable "DMA continuous request" to ensure that the conversion results are sent by DMA to memory continuously.
We set the "Number of conversions" to 3, considering the 3 different channels to measure.
For the T and Vref channels, we set the sampling time to 480 cycles, as these signals change slowly.
We initiate the conversion by software.

- We employ two DMA controllers, one for ADC data and the other for the USART protocol. We configure the ADC DMA to operate in "circular" mode, which ensures it automatically restarts to write to the first memory cell when the last memory cell is written. We set the direction of the ADC DMA as "peripheral to memory" and the USART DMA as "memory to peripheral."

- USART2 is activated in asynchronous mode, with the following configurations:



- The NVIC table must be set as follow:



The DMA interrupts are automatically set up when we enable the DMA peripherals.
We enable the global interrupt for USART2 to ensure that the USART DMA continues sending data.
There's no need to activate the ADC1 global interrupt since we've observed that the ADC DMA interrupt serves the same purpose.

Code:
In the infinite loop, first the ADC is started in DMA mode with the “HAL_ADC_Start_DMA()” function, and after that a simple 1s delay is introduced.
The HAL_ADC_Start_DMA() requires as parameters: the ADC handler, an array of data where the DMA pheripheral will store the conversion results, called adc_val[3], and a parameter that specifies the size of the memory pointed by the DMA.
In the "void HAL_ADC_ConvCpltCallback()" routine, which is called after the ADC completes all three conversions and the DMA fills the memory, we handle all the conversions as follows:

```
//convert the ADC value from the potentiometer to voltage
float pot = (adc_val[0]*VMAX)/4095;
//convert the ADC value from the T sensor to voltage
float temp = 25 + ( ( ( (adc_val[1]*VMAX)/4095) - V25 ) / AVG_SLOPE);
//convert the ADC value from the internal reference to voltage
float ref = (adc_val[2]*VMAX)/4095;
```

- for the potentiometer, the ADC value is simply converted into voltage
- for the Temperature, the following formula is used:

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

- for the reference voltage, again the value is simply converted into voltage

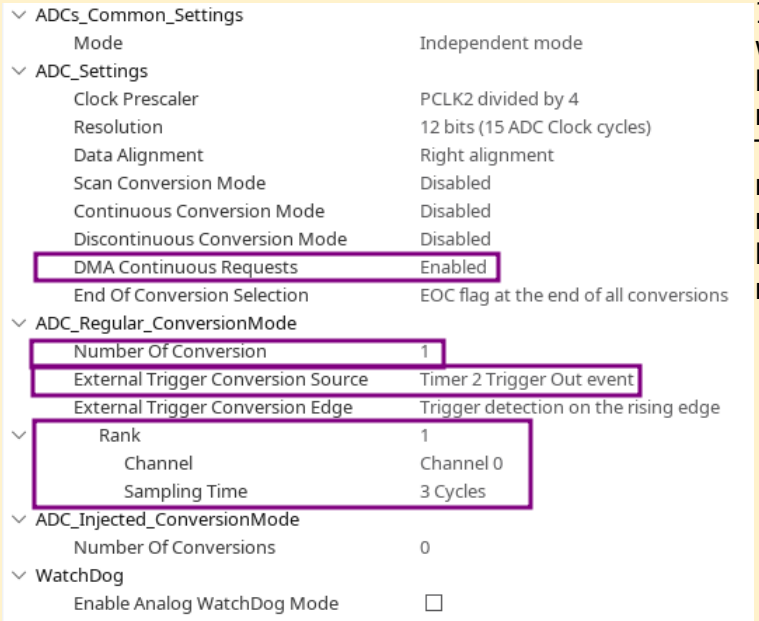
All the float results are then written into a buffer string with the “snprintf” function and sent over UART through the HAL-function “HAL_UART_Transmit_DMA”.
The program counter will go back to the infinite loop, where the ADC will be started again after the 1s delay.

Excercise 3b – Light dependent resistor:

CubeIDE Settings:

- From the “Green PCB Board” schematic we found that LDR is connected to the PA0 of the Nucleo board, therefore we begin by setting PA0 as an analog input in order to read the LDR. The PA0 pin is connected to the ADC1_IN0 channel

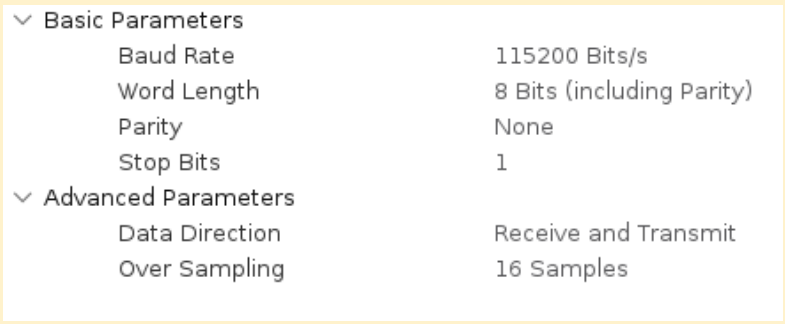
- In the “Analog” section we activate IN0 in the “mode” pannel. The configuration must be as follow:



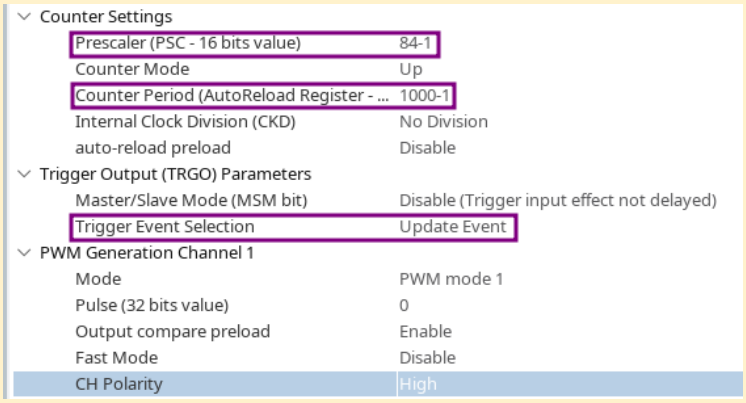
12 bits resolution is used, with right alignment, and DMA Continuous request must be enabled.
The number of conversions must be set to 1 since we read only one channel.
External trigger source must be the Timer 2.

- We employ two DMA controllers, one for ADC data and the other for the USART protocol. We configure the ADC DMA to operate in "circular" mode, which ensures it automatically restarts to write to the first memory cell when the last memory cell is written. We set the direction of the ADC DMA as "peripheral to memory" and the USART DMA as "memory to peripheral."

- USART2 is activated in asynchronous mode, with the following configurations:



- In the Timer section, TIM2 must be activated by setting its channel 1 to “PWM generation no output”. By setting the Prescaler equal to 84-1 and the Counter Period equal to 1000-1 the total period is 1ms. The “Trigger Event Selection” must be set to “Update Event” such as every time a period elapses, the ADC is started.



- In the NVIC settings, the ADC1 Global interrupt and the USART2 Global Interrupt must be enabled.

Code:

In the main(), before the infinite loop, first the ADC is started in DMA mode with the “HAL_ADC_Start_DMA()” function, and then the timer is started with the function “HAL_TIM_PWM_Start()”. The HAL_ADC_Start_DMA() requires as parameters: the ADC handler, an array of data where the DMA pheripheral will store the conversion results, called adc_val[1000], and a parameter that specifies the size of the memory pointed by the DMA.

In the "void HAL_ADC_ConvCpltCallback()" routine, which is called after the ADC completes all 1000 conversions and the DMA fills the memory, we first calculate the average of the 1000 samples. The average is also converted into voltage.

```
//sum in a variable all the 1000 samples
for(uint16_t i = 0 ; i<SAMPLE_SIZE; i++)
    sum += adc_val[i];

//calculate the average and convert it into voltage
float adc_avg = ((sum / SAMPLE_SIZE)*VDD)/4095.0;
```

The voltage is then converted into resistance following the formula:

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

And the resistance is converted into LUX following:

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

The code is:

```
//convert the voltage into resistance value
float ldr = (adc_avg * LDR_CONV)/(VDD  - adc_avg);
//convert the resistance value into LUX value
float lux = 10*pow((LDR_CONV/ldr), 1.25);
```

note how the pow() function from the math library had to be employed to calculate the power of 1.25.

All the float results are then written into a buffer string with the “snprintf” function and sent over UART through the HAL-function “HAL_UART_Transmit_DMA”.
The ADC ISR will be called every 1s, since the DMA waits for 1000 samples and the timer trigger a conversion each 1ms.

Professor comments: