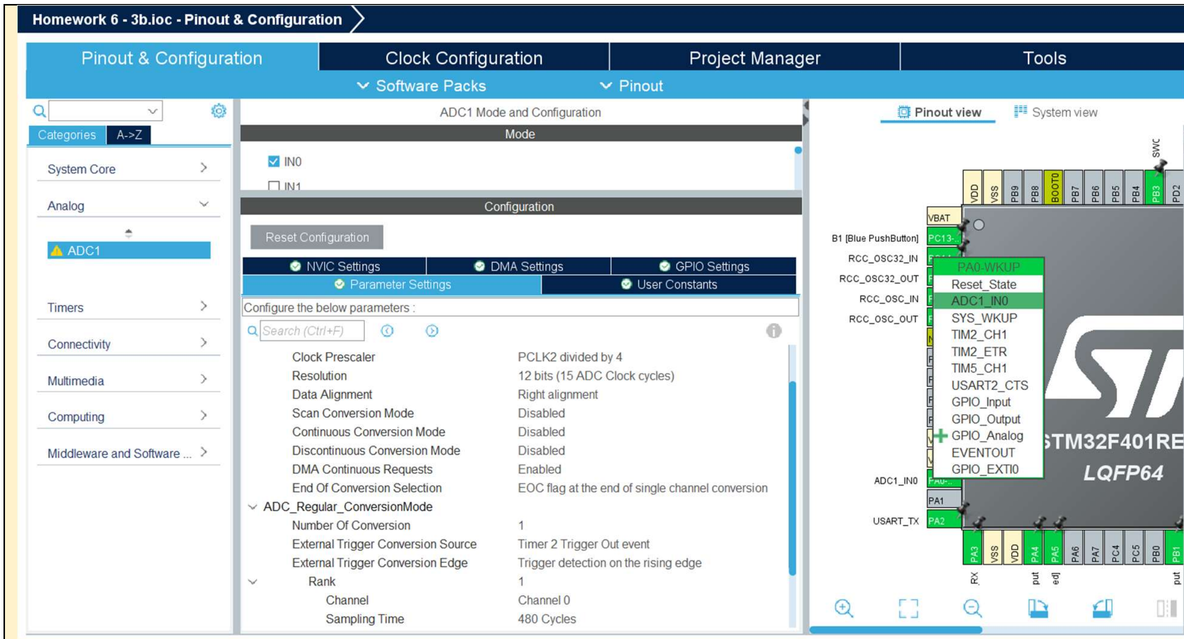
Mark	/11

Team name:	*A14*		
Homework number:	*HOMEWORK 06*		
Due date:	26/10/25		
Contribution	NO	Partial	Full
Mattia Di Mauro			*x*
Francesca Biondi			*x*
Lorenzo Castelli			*x*
Carmen Maria Niro			*x*
Pietro Albrigi			*x*
Notes: none			

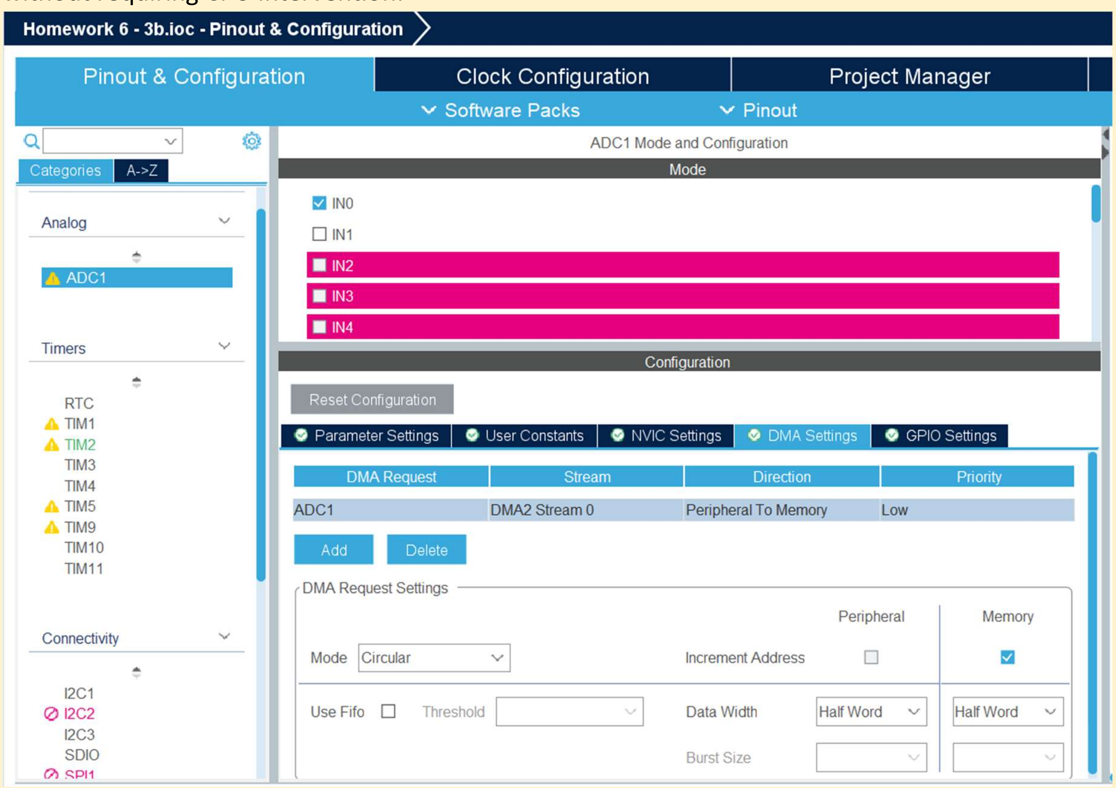
Project name	HOMEWORK 6		
Not done	Partially done (major problems)	Partially done (minor problems)	Completed
			x

Project 2b:

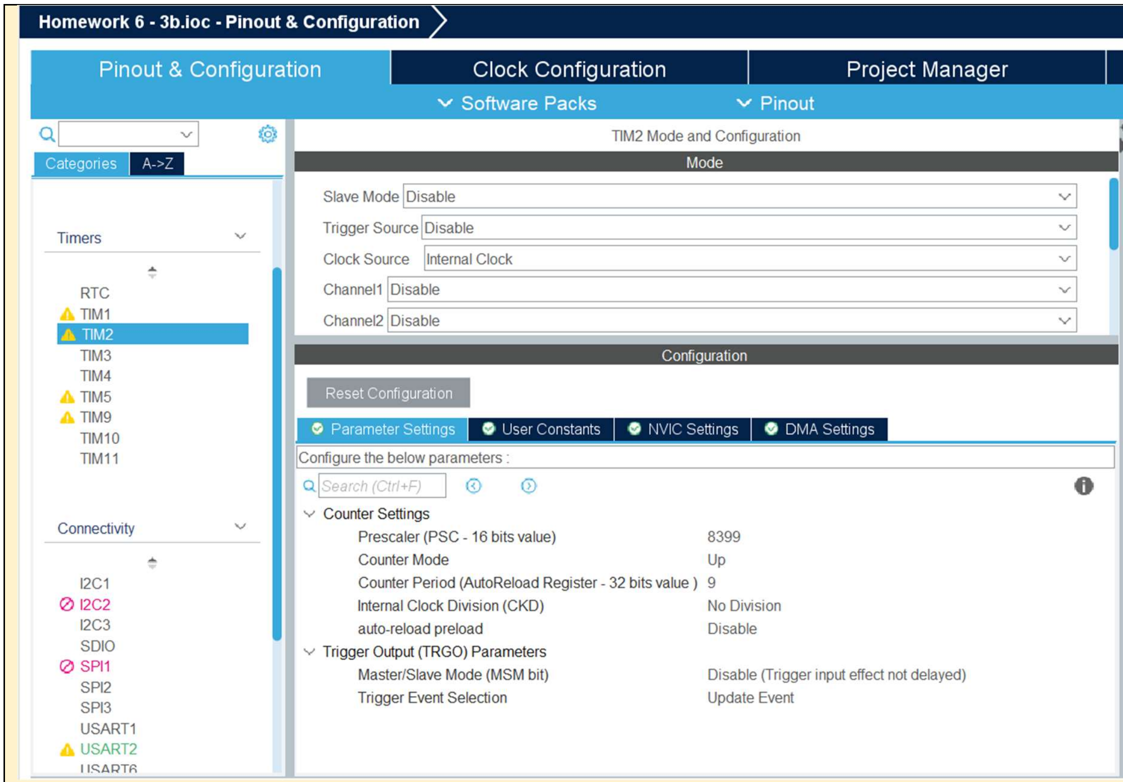
We identified the LDR pin (PA0) in the Hands-on Lab schematics and configured it in the IOC file as ADC1_IN0. We enabled the DMA Continuous Request and selected Timer 2 Trigger Output Event as the external trigger conversion source, with trigger detection set on the rising edge. The sampling time was set to 480 cycles, ensuring stable and accurate readings from the LDR:



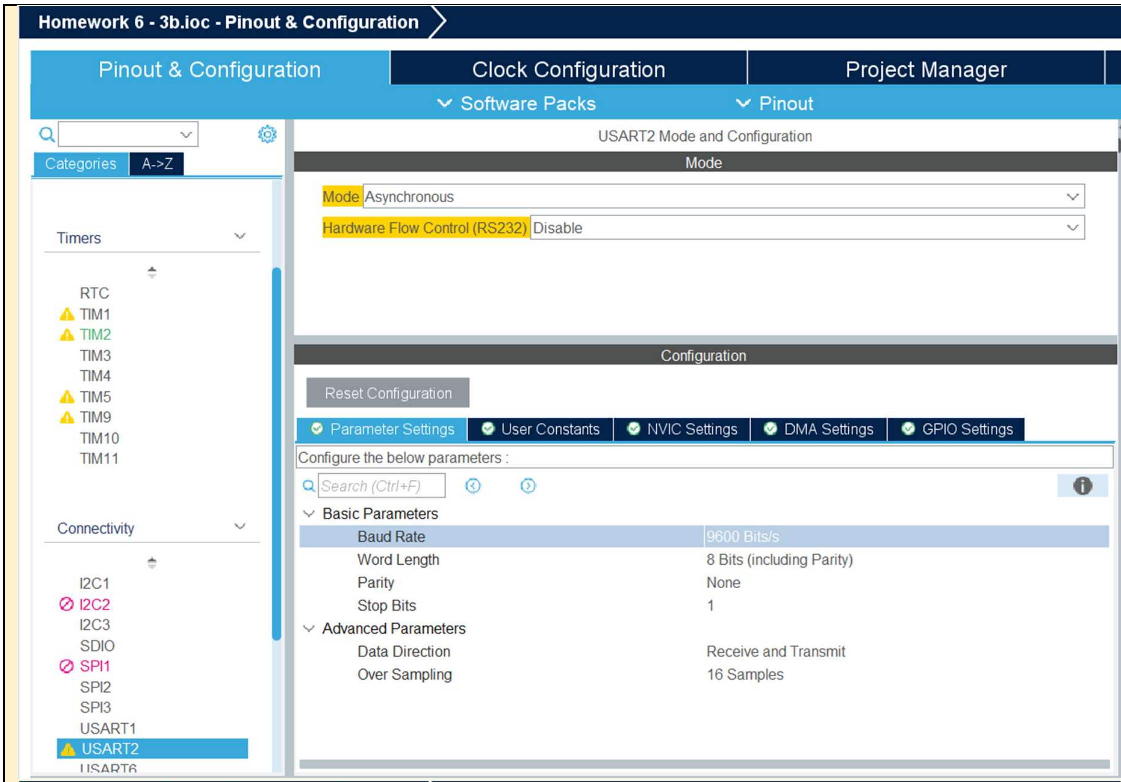
In the DMA configuration, ADC1 was added with mode set to circular, allowing continuous data transfers without requiring CPU intervention:

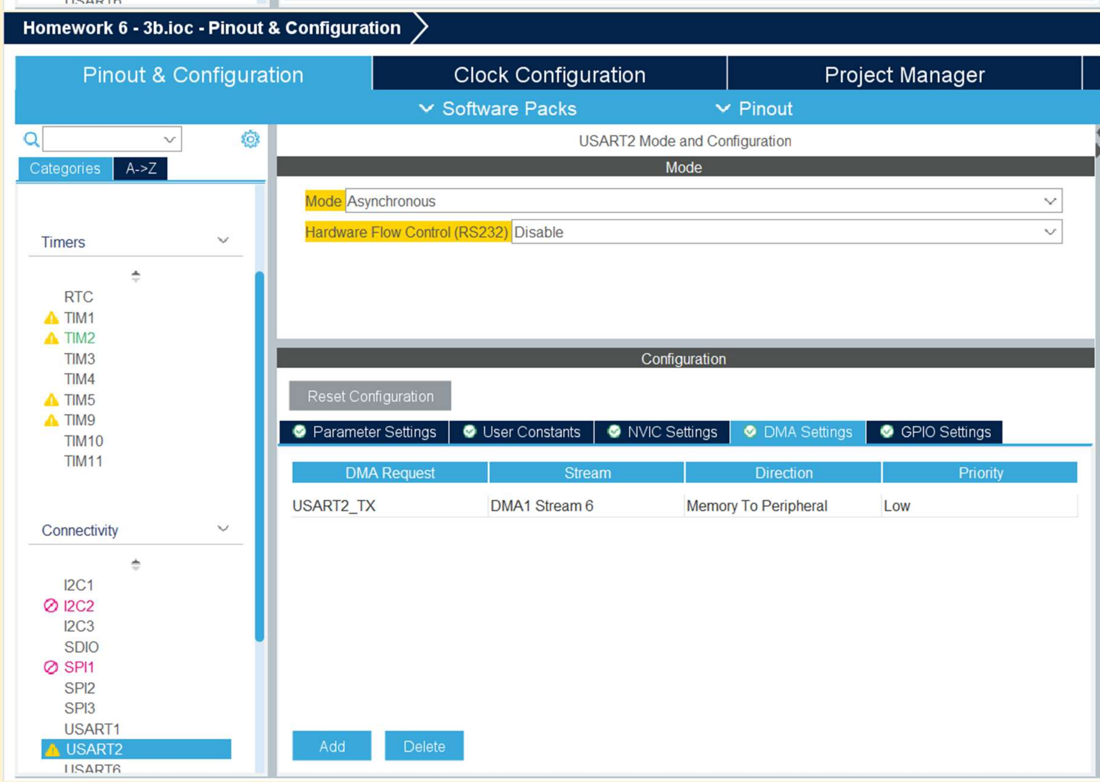


For Timer 2, we selected the internal clock source and configured a prescaler of 8399, an auto-reload register (ARR) value of 9 and Update Event as Trigger Event Selection. This means that Timer 2 generates an update event, and thus a trigger for the ADC, every millisecond. Since the ADC is configured to start a conversion on each rising edge of the trigger, this setup results in one ADC conversion being started every millisecond, precisely synchronized with the timer overflow.



We also enabled USART communication using DMA at a baud rate of 9600:





From the coding perspective, we begin by including the <math.h> library. We then defined the constant SAMPLES (1000) and the global variable LDR_buffer[SAMPLES], which is used by the ADC to automatically store conversion results through DMA.

In the main() function, we start Timer 2 using HAL_TIM_Base_Start(&htim2) and initialize the ADC in DMA mode with the call HAL_ADC_Start_DMA(&hadc1, &LDRBuffer, SAMPLES). This ensures that one sample is transferred to LDR_buffer every millisecond without any CPU overhead.

```c
/* USER CODE BEGIN 2 */
HAL_TIM_Base_Start(&htim2);
HAL_ADC_Start_DMA(&hadc1, &LDR_buffer, SAMPLES);
/* USER CODE END 2 */
```

Outside the main() function, two callback functions are defined: HAL_ADC_ConvHalfCpltCallback() and HAL_ADC_ConvCpltCallback(). These functions are automatically invoked by the HAL library when the ADC, operating in DMA mode, completes half or all of the buffer conversions, respectively. The half-complete callback is triggered when half of the ADC buffer is filled, enabling intermediate data processing while maintaining continuous sampling through DMA.

In both callbacks, the ADC samples stored in the LDR_buffer array are first converted into a corresponding voltage based on the ADC's 12-bit resolution and the 3.3 V reference.

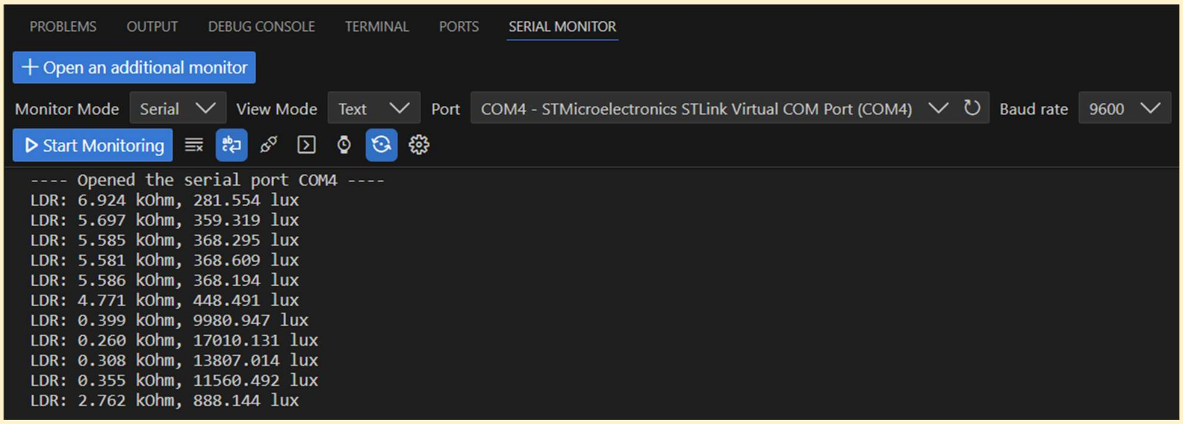
Using this voltage, the resistance of the LDR is calculated. The computed resistance values are then accumulated in the variable average, while the counter count is incremented each time a new value is acquired.

```c
/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
void HAL_ADC_ConvHalfCpltCallback(ADC_HandleTypeDef* hadc) {
    for (int i = 0; i < SAMPLES / 2; i++) {
        float voltage = LDR_buffer[i] * (3.3 / 4095);
        float R_LDR = (voltage * 100)/ (3.3 - voltage);
        average += R_LDR;
        count++;
    }
}

void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc) {
    for (int i = SAMPLES / 2; i < SAMPLES; i++) {
        float voltage = LDR_buffer[i] * (3.3 / 4095);
        float R_LDR = (voltage * 100)/ (3.3 - voltage);
        average += R_LDR;
        count++;
    }

    if (count >= SAMPLES) {
        average = average / SAMPLES;
        float lux = 10 * pow((100/average), 1.25);
        static char string[50];
        snprintf(string, sizeof(string), "LDR: %.3f kOhm, %.3f lux \r\n", average, lux);
        HAL_UART_Transmit_DMA(&huart2, string, sizeof(string));
        average = 0;
        count = 0;
    }
}
/* USER CODE END 0 */
```

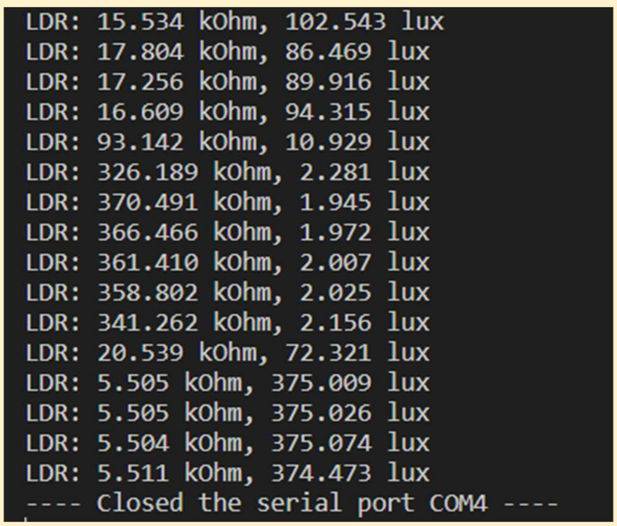
Once the counter reaches 1000, meaning approximately one second has passed (since we are sampling at 1 kHz), we compute the average resistance by dividing the accumulated sum by the number of samples. This average resistance is then used to calculate the corresponding light intensity in lux, using the formulas provided in the lecture slides. Finally, the computed values (such as average resistance and lux) are formatted into a string and transmitted to the serial terminal via USART using DMA.

Before exiting the conditional block, both the average sum and the count are reset to begin the next one-second measurement cycle.

The code was then tested on the board and confirmed to work as expected. From the remote terminal, plausible values were observed under different lighting conditions: in a normally lit room (around 5 kOhm/ 300 lux), when the sensor was illuminated with a flashlight (less than 1 kOhm / over 10 klux):



and when the sensor was covered to simulate darkness (around 350 kOhm / only a few lux):



Professor comments: