

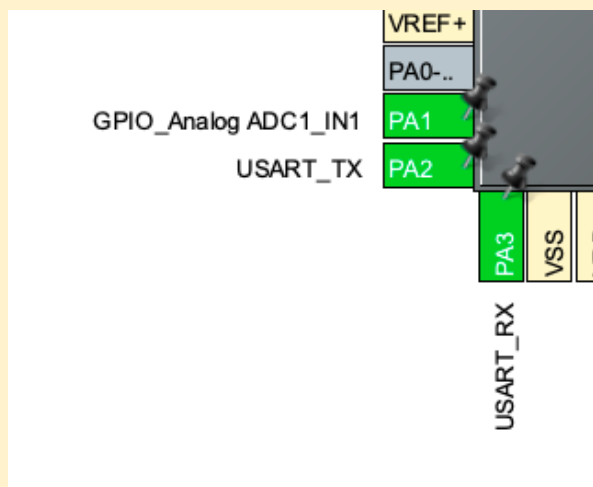
Mark	/11
------	-----

Team name:	A1		
Homework number:	HOMEWORK 06		
Due date:	28/10/24		
Contribution	NO	Partial	Full
Piombo			x
Fumagalli			x
Pierfederici			x
Zenoni			x
Ferraro			x
Notes:			

Project name	ADC DMA + LDR		
Not done	Partially done (major problems)	Partially done (minor problems)	Completed
			x

Part 1:

First of all, we set the potentiometer pin (PA1) as analog input (ADC1) and we checked the UART pins



We configured the ADC as below in order to scan the 3 channels in sequence (scanning mode), operating it in DMA mode, with the conversion started by the timer (TIM2)

ADC1 Mode and Configuration

Mode

☐ IN0

☒ IN1

☒ IN2

☒ IN3

☐ IN4

☒ IN5

☐ IN6

☐ IN7

☐ IN8

☐ IN9

☐ IN10

☐ IN11

☐ IN12

☐ IN13

☐ IN14

☐ IN15

☒ Temperature Sensor Channel

☒ Vrefint Channel

☐ Vbat Channel

Parameter Settings

User Constants

NVIC Settings

DMA Settings

GPIO Settings

Configure the below parameters :

Search (Ctrl+F)

ADCs_Common_Settings

ModeIndependent mode

ADC_Settings

Clock PrescalerPCLK2 divided by 4

Resolution12 bits (15 ADC Clock cycles)

Data AlignmentRight alignment

Scan Conversion ModeEnabled

Continuous Conversion ModeDisabled

Discontinuous Conversion ModeDisabled

DMA Continuous RequestsEnabled

End Of Conversion SelectionEOC flag at the end of all conversions

ADC_Regular_ConversionMode

Number Of Conversion3

External Trigger Conversion SourceTimer 2 Trigger Out event

External Trigger Conversion EdgeTrigger detection on the rising edge

Rank1

Rank2

Rank3

ADC_Injected_ConversionMode

Number Of Conversions0

WatchDog

Enable Analog WatchDog Mode

This is the scanning order operated by the ADC

🔑	Rank	1
	Channel	Channel 1
	Sampling Time	480 Cycles
🔑	Rank	2
	Channel	Channel Temperature Sensor
	Sampling Time	480 Cycles
🔑	Rank	3
	Channel	Channel Vrefint
	Sampling Time	480 Cycles

We needed the ADC interrupt enabled

Parameter Settings	User Constants	NVIC Settings	DMA Settings	GPIO Settings
NVIC Interrupt Table		Enabled	Preemption Priority	Sub Priority
ADC1 global interrupt		<input checked="" type="checkbox"/>	0	0
DMA2 stream0 global interrupt		<input checked="" type="checkbox"/>	0	0

This is the configuration of the DMA of the ADC (circular mode)

Configuration

Reset Configuration

Parameter SettingsUser ConstantsNVIC SettingsDMA SettingsGPIO Settings

DMA Request	Stream	Direction	Priority
ADC1	DMA2 Stream 0	Peripheral To Memory	Low

AddDelete

DMA Request Settings

Peripheral		Memory
Mode	<div>Circular</div>	Increment Address
		<input type="checkbox"/>
		<input checked="" type="checkbox"/>
Use Fifo	<input type="checkbox"/>	Threshold
		<div></div>
Data Width	<div>Half Word</div>	Half Word
		<div></div>
Burst Size	<div></div>	<div></div>

This is the timer 2 configuration (start the ADC conversion every second)

TIM2 Mode and Configuration

Mode

Slave Mode

Disable

Trigger Source

Disable

Clock Source

Internal Clock

Channel1

Disable

Channel2

Disable

Configuration

Reset Configuration

Parameter Settings

User Constants

NVIC Settings

DMA Settings

Configure the below parameters :

Search (Ctrl+F)

Counter Settings

Prescaler (PSC – 16 bits value)

8400-1

Counter Mode

Up

Counter Period (AutoReload Register – 32 bits value)

10000-1

Internal Clock Division (CKD)

No Division

auto-reload preload

Disable

Trigger Output (TRGO) Parameters

Master/Slave Mode (MSM bit)

Disable (Trigger input effect not delayed)

Trigger Event Selection

Update Event

In order to send data to the PC we used UART in DMA mode (we also enabled UART interrupt)

Configuration

Reset Configuration

Parameter Settings

User Constants

NVIC Settings

DMA Settings

GPIO Settings

DMA Request	Stream	Direction	Priority
USART2_TX	DMA1 Stream 6	Memory To Peripheral	Low

Add

Delete

DMA Request Settings

Mode

Normal

Increment Address

☐

Peripheral

☒

Memory

☒

Use Fifo

☐

Threshold

Data Width

Byte

Byte

Burst Size

Byte

These are the variables used in our code

```
/* USER CODE BEGIN PV */

uint16_t digital_data[BUF_LEN];    //store the 3 channel's digital values from ADC
float analog_voltage[BUF_LEN];    //store the 3 channel's analog values
float temperature;
char string[STR_LEN];
int length;

/* USER CODE END PV */
```

In our main function we configured the ADC in DMA mode and started our timer

```
HAL_ADC_Start_DMA(&hadc1, digital_data, BUF_LEN);    //properly configure the ADC in DMA mode
HAL_TIM_Base_Start(&htim2);    // start timer
```

When the ADC conversion is finished, this function is called: the data are elaborated and sent to the PC console (LSB is defined as 3.3/4096.0)

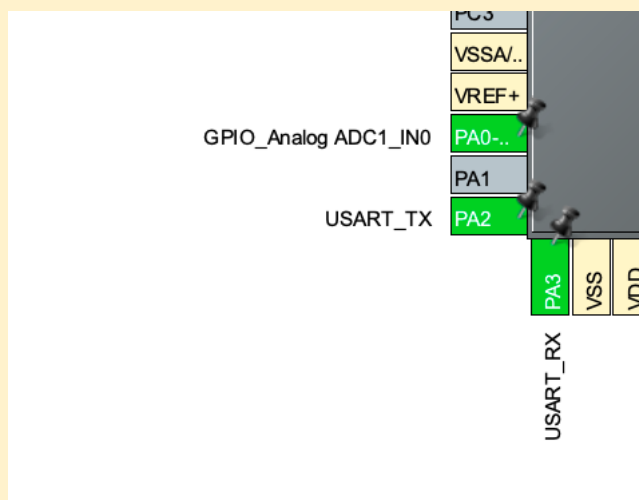
```
void HAL_ADC_ConvCpltCallback (ADC_HandleTypeDef *hadc) {
    if (hadc == &hadc1) {
        for (int i = 0; i < BUF_LEN; i++) {
            analog_voltage[i] = LSB*digital_data[i];
        }
        temperature = ((analog_voltage[1]-V_25)/AVG_SLOPE) + 25;    // conversion V -> °C
        length = snprintf(string, sizeof(string), "POT: %.3fV TMP: %.3f°C VREF: %.3fV\n", analog_voltage[0], temperature, analog_voltage[2]);
        HAL_UART_Transmit_DMA(&huart2, string, length);
    }
}
```

We computed the temperature with the provided formula

$$Temperature(in\ ^\circ C) = \frac{V_{sense} - V_{25}}{Avg_Slope} + 25$$

Part 2:

We set the light sensor pin (PA0) as analog input (ADC1) and we checked the UART pins



We configured the ADC as the part 1 (DMA **circular mode**, started by timer 2, ADC interrupt enabled). The only difference is that now we operate on a single channel.

This is timer 2 configuration, in order to start the ADC conversion every ms

Configure the below parameters :	
Search (Ctrl+F)	
Counter Settings	
Prescaler (PSC - 16 bits value)	8400-1
Counter Mode	Up
Counter Period (AutoReload Register - 32 bits value)	10-1
Internal Clock Division (CKD)	No Division
auto-reload preload	Disable
Trigger Output (TRGO) Parameters	
Master/Slave Mode (MSM bit)	Disable (Trigger input effect not delayed)
Trigger Event Selection	Update Event

Then we configured UART in DMA mode as part 1, to send our data to the PC.

We needed to acquire 1000 samples from the LDR and average them. In order to be as fast as possible to avoid the DMA buffer (digital_data) being overwritten while we still have to process our data, we exploited also the half DMA callback, in which we processed the first 500 samples. In this way the ADC with DMA can collect data in parallel to the data elaboration performed by the CPU, avoiding data corruption.

These are our variables (NUM_OF_SAMPLES is defined as 1000)

```
/* USER CODE BEGIN PV */

uint16_t digital_data[NUM_OF_SAMPLES];           //store all ldr values sampled every ms
uint32_t digital_sum1 = 0;                       //store sum of 1st 500 samples
uint32_t digital_sum2 = 0;                       //store sum of 2nd 500 samples
uint32_t digital_average = 0;                   //store ldr average value
float analog_voltage = 0;                       //store analog voltage value (average)
float ldr = 0; // in kOhm
float lux = 0; // in lx
char string[STR_LEN];
int length;

/* USER CODE END PV */
```

We configured the ADC and started the timer in the main function

```
HAL_ADC_Start_DMA(&hadc1, digital_data, NUM_OF_SAMPLES); //properly configure ADC in DMA mode
HAL_TIM_Base_Start(&htim2);                             // start timer
```

These are our callbacks (LSB is defined as 3.3/4096.0)

```
void HAL_ADC_ConvHalfCpltCallback (ADC_HandleTypeDef *hadc) {
    if (hadc == &hadc1) {
        digital_sum1 = 0;
        for (int i = 0; i < NUM_OF_SAMPLES/2; i++) {    //process here the 1st 500 samples
            digital_sum1 += digital_data[i];
        }
    }
}

void HAL_ADC_ConvCpltCallback (ADC_HandleTypeDef *hadc) {
    if (hadc == &hadc1) {
        digital_sum2 = 0;
        for (int i = 500; i < NUM_OF_SAMPLES; i++) {    //process here the 2nd 500 samples
            digital_sum2 += digital_data[i];
        }
        digital_average = (digital_sum1 + digital_sum2)/NUM_OF_SAMPLES; //compute the average
        analog_voltage = LSB*digital_average;
        ldr = (analog_voltage*100)/(3.3-analog_voltage);    //conversion V -> kOhm
        lux = 10*powf(100/ldr, 1.25);    //conversion kOhm -> lx
        length = snprintf(string, sizeof(string), "LDR: %.3fkOhm LUX: %.3flx\n", ldr, lux);
        HAL_UART_Transmit_DMA(&huart2, string, length);
    }
}
```

We calculated resistance value and lux value with the provided formulas

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

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