Use of ADC with DMA on Microcontroller

Targeted competences: Use of ADC in standard mode with DMA.

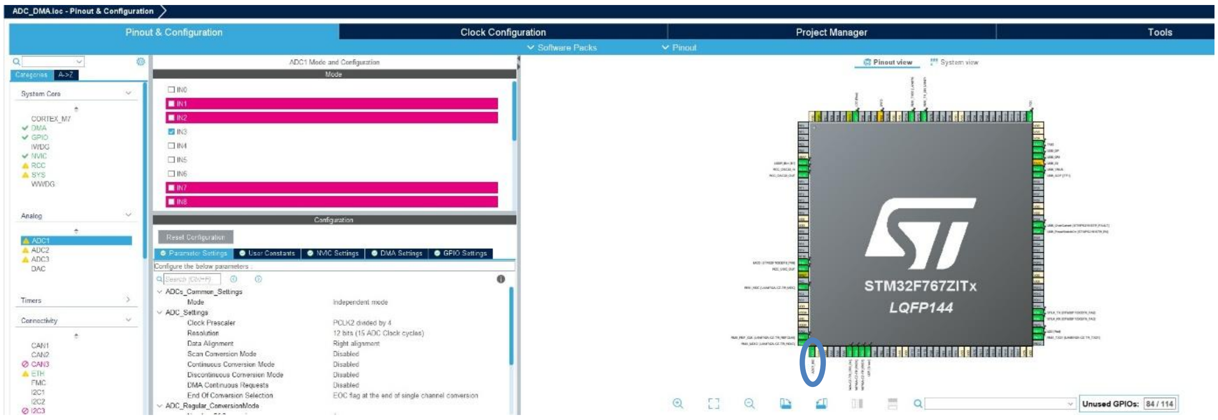
Hardware: STM32F7 Nucleo board

Framework: STM32CubeIDE 1.19

The aim of this example is to study how use an ADC in standard with the DMA. In a first time, we use a sampling period in order to just convert an analog value comes from a GPIO configured in analog mode. In order to transfer the converted value into a variable, we use the DMA (Direct Memory Access) instead of using the ADC_Get_Value function of the HAL driver.

Instead of realizing a pooling on the ADC to know if it is busy or not, we will use a timer to generate a sampling period. At each sampling period, we will convert the analog input and store the value into a variable.

The first step is to use the CubeMx program in order to configure the micro controller. Here we use the NUCLEO STM32F767ZI board. We choose to use the PA3 pin in ADC1_IN3 (See Figure 1) that means that the pin PA3 is linked to the input channel 3 of the ADC1.

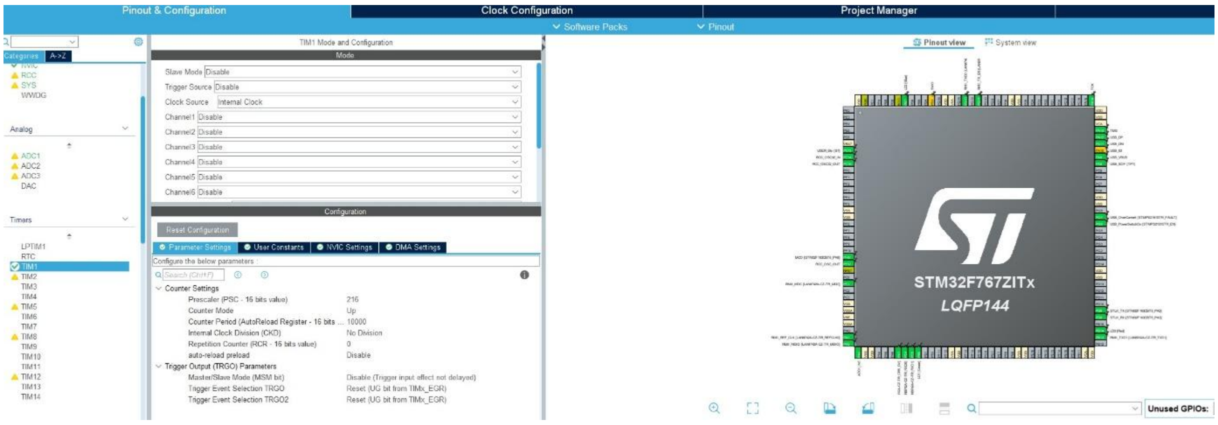


Configure the below parameters :

```
ADCs_Common_Settings
    Mode                              Independent mode
ADC_Settings
    Clock Prescaler                   PCLK2 divided by 4
    Resolution                        12 bits (15 ADC Clock cycles)
    Data Alignment                    Right alignment
    Scan Conversion Mode              Disabled
    Continuous Conversion Mode        Disabled
    Discontinuous Conversion Mode     Disabled
    DMA Continuous Requests           Disabled
    End Of Conversion Selection       EOC flag at the end of single channel conversion
ADC_Regular_ConversionMode
```

FIGURE 1: CONFIGURATION OF THE GPIO PIN

Then the second step is to select a timer in order to realize the sampling period. I chose to use the timer 1 and I configure it as shown in Figure 2 (just select the clock source).



TIM1 Mode and Configuration

Mode

Slave Mode: Disable
Trigger Source: Disable
Clock Source: Internal Clock
Channel1: Disable
Channel2: Disable
Channel3: Disable
Channel4: Disable
Channel5: Disable
Channel6: Disable

Configuration

Counter Settings

Prescaler (PSC - 16 bits value)	216
Counter Mode	Up
Counter Period (AutoReload Register - 16 bits ...	10000
Internal Clock Division (CKD)	No Division
Repetition Counter (RCR - 16 bits value)	0
auto-reload preload	Disable

Trigger Output (TRGO) Parameters

Master/Slave Mode (MSM bit)	Disable (Trigger input effect not delayed)
Trigger Event Selection TRGO	Reset (UG bit from TIMx_EGR)
Trigger Event Selection TRGO2	Reset (UG bit from TIMx_EGR)

FIGURE 2: TIMER CONFIGURATION ON CUBEMX

Now, we have to configure the clock tree for the board, in this example I chose to use the maximum frequency of the CPU (216MHz); the configuration can be found in Figure 3.

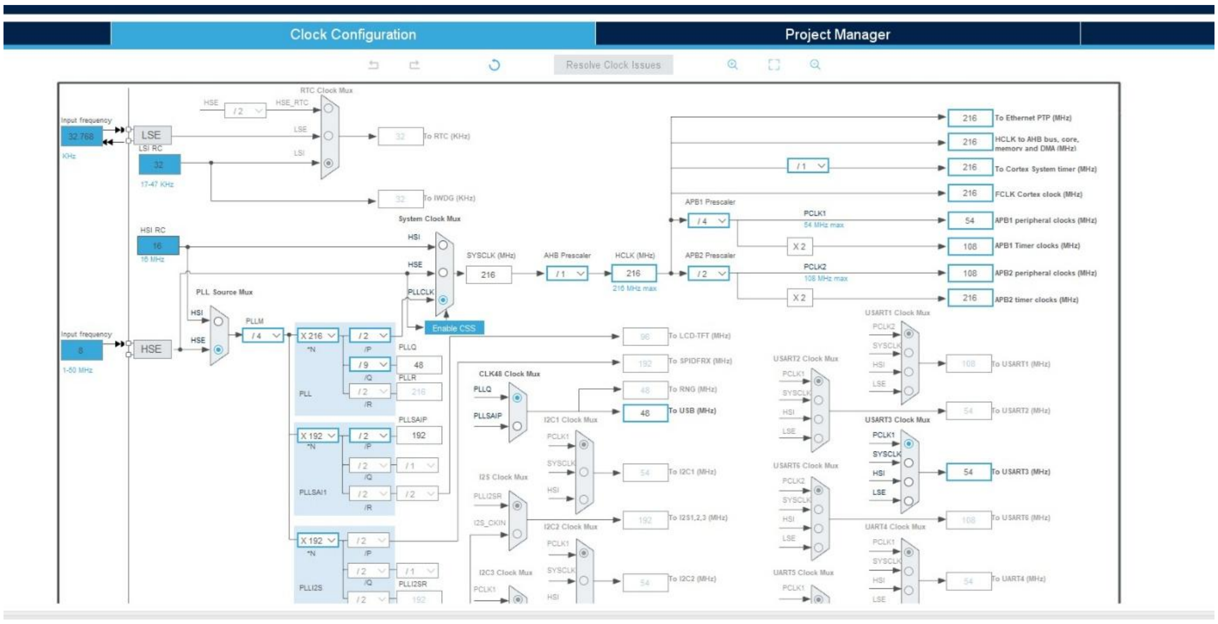
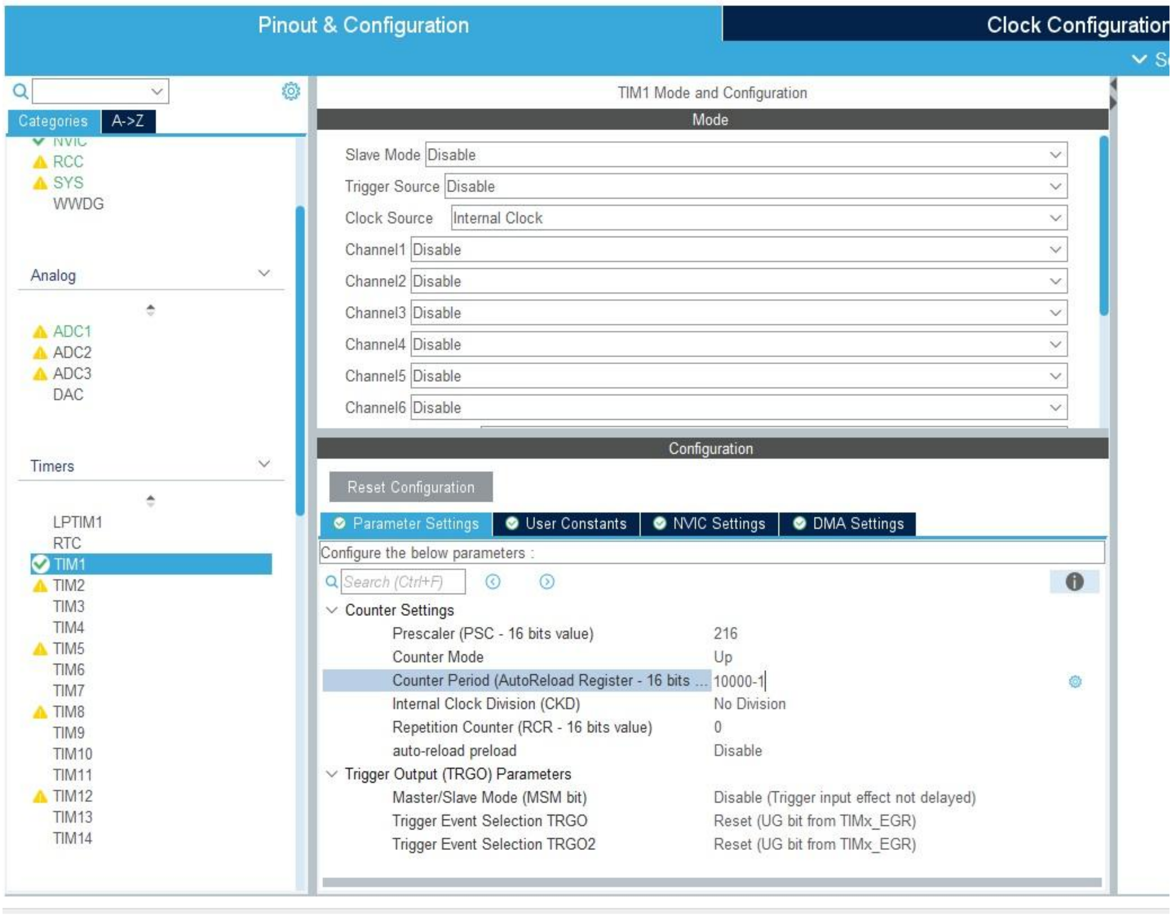


FIGURE 3: CONFIGURATION OF THE CLOCK TREE OF THE BOARD

The last step of the configuration is to configure the timer 1; to do that click on the timer1 panel. Now we want to scan the input pin every 10 millisecond so, we have to configure the right value in the Prescaler and in the Counter Period. The base time of the timer is compute with the frequency of the bus timer and the value of the prescaler. The timer 1 is connected to the APB2 bus so here the frequency of the bus is 216Mhz. So the counter period will be calculated by the following equation:

$$sampling\ period = \frac{1}{\frac{F_{timer}}{Prescaler\_value}} * Counter\ period$$

As F_{timer} is equal to 216Mhz and as the prescaler value is chosen to be equal to 216 then the counter period will be equal to (10000-1) in order to obtain a sampling period of 10 milliseconds. The configuration of the timer is shown in Figure 4.


TIM1 Mode and Configuration

Mode

Slave Mode	Disable
Trigger Source	Disable
Clock Source	Internal Clock
Channel1	Disable
Channel2	Disable
Channel3	Disable
Channel4	Disable
Channel5	Disable
Channel6	Disable

Configuration

Configure the below parameters :

Counter Settings

Prescaler (PSC - 16 bits value)	216
Counter Mode	Up
Counter Period (AutoReload Register - 16 bits ...	10000-1
Internal Clock Division (CKD)	No Division
Repetition Counter (RCR - 16 bits value)	0
auto-reload preload	Disable

Trigger Output (TRGO) Parameters

Master/Slave Mode (MSM bit)	Disable (Trigger input effect not delayed)
Trigger Event Selection TRGO	Reset (UG bit from TIMx_EGR)
Trigger Event Selection TRGO2	Reset (UG bit from TIMx_EGR)

FIGURE 4: TIMER CONFIGURATION IN CUBEMX

Now we are going to configure the timer in order to obtain an interruption when the timer reaches a period of 10 milliseconds. To do that we have only to enable the TIM1 update interrupt as shown in Figure 5.

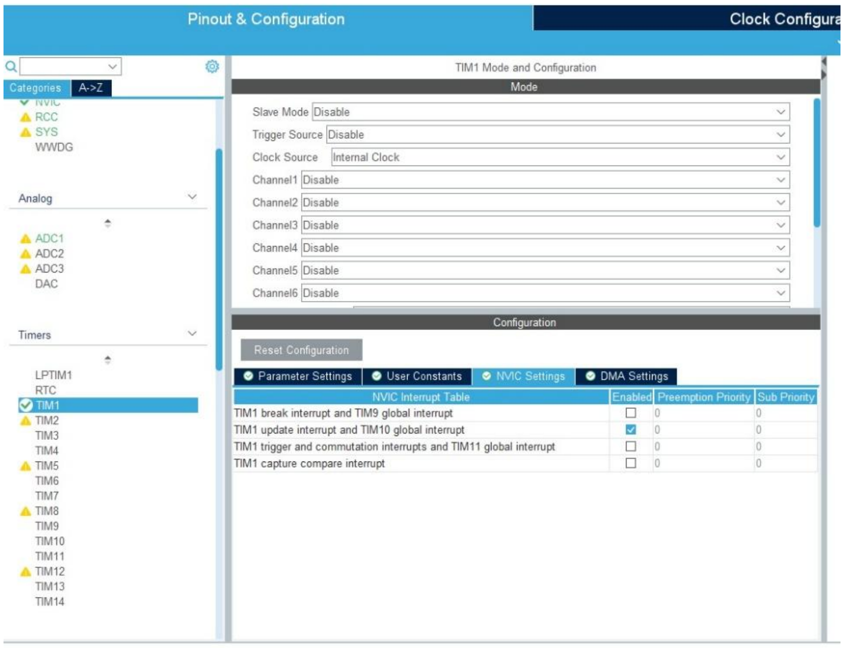
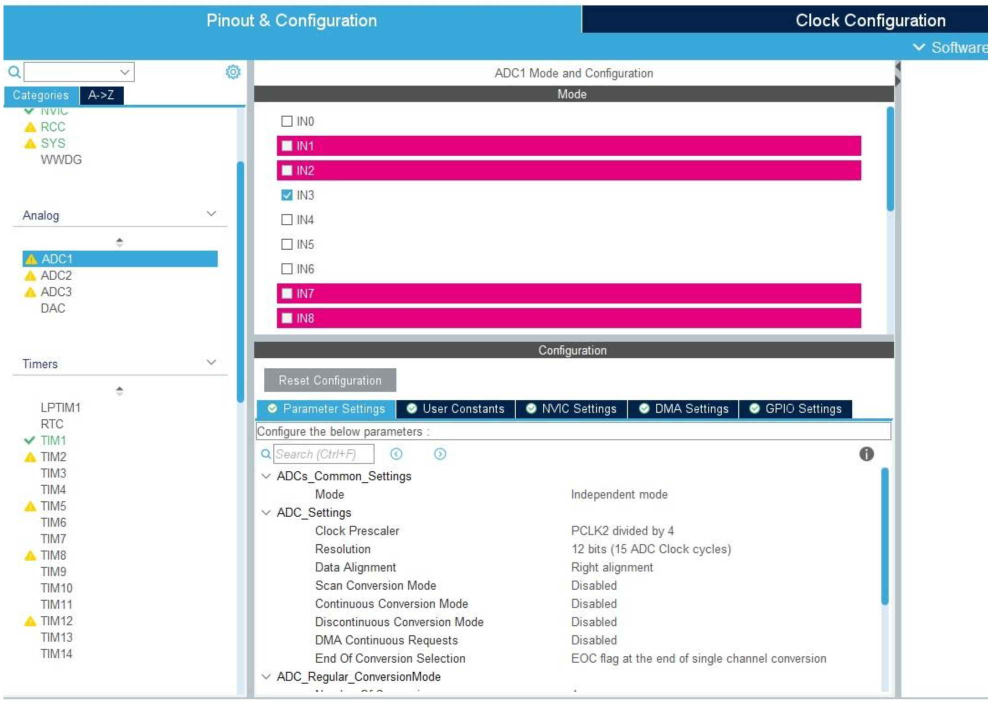

NVIC Interrupt Table	Enabled	Preemption Priority	Sub Priority
TIM1 break interrupt and TIM9 global interrupt	☐	0	0
TIM1 update interrupt and TIM10 global interrupt	✓	0	0
TIM1 trigger and commutation interrupts and TIM11 global interrupt	☐	0	0
TIM1 capture compare interrupt	☐	0	0

FIGURE 5: TIMER INTERRUPT CONFIGURATION

The next step is to configure the ADC. To do that, click on the ADC1 in the system view of CubeMX .



ADCs_Common_Settings

Mode	Independent mode

ADC_Settings

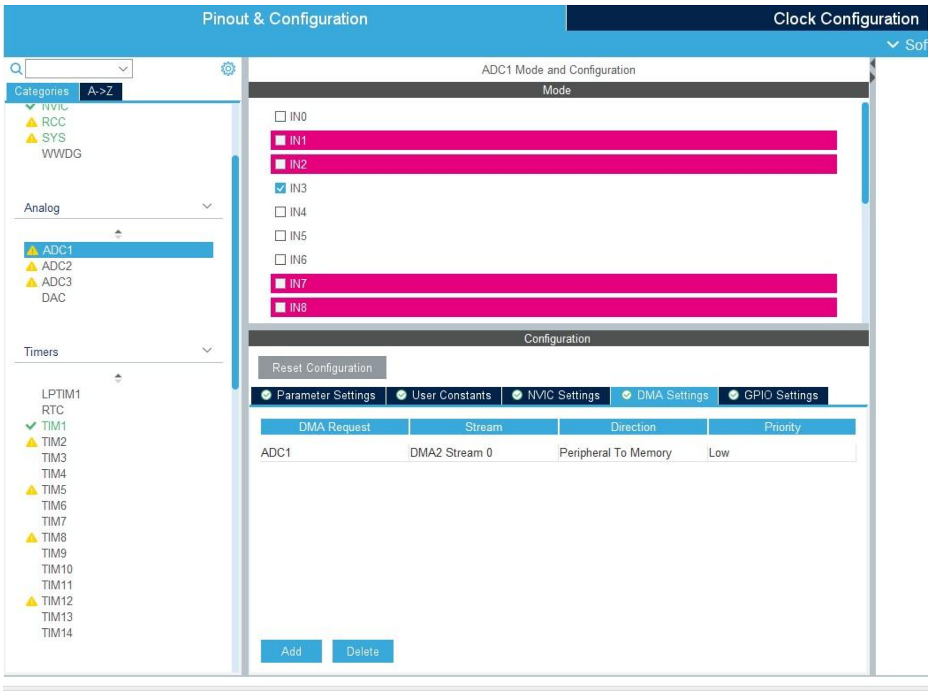
Clock Prescaler	PCLK2 divided by 4
Resolution	12 bits (15 ADC Clock cycles)
Data Alignment	Right alignment
Scan Conversion Mode	Disabled
Continuous Conversion Mode	Disabled
Discontinuous Conversion Mode	Disabled
DMA Continuous Requests	Disabled
End Of Conversion Selection	EOC flag at the end of single channel conversion

ADC_Regular_ConversionMode

FIGURE 6: ADC CONFIGURATION PANEL

Here we use the ADC in independent mode (other modes more complex are possible see the datasheet of the component). Also, we must choose the resolution of the ADC. Indeed, the ADC can use 12, 10, 8 or 6 bits for the resolution. More the resolution is high and more the precision of the conversion is high but the necessary time to realize the conversion increases. For this example, I use 12 bits for the resolution. We must choose the data alignment for the converted value since the maximum resolution is 12 bits and the data register is 16 bits wide. I choose the right alignment so the value of the conversion uses the 12 LSB. As in this example, I do not want to scan more than one input I disabled the scan mode and the continuous conversion mode; the conversion will be done only if the timer interruption is generated. The ADC will generate an interruption as soon as the conversion will be ended so in the End of Conversion Selection, the EOC flag is selected. Now we have to enable the interruption of the ADC; to do that click on the NVIC Settings and enables the interruption. The interrupt level can be the same as that of the timer, but in practice it is preferable for the timer to have higher priority, since it is the timer that generates the sampling frequency.

As in this example, we want to use the DMA to store the converted value in memory without using the ADC driver function (Get_ADC_Value), so we need to configure the DMA. We therefore need to add a DMA channel to connect the ADC to the memory. To do this, simply go to the DMA settings tab. The use of the DMA will generate an interruption every ADC conversion as soon as the transfer between the ADC and the memory will be finish.

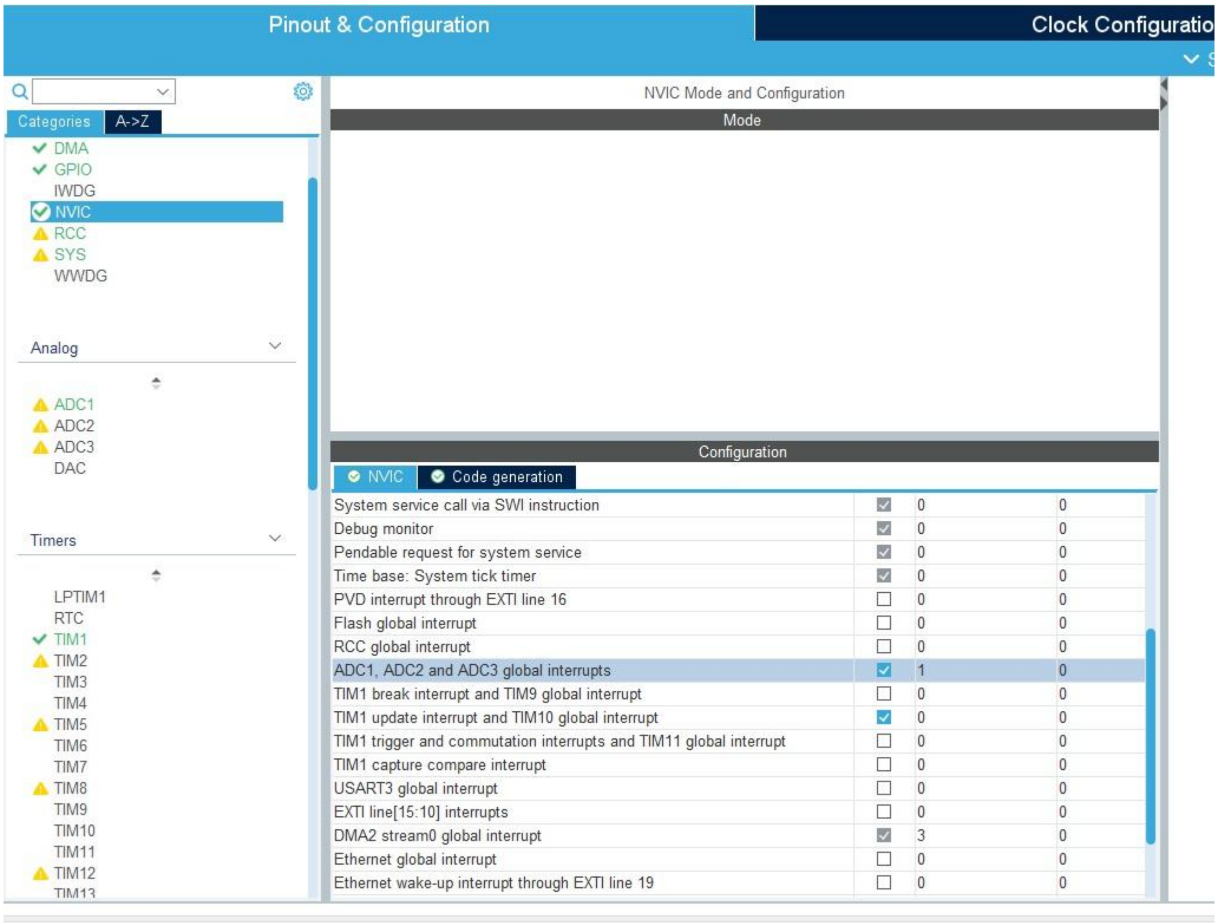


☐ IN0
☐ IN1
☐ IN2
☑ IN3
☐ IN4
☐ IN5
☐ IN6
☐ IN7
☐ IN8

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

FIGURE 7: DMA CONFIGURATION FOR THE ADC

Now we have three interruptions in our code so we must change their priorities in order to have no conflict between them. So we choose, in the system view, the NVIC.

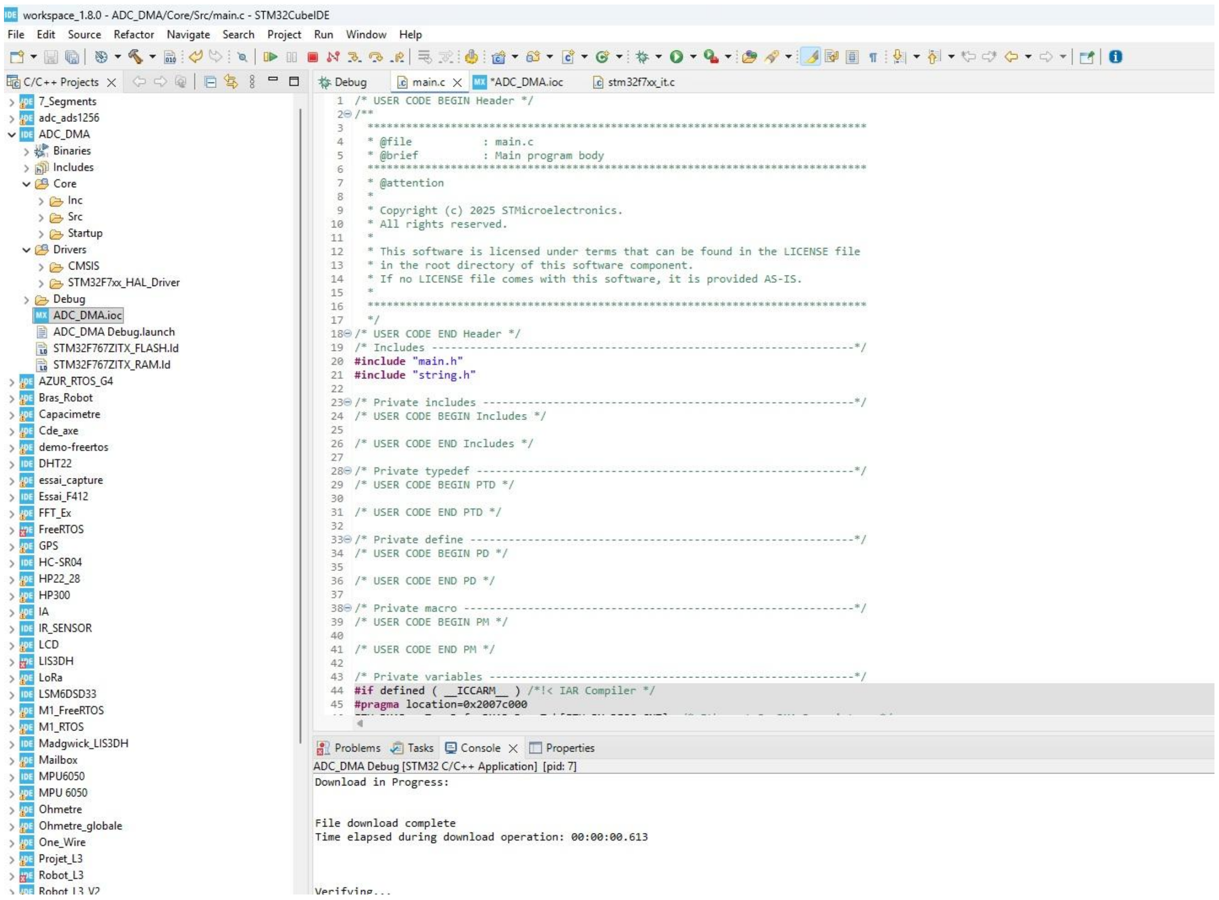


System service call via SWI instruction	✓	0	0
Debug monitor	✓	0	0
Pendable request for system service	✓	0	0
Time base: System tick timer	✓	0	0
PVD interrupt through EXTI line 16	☐	0	0
Flash global interrupt	☐	0	0
RCC global interrupt	☐	0	0
ADC1, ADC2 and ADC3 global interrupts	✓	1	0
TIM1 break interrupt and TIM9 global interrupt	☐	0	0
TIM1 update interrupt and TIM10 global interrupt	✓	0	0
TIM1 trigger and commutation interrupts and TIM11 global interrupt	☐	0	0
TIM1 capture compare interrupt	☐	0	0
USART3 global interrupt	☐	0	0
EXTI line[15:10] interrupts	☐	0	0
DMA2 stream0 global interrupt	✓	3	0
Ethernet global interrupt	☐	0	0
Ethernet wake-up interrupt through EXTI line 19	☐	0	0

FIGURE 8: NVIC PANEL

In the NVIC panel, we can see all the interruption of our system and in particular, the interruption of the Timer 1, of the ADC and of the DMA. Here, we will configure the preemption priority of the TIM1 to level 0, to level 1 for the ADC and to level 3 for the DMA. More the value is high and less is the priority so in this case the timer will have an interruption more priority than the ADC or the DMA.

Now that all our devices are configured, we need to generate the project code. To do this, simply click on the Device Configuration Tool Code Generation icon.



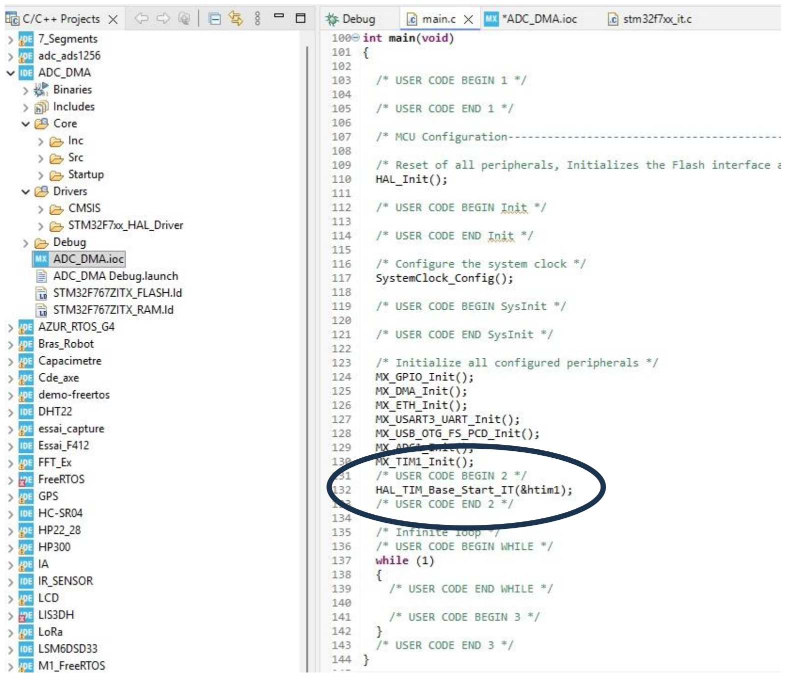
```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * Copyright (c) 2025 STMicroelectronics.
  * All rights reserved.
  *
  * This software is licensed under terms that can be found in the LICENSE file
  * in the root directory of this software component.
  * If no LICENSE file comes with this software, it is provided AS-IS.
  *
  ******************************************************************************
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "string.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */

/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */

/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
#if defined ( __ICCARM__ ) /*!< IAR Compiler */
#pragma location=0x2007c000
```

```
ADC_DMA Debug [STM32 C/C++ Application] [pid: 7]
Download in Progress:


File download complete
Time elapsed during download operation: 00:00:00.613


Verifying...
```

FIGURE 9: STM32CUBEIDE FRAMEWORK

The second part of the development is to program the start of the timer to generate a sampling period in order to launch the ADC conversion to acquire the value of the analog sensor.

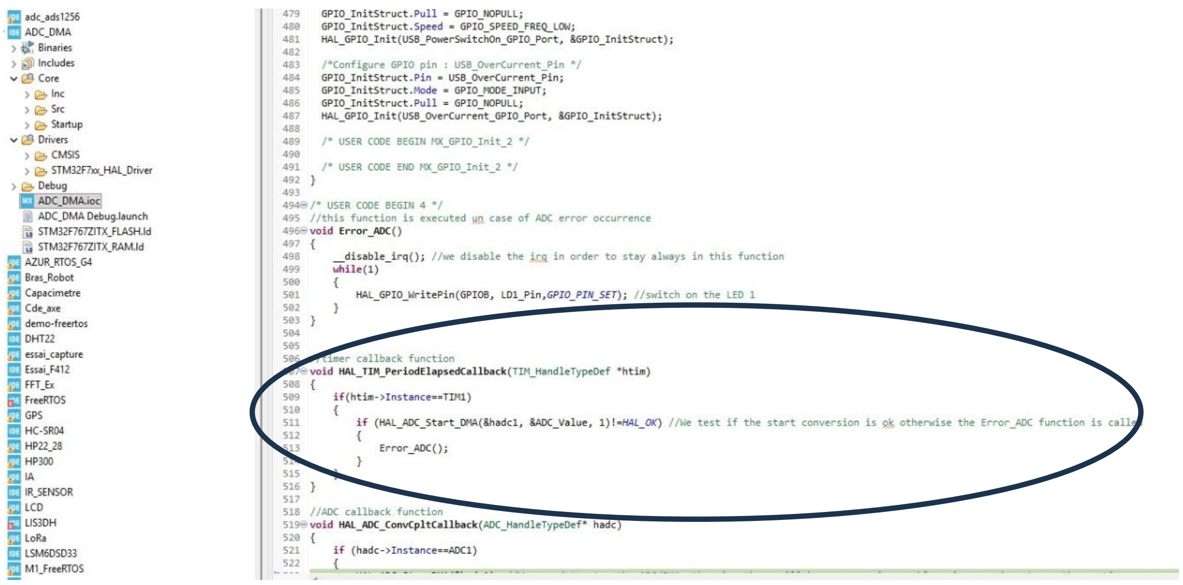
The first step is to start the timer 1 in order to obtain a sampling period of 10 milliseconds; this step is realized by starting the timer 1 in interrupt mode as shown in Figure 10. All the Timer functions API can be found in the HAL_tim.c file.



```c
int main(void)
{

  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration-----------------------------------------

  /* Reset of all peripherals, Initializes the Flash interface a
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_DMA_Init();
  MX_ETH_Init();
  MX_USART3_UART_Init();
  MX_USB_OTG_FS_PCD_Init();
  MX_ADC1_Init();
  MX_TIM1_Init();
  /* USER CODE BEGIN 2 */
  HAL_TIM_Base_Start_IT(&htim1);
  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}
```

FIGURE 10: LAUNCHING OF THE TIMER

Now, the timer will generate an interruption every 10 milliseconds so we have to write the code in order to start the ADC conversion. This function must be coded in the Timer_1 interruption routine. I also define a global variable named ADC_Value (uint32_t in the main.c file) to store via the DMA the converted value of the ADC.

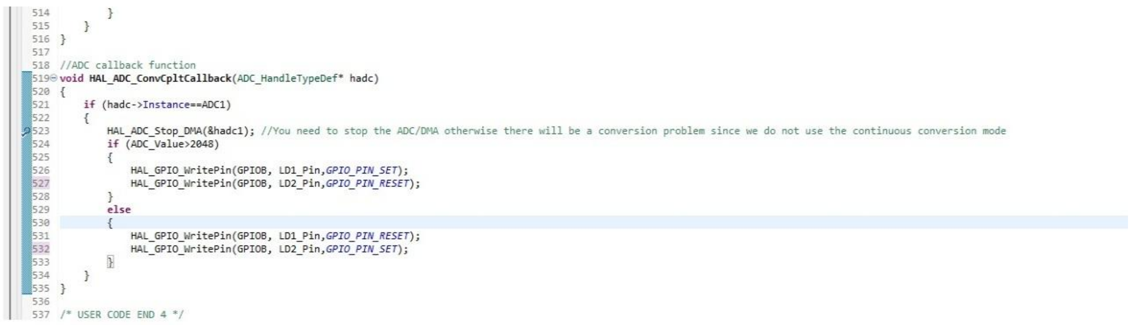


```c
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(USB_PowerSwitchOn_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pin : USB_OverCurrent_Pin */
  GPIO_InitStruct.Pin = USB_OverCurrent_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(USB_OverCurrent_GPIO_Port, &GPIO_InitStruct);

  /* USER CODE BEGIN MX_GPIO_Init_2 */

  /* USER CODE END MX_GPIO_Init_2 */
}

/* USER CODE BEGIN 4 */
//this function is executed un case of ADC error occurrence
void Error_ADC()
{
    __disable_irq(); //we disable the irq in order to stay always in this function
    while(1)
    {
        HAL_GPIO_WritePin(GPIOB, LD1_Pin,GPIO_PIN_SET); //switch on the LED 1
    }
}


//timer callback function
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
    if(htim->Instance==TIM1)
    {
        if (HAL_ADC_Start_DMA(&hadc1, &ADC_Value, 1)!=HAL_OK) //We test if the start conversion is ok otherwise the Error_ADC function is calle
        {
            Error_ADC();
        }
    }
}

//ADC callback function
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)
{
    if (hadc->Instance==ADC1)
    {
```

FIGURE 11:CODE TO START THE ADC CONVERSION AND THE STORAGE OF THE CONVERTED VALUE VIA THE DMA

In the Callback function of the timer, I test first if it's the Timer1 that has generated the interruption. If the condition is true thus I start the ADC conversion and the storage of the converted value via the DMA; I test whether the function is working properly, and if not, then we call the error function. You may not code this error function because it is not necessary for the application. As soon as the ADC conversion is finished, the ADC generates an interruption so in the Callback function of the

ADC, I compare the converted value to a threshold and if this value is higher than the threshold I switch on the LD1 and switch off the LD2 or the reversal otherwise.



```c
            }
        }
}

//ADC callback function
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)
{
    if (hadc->Instance==ADC1)
    {
        HAL_ADC_Stop_DMA(&hadc1); //You need to stop the ADC/DMA otherwise there will be a conversion problem since we do not use the continuous conversion mode
        if (ADC_Value>2048)
        {
            HAL_GPIO_WritePin(GPIOB, LD1_Pin,GPIO_PIN_SET);
            HAL_GPIO_WritePin(GPIOB, LD2_Pin,GPIO_PIN_RESET);
        }
        else
        {
            HAL_GPIO_WritePin(GPIOB, LD1_Pin,GPIO_PIN_RESET);
            HAL_GPIO_WritePin(GPIOB, LD2_Pin,GPIO_PIN_SET);
        }
    }
}

/* USER CODE END 4 */
```

FIGURE 12: ADC CALLBACK FUNCTION WHERE THE CONVERTED VALUE IS COMPARED WITH A THRESHOLD

It should be noted that the ADC conversion must be stopped in the callback function otherwise you will be a problem since the ADC is not configurated in continuous mode.