

Use of ADC on Microcontroller

Targeted competences: Use of ADC in standard and scan mode.

Hardware: STM32F7 Nucleo board

Framework: Atollic TrueStudio and CubeMX from STMicroelectronics

The aim of this example is to study how use an ADC in standard and in scan mode. 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 a second time, we will use the ADC in scan mode in order to convert more than one analog input.

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 create the CubeMx project for the board that we will use. Here we use the NUCLEO STM32F767ZI board. We choose to use the PA4 pin in ADC1_IN4 (See Figure 1) that means that the pin PA4 is linked to the input channel 4 of the ADC1.

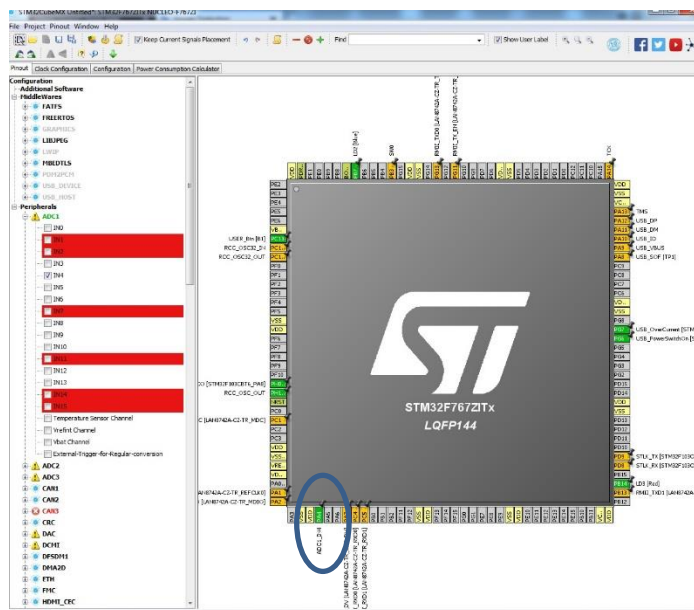


FIGURE 1: CONFIGURATION OF THE GPIO PIN

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). I also configure the RCC in order to use the HSE (High Speed Clock).

As F_{timer} is equal to 216Mhz and as the prescaler value is chosen to be equal to 65000 then the counter period will be equal to 3334 in order to obtain a sampling period of 1 second. The configuration of the timer is shown in Figure 5.

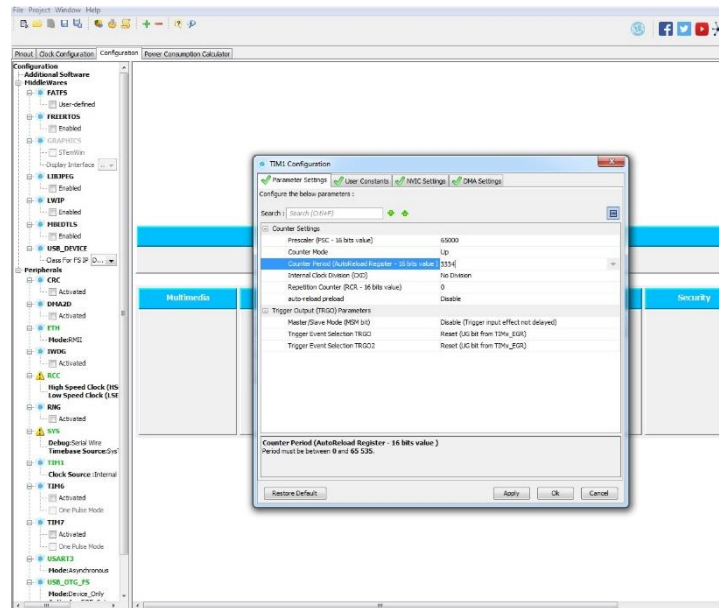


FIGURE 4: TIMER CONFIGURATION IN CUBEMX

Now we are going to configure the timer in order to obtain an interruption when the timer reaches a period of 1 second. To do that we have only to enable the TIM1 update interrupt as shown in Figure 6.

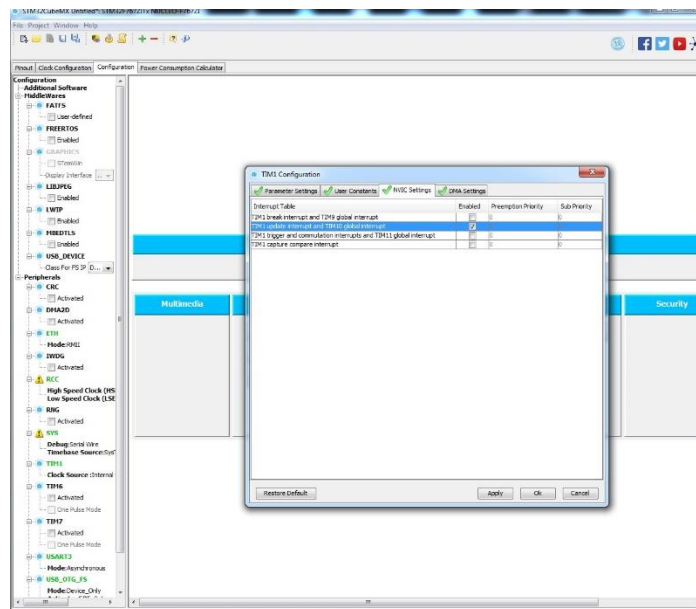


FIGURE 5: TIMER INTERRUPT CONFIGURATION

As we have only one interruption in the code so we do not need to configure the NVIC (Nested Vector Interrupt Controller) but if in your code, you have more than one interruption it is necessary to do it in order to obtain different level of priorities between the different interruptions.

The next step is to configure the ADC. To do that, click on the ADC1 in the system view of CubeMX (Attention I have updated the version of CubeMX so the graphic interface changed).

Now we have two 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.

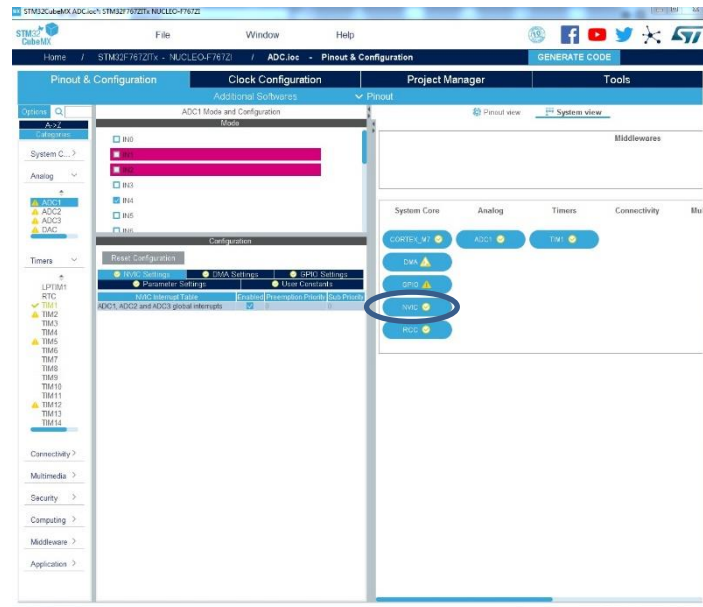


FIGURE 8: NVIC PANEL

In the NVIC panel, we can see all the interruption of our system and in particular, the interruption of the Timer 1 and of the ADC. Here, we will configure the preemption priority of the TIM1 to level 1 and to level 2 for the ADC. 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.

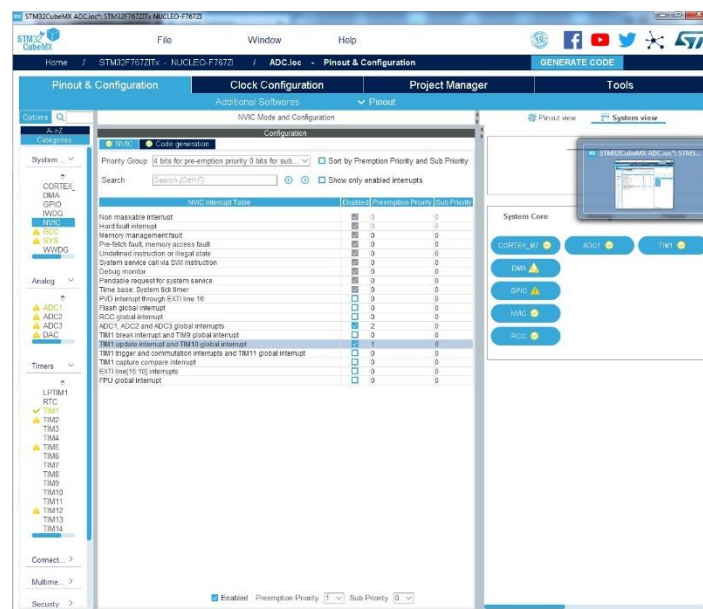


FIGURE 9: LEVEL OF PRIORITY FOR THE TIMER AND ADC

Now, we are going to generate the project code; to do that click on Project Manager and configure the generator as presented in Figure 10.

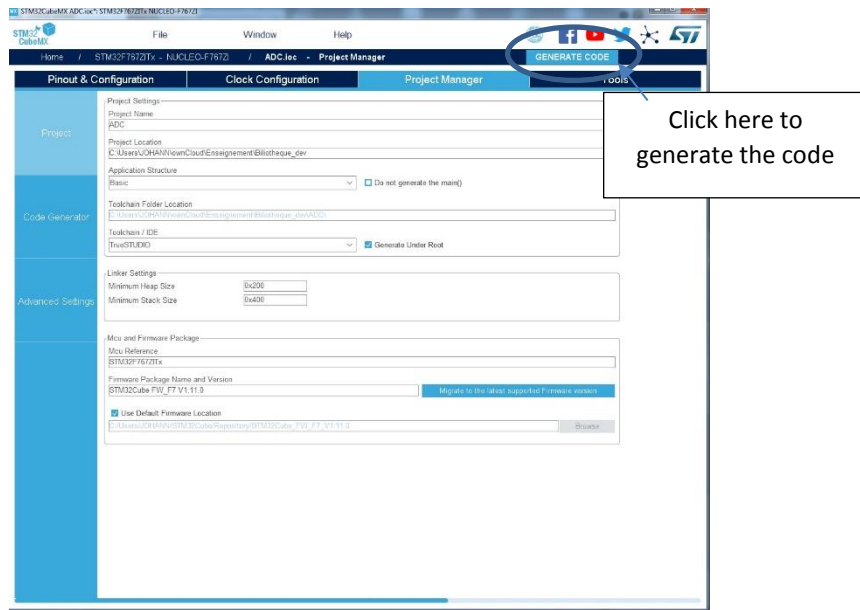


FIGURE 10: CONFIGURATION OF CODE SOURCE GENERATOR

In this example, I choose to use TrueStudio as IDE but you can choose another IDE if you want.

When the code is generated, the TrueStudio framework opens and we can see that the different source codes have been created (see Figure 11).

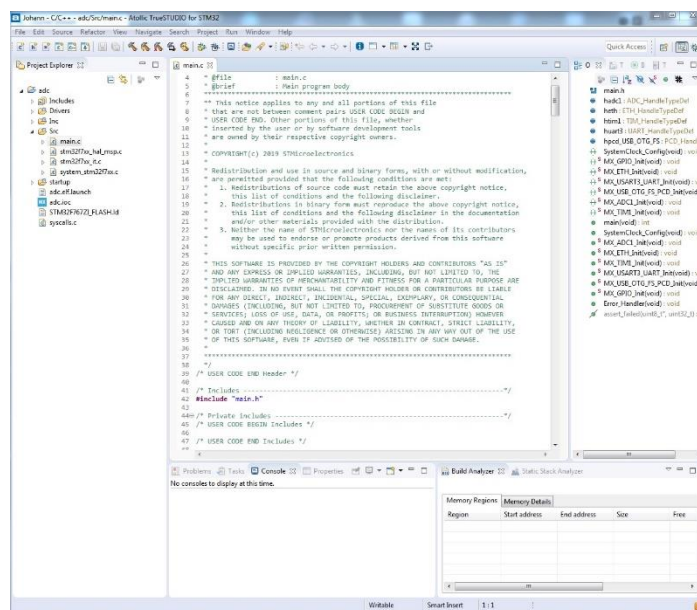


FIGURE 11: TRUE STUDIO FRAMEWORK

The second part of the development is to start 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 1 second; this step is realized by starting the timer 1 in interrupt mode as shown in Figure 12. All the Timer functions API can be found in the HAL_tim.c file.

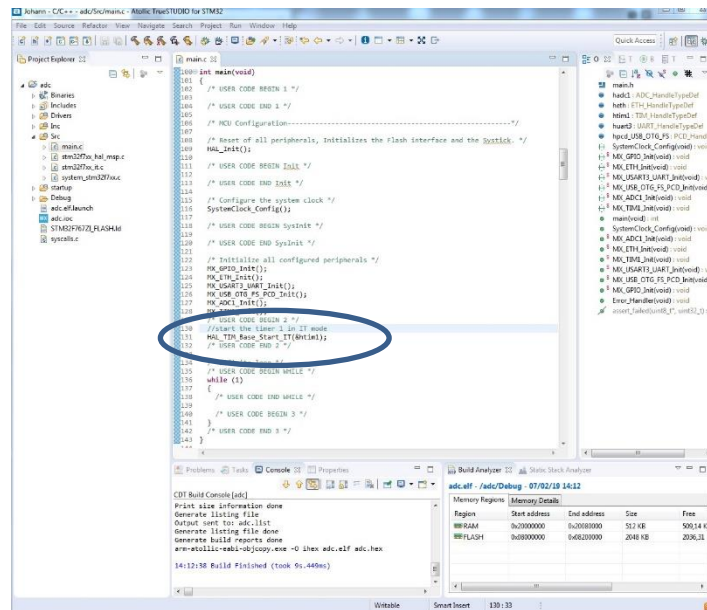


FIGURE 12: LAUNCHING OF THE TIMER

Now, the timer will generate an interruption every second 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 (uint16_t in the main.c file) to store the converted value of the ADC. In the stm32f7xx_it.c file this variable is defined as extern. To know if the value is greater than a threshold I switch on a led (LD2 on the board) and otherwise I switch on another led (LD3 on the board).

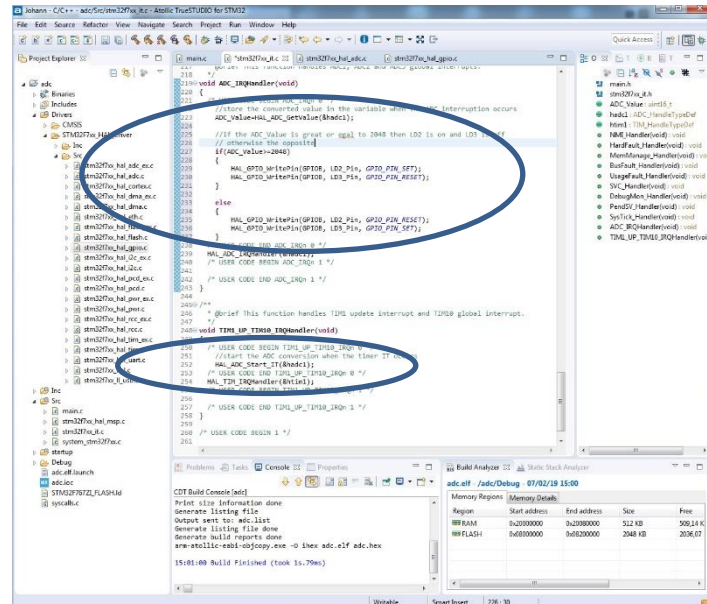


FIGURE 13: CODE TO START THE TIMER AND THE ADC, TO STORE THE VALUE AND TO SWITCH ON THE LED

A video of the code behavior can be found here: <https://youtu.be/VfPhpfE9Rb4>

