Use of ADC in scan mode on Microcontroller

Targeted competences: Use of ADC in scan mode.

Hardware: STM32F7 Nucleo board

Framework: Atollic TrueStudio and CubeMX from STMircoelectronics

The aim of this example is to study how use an ADC in scan mode. In a first time, we use a sampling period in order to just convert an analog value comes from GPIOs configured in analog mode. Then, 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 and to use the PA5 pin in ADC1_IN5.

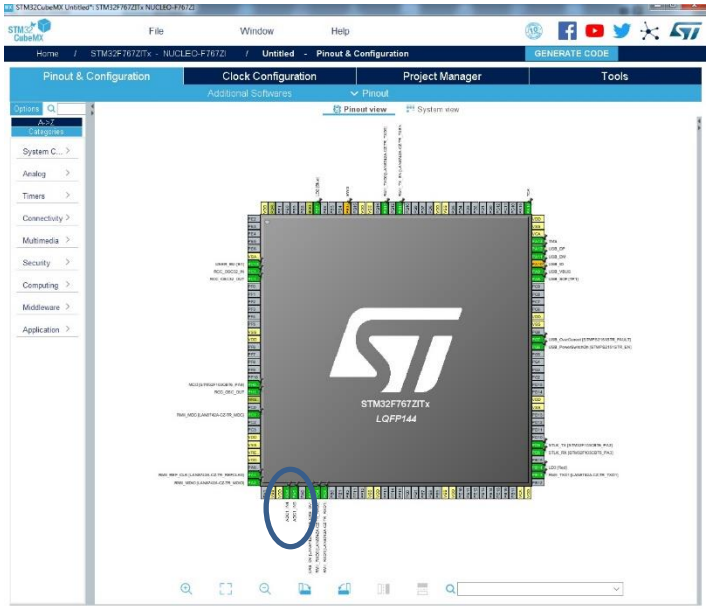

FIGURE 1: CONFIGURATION OF THE GPIO PINS

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).

FIGURE 2: TIMER CONFIGURATION ON CUBEMX

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 all the second 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 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 3.

FIGURE 3: 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 4.

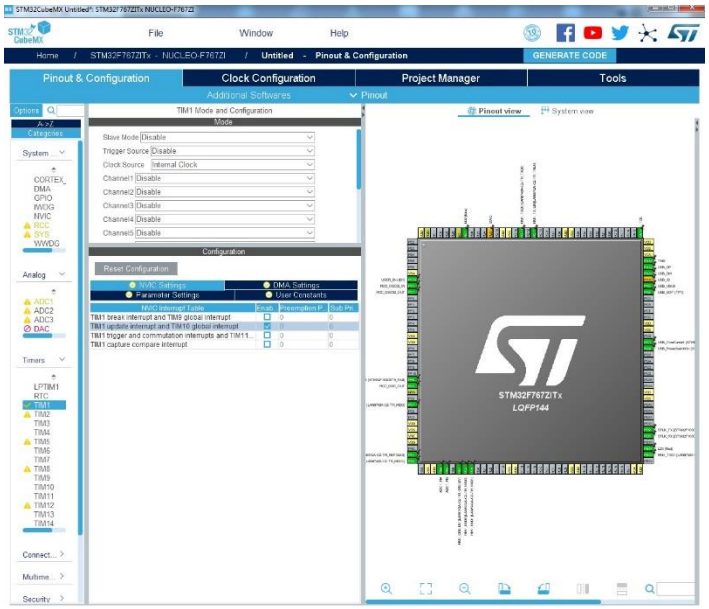

FIGURE 4: TIMER INTERRUPT CONFIGURATION

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.

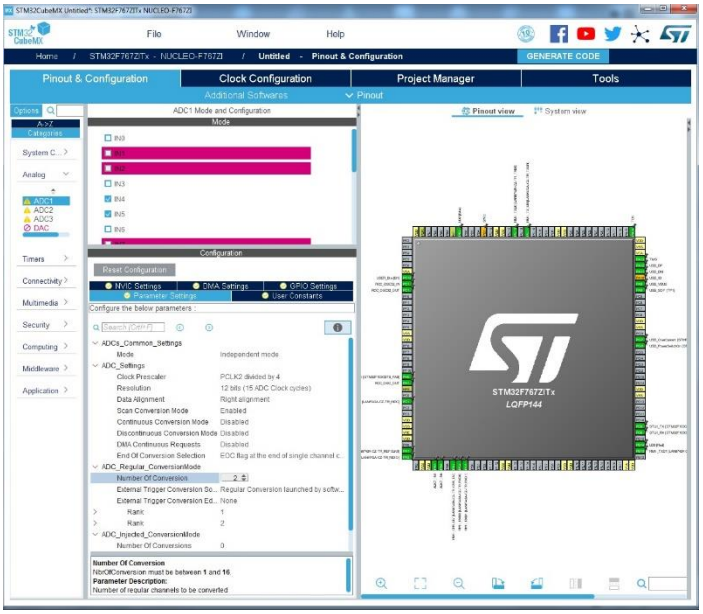

FIGURE 5: ADC CONFIGURATION PANEL

Here we use the ADC in independent mode (other modes more complexe are possible see the datasheet of the component). In addition, 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 want to scan more than one input I enable the scan mode and disable the continuous conversion mode; the conversion will be done only if the timer interruption is generated. The number of conversion must be equal to two since I want to scan two different inputs of the ADC. The maximum number could be 16 since the ADC owns 16 inputs.

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.

FIGURE 6: NVIC CONFIGURATION FOR THE ADC

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.

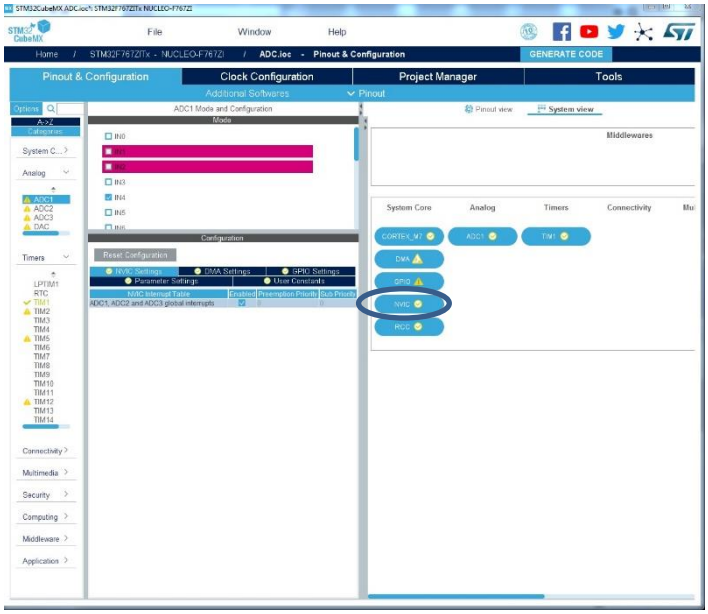

FIGURE 7: NVIC PANEL

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.

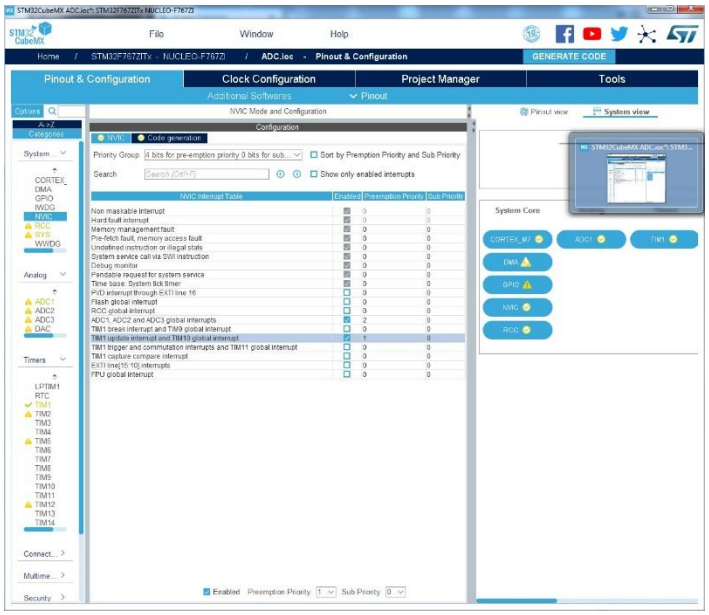

FIGURE 8: LEVEL OF PRIORITY FOR THE TIMER AND ADC

Now, we are going to generate the project code; to do that click on Project Manager and configure the generator as presented in Figure 9.

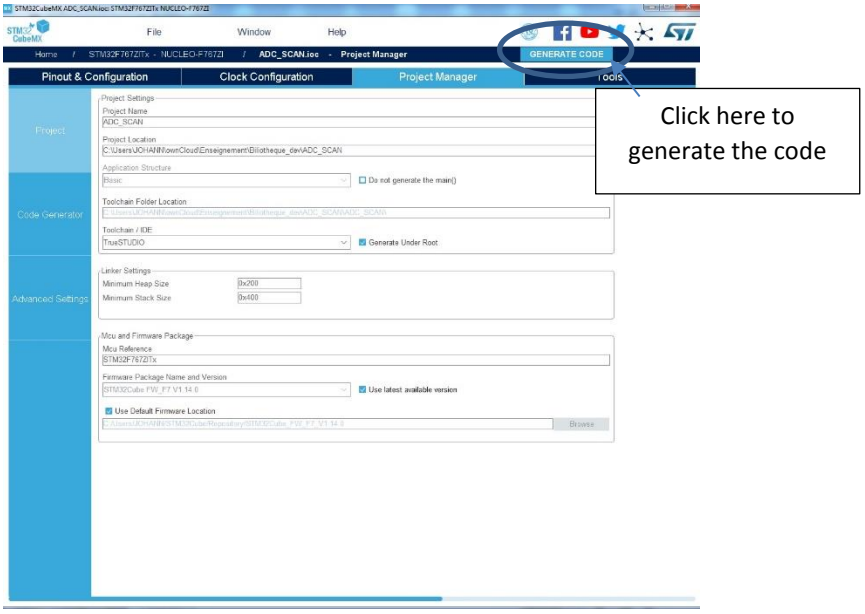

FIGURE 9: CONFIGURATION OF CODE SOURCE GENERATOR

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 10).

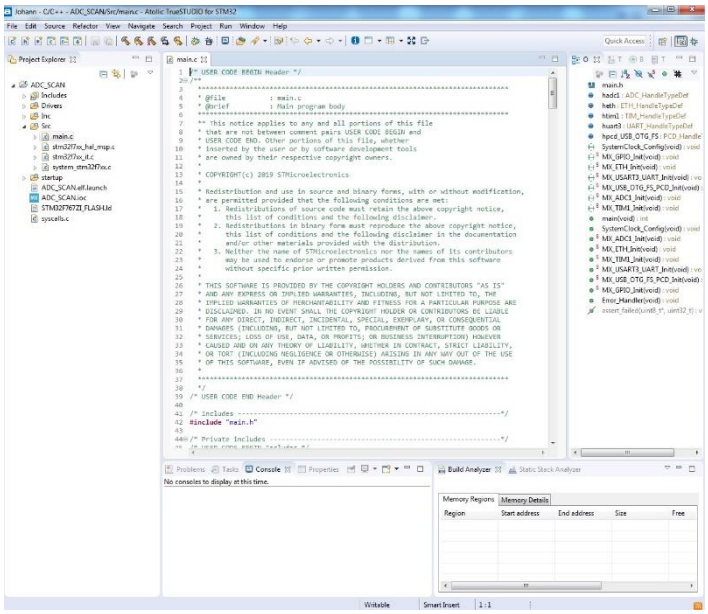

FIGURE 30: TRUE STUDIO FRAMEWORK

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 11. All the Timer functions API can be found in the HAL_tim.c file.

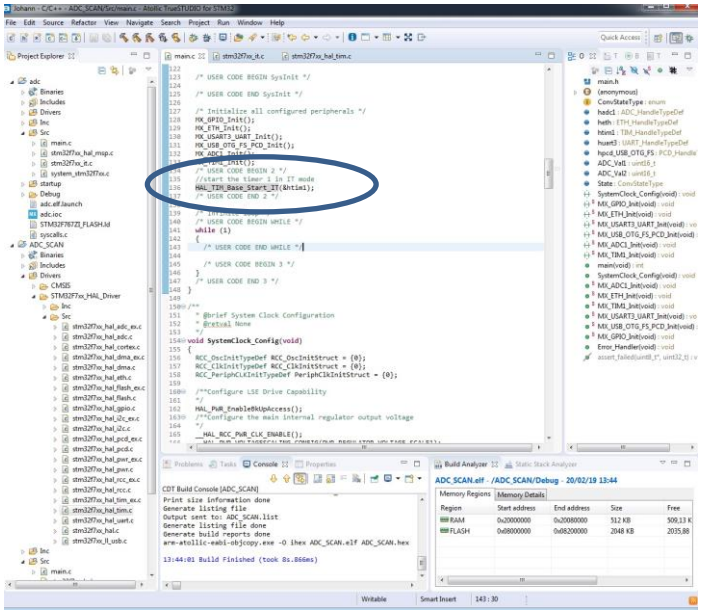

FIGURE 11: LAUNCHING OF THE TIMER

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 written in the Timer_1 interruption routine. I also define two global variables named respectively ADC_Val1 and ADC_Val2 (uint16_t in the main.c file) to store the converted values of the ADC. In the stm32f7xx_it.c file these variables are defined as extern. I also define a state variable (State) in order to know if the converted value is the one for the first or the second input. If the first value is less than a threshold 1 and the second is greater than a

threshold 2, I switch on the LD2 on the board and switch off the LD3 on the board. If the first value is greater than a threshold 1 and the second is less than a threshold 2, I switch off the LD2 on the board and switch on the LD3 on the board. If the first value is greater than a threshold 1 and the second is greater than a threshold 2, I switch on the LD2 on the board and switch on the LD3 on the board. Otherwise, the LD2 and LD3 are switched off.

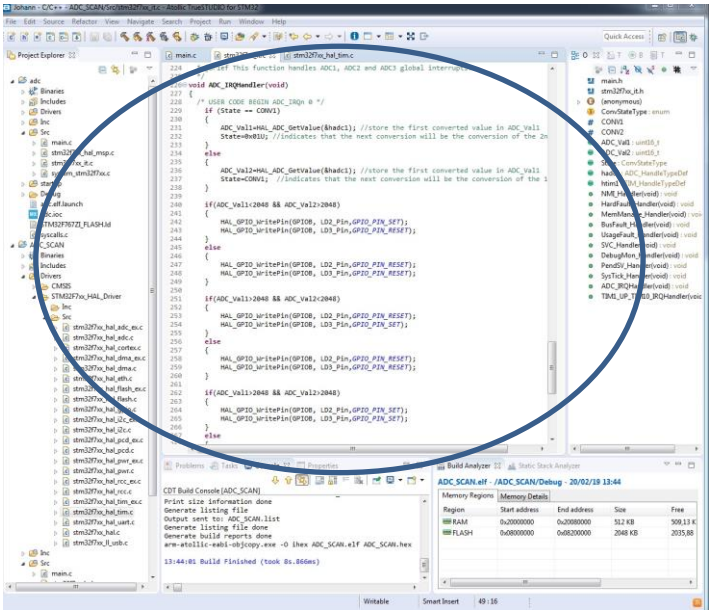

FIGURE 12: CODE TO STORE THE VALUES AND TO SWITCH ON THE LEDS