Use of Complete Analog Stream on Microcontroller

Targeted competences: Use of Digital to Analog converter, of Analog to Digital Converter and use of timer for sampling rate.

Hardware: STM32F7 Nucleo board

Framework: Keil µvision and CubeMX from STMircoelectronics

The aim of this document is to show how to use the DAC and ADC components in order to realize a complete analog acquisition stream. The Figure 1 presents the stream that we want to obtain. This is a classical way to get an analog input and to provide an analog output. Of course, in this document, we do not present all the ways whose the ADC and DAC components can be used but this is a first approach in order to be able to use this kind of component.

Figure 1: Acquisition Stream

To be able to get an analog input we can use an ADC; this ADC needs to have a sampling frequency in order to transform the continuous value (analog) into a discrete digital one. So, to obtain this sampling frequency we need to add a timer into this scheme. Furthermore, we need to provide the output signal regularly so this frequency will be also used for the output value. It is possible to use one timer for each component (ADC and DAC) but in several cases the output value will be provide at the same speed than the input one will be sampled even if a mathematical treatment will be realized between the two.

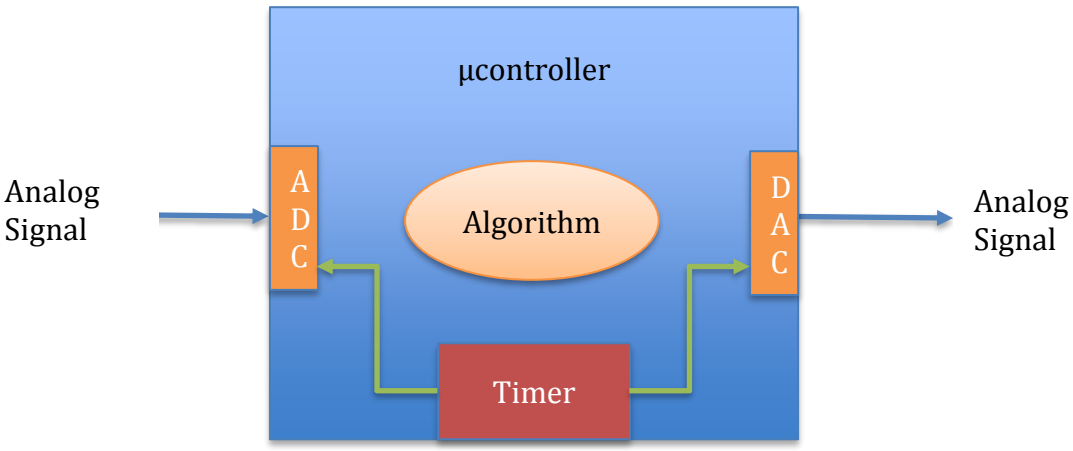

Figure 2: Global Scheme

1. Microcontroller configuration

The first step is to configure the microcontroller. In our case, we use the NUCLEO-F767ZI platform based on a STM32F7 architecture. In order to configure this board we will use the CubeMx software. After the right board was chosen on CubeMx, you can obtain this:

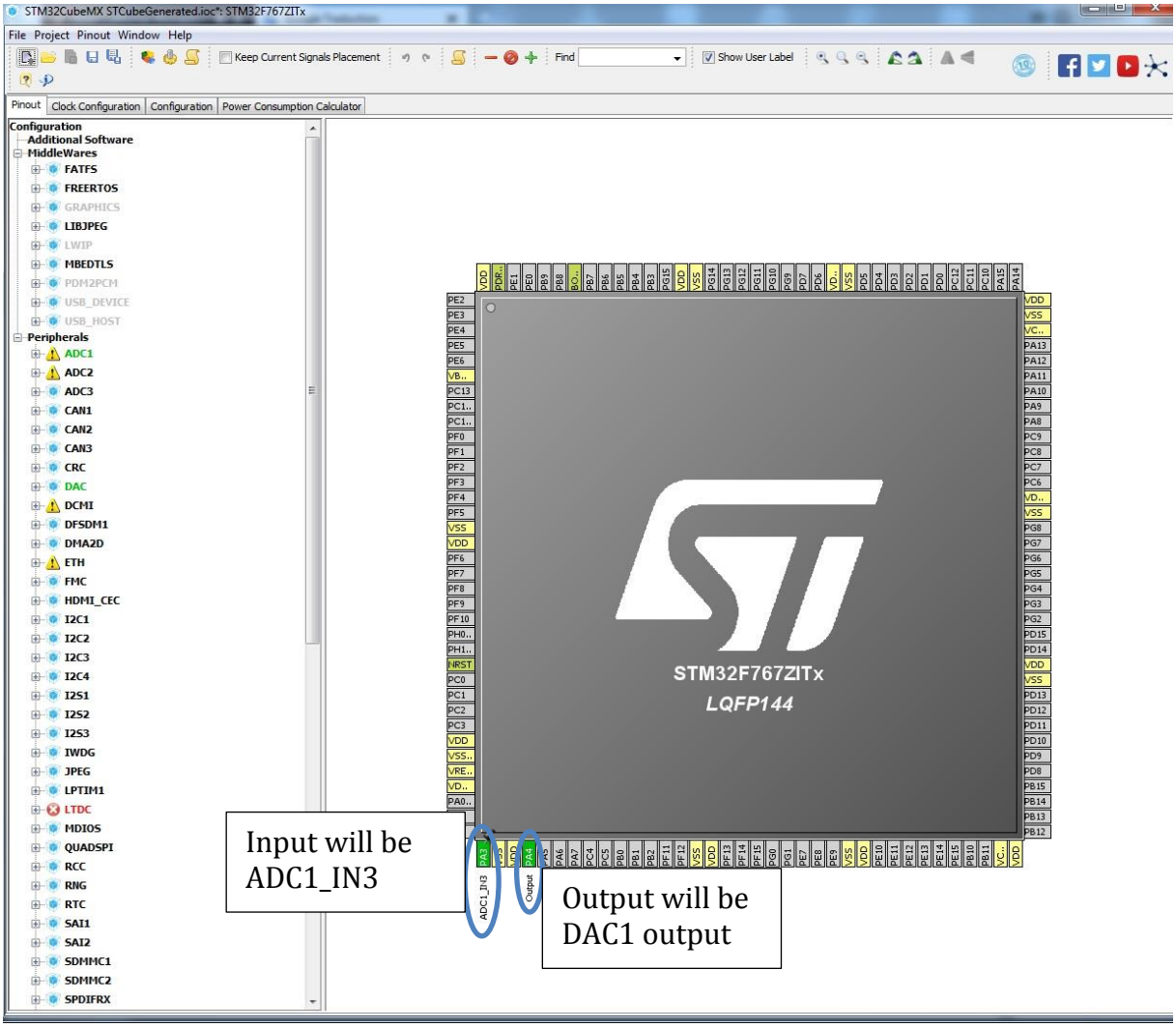

Figure 3: CubeMx configuration for ADC and DAC

Here, I have already chosen the ADC and DAC component that I would like to use. I decided that the output of the DAC will be the output of DAC1 so I configure the Pin PA4 as DAC_OUT1 by doing a left click on the Pin PA4. I have also configure the Pin PA3 as the input for the ADC1 by doing a left click on the Pin PA3 (ADC1_IN3).

The next step is to configure one timer to generate the sampling frequency. In this example, I chose to use the Timer Tim2 and I realized the configuration shown below (Figure 4).

You can see that I use the timer in its simplest mode because I would like only that this timer generate an interruption at the frequency F_e. So, we have only to choose the clock source for the timer; here we chose the internal clock.

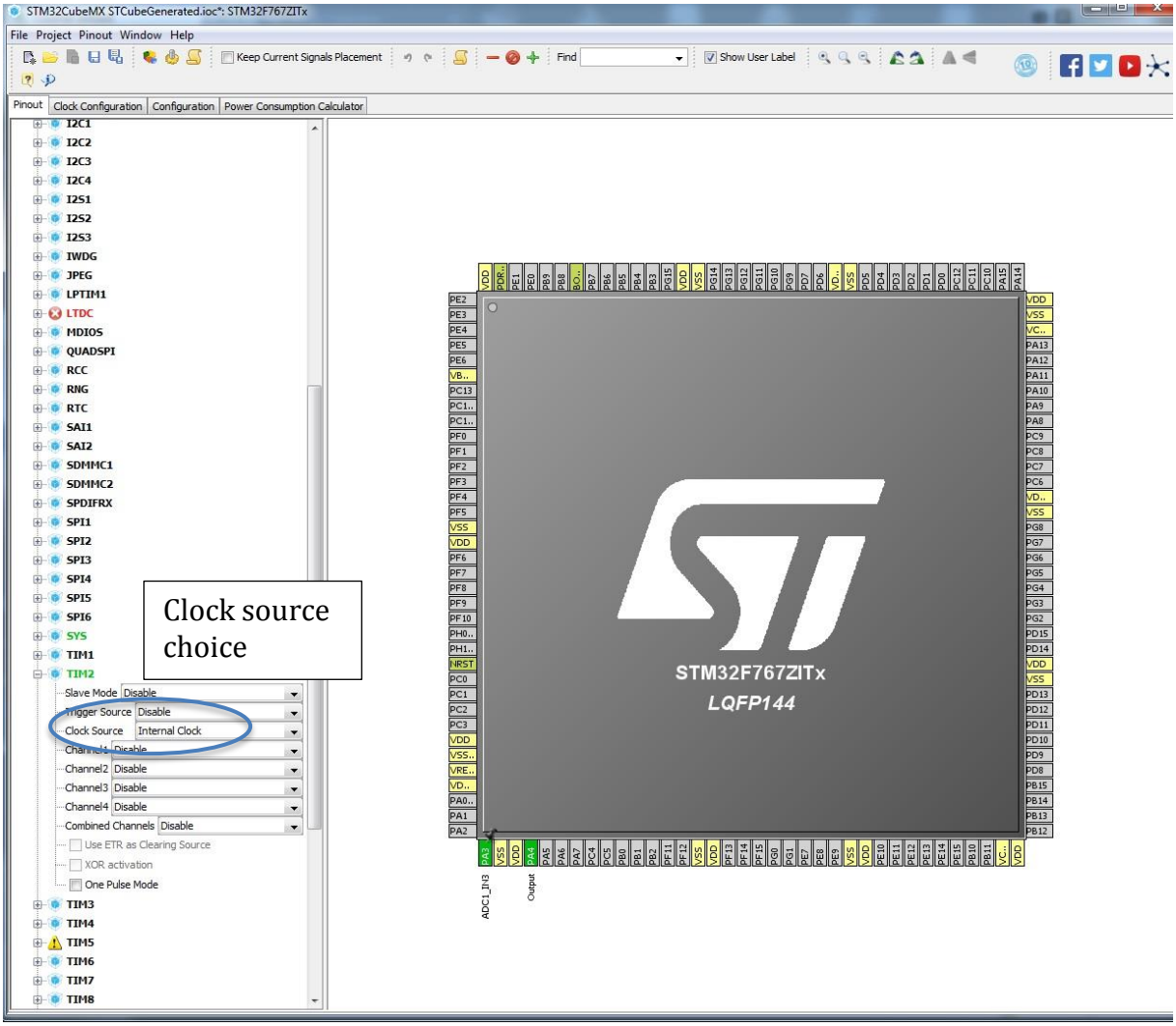

Figure 4: CubeMx configuration for Timer

Now we have to configure the clock for the whole board; to do this click on clock configuration item. With this configuration panel, we can choose the frequency of different µcontroller components as CPU frequency, AHB, APB1 buses and so on. I decide to use the maximum frequency of the SYSCLK that is 216MHz. In this case, the frequency of the APB1 and APB2 timer clocks are respectively 108MHz and 216MHz (see Figure 5). These frequencies will be very important in the next step when we will have to configure the sampling frequency.

Now we are going to configure the timer so that generates the sampling frequency. First click and configuration panel then on TIM2 now, we are going to determine the values for the prescaler and the counter period. The prescaler value divides the clock timer frequency in order to obtain a small one; it is useful if you want to have a small sampling frequency. Indeed, as the frequency in our case is 216 MHz and as the maximum count value is 4294967295 (32 bits timer) the minimum of the frequency scaling can be 0.05Hz. The prescaler parameter has more importance when you use a 16 bits timer.

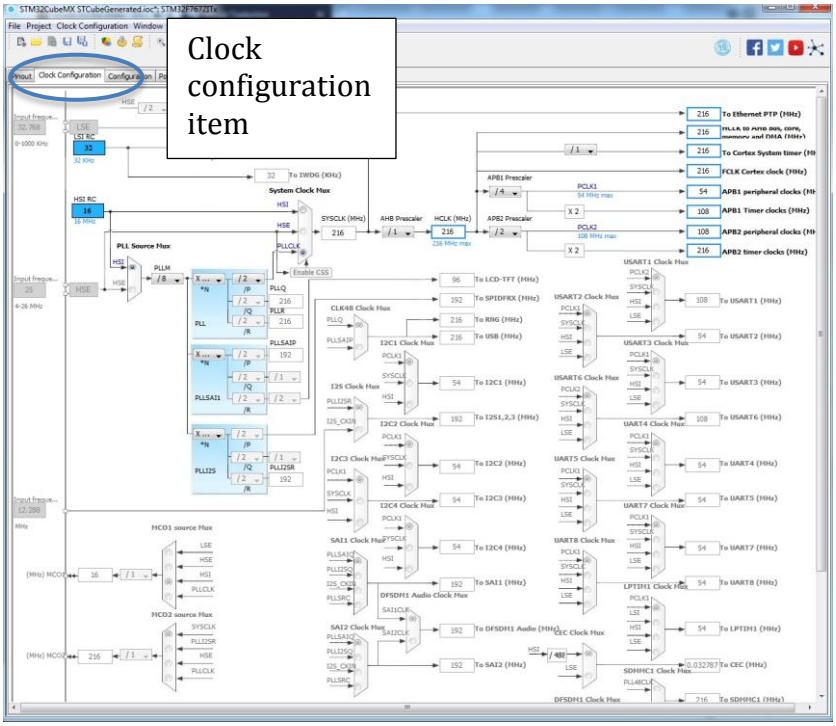

Figure 5: Clock configuration for the board

For this example, I would like a sampling frequency of 2.1Hz so I chose the following values: prescaler value 30000 (in this case, the clock frequency of the timer will be 7.2 KHz instead of 216 MHz), the counter period value 3323 (3323 ticks at 7.2 KHz represents a sampling period of 0.46s).

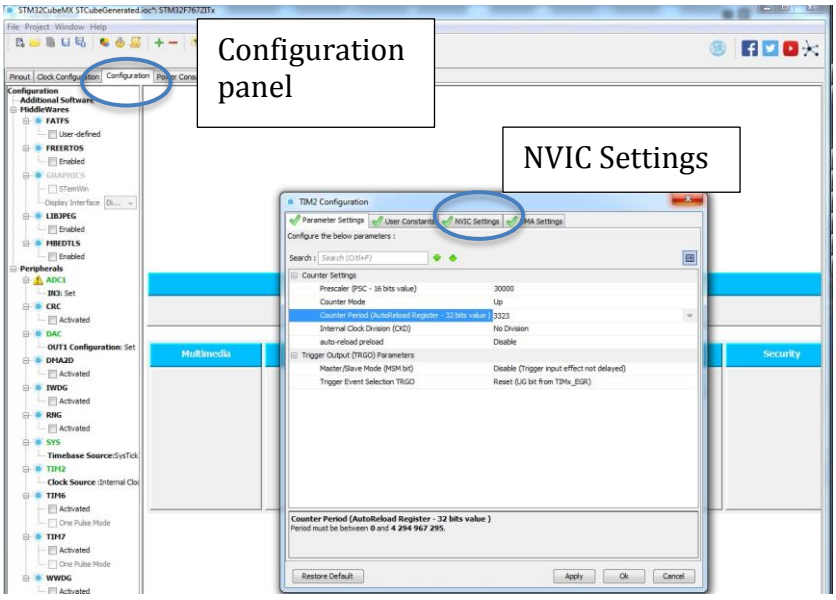

Figure 6: Timer configuration

The last thing to do here is to enable the interruption for the timer. To do this, click on the NVIC Settings panel and enables the TIM2 global interruption by checking Enabled. Then click on Ok.

Now, we are going to configure the ADC component for that click on ADC1 panel; the figure below appears.

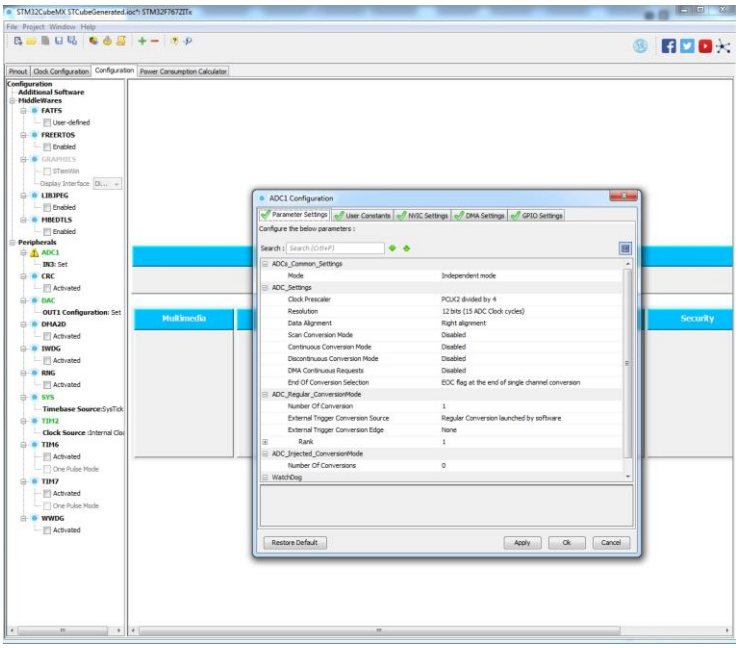

Figure 7: ADC configuration

Here we are going to use the ADC in its simplest mode so we choose the configuration presented in figure 7 (for more details see the ADC document that I wrote).

The last component to configure is the DAC. For that, we click on the DAC configuration panel and we choose the settings presented in the Figure below.

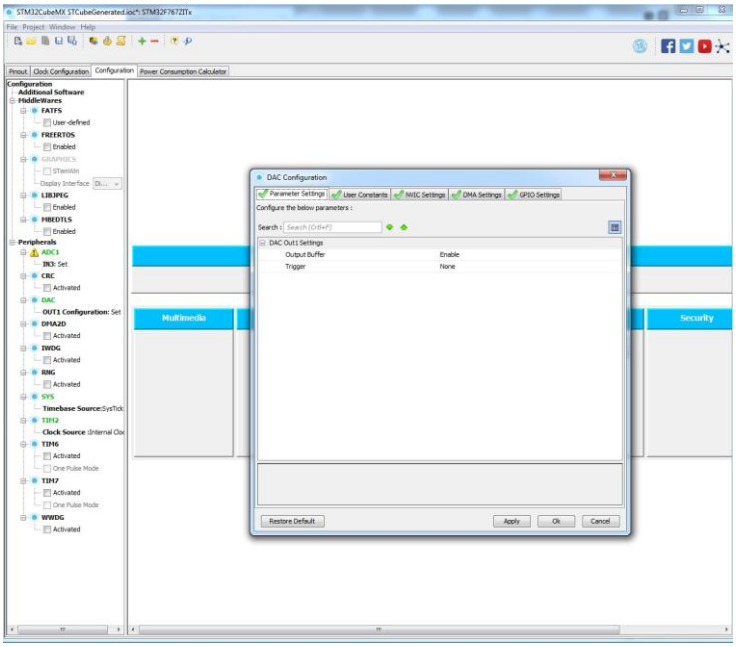

Figure 8: DAC configuration panel

Now, we configured all the necessary components to realize the application. As, most of them used interruption capability we have to define the priority of these interruptions in order to obtain the desired behavior.

To do this, choose the NVIC configuration panel and change the preemption priority so that the TIMER 2 had the higher priority level (more the priority is high most the number is near 0).

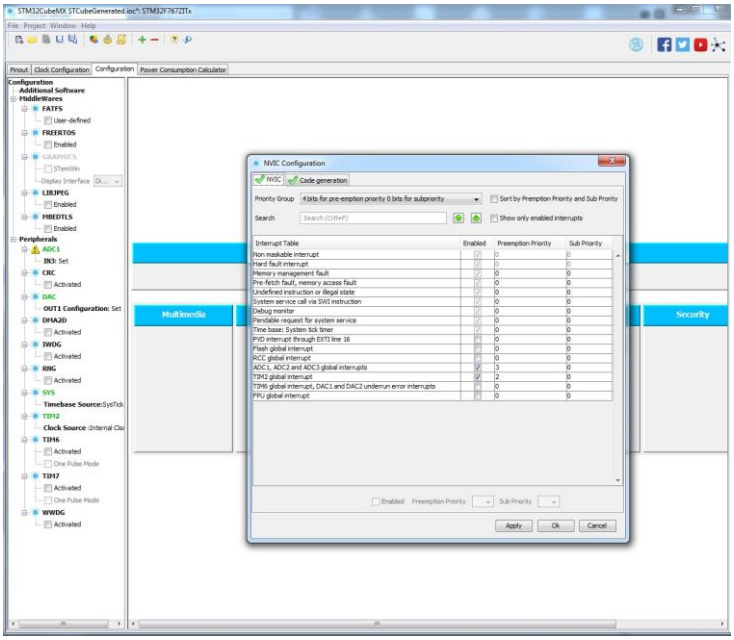

Figure 9: NVIC configuration panel

After this last step, all the configurations for this application have been realized so we can generate the code for our application. To do this, click on the code generation icon as shown in Figure 10. Then give the name for your project and the path for its location finally choose the toolchain/ide; in this example, I chose to use the MDK-ARM environment.

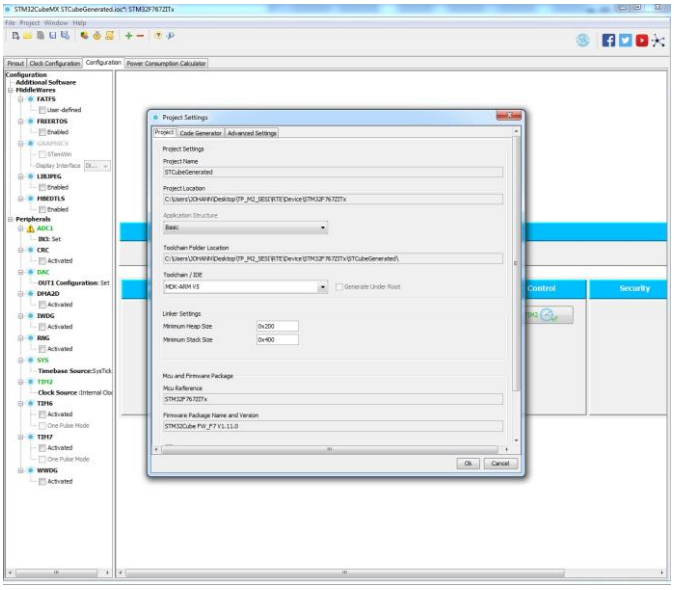

Figure 10: Project configuration panel

After clicking on ok, CubeMx generates the application code and create the MDK-ARM project as shown in the figure below.

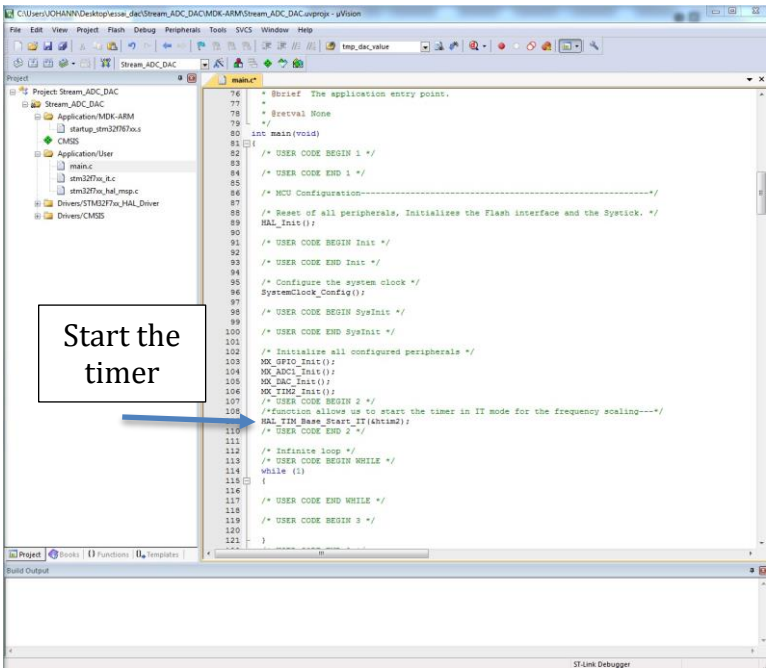

Figure 11: MDK-ARM code generation

At the code level, few things are to do indeed, we only want that the sample provided from the ADC to be transferred to the DAC at each sampling period. Therefore, the first thing to do is to start the timer in interruption mode. In the main.c file it is the only thing to write.

The second step is to write the code for the interrupt functions. When the timer interrupt occurs, we have to start the ADC conversion and in function of the etat value (0 or 1) either we transfer the ADC value to the DAC output or we transfer a '0' value to the DAC output. This is just an example; the behavior can be another one.

The code to realize this behavior is shown in figure 12 and 13.

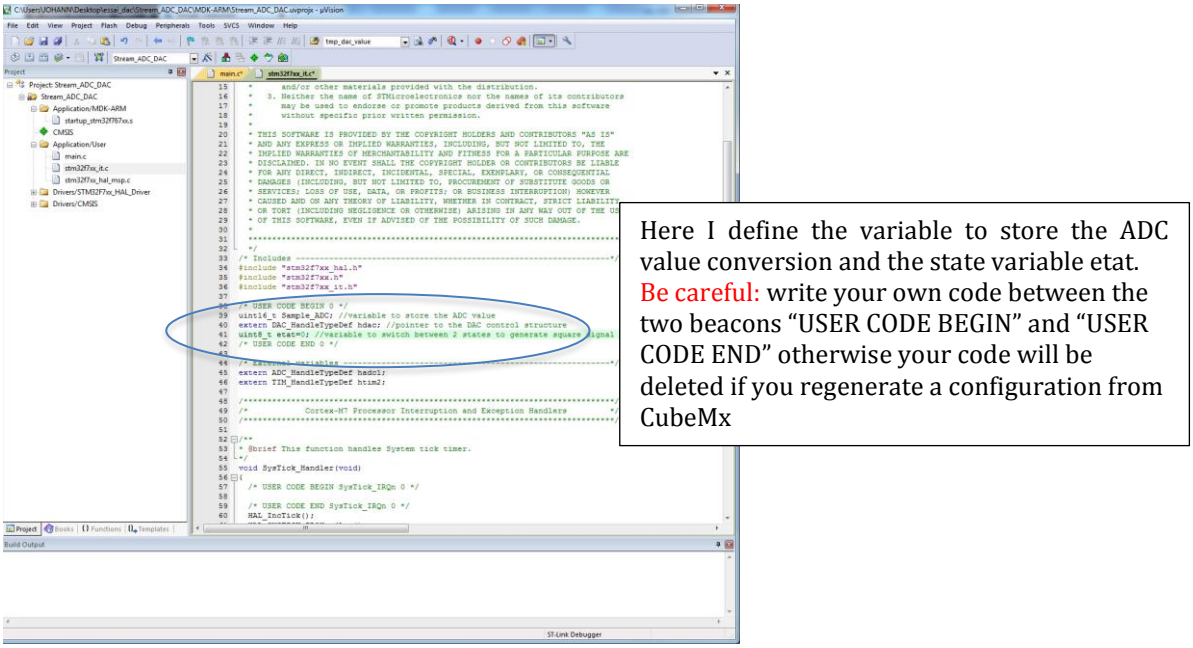

Figure 12: Interruption variable statement

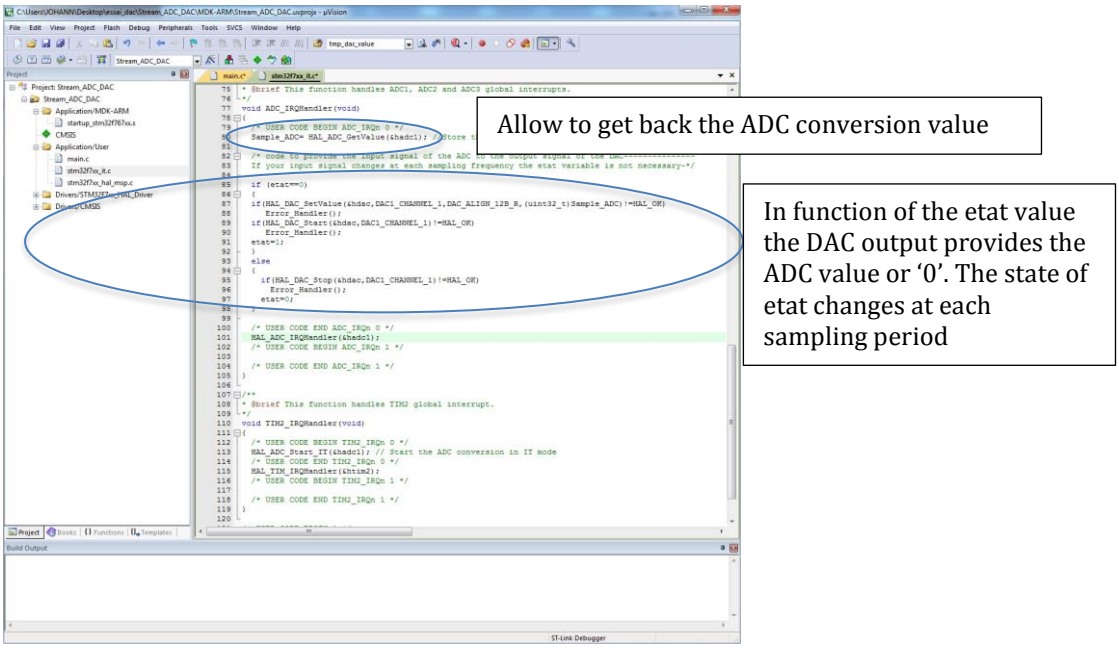

Figure 13: interruption code function

At the end we have to compile the code in order to obtain the executable file for the STM32F7. When the compilation is done without error, we can transfer the code into the board. A demo video can be found here: https://youtu.be/_u9r6jeGoiE