Use of I²C Component on Microcontroller

Targeted competences: Use of I²C triple axis accelerometer with a sampling frequency of 100Hz

Hardware: STM32F7 Nucleo board

Framework: Keil µvision (V5.25.2.0) and CubeMX (V5.0.1) from STMircoelectronics

The aim of this document is to show how to use an I²C bus in order to read the data from a 3-Axis accelerometer. In function of values provided by the accelerometer, we will switch on a led.

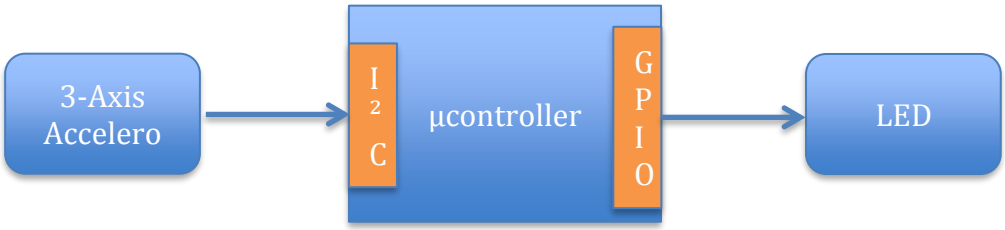

Figure 1: Acquisition Stream

To obtain the sampling frequency we need to add a timer into this scheme. This timer will avoid making a blocking polling on the I²C bus and avoid obtaining too many unnecessary values from the sensor. Here, we choose to sample at 400Hz; this value can be choose in function of your application (respect the frequency period of your sensors).

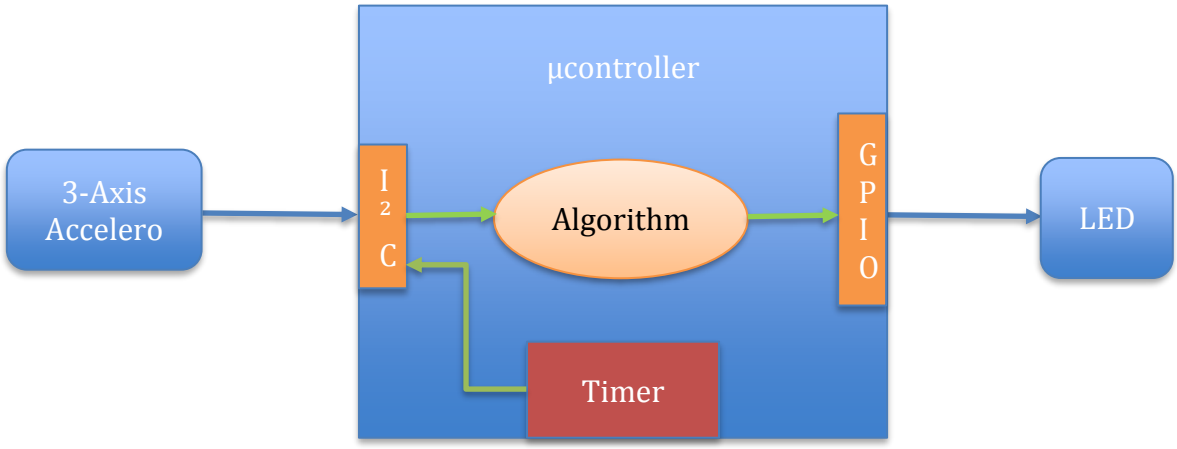

Figure 2: Global Scheme

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 choice of the board (attention the configuration can depend on the board used) you obtain the figure below.

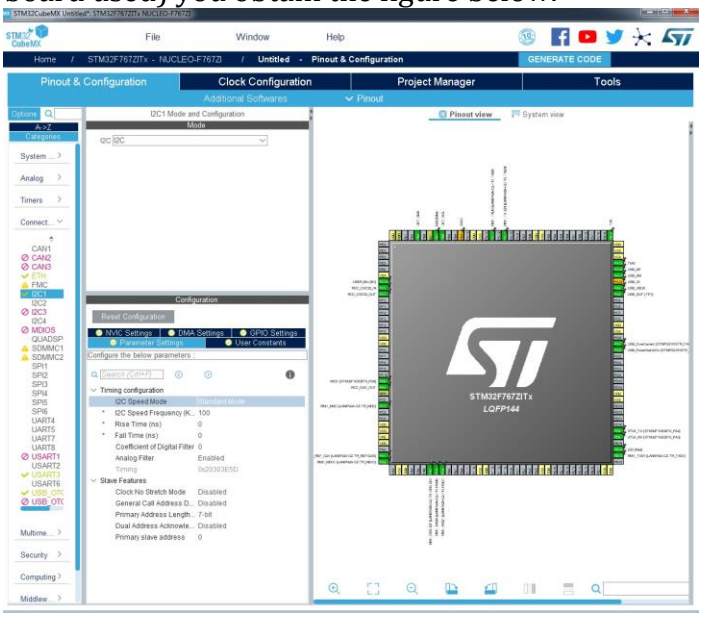

Figure 3: CubeMx configuration for I²C

Here, I chose to use the I²C bus in standard mode so the frequency of the bus is equal to 400KHz furthermore the address length for the peripherals is in 7-bit mode (see our component datasheet if you use a different peripheral than the one is used). The pin PB6 is used for the SCL signal and the pin PB9 for the SDA signal. Here I cannot explain the behavior of the I²C bus; you can find information at the following address: https://en.wikipedia.org/wiki/I%C2%B2C

I configure also the NVIC (enabled the interruption in NVIC settings) in order to interruptions generated when an error or an event appears on the bus.

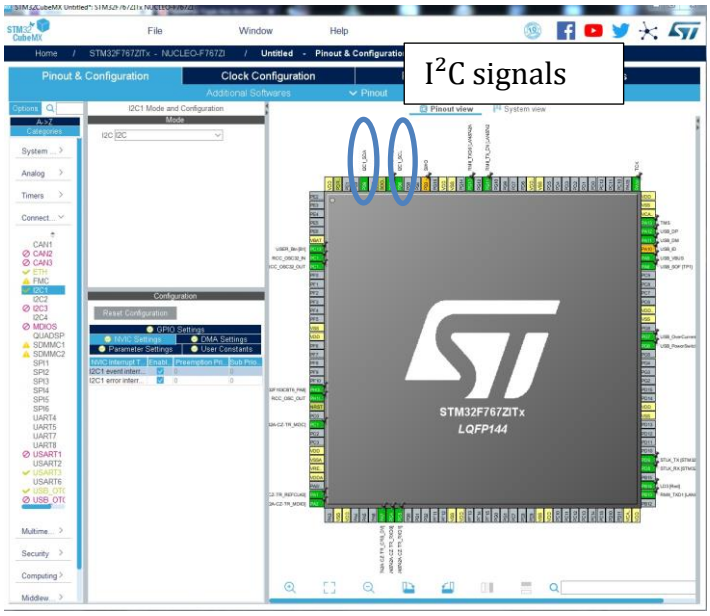

Figure 4: I²C signals and NVIC settings

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.

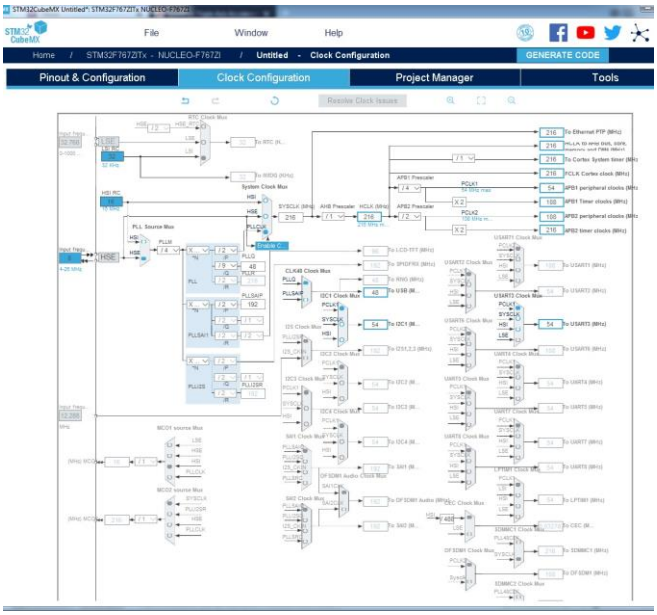

Figure 5: Clock configuration for the board

Now we are going to configure the timer so that generates the sampling frequency. First click and configuration panel then on TIM1 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 65535 (16-bit timer) the minimum of the frequency scaling can be 3296Hz.

For this example, I would like a sampling frequency of 400Hz so I chose the following values: prescaler value 216 (in this case, the clock frequency of the timer will be 1MHz instead of 216 MHz), the counter period value 2500 (2500 ticks at 1 MHz represents a sampling period of 2.5ms).

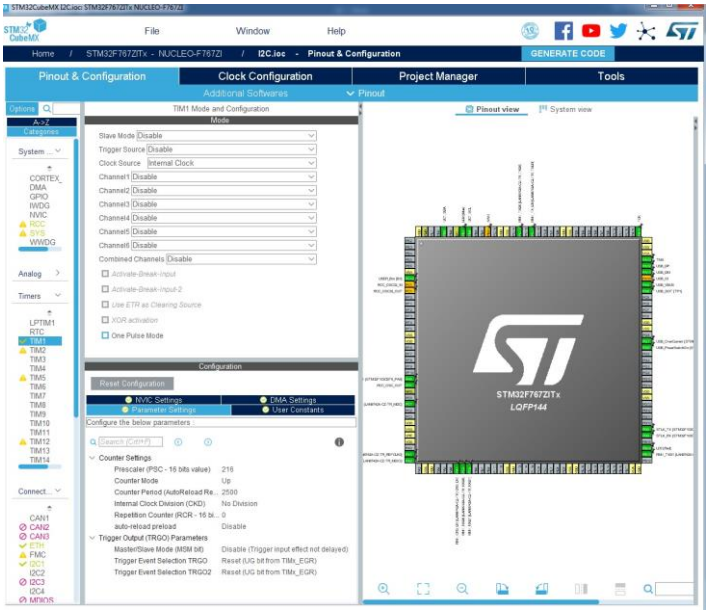

Figure 6: Timer configuration to create the 400Hz sampling frequency

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 TIM1 update interrupt and TIM10 global interrupt by checking Enabled.

Figure 7: Timer interrupt configuration

Now, we are going to configure the project manager to generate the different useful files.

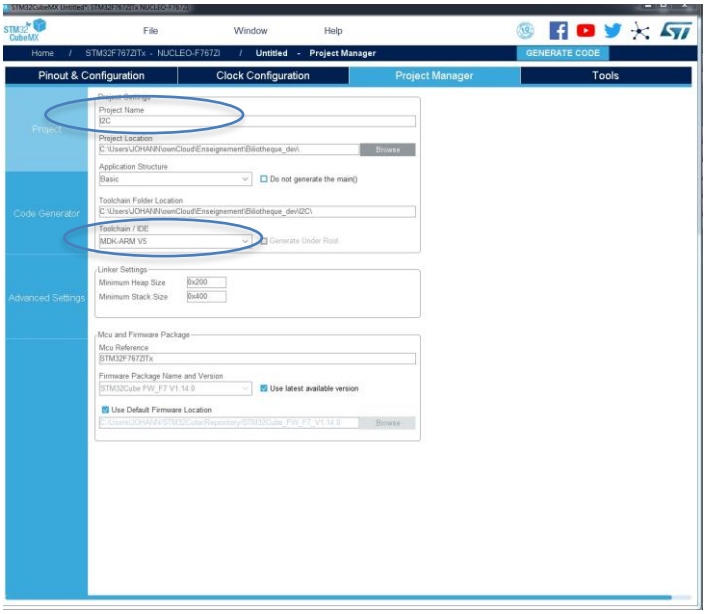

Figure 8: Project manager configuration

You must give a name for your project and choose the toolchain that you want to use; here I choose to use MDK-ARM toolchain.

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

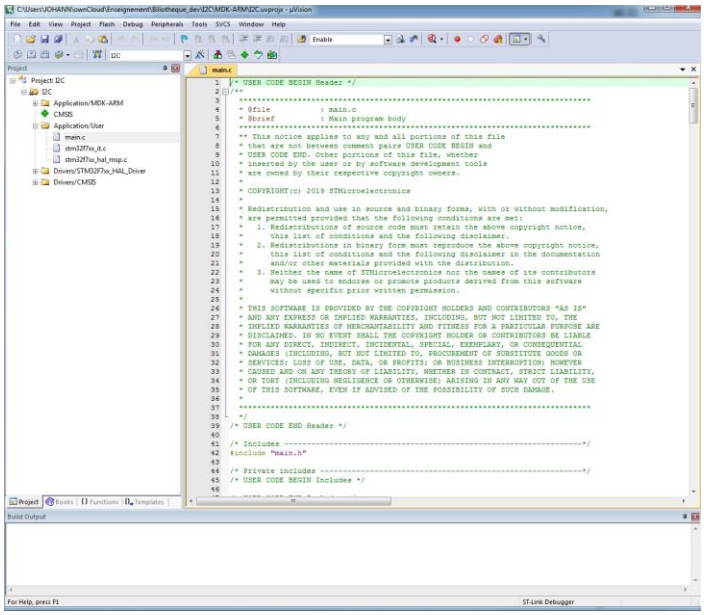

Figure 9: Code generation

The next step is to define all the register addresses of the peripheral. In my case, I use a LIS3DH, which is a 3-axis accelerometer. In order to define all the addresses I wrote an include file (accelero.h); you can find below a part of this file.

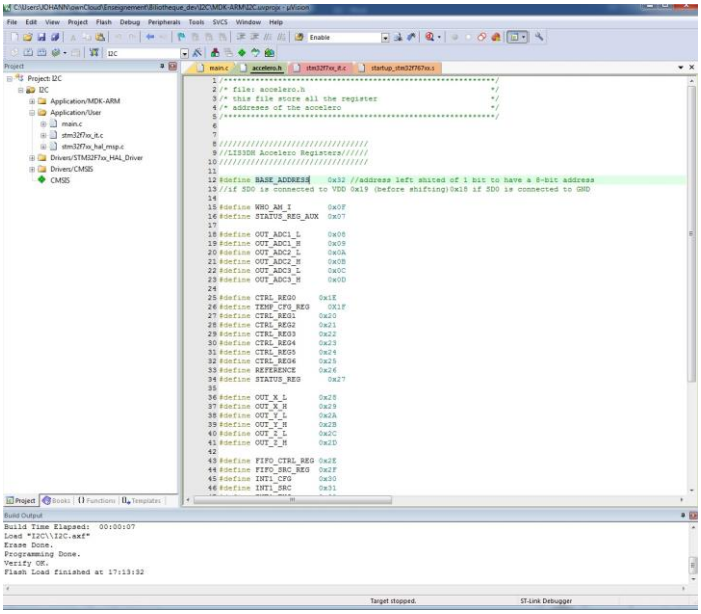

Figure 10: Include file defining the addresses of the LIS3DH component

Now, we must setup the I^2C sensor and start the timer in order to generate the sampling frequency of 400Hz. To do that we have just to use the HAL driver function for the timer and write a function to setup the sensor.

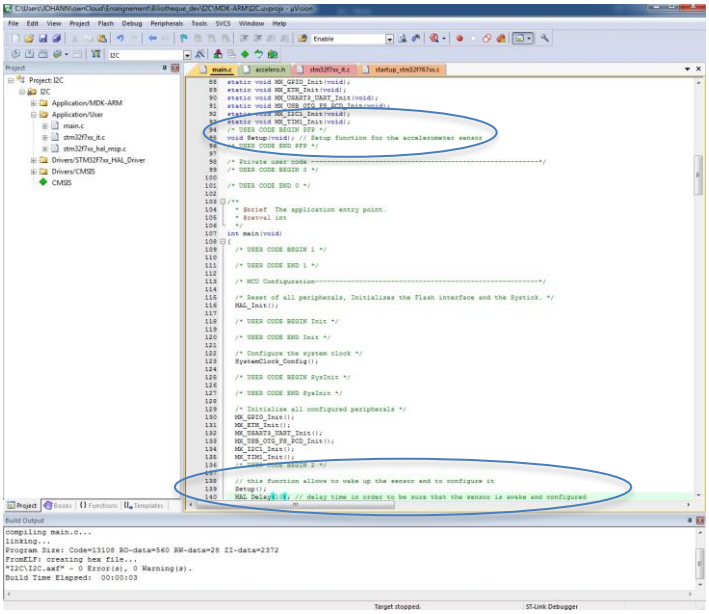

Figure 11: Declaration of the Setup Function and its call

Below, you can find the Setup function code that allows configuring the LIS3DH sensor. I configure it to use a 400Hz sampling frequency and enable all the axis of the accelerometer.

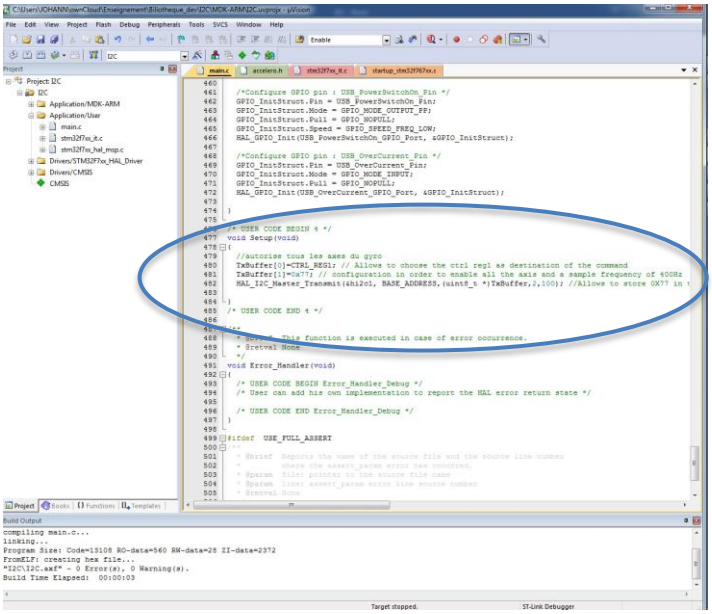

Figure 12: Setup function code to configure the sensor

Now, the sensor is ready so we can start the timer in IT mode in order to generate the sampling frequency.

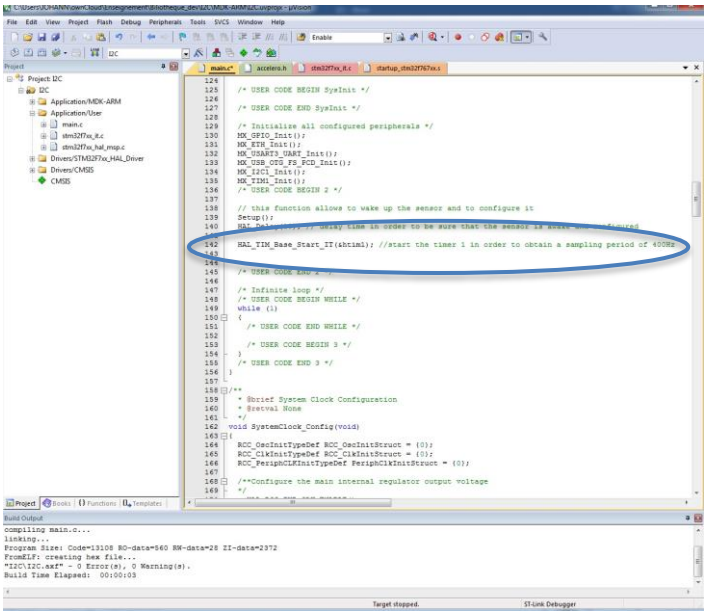

Figure 13: Starting the sampling timer

To be able to store the 3-axis value of the accelerometer, we declare three variables to store them.

Paid attention, each axis uses two registers to store its value one for the 8-bit LSB and another for the 8-bit MSB. Therefore, to store the complete value I shift the 8-bit MSB to the left (shifted by 8) and then I do a mask (a OR function) between this value and the 8-bit LSB otherwise the MSB value will be modified. I realized that for the three axis of course. This requires realizing two reads one for each register as you can see on the next figure.

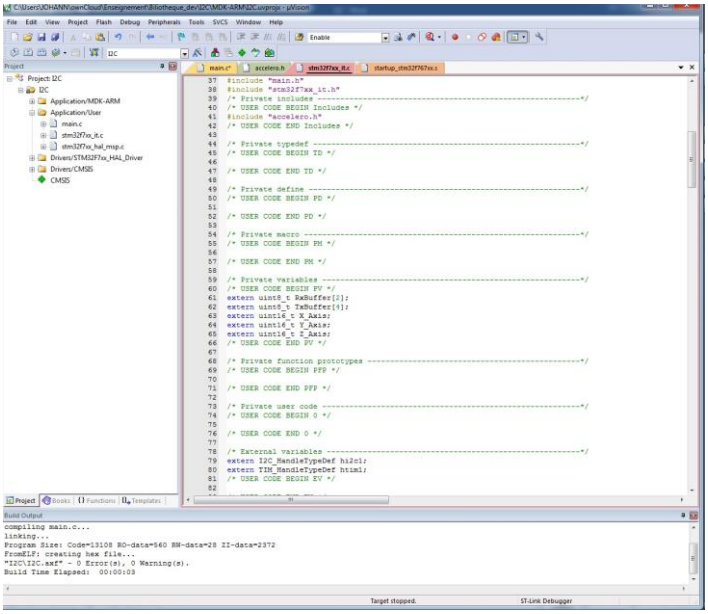

Figure 14: Variable declarations to store the 3-axis values

The first step is to know if the peripheral is the good one (in our case the LIS3DH). To do that I read the Who_am_I register of the peripheral and I compare it with the internal address of the sensor (here the address value is 0x33). If the sensor is the good one then I read the two register of each axis. To do this, I write the address of the desired register in the TxBuffer and write it on the I^2C bus (by using the HAL_I2C_Master_Transmit function), then I read the returned value on the I^2C bus (by using the HAL_I2C_Master_Receive function). This entire job is done in the timer interrupt function.

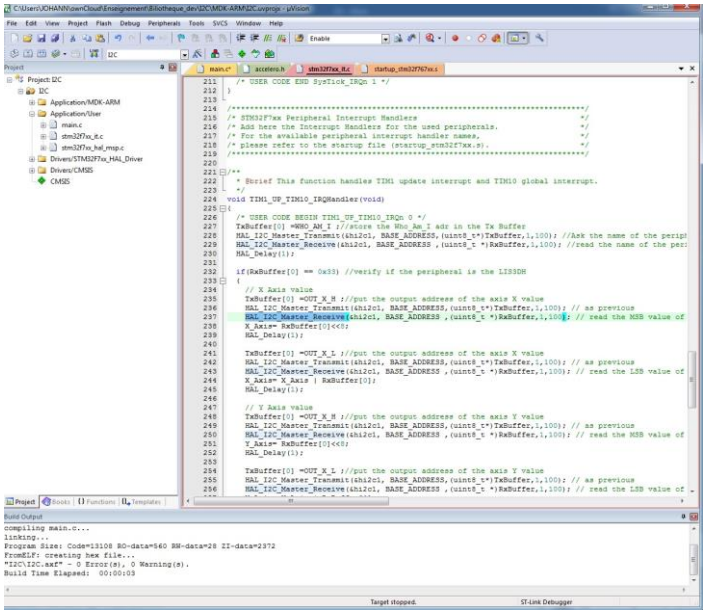

Figure 15: Code to read the axis values

To be sure that the code is ok I switch on a led (LD3) if the X-axis value is greater than 10000 (the value of the x-axis can be from 0 to 65535 since it is stored in 16-bit variable).

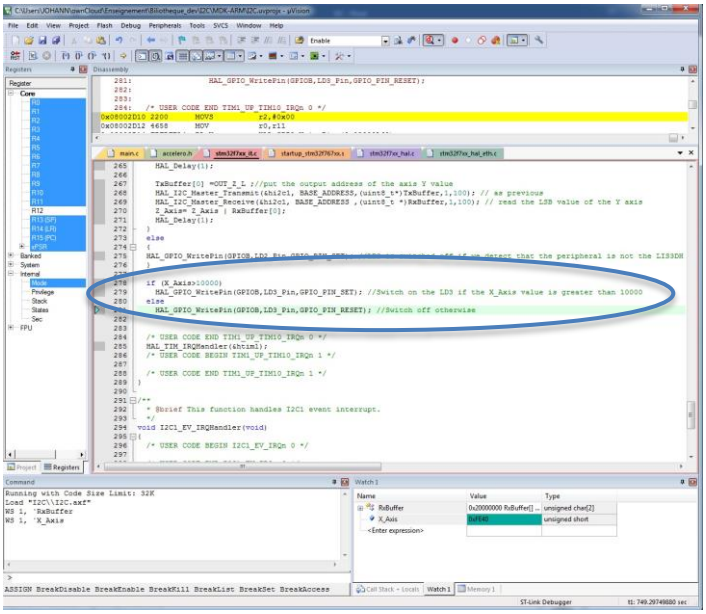

Figure 16: Code to switch on or off the LD3 in function of the X-axis value

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/cdO2M0oV9yo