Internship Arobs 2021

Car Project

**Mentors, Interns,**

***Irina Leoveanu Iulia Tia Stoian***

***Andrei Ciocan Nicoleta Trifan***

***Vlad Dobra Andrei Borza***

***Balazs Benedek***

***Laza Sergiu***

***Sebastian Cioată***

[**CAN GATEWAY**](#_ybtwob2l1ghi) **3**

[Introduction](#_pgtm2ksjue8x) 3

[STM32CubeMX Configuration](#_esk1cbk3b93l) 3

[Code Implementation](#_67w137yutmwb) 6

[Configure the CAN filter](#_kwsu58ami56n) 6

[Start the CAN communication](#_msmjcbb6rsle) 7

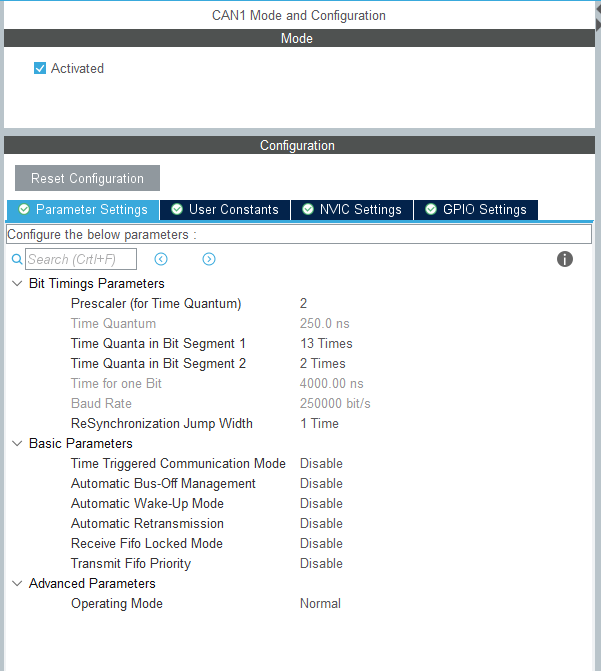
[Activation of CAN Interrupts](#_ji9rxyuq093h) 7

[Transmit data using CAN communication](#_cb4229oaw5ax) 8

[Receive data using CAN communication](#_g36z8mtne1ab) 9

# **CAN GATEWAY**

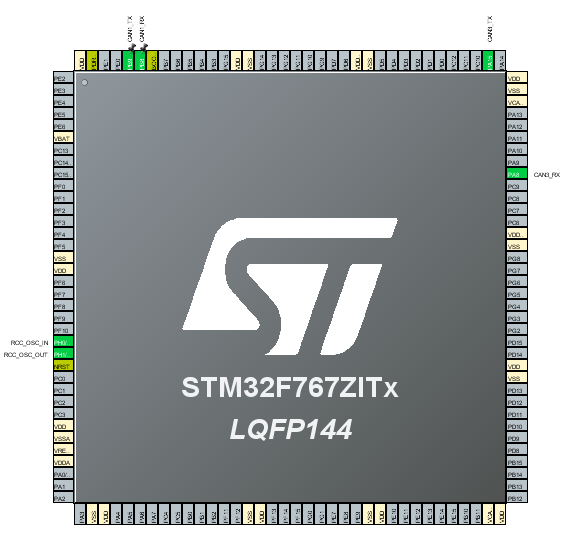
## **Introduction**

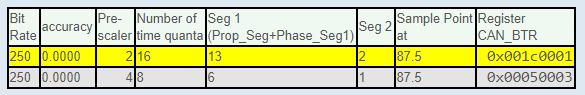
A CAN gateway acts as an interface between different networks, enabling the possibility of connecting different CAN networks (or sub-networks) with different baud rates and protocols, or integrating with different network types.

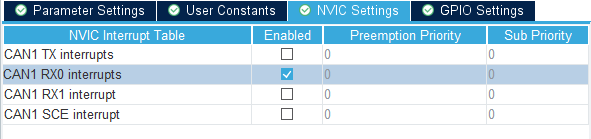
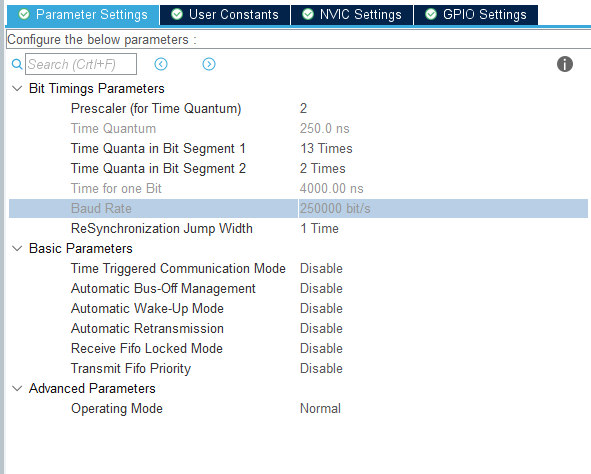


## **STM32CubeMX Configuration**

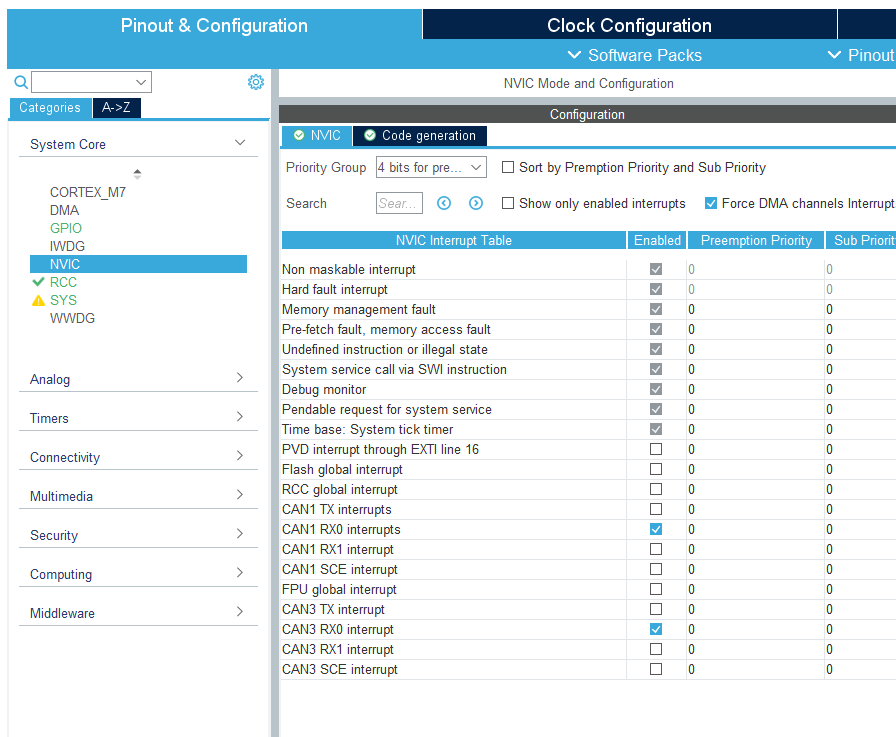
In our case we will use the STM32 F767ZI development board as the CAN gateway. This board has 3 CANs (CAN1, CAN2 AND CAN3), but for our project we will use only CAN1 and CAN3.

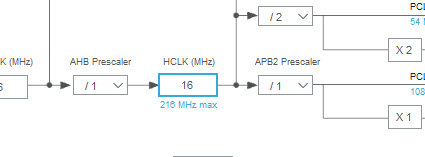


To configure the two CAN buses we will take into account the frequency at which the board works, Sample-Point which is usually 87.5%, SJW which is 1 preferred and also the value in kbit / s for which we want to calculate these parameters. To make the process easier we will use the following website where we can introduce our desired values <http://www.bittiming.can-wiki.info/>. In order to obtain a Time Quantum of 250.0 ns and a Baud Rate of 250 kbit/s the following parameters should be used. 

The same settings will be made for CAN3. Next we will activate the interrupts on RX because we want to use the method of interruptions and not polling. We go to the NVIC Settings menu and activate CAN1 RX0 interrupts.

Finally, after applying the same configurations for CAN3, we should have the following configurations in System Core - NVIC.



In Clock Configuration we will set HCLCK (MHz) to 16 MHz and leave the settings made automatically by STM32CubeMX.

## **Code Implementation**

### **Configure the CAN filter**

To prepare CAN communication and filter the information and messages received we will use the following function:

*HAL\_StatusTypeDef HAL\_CAN\_ConfigFilter(CAN\_HandleTypeDef \*hcan, CAN\_FilterTypeDef \*sFilterConfig)*

We will need to declare a variable of type CAN\_HandleTypeDef in order to use this function. We will declare two variables in the User Code Begin Private variables area, one for CAN1 and one for CAN3.

*CAN\_HandleTypeDef hcan1;*

*CAN\_HandleTypeDef hcan3;*

For *CAN\_FilterTypeDef \*sFilterConfig* we need to declare a variable in the User Code Begin Private Variable area *CAN\_FilterTypeDef sFilterConfig;* After declaration we have to implement this structure in oder to have a working communication.

*sFilterConfig.FilterFIFOAssignment = CAN\_FILTER\_FIFO0;* specifies the FIFO (0 or 1) which will be assigned to the filter.

*sFilterConfig.FilterIdHigh = 0;* specifies the filter identification number (MSBs for a 32-bit configuration, first one for a 16-bit configuration). This parameter must be a number between Min\_Data = 0x0000 and Max\_Data = 0xFFFF.

*sFilterConfig.FilterIdLow = 0;* specifies the filter identification number (LSBs for a 32-bit configuration, second one for a 16-bit configuration). This parameter must be a number between Min\_Data = 0x0000 and Max\_Data = 0xFFFF.

*sFilterConfig.FilterMaskIdHigh = 0;* specifies the filter mask number or identification number, according to the mode (MSBs for a 32-bit configuration, first one for a 16-bit configuration). This parameter must be a number between Min\_Data = 0x0000 and Max\_Data = 0xFFFF.

*sFilterConfig.FilterIdLow = 0;* specifies the filter mask number or identification number, according to the mode (LSBs for a 32-bit configuration, first one for a 16-bit configuration). This parameter must be a number between Min\_Data = 0x0000 and Max\_Data = 0xFFFF.

*sFilterConfig.FilterScale = CAN\_FILTERSCALE\_32BIT;* specifies the filter scale. We can choose between 16 bit and 32 bit.

*sFilterConfig.FilterActivation = ENABLE;* enable or disable the filter.

Finally we have to call the following function for the configuration to be successful.

*HAL\_CAN\_ConfigFilter(&hcan1, &sFilterConfig);*

An example on how we can configure can be seen here for STID[10:0] & EXTID[17:13] and EXID[12:5] & 3 Reserved bits.

*sFilterConfig.FilterIdHigh = ((filter\_id << 5) | (filter\_id >> (32 - 5))) & 0xFFFF;*

*filter.FilterIdLow = (filter\_id >> (11 - 3)) & 0xFFF8;*

*sFilterConfig.FilterMaskIdHigh = ((filter\_mask << 5) | (filter\_mask >> (32 - 5))) & 0xFFFF;*

*sFilterConfig.FilterMaskIdLow = (filter\_mask >> (11 - 3)) & 0xFFF8;*

### **Start the CAN communication**

To program the STM32 F767ZI development board we will start by studying the stm32f7xx\_hal\_can.h and stm32f7xx\_hal\_can.c files. In these files and in the datasheet of the development board we will find the necessary information to be able to program it so that the 2 CAN buses (CAN1 and CAN3) will be available. We will start with the first function that will start the CAN communication. The prototype of the function is the following:

*HAL\_StatusTypeDef HAL\_CAN\_Start(CAN\_HandleTypeDef \*hcan)*

The variable is already defined so we will use the following syntax in order to start the CAN communications:

*HAL\_CAN\_Start(&hcan1);*

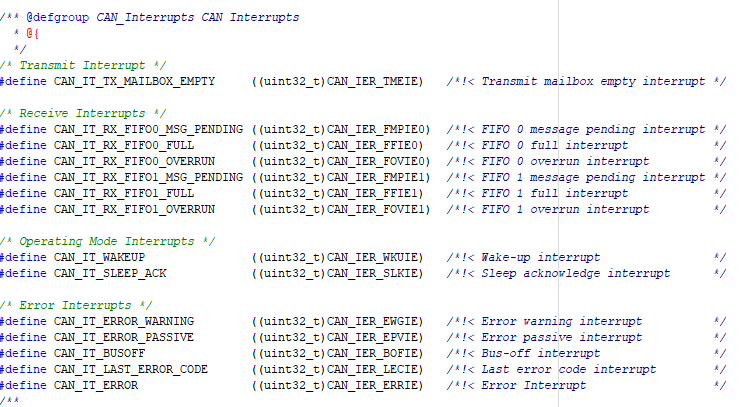
*HAL\_CAN\_Start(&hcan3);*

### **Activation of CAN Interrupts**

After activating the communication, we will have to activate the reception interruptions. To do this we will use another function, its prototype being the following:

*HAL\_StatusTypeDef HAL\_CAN\_ActivateNotification(CAN\_HandleTypeDef \*hcan, uint32\_t ActiveITs);*

For this function we need two parameters. One of these was previously created, the second parameter ActiveITs, must be chosen from a list of parameters in the header file. These parameters can be chosen from CAN\_Interrupts.



In our case we will use *CAN\_IT\_RX\_FIFO0\_MSG\_PENDING*, because we want to activate the interruption all the time when we have a message on the reception line, using FIFO0. To call the functions we will use the following syntax:

*HAL\_CAN\_ActivateNotification(&hcan1, CAN\_IT\_RX\_FIFO0\_MSG\_PENDING);*

*HAL\_CAN\_ActivateNotification(&hcan3, CAN\_IT\_RX\_FIFO0\_MSG\_PENDING);*

### **Transmit data using CAN communication**

We are going to add messages on the transmission line. Here we will use the following function:

*HAL\_StatusTypeDef HAL\_CAN\_AddTxMessage(CAN\_HandleTypeDef \*hcan, CAN\_TxHeaderTypeDef \*pHeader, uint8\_t aData[], uint32\_t \*pTxMailbox)*

The first parameter is already defined, we will have to define the other three parameters.

*CAN\_TxHeaderTypeDef \*pHeader* represents the data structure of the transmitter. Here we will specify the size of the data area, the transmitter id and other important parameters for making the connection. We will declare the variable with the following format *CAN\_TxHeaderTypeDef pTxHeader;.*

Moving forward we will initialize the structure. We will start with DLC. *pTxHeader.DLC = 8;* DLC specifies the length of the frame that will be transmitted. This parameter must be a number between Min\_Data = 0 and Max\_Data = 8.

*pTxHeader.IDE = CAN\_ID\_STD;* IDE Specifies the type of identifier for the message that will be transmitted. We can choose between Standard Id CAN\_ID\_STD and Extended Id CAN\_ID\_EXT.

*pTxHeader.RTR = CAN\_RTR\_DATA;* specifies the type of frame for the message that will be transmitted. CAN\_RTR\_DATA for Data frame and CAN\_RTR\_REMOTE for Remote frame.

*pTxHeader.StdId = 0x244;* specifies the standard identifier. This parameter must be a number between Min\_Data = 0 and Max\_Data = 0x7FF.

*uint8\_t aData[]* represents the 8-bit data packet. It will be defined in the User Code Begin Private Variables area.

*uint32\_t \*pTxMailbox* it is the mailbox where the message will be stored, we will define in the User Code Begin Private variables area an uin32\_t variable.

### **Receive data using CAN communication**

To receive data, as mentioned above, we will use interrupts. Therefore, we will go in the interrupt request handler intended for each CAN and we will use the following function:

*HAL\_StatusTypeDef HAL\_CAN\_GetRxMessage(CAN\_HandleTypeDef \*hcan, uint32\_t RxFifo, CAN\_RxHeaderTypeDef \*pHeader, uint8\_t aData[])*

The first parameter is already defined, we will continue with the other parameters.

*uint32\_t RxFifo* here we have to select on which RxFifo we want to receive data. We can chose between CAN\_RX\_FIFO0 and CAN\_RX\_FIFO1.

*CAN\_RxHeaderTypeDef \*pHeader* will be declared in the User Code Begin Private Variables area, *CAN\_RxHeaderTypeDef pRxHeader;* This is the struct that will be used in order to receive a message on the CAN communication, and to receive the proper information about the sender.

*uint8\_t aData[]* is the data received from the sender. In this variable we will store the message transmitted. It will be also declared in the User Code Begin Private Variable area.